

Run-time Library Reference

© 1998 Sony Computer Entertainment Inc.

Publication date: August 1998

Sony Computer Entertainment America
919 E. Hillsdale Blvd., 2nd floor
Foster City, CA 94404

Sony Computer Entertainment Europe
Waverley House
7-12 Noel Street
London W1V 4HH, England

The *Run-time Library Reference* manual is supplied pursuant to and subject to the terms of the Sony Computer Entertainment PlayStation® License and Development Tools Agreements, the Licensed Publisher Agreement and/or the Licensed Developer Agreement.

The *Run-time Library Reference* manual is intended for distribution to and use by only Sony Computer Entertainment licensed Developers and Publishers in accordance with the PlayStation® License and Development Tools Agreements, the Licensed Publisher Agreement and/or the Licensed Developer Agreement.

Unauthorized reproduction, distribution, lending, rental or disclosure to any third party, in whole or in part, of this book is expressly prohibited by law and by the terms of the Sony Computer Entertainment PlayStation® License and Development Tools Agreements, the Licensed Publisher Agreement and/or the Licensed Developer Agreement.

Ownership of the physical property of the book is retained by and reserved by Sony Computer Entertainment. Alteration to or deletion, in whole or in part, of the book, its presentation, or its contents is prohibited.

The information in the *Run-time Library Reference* manual is subject to change without notice. The content of this book is Confidential Information of Sony Computer Entertainment.

PlayStation and PlayStation logos are registered trademarks of Sony Computer Entertainment Inc. All other trademarks are property of their respective owners and/or their licensors.

Table of Contents

ABOUT THIS MANUAL

Changes Since Last Release	ix
Related Documentation	xiv
Manual Structure	xiv
Developer Reference Series	xiv
Typographical Conventions	xv
Developer Support	xv

CH 1 KERNEL LIBRARY

Structures	1-3
Functions	1-9

CH 2 STANDARD C LIBRARY

Functions	2-3
-----------	-----

CH 3 MATH LIBRARY

Functions	3-3
-----------	-----

CH 4 MEMORY CARD LIBRARY

Functions	4-3
-----------	-----

CH 5 EXTENDED MEMORY CARD LIBRARY

Functions	5-3
-----------	-----

CH 6 DATA COMPRESSION LIBRARY

Structures	6-3
Functions	6-5

CH 7 BASIC GRAPHICS LIBRARY

Structures	7-5
Functions	7-37
Macros	7-130

CH 8 BASIC GEOMETRY LIBRARY

Structures	8-5
Functions	8-22

CH 9 EXTENDED GRAPHICS LIBRARY

Structures	9-3
Functions	9-33
Macros	9-121
External Variables	9-123

CH 10 CD/STREAMING LIBRARY

Structures	10-3
Functions	10-8

CH 11 EXTENDED CD-ROM LIBRARY

Structures	11-3
Functions	11-7

CH 12 CONTROLLER/PERIPHERALS LIBRARY

Functions	12-3
-----------	------

CH 13 LINK CABLE LIBRARY

Functions	13-3
Macros	13-7

CH 14 EXTENDED SOUND LIBRARY

Structures	14-5
------------	------

Functions	14-15
CH 15 BASIC SOUND LIBRARY	
Structures	15-5
Functions	15-15
CH 16 SERIAL INPUT/OUTPUT LIBRARY	
Functions	16-3
CH 17 HMD LIBRARY	
Structures	17-3
Functions	17-25

List of Figures

Figure 7-1	7-22
Figure 7-2	7-24
Figure 8-1	8-218
Figure 8-2	8-220
Figure 9-1: Projection	9-87
Figure 15-1: ADSR Conceptual Diagram	15-117

List of Tables

Table 1-1: Description of Each Bit of the Cause Register	1-33
Table 1-2: Description of Each Bit of the Status Register	1-37
Table 1-3: Data Formats Stored in the Receive Buffer	1-42
Table 1-4: <i>flag</i> Designations	1-50
Table 1-5: Common Parts	1-54
Table 2-1	2-6
Table 2-2	2-16
Table 2-3	2-21
Table 2-4	2-37
Table 2-5	2-42
Table 2-6	2-49
Table 2-7	2-50
Table 3-1	3-3
Table 3-2	3-4
Table 3-3	3-7
Table 3-4	3-18
Table 3-5	3-19
Table 3-6	3-21
Table 3-7	3-26
Table 3-8	3-27
Table 4-1: Posts an event on completion of processing	4-9
Table 4-2: Posts an event on completion of processing	4-11
Table 4-3: Posts an event on completion of processing	4-12
Table 4-4: Posts an event on completion of processing	4-13
Table 4-5	4-14
Table 4-6: Posts an event on completion of processing	4-16
Table 6-1	6-7
Table 6-2	6-9
Table 6-3	6-12
Table 6-4	6-14
Table 7-1	7-8
Table 7-2	7-52
Table 7-3	7-96
Table 7-4	7-97

Table 7-5	7-98
Table 7-6	7-99
Table 7-7	7-103
Table 7-8	7-109
Table 7-9	7-116
Table 7-10	7-118
Table 7-11	7-121
Table 7-12	7-128
Table 7-13	7-128
Table 8-1	8-14
Table 8-2	8-15
Table 8-3	8-57
Table 8-4	8-58
Table 8-5	8-59
Table 8-6	8-60
Table 8-7	8-61
Table 8-8	8-62
Table 8-9	8-63
Table 8-10	8-64
Table 9-1	9-6
Table 9-2	9-6
Table 9-3: Lighting modes	9-10
Table 9-4: Semi-transparency Rate	9-10
Table 9-5: Lighting Modes	9-13
Table 9-6: Semi-transparency Rate	9-26
Table 9-7: Resolution and Aspect Ration	9-87
Table 10-1	10-8
Table 10-2: Primitive command overview	10-9
Table 10-3: Primitive commands that take arguments and their arguments	10-10
Table 10-4: Return values of primitive commands	10-10
Table 10-5	10-23
Table 10-6	10-25
Table 10-7	10-37
Table 10-8	10-37
Table 10-9	10-38
Table 10-10	10-38
Table 10-11	10-40
Table 10-12	10-42
Table 10-13	10-44
Table 10-14	10-44
Table 10-15	10-56
Table 10-16	10-58
Table 12-1	12-6
Table 12-2: System Clock/Pixel Clock Variable Table	12-7
Table 12-3: Buffer Data Format	12-9
Table 12-4: 16 button analog, analog joystick	12-9
Table 12-5: Gun controller, 16 buttons	12-10
Table 12-6: Multi-tap received data configuration	12-10
Table 12-7	12-45
Table 13-1: Control Line Status	13-13
Table 13-2: Communication Mode	13-16
Table 13-3: Control Line Status	13-22
Table 13-4: Communication Mode	13-23
Table 13-5: Serial Controller Status	13-26
Table 14-1	14-16
Table 14-2	14-27
Table 14-3	14-27

vi Table of Contents

Table 14-4	14-28
Table 14-5	14-29
Table 14-6	14-36
Table 14-7	14-44
Table 14-8	14-58
Table 14-9	14-74
Table 14-10	14-82
Table 14-11	14-82
Table 14-12	14-82
Table 14-13	14-83
Table 14-14	14-88
Table 14-15: Reverb Type Overview (See Sound Delicatessen DSP)	14-123
Table 14-16	14-135
Table 14-17	14-136
Table 14-18	14-140
Table 15-1	15-18
Table 15-2	15-28
Table 15-3	15-29
Table 15-4	15-31
Table 15-5	15-32
Table 15-6	15-33
Table 15-7	15-39
Table 15-8	15-66
Table 15-9	15-67
Table 15-10: Arrangement of Data	15-74
Table 15-11	15-75
Table 15-12	15-76
Table 15-13	15-77
Table 15-14	15-78
Table 15-15	15-80
Table 15-16	15-81
Table 15-17	15-81
Table 15-18	15-81
Table 15-19	15-81
Table 15-20	15-88
Table 15-21	15-88
Table 15-22	15-91
Table 15-23	15-92
Table 15-24	15-94
Table 15-25	15-96
Table 15-26	15-96
Table 15-27	15-97
Table 15-28	15-99
Table 15-29	15-103
Table 15-30: Reverb Mode and Other Attributes	15-103
Table 15-31: Volume Occupied by Reverb Mode In Sound Buffer	15-104
Table 15-32	15-106
Table 15-33	15-106
Table 15-34	15-108
Table 15-35	15-114
Table 15-36: Volume Mode and Volume Setting Ranges	15-115
Table 15-37: Pitch Specification Values and Interval	15-116
Table 15-38: Note Specification Values	15-116
Table 15-39: Waveform Data Sample Note Specification Values	15-116
Table 15-40: Parameters and Structure Members	15-117
Table 15-41: Rate and Level Setting Ranges	15-117
Table 15-42: ADSR Rate Modes	15-117

Table 15-43	15-133
Table 15-44	15-137
Table 15-45	15-138
Table 16-1: Command Summary	16-7
Table 16-2: Driver Status	16-7
Table 16-3: Control Register	16-8
Table 16-4: Mode Register	16-8

About This Manual

This manual is the latest release of the PlayStation® library reference as of Run-time Library release 4.3. The purpose of this manual is to define all available PlayStation run-time library functions, macros and structures. The companion *Overview* volume describes the structure and purpose of the libraries in programming software for the PlayStation.

Changes Since Last Release

This manual has been expanded in content since release 4.2 of the Run-time Library. It combines all previously released material with the latest Run-time Library 4.3 information.

The changes in this Reference Run-time Library Release 4.3 are described below:

Kernel Library

Function revised:

format

Math Library

Functions revised:

ldexp
printf2
sprintf2

Memory Card Library

Function revised:

_card_format()

Extended Memory Card Library

Function added:

MemCardUnFormat

Functions revised:

MemCardOpen
MemCardCreateFile
MemCardDeleteFile
MemCardGetDirent
MemCardFormat
MemCardSync

Data Compression Library

Structure revised:

ENCSPUENV

Functions revised:

DecDCTinSync

EncSPU

Basic Graphics Library

Structure added:

RECT32

Functions added:

GetDrawArea

GetDrawOffset

GetDrawMode

GetTexWindow

Macros added:

dumpMatrix

dumpRECT

dumpVector

dumpWH...

setTPage

setCLUT

setXYWH

Structure revised:

DRAWENV

Functions revised:

DrawPrim

KanjiFntClose

KanjiFntFlush

Krom2Tim

SetDefDispEnv

SetDrawLoad

SetTexWindow...

SetTexWindow (macro version)

Basic Geometry Library

Structure added:

QMESH

Functions added:

RotMatrix XZY*

RotMatrix YZX*

RotMatrix ZXY*

*These functions are included in RotMatrix... together with RotMatrixYXZ and RotMatrixZYX .

Functions revised:

SquareSS0
 SquareSS12
 ScaleMatrix
 ScaleMatrixL
 SetFogNearFar

Extended Graphics Library*Functions deleted:*

GsCreateNewObj2()
 GsInitObjTable2()
 GsRemoveObj2()
 GsSearchObjById2()
 GsSearchTmdById()
 GsSetTodFrame2()
 GsSetTodPacket2()

Structures moved to HMD Library:

GsARGUNIT
 GsARGUNIT_ANIM
 GsARGUNIT_GND...
 GsARGUNIT_IMAGE
 GsARGUNIT_JutMIMe
 GsARGUNIT_NORMAL
 GsARGUNIT_RstJntMIMe
 GsARGUNIT_RstVNMIMe
 GsARGUNIT_SHARED
 GsARGUNIT_VNMIMe
 GsCOORDUNIT
 GsRVIEWUNIT
 GsSEH
 GsSEQ
 GsTYPEUNIT
 GsUNIT
 GsVIEWUNIT
 GsWORKUNIT

Functions moved to HMD Library:

GsGetHeadUnit
 GsGetLsUnit
 GsGetLwsUnit
 GsGetLwUnit
 GsInitRstNrmMIMe
 GsInitRstVtxMIMe
 GsLinkAnim
 GsMapCoordUnit
 GsMapUnit
 GsScanAnim
 GsScanUnit
 GsSetRefViewLUnit
 GsSetRefViewUnit
 GsSetViewUnit
 GsSortUnit
 GsU_00000008...

GsU_03000000...
GsU_03000001...
GsU_03010110...
GsU_04010010...

Functions revised:

GsTMDfastF4GL
GsTMDfastF4GLFG
GsTMDfastF4GNL(
GsTMDfastG4GL
GsTMDfastG4GLFG
GsTMDfastG4GNL

Functions reassigned as macros:

GsIncFrame
GsSetAzwh

Structure revised:

_GsFCALL
GsF_LIGHT
GsOT
GsIMAGE

Functions revised:

GsTMDdiv...
GsSortOT

CD-ROM/Streaming Library

Functions revised:

CdDataCallback
CdGetSector
CdGetSector2
CdReadExec
CdSearchFile
StSetStream
StSetEmulate

Extended CD-ROM Library

Functions added:

DsLastCom
DsComstr
DsInstr

Functions revised:

DsReadExec
DsGetSector
DsGetSector2
DsSearchFile

Controller/Peripherals Library

Functions revised:

InitGun
PadSetAct
PadInfoAct
PadInfoComb
PadSetAct
PadSetActAlign
PadSetMainMode
StartTAP

Extended Sound Library

Functions revised:

SsGetMute
SsSetVoiceMask

Basic Sound Library

Functions revised:

SpuRGetAllKeysStatus
SpuRSetVoiceAttr
SpuStGetVoiceStatus
SpuReadDecodedData
SpuSetIRQAddr
SpuWrite
SpuWritePartly
SpuRead

Serial Input/Output Library

Revised functions:

AddSIO
DelSIO

Related Documentation

This manual should be read in conjunction with the *Run-time Library Overview*, since the *Overview* summarizes the use of the libraries.

Note: the Developer Support Website posts current developments regarding the run-time libraries and also provides notice of future documentation releases and upgrades.

Manual Structure

The Library Reference contains seventeen chapters providing definitions of library structures and functions.

Generally, each chapter defines the structures and/or functions of a single library. Note, however, that some chapters provide definitions for several related libraries. In particular, note that Chapter 2, the Standard C Library, describes `libc` and `libc2`. Chapter 12, the Controller/Peripherals Library, describes `libetc`, `libgun`, `libpad` and `libtap`.

Developer Reference Series

This manual is part of the *Developer Reference Series*, a series of technical reference volumes covering all aspects of PlayStation development. The complete series is listed below:

Manual	Description
PlayStation Hardware	Describes the PlayStation hardware architecture and overviews its subsystems.
PlayStation Operating System	Describes the PlayStation operating system and related programming fundamentals.
Run-Time Library Overview	Describes the structure and purpose of the run-time libraries provided for PlayStation software development.
Run-Time Library Reference	Defines all available PlayStation run-time library functions, macros and structures.
Inline Programming Reference	Describes in-line programming using DMPSX, GTE inline macro and GTE register information.
SDevTC Development Environment	Describes the SDevTC (formerly "Psy-Q") Development Environment for PlayStation software development.
3D Graphics Tools	Describes how to use the PlayStation 3D Graphics Tools, including the animation and material editors.
Sprite Editor	Describes the Sprite Editor tool for creating sprite data and background picture components.
Sound Artist Tool	Provides installation and operation instructions for the DTL-H800 Sound Artist Board and explains how to use the Sound Artist Tool software.
File Formats	Describes all native PlayStation data formats.
Data Conversion Utilities	Describes all available PlayStation data conversion utilities, including both stand-alone and plug-in programs.
CD Emulator	Provides installation and operation instructions for the CD Emulator subsystem and related software.
CD-ROM Generator	Describes how to use the CD-ROM Generator software to write CD-R discs.

Performance Analyzer User Guide	Provides general instructions for using the Performance Analyzer software.
Performance Analyzer Technical Reference	Describes how to measure software performance and interpret the results using the Performance Analyzer.
DTL-H2000 Installation and Operation	Provides installation and operation instructions for the DTL-H2000 Development System.
DTL-H2500/2700 Installation and Operation	Provides installation and operation instructions for the DTL-H2500/H2700 Development Systems.

Typographic Conventions

Certain Typographic Conventions are used through out this manual to clarify the meaning of the text. The following conventions apply to all narrative text except for structure and function descriptions:

<i>Convention</i>	<i>Meaning</i>
<code>courier</code>	Indicates literal program code.
*	New function or structure
Bold	Indicates a document, chapter or section title.

The following conventions apply within structure and function descriptions only:

<i>Convention</i>	<i>Meaning</i>
Medium Bold	Denotes structure or function types and names.
<i>Italic</i>	Denotes function arguments and structure members.

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
<i>In North America</i> Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd., 2nd floor Foster City, CA 94404 Tel: (650) 655-8000	<i>In North America</i> E-mail: DevTech_Support@playstation.sony.com Web: http://www.scea.sony.com/dev Developer Support Hotline: (650) 655-8181 (Call Monday through Friday, 8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
<i>In Europe</i> Attn: Production Coordinator Sony Computer Entertainment Europe Waverley House 7-12 Noel Street London W1V 4HH Tel: +44 (0) 171 447 1600	<i>In Europe</i> E-mail: dev_support@playstation.co.uk Web: https://www-s.playstation.co.uk Developer Support Hotline: +44 (0) 171 447 1680 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT or BST/BDT)

Chapter 1: Kernel Library

Table of Contents

Structures	
DIRENTRY	1-3
EvCB	1-4
EXEC	1-5
TCB	1-6
TCBH	1-7
ToT	1-8
Functions	
calloc2	1-9
calloc3	1-10
cd	1-11
ChangeClearPAD	1-12
ChangeTh	1-13
close	1-14
CloseEvent	1-15
CloseTh	1-16
delete	1-17
DeliverEvent	1-18
DisableEvent	1-19
DisablePAD	1-20
EnableEvent	1-21
EnablePAD	1-22
EnterCriticalSection	1-23
Exception	1-24
Exec	1-25
ExitCriticalSection	1-26
firstfile	1-27
FlushCache	1-28
format	1-29
free2	1-30
free3	1-31
GetConf	1-32
GetCr	1-33
GetGp	1-34
GetRCnt	1-35
GetSp	1-36
GetSr	1-37
GetSysSp	1-38
InitHeap	1-39
InitHeap2	1-40
InitHeap3	1-41
InitPAD	1-42
ioctl	1-44
Krom2RawAdd	1-45
Krom2RawAdd2	1-46
Load	1-47
LoadExec	1-48
LoadTest	1-49
lseek	1-50
malloc2	1-51
malloc3	1-52
nextfile	1-53
open	1-54
OpenEvent	1-55

OpenTh	1-56
read	1-57
realloc2	1-58
realloc3	1-59
rename	1-60
ResetRCnt	1-61
ReturnFromException	1-62
SetConf	1-63
SetMem	1-64
SetRCnt	1-65
SetSp	1-66
StartPAD	1-67
StartRCnt	1-68
StopPAD	1-69
StopRCnt	1-70
SwEnterCriticalSection	1-71
SwExitCriticalSection	1-72
SystemError	1-73
TestEvent	1-74
undelele	1-75
UnDeliverEvent	1-76
WaitEvent	1-77
write	1-78
_96_init	1-79
_96_remove	1-80
_boot	1-81
_get_errno	1-82
_get_error	1-83

DIRENTRY

Data structure of directory entries.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Structure

```
struct DIRENTRY {
    char name[20];
    long attr;
    long size;
    struct DIRENTRY *next;
    char system[8];
};
```

Members

<i>name</i>	Filename
<i>attr</i>	Attributes (dependent on file system)
<i>size</i>	File size (in bytes)
<i>next</i>	Pointer to next file entry (for user)
<i>system</i>	Reserved by system

Explanation

This structure stores information relating to files registered in the file system.

Remarks

See also: firstfile (p. 1-27), nextfile (p. 1-51).

EvCB

Event Control Block.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Structure

```
struct EvCB {
    unsigned long desc;
    long status;
    long spec;
    long mode;
    (long *FHandler);
    long system[2];
};
```

Members

desc Cause descriptor
status Status
spec Event type
mode Mode
FHandler Pointer to a function type handler
system Reserved by system

Explanation

Used for event management.

Remarks

See also: Open Event (p. 1-55), GetConf (p. 1-30), SetConf (p. 1-63).

EXEC

The data structure of an execute file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	5/22/97

Structure

```
struct EXEC {
    unsigned long pc0;
    unsigned long gp0;
    unsigned long t_addr;
    unsigned long t_size;
    unsigned long d_addr;
    unsigned long d_size;
    unsigned long b_addr;
    unsigned long b_size;
    unsigned long s_addr;
    unsigned long s_size;
    unsigned long sp;
    unsigned long fp;
    unsigned long gp;
    unsigned long ret;
    unsigned long base;
};
```

Members

<i>pc0</i>	Execution start address
<i>gp0</i>	gp register initial value
<i>t_addr</i>	Starting address of initialized text section
<i>t_size</i>	Size of text section
<i>d_addr</i>	Starting address of initialized data section
<i>d_size</i>	Size of initialized data section
<i>b_addr</i>	Uninitialized data section start address
<i>b_size</i>	Uninitialized data section size
<i>s_addr</i>	Stack start address (specified by the user)
<i>s_size</i>	Stack size (specified by the user)
<i>sp</i>	Register shunt variable
<i>fp</i>	Register shunt variable
<i>gp</i>	Register shunt variable
<i>ret</i>	Register shunt variable
<i>base</i>	Register shunt variable

Explanation

This structure retains the information for loading and executing a program stored in a file and arranged within the first 2K bytes of the execution file (PS-X EXE format). By adding stack information and transferring it to the Exec () function, the program is activated.

Remarks

See also: Exec (p. 1-25).

TCB

Task Control Block.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Structure

```
struct TCB {
    long status;
    long mode;
    unsigned long reg[NREGS];
    long system[6];
};
```

Members

status Status
mode Mode
reg Register saving area (specified by register designation macro)
system Reserved by system

Explanation

Data block where a context (the contents of the registers) is stored for thread management.

Remarks

See also: Open Th (p. 1-56), ChangeTh (p. 1-13), GetConf (p. 1-30), SetConf (p. 1-63).

TCBH

Task Execute Queue.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Structure

```
struct TCBH {
    struct TCB *entry;
};
```

Members

entry Pointer to execute TCB

Explanation

Used for thread management. The execute TCB is linked to *entry*.

Remarks

See also: ChangeTh (p. 1-13).

ToT

System Table Information.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Structure

```
struct ToT {
    unsigned long *head;
    long size;
};
```

Members

head Pointer to a system table start address
size System table size (in bytes)

Explanation

Table information which enables organized handling of various system tables which are used by the kernel. The placement address is 0x00000100.

Remarks

calloc2

Allocates main memory.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	10/23/96

Syntax

```
void *calloc2 (
    size_t n,
    size_t s
)
```

Arguments

n Number of partitions
s Size of one partition

Explanation

This function allocates a block of *n***s* bytes. Corresponds to InitHeap2().

Return value

Returns a pointer to the allocated memory block. If allocation fails, NULL will be returned.

Remarks

See also: malloc2(),realloc2(),free2()

calloc3

Allocates main memory.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	4.0	5/22/97

Syntax

```
void *calloc3 (  
    size_t n,  
    size_t s  
)
```

Arguments

<i>n</i>	Number of partitions
<i>s</i>	Size of one partition

Explanation

This function allocates a block of $n*s$ bytes from the heap memory and initializes at 0. Corresponds to InitHeap3.

Return value

Returns a pointer to the allocated memory block. If allocation fails, NULL will be returned.

Remarks

See also: InitHeap3(), malloc3(), realloc3(), free3)

cd

Change default directory.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long cd(*path)
char *path;
```

Arguments

path Pointer to the default directory path

Explanation

Changes the default directory path for the file system. The file system is specified by the device name at the beginning of the path.

Return value

Returns “1” if it succeeds, and “0” otherwise.

Remarks

See also:

ChangeClearPAD

Sets the control driver.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
void ChangeClearPAD(val)
long val;
```

Arguments

val Vertical retrace line interruption clear flag

Explanation

This function specifies whether to complete interrupt processing in a control driver started by a vertical retrace line interrupt, or to pass processing to a lower priority interrupt module without completion. A *val* value of 1 specifies completion, while a *val* value of 0 specifies passing.

Return value

None

Remarks

See also: StartPAD (p. 1-67), StopPAD (p. 1-69), StartCARD (see libcard), StopCARD (see libcard).

ChangeTh

Changing a thread to be executed.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long ChangeTh(thread)
unsigned long thread;
```

Arguments

thread Thread descriptor

Explanation

Execution is transferred to the thread specified by *thread*. The current thread is saved in a TCB during execution of this function. It returns from this function when the original thread is restored.

Return value

On success and re-execution, the function returns 1. On failure, it returns 0. The Return value on re-execution can be changed by any other thread.

Remarks

Before executing ChangeTh(), initialize TCB reg [R-SR] to the following:

- The interrupt context is 0X404
- The main flow is 0X401

See also: TCB structure (p.1-6), TCBH structure (p. 1-7), OpenTh (p. 1-56).

close

Closing a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
int close(fd)
```

```
int fd;
```

Arguments

fd File descriptor

Explanation

This function closes a file descriptor.

Return value

On success, the function returns *fd*. On failure, it returns -1.

Remarks

See also: Open (p. 1-53).

CloseEvent

Closing an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long CloseEvent(event)
unsigned long event;
```

Arguments

event Event descriptor

Explanation

Releases the EvCB specified by *event*.

Return value

On success, the function returns 1. On failure, it returns 0.

Remarks

To be executed in a critical section.

See also: OpenEvent (p. 1-54), EnterCriticalSection (p. 1-22), SwEnterCriticalSection (p. 1-71).

CloseTh

Closes a thread.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long closeTh(thread)
unsigned long thread;
```

Arguments

thread Thread descriptor

Explanation

This function closes a thread and releases its TCB.

Return value

On success, the function returns 1. On failure, it returns 0.

Remarks

To be executed in a critical section.

See also: OpenTh (p. 1-55), EnterCriticalSection (p. 1-22), SwEnterCriticalSection (p. 1-71).

delete

Deletes a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long delete(*name)
char *name;
```

Arguments

name Pointer to a filename

Explanation

Deletes the file specified by *name*.

Return value

Returns “1” if it succeeds, and “0” otherwise.

Remarks

See also: undelete (p. 1-74).

DeliverEvent

Generates an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
void DeliverEvent(ev1, ev2)
unsigned long ev1;
long ev2;
```

Arguments

ev1 Cause descriptor
ev2 Event class

Explanation

This function delivers an event if that event's current status is EvStACTIVE (event not yet generated, generation possible). If the event mode is EvMdNOINTR, the event handler function is called. If the event mode is EvMdINTR, the event status is changed to EvStALREADY (event already occurred, generation prohibited).

Return value

None

Remarks

This function must be executed in a critical section.

See also: UnDeliverEvent (p. 1-76), OpenEvent (p. 1-55), TestEvent (p. 1-74), EnterCriticalSection (p. 1-22), SwEnterCriticalSection (p. 1-71), DisableEvent (p. 1-19), EnableEvent (p. 1-20) WaitEvent (p. 1-77), CloseEvent (p. 1-15).

DisableEvent

Disables an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long DisableEvent(event)
unsigned long event;
```

Arguments

event Event descriptor

Explanation

This function inhibits occurrence of an event specified by the descriptor *event*. It changes the event status to EvStWAIT (event generation prohibited).

Return value

On success, the function returns 1. On failure, it returns 0.

Remarks

See also: EnableEvent (p. 1-20).

DisablePAD

Disables communication with the controller

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	5/22/97

Syntax

void DisablePad(*void*)

Arguments

None

Explanation

Temporarily disables communication with the controller.

Although StopTAP() deletes the controller handler activated by Vsync interrupts, this function simply skips controller communication with a flag operation.

Return value

None

Remarks

Although a normal controller communicates via Vsync interrupts, this function is used only with timing longer than 1/60 sec when the controller status is not needed.

See also: EnablePAD()

EnableEvent

Enables occurrence of an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long EnableEvent(event)
unsigned long event;
```

Arguments

event Event descriptor

Explanation

This function enables occurrence of an event specified by the descriptor *event*. It changes the event status to EvStACTIVE (event not yet generated, generation possible).

Return value

On success, the function returns 1. On failure, it returns 0.

Remarks

See also: DisableEvent (p. 1-19), TestEvent (p. 1-74).

EnablePAD

Enables occurrence of an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	5/22/97

Syntax

```
void EnablePAD()
```

Arguments

None

Explanation

Enables communication with a controller which was disabled with DisablePAD().

Return value

None

Remarks

Although a normal controller communicates via Vsync interrupts, this function is used only with timing longer than 1/60 sec when the controller status is not needed.

See also: DisablePAD()

EnterCriticalSection

Enter a critical section.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	10/01/97

Syntax

```
void EnterCriticalSection(void)
```

Arguments

None

Explanation

This function stops interrupts, and enters a critical section. This occurs immediately after kernel startup.

Return value

When this function is called in Critical Section, 0 is returned. In all other cases, 1 is returned.

Remarks

Executes an internal system call and destroys the interrupt context. However, does not call the main function from the event handler callback interrupt context.

See also: TCBH (p. 1-7), TCB (p. 1-6), ExitCriticalSection (p. 1-26).

Exception

Causes an interrupt.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

void Exception(*void*)

Arguments

None

Explanation

This function causes an interrupt, and stores the current context in the execute TCB. It is also valid in a critical section.

Return value

None

Remarks

Executes an internal call and destroys the exception context.

See also: TCBH (p. 1-7), TCB (p. 1-6), ChangeTh (p. 1-13), ReturnFromException (p. 1-62).

Exec

Executes an execute file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long Exec(*exec, argc, *argv)
struct EXEC *exec;
long argc;
char *argv;
```

Arguments

exec Pointer to execute file information
argc Number of arguments
argv Pointer to argument

Explanation

According to the execute file information specified by *exec*, this function executes a module already loaded in memory. If *exec*->*s_addr* is 0, neither stack or frame pointer is set.

The function performs the following:

- A data section without initial values is cleared to zero.
- *sp*, *fp*, and *gp* are saved, and then initialized. (*fp* is set to the same value as *sp*.)
- The arguments of *main()* are set (in the *a0* and *a1* registers).
- The execution start address is called.
- After a return is made, *sp*, *fp*, and *gp* are restored.

Return value

On success, the function returns 1. On failure, it returns 0.

Remarks

To be executed in a critical section.

This function needs the ISO 9660 file system to run properly. *_96_init()* must be called to initialize this system and *_96_remove* must be called to exit this system.

See also: EXEC (p. 1-5), Load (p. 1-46), *_96_init* (p. 1-79), *_96_remove* (p. 1-80).

ExitCriticalSection

Exits critical section.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	10/01/97

Syntax

void ExitCriticalSection(*void*)

Arguments

None

Explanation

This function enables interrupts, and exits from the critical section.

Return value

None

Remarks

Executes an internal system call and destroys the interrupt context. However, does not call the main function from the event handler callback interrupt context.

See also: TCBH (p. 1-7), TCB (p. 1-6), EnterCriticalSection (p. 1-22).

firstfile

Looks up the first file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
struct DIRENTRY *firstfile(*name, *dir)
char *name;
struct DIRENTRY *dir;
```

Arguments

name Pointer to a filename
dir Pointer to the buffer holding information relating to the referenced file.

Explanation

Looks up the file corresponding to the filename pattern *name*, and stores data relating to this file in the directory *dir*.

Return value

Returns *dir* if it succeeds, and "0" otherwise.

Remarks

The wildcard characters "?" (standing for any one character) and "*" (standing for a character string of any length) can be used in the filename pattern. Characters specified after "*" are ignored.

See also: DIRENTRY (p. 1-3), nextfile (p. 1-50).

FlushCache

Flushes instruction cache (I cache).

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
void FlushCache(void)
```

Arguments

None

Explanation

Flushes I cache. Code is not executed when written to memory.

Return value

None

Remarks

To be executed in a critical section.

format

Initializes file system.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	6/22/98

Syntax

```
long format(*fs)
char *fs;
```

Arguments

fs Pointer to file system name

Explanation

Initializes file system *fs*.

Return value

Returns “1” if it succeeds, and “0” otherwise.

Note: This function has a bug where 1 is returned regardless of success or failure.

When initializing the Memory Card, use the `_card_format` function within `libcard.lib`. If use of this function is unavoidable for reasons such as memory capacity, perform a Memory Card status confirmation using `_card_load`, etc. immediately after executing this function.

Remarks

This function is only effective on writeable file systems.

See also:

free2

Frees allocated memory blocks.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	10/23/96

Syntax

```
void free2  
(void *block)
```

Arguments

block Area to be released

Explanation

This function releases a memory block that was allocated by `calloc2`, `malloc2`, and `realloc2`. Corresponds to `InitHeap2()`.

Return value

None

Remarks

See also: `calloc2()`, `malloc2()`, `realloc2()`

free3

Frees allocated memory blocks.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	4.0	5/22/97

Syntax

```
void free3
(void *block)
```

Arguments

block Area to be released

Explanation

This function releases a memory block that was allocated by `calloc3`, `malloc3`, and `realloc3`.

Return value

None

Remarks

See also: `calloc3()`, `malloc3()`, `realloc3()`

GetConf

Obtains the kernel configuration.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

```
void GetConf(*ev, *tcb, *sp)
unsigned long *ev;
unsigned long *tcb;
unsigned long *sp;
```

Arguments

ev Pointer to the address that stores the number of event management block elements
tcb Pointer to the address that stores the number of task management block elements
sp Ignored

Explanation

This function stores a system configuration parameter set by SetConf () to the address given by the pointer as the argument.

Return value

None

Remarks

This function returns an undefined value before the execution of SetConf () because this function refers to its internal parameter.

See also: SetConf (p. 1-63).

GetCr

Gets a cause register value.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

unsigned long GetCr(void)

Arguments

None

Explanation

This function gets the control register cause register value.

Table 1-1: Description of Each Bit of the Cause Register

Bit	Description
31-6	Reserved by the system
5-2	Exception code
	0000 External interrupt
	0001 Not used
	0010 Not used
	0011 Not used
	0100 Address read error
	0101 Address write error
	0110 Command bus error
	0111 Data bus error
	1000 System call
	1001 Break point
	1010 Undefined command
	1011 Co-processor not mounted
	1100 Overflow
1-0	Reserved by the system

Return value

The current cause register value is returned.

Remarks

See also: OpenTh (p. 1-56).

GetGp

Gets a gp register value.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

unsigned long GetGp(*void*)

Arguments

None

Explanation

This function gets a gp register value.

Return value

The current gp register value is returned.

Remarks

See also: EXEC structure (p. 1-5), OpenTh (p. 1-56), Load (p. 1-46), Exec (p. 1-25).

GetRCnt

Acquires a root counter.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long GetRCnt(spec)
long spec;
```

Arguments

spec Root counter

Explanation

Returns the current value of root counter *spec*. To be used when root counter *spec* has been set by SetRCnt to a polling mode (RCntMdNOINTR).

Return value

On success, the function returns the 32-bit unsigned expanded counter value. On failure, it returns -1.

Remarks

See also: SetRCnt (p. 1-65), StartRCnt (p. 1-68), StopRCnt (p. 1-70), ResetRCnt (p. 1-61).

GetSp

Gets an sp register value.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

unsigned long GetSp(*void*)

Arguments

None

Explanation

This function gets an sp register value.

Return value

A current sp register value is returned.

Remarks

See also: EXEC (p. 1-4), OpenTh (p. 1-55), Load (p. 1-45), Exec (p. 1-25).

GetSr

Gets a status register value.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

unsigned long GetSr(*void*)

Arguments

None

Explanation

This function gets the control register status register value.

Table 1-2: Description of Each Bit of the Status Register

Bit	Description
31-28	Co-processor installation flag (1: Installed) Bit 29 is GTE.
27-11	Reserved by the system
10	Always 1
9-3	Reserved by the system
2	Main flow interrupt permission (1: Permission)
1	Reserved by the system
0	Interrupt permission (1: Permission)

Return value

The current status register value is returned.

Remarks

See also: OpenTh (p. 1-55).

GetSysSp

Gets a system stack.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

long GetSysSp(*void*)

Arguments

None

Explanation

This function acquires the highest address of a system stack area for event handler function execution.

The size of the stack area is 2 K-bytes.

Return value

Highest address of the system stack area

Remarks

See also:

InitHeap

Initializes a heap area.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.0	7/31/96

Syntax

```
void InitHeap(*head, size)
unsigned long *head;
unsigned long size;
```

Arguments

head Pointer to heap start address
size Heap size (a multiple of 4, in bytes)

Explanation

This function initializes a group of standard function library memory control functions. After using this function, malloc(), etc. are usable.

There is an overhead so the entire size in bytes cannot be used.

Return value

None

Remarks

To be executed in a critical section. If several executions of this function overlap, the memory control information previously held will be lost.

See also: malloc (see libc/libc2).

InitHeap2

Initializes heap area.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	10/23/96

Syntax

```
void InitHeap2(*head, size)
void *head;
long size;
```

Arguments

head Pointer to heap start address
size Heap size (a multiple of 4, in bytes)

Explanation

This function initializes a group of standard function library memory control functions. After using this function, `malloc2()`, etc. are usable.

There is an overhead so the entire "size" in bytes cannot be used. This is the bug fix version of `InitHeap()` but has larger program size since this is a memory resident function

Return value

None

Remarks

If several executions of this function overlap, the memory control information previously held will be lost.

See also: `InitHeap()`, `malloc2()`, `realloc2()`, `calloc2()`, `free2()`

InitHeap3

Initializes heap area.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	4.0	10/01/97

Syntax

```
void InitHeap3(*head, size)
void *head;
long size;
```

Arguments

head Pointer to heap start address
size Heap size (a multiple of 8, in bytes)

Explanation

This function initializes a group of library memory control functions. After using this function, malloc 3, etc. will be usable. However, since there is an overhead, the entire "size" in bytes cannot be used.

Return value

None

Remarks

If several executions of this function overlap, the memory control information previously held will be lost.

This function is a higher speed than the InitHeap2() system and is smaller in size.

When a number other than a multiple of 8 is specified as the size, the surplus which cannot be divided by 8 is discarded and is not allocated.

Refer to the section entitled "Memory Allocation Functions" in the Kernel chapter of the Library Overview for the differences between the various malloc systems.

See also: InitHeap(), InitHeap2(), malloc3(), realloc3(), calloc3(), free3()

InitPAD

Initializes the controller.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	5/22/97

Syntax

```
long InitPAD(*bufA, lenA, *bufB, lenB)
char *bufA, *bufB;
long lenA, lenB;
```

Arguments

bufA, bufB Pointers to incoming data buffers
lenA, lenB Length of incoming data buffers (in bytes)

Explanation

This function registers a receive data buffer for the controller. The format of received data is given in the Library Overview. ChangeClearPAD() is not executed internally.

Return value

Always 1.

Remarks

Since it is possible for mistakes to occur when an unexpected controller is connected to the receive data length, always maintain 34 bytes.

Table 1-3: Data Formats Stored in the Receive Buffer

Byte	Contents
0	Receive result 0x00: Success, 0xff: Failure
1	Upper 4 bits: Terminal classification 0x1: Mouse 0x2: 16 button analog 0x3: Gun controller 0x4: 16 button 0x5: Analog joystick 0x7: Analog controller 0x8: Multi-tap Lower 4 bits: No. of receive data bytes/2
Mouse	
3	Button status 1: release, 0: push 2nd bit: right, 3rd bit: left
4	Shift volume X direction (-128~127)
5	Shift volume Y direction (-128~127)
6	Shift volume Z direction (-128~127)
7,8,-	Shift volume ? direction (-128~127)
16 button analog, analog joystick, analog controller	
2,3	Button status 1: release, 0: push
4,5,6,7,-	Analog channel value
Gun Controller, 16 button	

2,3	Button status 1: release, 0: push
-----	-----------------------------------

The upper four bits of the first buffer byte are the terminal classification. The lower four bits are half the value of the number of bytes of received data (stored from the third byte of the buffer onwards) from the terminal. Refer to the documentation for each terminal for the physical positioning and handling of the buttons and channels.

Existing terminals

The standard controller is a 16 button type.

The mouse produced by Sony Computer Entertainment supports only x and y directional movement. The 'NeGcon' from Namco Corporation is a four channel, 16 button analog type, in other words, the upper four bits of the first buffer byte are 3 ((2 byte button section + 4 byte 4 channel section) / 2).

When using the multi-tap, use InitTAP () and when using the gun controller, use InitGUN ().

See also: StartPAD (p. 1-67), StopPAD (p. 1-69), ChangeClearPAD (p. 1-12).

ioctl

Controls devices.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

long *ioctl*(*fd*, *com*, *arg*)

int *fd*;

int *com*;

int *arg*;

Arguments

fd File descriptor
com Control command
arg Control command argument

Explanation

Executes all types of control commands on the device. Details of the commands and their arguments are given separately for each device.

Return value

Returns the value “1” if it succeeds and the value “0” otherwise.

Remarks

See also: open (p. 1-53).

Krom2RawAdd

Collects Kanji font pattern addresses.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	02/15/98

Syntax

```
unsigned long Krom2RawAdd(sjiscode)
unsigned short sjiscode;
```

Arguments

sjiscode Shift JIS code

Explanation

This function acquires the starting address in the kernel of the font pattern corresponding to the Kanji character specified by *sjis code*.

Return value

The head address of a Kanji font pattern is returned. If there is no font data corresponding to the specified Kanji character, a value of -1 is returned.

Remarks

This function is considered a normal argument when the Shift JIS code values of 0x8140~0x84BE or 0x889F~0x9872 are provided as the arguments. Because it is converted to the font pattern head address, when Shift JIS code within that region which cannot be used (areas which are blank on the code table) are provided as arguments, a font pattern unrelated to that code is returned as the head address (this problem has been corrected in Krom2RawAdd2, so we recommend using Krom2RawAdd2 to obtain the font pattern head address).

Refer to the codeview sample in \psx\kanji\sjiscode for a list of usable fonts and the actual fonts themselves.

See also:

Krom2RawAdd2

Collects Kanji font pattern addresses.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.2	2/15/98

Syntax

```
unsigned long Krom2RawAdd2(sjiscode)
unsigned short sjiscode;
```

Arguments

sjiscode Shift JIS code

Explanation

Acquires the head address in the font pattern kernel corresponding to the non-Kanji/Kanji No. 1 level/foreign language specified by the *sjiscode*.

Return value

Returns the kanji font pattern head address.

When the font data corresponding to the specified kanji is not prepared, a head address which contains a full space font pattern is returned.

Refer to the codeview sample in \psx\kanji\sjiscode for a list of usable fonts and the actual fonts themselves.

Remarks

See also:

Load

Loads an execute file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long Load(*name, *exec)
char *name;
struct EXEC *exec;
```

Arguments

name Pointer to filename
exec Pointer to execute file information

Explanation

This function reads the PS-X EXE format file *name* to the address specified by its internal header, and writes internal information to *exec*.

Return value

On success, the function returns 1. On failure, it returns 0.

Remarks

This function needs the ISO 9660 file system to run properly. `_96_init()` must be called to initialize this system and `_96_remove` must be called to exit this system. Calls `FlushCache()` internally.

See also: EXEC structure (p. 1-5), Exec (p.1-25), `_96_init` (p. 1-79), `_96_remove` (p. 1-80).

LoadExec

Executes a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

Void LoadExec(**name*, *s_addr*, *s_size*)

char **name*;

unsigned long *s_addr*;

unsigned long *s_size*;

Arguments

name Pointer to a PS-X EXE format execution file name (fewer than 19 characters)

s_addr Stack area starting address

s_size Number of bytes in stack area

Explanation

This function calls Load() and Exec(), then reads a file name into memory and executes the file. *s_addr* and *s_size* are passed to Exec() and set by the structure EXEC.

Return value

None There is no return value when the function executes normally.

Remarks

This function needs the ISO 9660 file system to run properly. `_96_init()` must be called to initialize this system and `_96_remove` must be called to exit this system.

See also: EXEC (p. 1-4), Load (p. 1-45), Exec (p. 1-25), `_96_init` (p. 1-79), `_96_remove` (p. 1-80).

LoadTest

Load test.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long LoadTest(*name, exec)
char *name;
struct EXEC *exec;
```

Arguments

name Pointer to filename
exec Pointer to data in an execute file

Explanation

This function writes internal information from a PS-X EXE format file *name* to *exec*.

Return value

On success, the function returns the execution starting address. On failure, it returns 0.

Remarks

See also: EXEC (p. 1-4), Load (p. 1-45).

lseek

Moves a file pointer.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

unsigned long lseek(*fd*, *offset*, *flag*)

int *fd*;

unsigned int *offset*;

int *flag*;

Arguments

fd File descriptor

offset Offset

flag Start point flag

Explanation

This function moves a file pointer to the device indicated by the descriptor *fd*. *offset* stands for the number of bytes to be moved. The starting point of the movement varies with the value of the *flag*. However, it does not apply to a tty driver.

Table 1-4: *flag* Designations

flag macro	Operation
SEEK_SET	Start of file
SEEK_CUR	Current position

Return value

On success, the function returns the current file pointer. On failure, it returns -1.

Remarks

See also: open (p. 1-54), read (p. 1-57), write (p. 1-78).

malloc2

Allocates main memory.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	10/23/96

Syntax

```
void *malloc2 (
    size_t s
)
```

Arguments

s Size of memory block to be allocated

Explanation

This function allocates s bytes of memory block from the heap memory. Corresponds to InitHeap2().

Return value

Returns a pointer to allocated memory block. If failed, NULL is returned. *Heap memory is defined as below:

Low Address	Module Highest Address + 4
High Address	On-board memory - 64KB

Remarks

See also: calloc2(),realloc2(),free2()

malloc3

Allocates main memory.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	4.0	10/01/97

Syntax

```
void *malloc3 (
size_t s
)
```

Arguments

s Size of memory block to be allocated

Explanation

This function allocates s bytes of memory block from the heap memory. Corresponds to InitHeap3.

Return value

Returns a pointer to the allocated memory block. If allocation fails, NULL will be returned.

Remarks

InitHeap3 must be executed in advance.

InitHeap and InitHeap2 cannot be used together as a pair.

Refer to the section entitled "Memory Allocation Functions" in the Kernel chapter of the Library Overview for the differences between the various malloc systems.

See also: InitHeap3(), calloc3(), realloc3(), free3()

nextfile

Looks up the next file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
struct DIRENTRY *nextfile(*dir)
struct DIRENTRY *dir;
```

Arguments

dir Pointer to a buffer holding information relating to the referenced file

Explanation

This function continues the lookup under the same conditions as the function "firstfile()", executed immediately beforehand. If it finds the corresponding file, it stores information relating to this file in *dir*.

Return value

Returns *dir* if it succeeds, and "0" otherwise.

Remarks

If the shell cover of the CD-ROM drive has been opened since the execution of the immediately preceding function "firstfile()", this function fails on execution, and reports that the file has not been found.

See also: DIRENTRY (p. 1-3), firstfile (p. 1-27).

open

Opens a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long open(*devname, flag)
```

```
char *devname;
```

```
int flag;
```

Arguments

devname Pointer to a filename

flag Open mode

Explanation

This function opens a device for low-level input/output, and returns the descriptor. *flag* is dependent on the device.

Table 1-5: Common Parts

Macro	Open mode
O_RDONLY	Read only
O_WRONLY	Write only
O_RDWR	Both read and write
O_CREAT	Create new file
O_NOBUF	Non-buffer mode
O_NOWAIT	Asynchronous mode

Return value

On success, the function returns the descriptor. On failure, it returns -1.

Remarks

See also: close (p.1-14).

OpenEvent

Opens an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long OpenEvent(desc, spec, mode, *func)
unsigned long desc;
long spec;
long mode;
long *func();
```

Arguments

desc Cause descriptor
spec Event type
mode Mode
func Pointer to the handler function

Explanation

This function secures the EvCB for an event with the descriptor *desc* and event class *spec*.

Return value

On success, the function returns an event descriptor. On failure, it returns -1.

Remarks

To be executed in a critical section.

See also: EvCB structure (p. 1-4), CloseEvent (p. 1-15), DeliverEvent (p.1-18).

OpenTh

Opens a thread.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
unsigned long OpenTh(*func, sp, gp)
unsigned long (*func)();
unsigned long sp;
unsigned long gp;
```

Arguments

func Pointer to the execution start function
sp Stack pointer value
gp Global pointer value

Explanation

This function secures a TCB, and initializes it according to the arguments. This TCB can be executed using ChangeTh().

Return value

On success, the function returns the descriptor. On failure, it returns -1.

Remarks

To be executed in a critical section.

See also: TCB structure (p. 1-6), CloseTh (p. 1-13), ChangTh (p. 1-13).

read

Reads data from a file

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long read (
long fd,
void *buf,
long n
)
```

Arguments

fd File descriptor
buf Pointer to read buffer address
n Number of bytes to be read

Explanation

This function reads *n* bytes from the descriptor *fd* to the area specified by *buf*.

Return value

On normal termination, the function returns the actual number of bytes read into the area. Any other value returns -1.

Remarks

See also: open (p. 1-54).

realloc2

Changes the heap memory allocation.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	10/23/96

Syntax

void *realloc2

(void **block*, size_t *s*)

Arguments

block Area to be reallocated

s Size of area to be reallocated

Explanation

This function increases/decreases the size of the memory block previously allocated to "s" bytes. Same as malloc2 when block is NULL. Corresponds to InitHeap2().

Return value

Returns a pointer to the reallocated memory block. The new pointer may have different address from the original. If reallocation fails, NULL will be returned, and original block will not be released.

Remarks

See also: calloc2(), malloc2(), free2()

realloc3

Changes the heap memory allocation.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	4.0	5/22/97

Syntax

void *realloc3

(void **block*, size_t *s*)

Arguments

block Area to be reallocated

s Size of area to be reallocated

Explanation

This function increases or decreases the size of the memory block previously allocated to '*s*' bytes. When the block is NULL, it operates in the same way as malloc3. This function corresponds to InitHeap3.

Return value

Returns a pointer to the reallocated block address. It is possible that this address may be different to the original. If reallocation fails, NULL will be returned. In such cases, the original block will not be released. NULL will also be returned if '*s*' is set to '0'.

Remarks

See also: calloc3(), malloc3(), free3()

rename

Changes a file name.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long rename(*src, *dest)
char *src;
char *dest;
```

Arguments

src Pointer to the old filename
dest Pointer to the new filename

Explanation

Changes the filename from *src* to *dest*. In both cases, the full path from the device name must be specified.

Return value

Returns “1” if it succeeds, and “0” otherwise.

Remarks

This function is only effective on writeable file systems.

See also:

ResetRCnt

Resets a root counter.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	5/22/97

Syntax

```
long ResetRCnt(spec)
long spec;
```

Arguments

spec Specifies a root counter

Explanation

This function resets a root counter *spec* to 0.

Return value

On success, the function returns 1. On failure, it returns 0.

Since RCntCNT3 cannot be set, 0 is always returned.

Remarks

See also: SetRCnt (p. 1-65), GetRCnt (p. 1-35), StartRCnt (p. 1-68), StopRCnt (p. 1-70).

ReturnFromException

Return from exception.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
void ReturnFromException(void)
```

Arguments

None

Explanation

Accesses the exception context and returns from exception processing. It is used in an event handler or callback function.

Return value

None if the function is executed normally.

Remarks

See also:

SetConf

Modifies the kernel configuration.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long SetConf(ev, tcb, sp)
unsigned long ev;
unsigned long tcb;
unsigned long sp;
```

Arguments

- ev* Number of event management block (EvCB) elements
- tcb* Number of task management block (TCB) elements
- sp* Ignored

Explanation

This function modifies system configuration parameters to reconfigure the kernel configuration, specifically the allocation of the system internal table.

All the contents of event and task management blocks and all the settings for event handlers and callback functions in each library are destroyed. However, file descriptors are not affected (all the descriptors should be closed before SetConf call) because most of the device drivers are driven by the event handler.

All patches to the kernel are holded.

Return value

1 will be returned on success of the modification. Otherwise, 0 will be returned.

Remarks

This function should be executed at the head of the first execution file. The operations of libraries initialized before the execution of this function are not ensured.

This function eliminates the ISO-9660 file system installed in the kernel immediately after activation (call _96_init() to reinstate). The result of operations on the opened files are not predictable.

If the number of the designated elements exceeds the maximum, the operation of the system after the execution of this function is not defined.

See also: GetContf (p. 1-29).

SetMem

Modifies the valid memory size.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
void SetMem(n)
unsigned long n;
```

Arguments

n Valid memory size (in megabytes)

Explanation

This function changes the valid memory size to the value specified by the argument. *n* must be 2 (2 megabytes) or 8 (8 megabytes). Any values other than these are ignored.

Return value

None

Remarks

Memory access out of the valid range results in the generation of CPU exception irrespective of the mounted physical memory.

See also:

SetRCnt

Sets a root counter.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	5/22/97

Syntax

long SetRCnt(*spec*, *target*, *mode*)

long *spec*;

unsigned short *target*;

long *mode*;

Arguments

spec Root counter specification

target Target value

mode Mode

Explanation

Set the root counter in *spec*, the target value in *target*, and the mode in *mode*.

Return value

On success, the function returns 1. On failure, it returns 0.

Since RCntCNT3 cannot be set, 0 is always returned.

Remarks

See also: GetRConf (p. 1-35), StartRCnt (p. 1-68), StopRCnt (p. 1-70), ResetRCnt (p. 1-61).

SetSp

Sets a stack pointer.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

unsigned long SetSp(*new-sp*)

unsigned long *new-sp*;

Arguments

new-sp value set in sp register

Explanation

Sets *new-sp* in the sp register.

Return value

Returns the sp register value before modification.

Remarks

See also: EXEC (p. 1-4), OpenTh (p. 1-55), Load (p. 1-45), Exec (p. 1-25).

StartPAD

Starts reading the controller.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

long StartPAD(*void*)

Arguments

None

Explanation

Triggered by the interruption of a vertical retrace line, this function starts to read the controller. ChangeClearPAD (1) is executed internally.

Return value

Always returns 1.

Remarks

Interruption is permitted.

See also: InitPAD (p. 1-39), ChangeClearPAD (p. 1-12).

StartRCnt

Starting a root counter.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	5/22/97

Syntax

```
long StartRCnt(spec)
```

```
long spec;
```

Arguments

spec Root counter

Explanation

This function enables interrupts for root counter *spec*.

Since RCntCNT3 cannot be set, 0 is always returned.

Return value

On success, the function returns 1. On failure, it returns 0.

Remarks

See also: GetRCnt (p. 1-35), ResetRCnt (p. 1-61), SetRCnt (p. 1-65), StopRCnt (p. 1-70).

StopPAD

Stops reading the controller.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

void StopPAD(*void*)

Arguments

None

Explanation

This function stops reading the controller. Interruption is not permitted.

Return value

None

Remarks

See also: InitPAD (p. 1-67), ChangeClearPAD (p. 1-12).

StopRCnt

Stops a root counter.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long StopRCnt(spec)
```

```
long spec;
```

Arguments

spec Root counter

Explanation

This function disables interrupts for root counter *spec*.

Return value

On success, the function returns 1. On failure, it returns 0.

Since RCntCNT3 cannot be set, 0 is always returned.

Remarks

See also: StartRCnt (p. 1-68), SetRCnt (p. 1-65), ResetRCnt (p. 1-61), GetRCnt (p. 1-35).

SwEnterCriticalSection

Suppresses interrupts.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

void SwEnterCriticalSection(*void*)

Arguments

None

Explanation

This function suppresses interrupts. Because no system call interrupt is generated internally, this function can be invoked in event handling and callback functions. It must be executed in a critical section.

Return value

None

Remarks

See also: EnterCriticalSection (p. 1-26), SwExitCriticalSection (p. 1-71).

SwExitCriticalSection

Permits interrupts.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

void SwExitCriticalSection(*void*)

Arguments

None

Explanation

This function permits interrupts. Because no system call interrupt is generated internally, the function can be invoked in event handling and callback functions. It must be executed in a critical section.

Return value

None

Remarks

See also: EnterCriticalSection (p. 1-22), SwExitCriticalSection (p. 1-71).

SystemError

Displays the system error screen.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
void SystemError(c, n)
char c;
long n;
```

Arguments

c Error identification character (Alphabetic character)
n Error identification code (0 to 999)

Explanation

This function displays a detected system error for the user (game player). In the PlayStation, `exit()` is called. Successful execution results in no return value.

Return value

None

Remarks

See also:

TestEvent

Testing an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long TestEvent(event)
unsigned long event;
```

Arguments

event Event descriptor

Explanation

This function checks to see whether or not the event specified by the descriptor *event* has occurred. If so, the function restores the event state to EvStACTIVE.

Return value

If the event is found to have occurred, the function returns 1. Otherwise, it returns 0.

Remarks

See also: DeliverEvent (p. 1-18), EnableEvent (p. 1-20), WaitEvent (p. 1-77), OpenEvent (p. 1-55), CloseEvent (p. 1-15), UnDeliverEvent (p. 1-76), DisableEvent (p. 1-19).

undelete

Resurrect a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long undelete(*name)
char *name;
```

Arguments

name Pointer to filename

Explanation

Resurrects the previously deleted file specified by *name*.

Return value

Returns “1” if it succeeds, and “0” otherwise.

Remarks

See also: delete (p. 1-17).

UnDeliverEvent

Cancels an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
void UnDeliverEvent(ev1, ev2)
unsigned long ev1;
long ev2;
```

Arguments

ev1 Cause descriptor
ev2 Event class

Explanation

This function returns event state from EvStALREADY (already occurred) to EvStACTIVE if the event mode is EvMdNOINTR.

Return value

None

Remarks

This function must be executed in a critical section.

See also: DeliverEvent (p. 1-18), EnableEvent (p. 1-20), OpenEvent (p. 1-54), TestEvent (p. 1-73), WaitEvent (p. 1-77), EnterCriticalSection (p. 1-22).

WaitEvent

Waits for the occurrence of an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
long WaitEvent(event)
unsigned long event;
```

Arguments

event Event descriptor

Explanation

This function waits until an event specified by the descriptor *event* occurs, and returns after restoring the event state to EvStACTIVE.

Return value

On success, the function returns 1. Otherwise, it returns 0.

Remarks

See also: TestEvent (p. 1-74), OpenEvent (p. 1-55), CloseEvent (p. 1-15), DeliverEvent (p. 1-18), UnDeliverEvent (p. 1-76), DisableEvent (p. 1-19), EnableEvent (p. 1-20).

write

Writes data to a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

```
int write(fd, *buf, n)
int fd;
char *buf;
int n;
```

Arguments

fd File descriptor
buf Pointer to the write buffer address
n Number of bytes to be written

Explanation

This function writes *n* bytes from the descriptor *fd* to the area specified by *buf*.

Return value

At normal termination, this function returns the number of bytes actually written to the area. Any other result returns -1.

Remarks

See also: open (p. 1-54).

_96_init

Installs the ISO-9660 file system.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	<i>2.x</i>	<i>7/31/96</i>

Syntax

void _96_init(void)

Arguments

None

Explanation

This function installs the ISO-9660 file system driver that manages access to the CD-ROM in the kernel.

Return value

None

Remarks

See also: `_96_remove` (p. 1-80).

`_96_remove`

Removes the ISO-9660 file system.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

`void _96_remove(void)`

Arguments

None

Explanation

This function removes the ISO-9660 file system driver that manages access to the CD-ROM from the kernel.

Return value

None

Remarks

See also: `_96_init` (p. 1-79).

`_boot`

Reboots the system.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	7/31/96

Syntax

`void _boot(void)`

Arguments

None

Explanation

This function reboots the system. This is an interface used to develop demonstration programs. Do not use it for general title applications.

Return value

None

Remarks

See also:

`_get_errno`

Collects the latest I/O error code.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

`int _get_errno(void)`

Arguments

None

Explanation

This function collects the latest error code through all file descriptors. Error codes are defined in `sys/errno.h`.

Return value

Error code

Remarks

See also:

`_get_error`

Collects an error code for a file descriptor.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

`int_get_error(fd)`

`int fd`

Arguments

fd File descriptor

Explanation

This function returns the code of the most recent error on the specified file descriptor. Error codes are defined in `sys/errno.h`.

Return value

Error code.

Remarks

See also:

Chapter 2: Standard C Library

Table of Contents

Functions	
abs	2-3
atoi	2-4
atol	2-5
bcmp	2-6
bcopy	2-7
bsearch	2-8
bzero	2-9
calloc	2-10
exit	2-11
free	2-12
getc	2-13
getchar	2-14
gets	2-15
isXXXX...	2-16
labs	2-17
longjmp	2-18
malloc	2-19
memchr	2-20
memcmp	2-21
memcpy	2-22
memmove	2-23
memset	2-24
printf	2-25
putc	2-26
putchar	2-27
puts	2-28
qsort	2-29
rand	2-30
realloc	2-31
setjmp	2-32
sprintf	2-33
srand	2-34
strcat	2-35
strchr	2-36
strcmp	2-37
strcpy	2-38
strcspn	2-39
strlen	2-40
strncat	2-41
strncmp	2-42
strncpy	2-43
strpbrk	2-44
strrchr	2-45
strspn	2-46
strstr	2-47
strtok	2-48
strtol	2-49
strtoul	2-50
toascii	2-51
tolower	2-52
toupper	2-53

abs

Calculates absolute value.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>abs.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
int abs(i)
```

```
int i;
```

Arguments

i Integer

Explanation

This function calculates the absolute value of the integer *i*. This is essentially a function for finding the absolute value of an integer of the type int, but in R3000, int and long are the same size, so on this system, this function is equivalent to the function labs() described later.

Return value

This function returns the absolute value of the argument.

Remarks

See also: labs (p. 2-17).

atoi

Converts a string to an integer.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>convert.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
int atoi(*s)  
char *s;
```

Arguments

s Pointer to a character string

Explanation

Converts a string to its integer equivalent. This function is the same as (long) strtol(s, (chr**) NULL). On this system, it is equivalent to atol(), described later.

Return value

This function returns the result obtained by converting the input value s to an integer.

Remarks

See also: atol (p. 2-4), strtol (p. 2-49).

atol

Converts a character string to a long.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>convert.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
long atol(*s)
char *s;
```

Arguments

s Pointer to a character string

Explanation

This function is the same as (long) strtol(s, (chr**) NULL).

Return Value

This function returns the result obtained by converting the input value s to a long.

Remarks

See also: atoi (p. 2-4), strtol (p. 2-49).

bcmp

Compares memory blocks.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	02/15/98

Syntax

```
int bcmp(*b1, *b2, n)
unsigned char *b1;
unsigned char *b2;
int n;
```

Arguments

b1 Pointer to comparison source 1
b2 Pointer to comparison source 2
n Number of bytes compared

Explanation

This function compares the first *n* bytes of *b1* and *b2*.

Return value

The return value may be as follows, depending on the results of the comparison.

Table 2-1

Result	Return value
<i>b1</i> < <i>b2</i>	<0
<i>b1</i> = <i>b2</i>	=0
<i>b1</i> > <i>b2</i>	>0

Remarks

See also: memcmp (p. 2-21).

bcopy

Copies a memory block.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	02/15/98

Syntax

```
void bcopy(*src, *dest, n)
unsigned char *src;
unsigned char *dest;
int n;
```

Arguments

src Pointer to copy source
dest Pointer to copy destination
n Number of bytes copied

Explanation

This function copies the first *n* bytes of *src* to *dest*.

Return value

None

Remarks

See also: memcpy (p. 2-22).

bsearch

Binary search.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdlib.h</i>	2.x	02/15/98

Syntax

```
void *bsearch(*key, *base, n, w, *fcmp)
unsigned char *key;
unsigned char *base;
size_t n;
size_t w;
int (*fcmp)();
```

Arguments

key Pointer to storage destination of the value to be searched for
base Pointer to storage destination of the array to be searched for
n Number of elements
w Size of one element
fcmp Pointer to address of comparison function

Explanation

This function carries out a binary search on a table of *n* items (of item size *w*) starting from *base*, for an item matching *key*.

Return value

This function returns the address of the first item matching the search key. If no matching item is found, it returns 0.

Remarks

See also:

bzero

Fills a memory block with zeros.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	02/15/98

Syntax

```
void bzero(*p, n)
unsigned char *p;
int n;
```

Arguments

p Pointer to memory block
n Size

Explanation

This function sets *n* bytes to the value 0, starting from *p*.

Return value

None

Remarks

See also:

calloc

Allocates main memory.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>malloc.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void *calloc(n, s)
```

```
size_t n;
```

```
size_t s;
```

Arguments

n Number of blocks

s Size of block

Explanation

This function secures *n* blocks of *s* bytes each from the heap and initializes at 0.

Return value

This function returns a pointer to the memory block secured. If the function fails, it returns NULL.

Remarks

See also: malloc (p. 2-19), realloc (p. 2-31), free (p. 2-12).

exit

Terminates a program normally.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdlib.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void exit(err)
int err;
```

Arguments

err Error code

Explanation

When this function is executed on the PlayStation itself, a system error notice window (including the error code) is displayed, and the system enters an infinite loop. When this function is executed on a development machine, the program currently being executed is terminated, and the system returns to the debug monitor.

Return value

None

Remarks

See also:

free

Releases allocated memory blocks.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>malloc.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void free(*block)
void *block;
```

Arguments

block Pointer to a memory block allocated by a function such as malloc().

Explanation

This function releases memory blocks secured by the functions calloc(), malloc() and realloc().

Return value

None

Remarks

See also: calloc (p. 2-10), malloc (p. 2-19), realloc (p. 2-31).

getc

Gets one character from the stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char getc(fd)
int fd;
```

Arguments

fd File descriptor

Explanation

This function gets one character from the file indicated by *fd*.

Return value

If this function succeeds, it returns the character it has read.

When *getc* reaches the end of the file, or when an error is generated, it returns EOF.

Remarks

Devices and systems with a block size of 1 may all be used as the standard input/output stream as follows.

- Close (0);
- Close (1);
- Open (<device name>, O_RDONLY);
- Open (<device name>, O_WRONLY);

See also: *getchar* (p. 2-14), *gets* (p. 2-15).

getchar

Gets one character from the standard input stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

char **getchar**(*void*)

Arguments

None

Explanation

This function gets one character from the standard input stream. It is the same as `getc(stdin)`.

Return value

The return value is the same as for `getc()`.

Remarks

See also: `getc` (p. 2-13), `gets` (p. 2-15).

gets

Reads a character string from the standard input.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char *gets(*s)
char *s;
```

Arguments

s Pointer to storage destination for input character string

Explanation

This function reads a character string from the standard input stream (stdin) and stores it in s until a new-line character is read.

Return value

If this function succeeds, it returns s the new-line character is discarded and a null character is written immediately after the last character read. If it reaches the end of the file, or if an error is generated, it returns NULL.

Remarks

See also: getc (p. 2-13), getchar (p. 2-14).

isXXXX...

Tests characters.

Library	Header File	Introduced	Documentation Date
libc\libc2.lib	ctype.h	2.x	02/15/98

Syntax

```
long isXXXX(c)
long c;
```

Arguments

c Character

Explanation

This function tests on the character c. All of the tests are macros. The test conditions are as follows.

Table 2–2

Name	Conditions
isalnum(c)	isalpha(c) isdigit(c)
isalpha(c)	isupper(c) islower(c)
isascii(c)	ASCII character
iscntrl(c)	Control character
isdigit(c)	Decimal
isgraph(c)	Printing characters other than space
islower(c)	Lower-case character
isprint(c)	Printing characters including space
ispunct(c)	Printing characters other than space and alphanumerics
isspace(c)	Space, new page, new line, restore, tab
isupper(c)	Upper-case character
isxdigit(c)	Hexadecimal

Return value

This function returns a value other than 0 if the character c satisfies the test conditions, and returns the value 0 if it does not satisfy the test conditions.

Remarks

See also:

labs

Absolute value.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>convert.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

long labs(*i*)

long *i*;

Arguments

i Long value

Explanation

This function calculates the absolute value of *i*.

Return value

This function returns the absolute value of the argument.

Remarks

See also: abs (p. 2-2).

longjmp

Non-local jump.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>setjmp.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void longjmp(p, val)  
jmp_buf p;  
int val;
```

Arguments

p Environment storage variable
val setjmp() Return value

Explanation

This function makes a non-local jump to the destination specified by *p*.

Return value

None If the function executes normally, it does not return.

Remarks

See also: setjmp (p. 2-32).

malloc

Allocates main memory.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>malloc.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void *malloc(s)
size_t s;
```

Arguments

s Number of bytes to be allocated

Explanation

This function secures a block of *s* bytes from the memory heap.

Return value

This function returns a pointer to the secured memory block. If it has failed to secure a block, it returns NULL.

Remarks

Note that the memory heap is defined as follows when the user program is activated:

Bottom address: top address of module + 4.

Top address: available memory -32KB.

There is a bug whereby the area is not completely released in `free()`. This function can be substituted with `malloc2()`, or `malloc3()` from LIBAPI. For details refer to 'Memory Control functions' in `overview/api.doc`.

See also: `calloc` (p. 2-10), `realloc` (p. 2-31), `free` (p. 2-12).

memchr

Searches memory block for a character.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void *memchr(*s, c, n)
unsigned char *s;
unsigned char c;
int n;
```

Arguments

s Pointer to memory block
c Character
n Number of bytes

Explanation

This function searches the memory block of *n* bytes starting from *s*, looking for the first appearance of the character *c*.

Return value

This function returns a pointer to the location at which *c* was found. If *c* was not found, it returns NULL.

Remarks

See also:

memcmp

Compares memory blocks.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	02/15/98

Syntax

```
void *memcmp(*s1, *s2, n)
unsigned char *s1;
unsigned char *s2;
int n;
```

Arguments

s1 Pointer to comparison source memory block1
s2 Pointer to comparison source memory block 2
n Number of bytes compared

Explanation

This function compares the first *n* bytes of *s1* and *s2*.

Return value

This function returns the values shown below, depending on the results of the comparison of *s1* and *s2*.

Table 2-3

Result	Return value
<i>s1</i> < <i>s2</i>	<0
<i>s1</i> = <i>s2</i>	=0
<i>s1</i> > <i>s2</i>	>0

Remarks

See also: *bcmp* (p. 2-6).

memcpy

Copies memory blocks.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	02/15/98

Syntax

```
void *memcpy(*dest, *src, n)
unsigned char *dest;
unsigned char *src;
int n;
```

Arguments

dest Pointer to copy destination memory block
src Pointer to copy source memory block
n Number of bytes copied

Explanation

This function copies the first *n* bytes of *src* to *dest*.

Return value

This function returns *dest*.

Remarks

See also: bcopy (p. 2-7).

memmove

Copies a memory block.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	02/15/98

Syntax

```
void *memmove(*dest, *src, n)
unsigned char *dest;
unsigned char *src;
int n;
```

Arguments

dest Pointer to copy destination memory block
src Pointer to copy source memory block
n Number of bytes copied

Explanation

This function copies the first *n* bytes of *src* to *dest*. The block is copied correctly, even between overlapping objects.

Return value

This function returns *dest*.

Remarks

See also:

memset

Writes specified characters to a memory block.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void *memset(*s, c, n)
unsigned char *s;
unsigned char c;
int n;
```

Arguments

s Pointer to memory block
c Character
n Number of characters

Explanation

This function writes *c* to a memory block of *n* bytes starting at *s*.

Return value

This function returns *s*.

Remarks

See also:

printf

Formatted output.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
int printf (
char *fmt[, argument ...]
)
```

Arguments

fmt Pointer to input format character string

Explanation

Omitted. See a C language reference. Conversion directives f, e, E, g and G cannot be used.

Return value

printf returns the length of the output character string. If an error is generated, the function returns NULL.

Remarks

See also:

putc

Outputs one character to the stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void putc(c, fd)
```

```
char c;
```

```
int fd;
```

Arguments

c Output character

fd File descriptor

Explanation

This function outputs a character *c* to the file indicated by *fd*.

Return value

This function returns *c* if it succeeds, and EOF if an error is generated.

Remarks

See also: putchar (p. 2-27), puts (p. 2-28).

putchar

Outputs one character to the standard output stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void putchar(c)  
char c;
```

Arguments

c Output character

Explanation

This function outputs a character *c* to the standard output. It is the same as `putc(stdout)`.

Return value

The return value is the same as for `putc()`.

Remarks

See also: `putc` (p. 2-26), `puts` (p. 2-28).

puts

Outputs a character string to the standard output stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void puts(*s)
char char *s
```

Arguments

s Pointer to output character string

Explanation

This function outputs a character string ending in NULL to the standard output stream (stdout), and finally outputs a newline character.

Return value

This function returns a non-negative value if it succeeds, and EOF if an error is generated.

Remarks

See also: putc (p. 2-26), putchar (p. 2-27).

qsort

Quick sort.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>qsort.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void qsort (*base, n, w, *fcmp)
void *base;
size_t n;
size_t w;
int (*fcmp)();
```

Arguments

base Pointer to storage destination of array to be sorted
n Number of elements
w Size of on element
fcmp Pointer to address of comparison function

Explanation

This function quick-sorts a table of *n* items (of item size *w*) starting with *base*, with *fcmp* as the comparison function.

Return value

None

Remarks

See also:

rand

Generates random numbers.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>rand.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

int rand(*void*)

Arguments

None

Explanation

This function generates a pseudo-random number from 0 to RAND_MAX (0x7FFF=32767).

Return value

This function returns the pseudo-random number which has been generated.

Remarks

See also: srand (p. 2-33).

realloc

Changing heap memory allocations.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>malloc.h</i>	2.x	02/15/98

Syntax

```
void *realloc(*block, s)
void *block;
size_t s;
```

Arguments

block Pointer to a block secured by a function such as malloc()
s New size

Explanation

This function takes a previously concerned *block* and contracts it or expands it to *s* bytes. If *block* is NULL, this function works in the same way as malloc.

Return value

This function returns the address of the reallocated block. This address may be different to the old address. If it fails to perform the allocation, the function returns NULL. In this case, the old block is not released.

Remarks

See also: calloc (p. 2-10), malloc (p. 2-19), free (p. 2-12).

setjmp

Defines non-local jump destination.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>setjmp.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
int setjmp(p)  
jmp_buf p;
```

Arguments

p Environment storage variable

Explanation

This function stores the destination information for a non-local jump at *p*. If `longjmp(p, val)` is executed, the system will return from `setjmp()`.

Return value

This function returns the value given to the second argument of `longjmp()` when the jump is executed.

Remarks

See also: `longjmp` (p. 2-18).

sprintf

Defines non-local jump destination.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
long sprintf (
char *s,
const char *fmt[,argument...]
)
```

Arguments

s Storage location for variable character string
fmt Input format character string

Explanation

Refer to the various C language references for a detailed explanation of the input format.

The conversion designators [f] [e] [E] [g] [G] are not supported. Use `sprintf2()` from the math library to display floating points.

Return value

The length of the output character string is returned. NULL is returned when an error occurs.

Remarks

See also: `printf()`, `sprintf2()`.

srand

Initializes the random number generator.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>rand.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void srand(seed)  
unsigned long seed;
```

Arguments

seed Random number seed

Explanation

This function sets a new starting point for random number generation. The default is 1.

Return value

None

Remarks

See also: rand (p. 2-30).

strcat

Concatenates character strings.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char *strcat(*dest, *src)
char *dest;
char *src;
```

Arguments

dest Pointer to concatenation target string
src Pointer to concatenation source string

Explanation

This function appends the character string *src* to the end of the character string *dest*.

Return value

This function returns *dest*.

Remarks

See also: `strncat` (p. 2-41).

strchr

Searches for the first location at which a specified character appears in a character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char *strchr(*s, c)
char *s;
char c;
```

Arguments

s Pointer to character string searched
c Character searched for

Explanation

This function searches for the first location at which the character *c* appears in the character string *s*.

Return value

This function returns the address of the location at which *c* appears. If *c* has not been found, it returns NULL.

Remarks

See also:

strcmp

Compares character strings.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	02/15/98

Syntax

```
int strcmp(*s1, *s2)
char *s1;
char *s2;
```

Arguments

s1 Pointer to character string 1
s2 Pointer to character string 2

Explanation

This function compares the character string *s2* with the character string *s1*, treating each character as an unsigned char.

Return value

This function returns one of the values shown below, depending on the comparison result.

Table 2-4

Result	Return value
<i>s1</i> < <i>s2</i>	<0
<i>s1</i> = <i>s2</i>	=0
<i>s1</i> > <i>s2</i>	>0

Remarks

See also:

strcpy

Copies a character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char *strcpy(*dest, *src)
char *dest;
char *src;
```

Arguments

dest Pointer to copy destination character string
src Pointer to copy source character string

Explanation

This function copies the character string *src* to the character string *dest*.

Return value

This function returns *dest*.

Remarks

See also: strncpy (p. 2-43).

strcspn

Search for a partial character string made up solely of characters not included in the specified character set.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	02/15/98

Syntax

```
int strcspn(*s1, *s2)
char *s1;
char *s2;
```

Arguments

s1 Pointer to character string
s2 Pointer to character group

Explanation

This function returns the length of the first part of the character string *s1* consisting only of characters not included in the character string *s2*.

Return value

This function returns the length of the partial character string found.

Remarks

See also:

strlen

Counts the number of characters in a character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
int strlen(*s)
char *s;
```

Arguments

s Pointer to character string

Explanation

This function counts the number of characters in a character string s.

Return value

This function returns the number of characters.

Remarks

See also:

strncat

Concatenates character strings.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char *strncat(*dest, *src, n)
char *dest;
char *src;
int n;
```

Arguments

dest Pointer to concatenation destination array
src Pointer to concatenation source character string
n Number of characters concatenated

Explanation

This function appends the first *n* characters from *src* to the end of the character string *dest*.

Return value

This function returns *dest*.

Remarks

See also:

strncmp

Compares character strings.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
int strcmp(*s1, *s2, n)
char *s1;
char *s2;
int n;
```

Arguments

- s1 Pointer to character string 1
- s2 Pointer to character string 2
- n Number of characters compared

Explanation

This function compares the first *n* characters of *s1* and *s2*, treating each character as unsigned char.

Return value

This function returns one of the following values, depending on the comparison result (the values are the same as for strcmp).

Table 2-5

Result	Return value
s1<s2	<0
s1=s2	=0
s1>s2	>0

Remarks

See also:

strncpy

Copies a character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char *strncpy(*dest, *src, n)
char *dest;
char *src;
int n;
```

Arguments

dest Pointer to copy destination array
src Pointer to copy source character string
n Number of bytes

Explanation

This function copies *n* bytes worth of *src* to the character string *dest*. When the number of characters copied reaches *n*, the copying is terminated.

Return value

This function returns *dest*.

Remarks

See also:

strpbrk

Searches for the first occurrence of a character within a specified character set.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char *strpbrk(*s1, *s2)
char *s1;
char *s2;
```

Arguments

s1 Pointer to character string searched
s2 Pointer to character group

Explanation

This function searches for the first location at which any of the characters contained in the character string *s2* appear within the character string *s1*.

Return value

This function returns the address of the character found. If no character was found, it returns NULL.

Remarks

See also:

strrchr

Searches for the last location of a specified character in a character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char *strrchr(*s, c)
char *s;
char c;
```

Arguments

s Pointer to character string searched
c Character searched for

Explanation

This function searches for the last occurrence of the character *c* within the character string *s*.

Return value

This function returns the address of *c*. If *c* does not occur, it returns NULL.

Remarks

See also:

strspn

Searches for the part of a character string consisting solely of characters contained in the specified character set.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	02/15/98

Syntax

```
int strspn(*s1, *s2)
```

```
char *s1;
```

```
char *s2;
```

Arguments

s1 Pointer to character string

s2 Pointer to character group

Explanation

This function returns the length of the first part of the character string *s1* which consists solely of characters included in the character string *s2*.

Return value

This function returns the length of the partial character string it has found.

Remarks

See also:

strstr

Searches for the location of a specified partial character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
char *strstr(*s1, *s2)
char *s1;
char *s2;
```

Arguments

s1 Pointer to character string searched
s2 Pointer to character string searched

Explanation

This function searches for the first location of character string *s2* within character string *s1*.

Return value

This function returns the address of *s2*. If it was not found, the function returns NULL.

Remarks

See also:

strtok

Searches for a character string demarcated by certain characters within a specified character set.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	02/15/98

Syntax

```
char *strtok(*s1, *s2)
char *s1;
char *s2;
```

Arguments

s1 Pointer to character string searched
s2 Pointer to separator characters

Explanation

This function treats character string *s1* as a set of tokens punctuated by one or more characters from the separator character string *s2*. The first call in the sequence searches *s1* for the first character that is not contained within *s2*.

The first time `strtok()` is called, the starting address of the first token of *s1* is returned, and a NULL character is written in immediately after this token. The address of *s1* is stored in the function, and then, when `strtok()` is called with NULL entered as the first argument, a search is carried out until there are no tokens left in the character string *s1*.

Return value

This function returns the starting address of the tokens found in *s1*. If it does not find any *s1* tokens, `strtok()` returns NULL.

Remarks

See also:

strtol

Performs long conversion of a character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>convert.h</i>	2.x	02/15/98

Syntax

```
long strtol(*s, **endp)
char *s;
char **endp;
unsigned int base;
```

Arguments

s Pointer to character string
endp Storage destination of pointer to a non-convertible character string
base Radix specification

Explanation

This function converts a character string *s* to long type (the same as int type in R3000). *s* must be formatted as follows.

[ws][sn][ddd]

- [ws] white space (may be omitted)
- [sn] sign (may be omitted)
- [ddd] number string (may be omitted)

The value of *base* determines the format of [ddd]. The letters a (or A) thru z (or Z) are ascribed values from 10-35. Only values less than *base* may be included in [ddd]. For some values of *base*, optional characters may precede the sequence of letters and digits following the sign (if present).

Table 2-6

Base value	Optimal characters
2	0b, 0B
8	"O," "o"
16	0x, 0X

The function `strtol()` stops converting when it encounters a non-convertible character, and if *endp* is not NULL, it sets *endp* as the pointer to the character at which it stopped converting.

Return value

This function returns the result obtained by converting the input value *s* to a long. If an error is generated, it returns 0.

Remarks

See also: `strtoul` (p. 2-49).

strtoul

Performs unsigned long conversion of a character string.

Library	Header File	Introduced	Documentation Date
libc\libc2.lib	convert.h	2.x	02/15/98

Syntax

```
unsigned long strtoul(*s, endp, base)
char *s;
char **endp;
int base;
```

Arguments

- s Pointer to character string
- endp Storage destination of pointer to a non-convertible character string
- base Radix specification

Explanation

This function converts a character string s to unsigned long type (the same as unsigned int type in R3000). s must be formatted as follows.

[ws][sn][ddd]

- [ws] white space (may be omitted)
- [sn] sign (may be omitted)
- [ddd] number string (may be omitted)

The value of base determines the format of [ddd]. The letters a (or A) thru z (or Z) are ascribed values from 10-35. Only values less than base may be included in [ddd]. For some values of base, optional characters may precede the sequence of letters and digits following the sign (if present).

Table 2-7

Base value	Optimal characters
2	0b, 0B
8	“O,” “o”
16	0x, 0X

The function strtoul() stops converting when it encounters a non-convertible character, and if endp is not NULL, it sets endp as the pointer to the character at which it stopped converting.

Return value

This function returns the result obtained by converting the input value s to a long.

Remarks

See also: strtol (p. 2-49).

toascii

Masks bit 7 of the input value.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>ctype.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
long toascii(c)
long c;
```

Arguments

c Value

Explanation

This is a macro which masks the 7th bit.

Return value

This macro returns a value obtained by masking the 7th bit of the input value *c*.

Remarks

See also:

tolower

Converts a letter to lower-case.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>ctype.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
long tolower(c)
```

```
long c;
```

Arguments

c Character

Explanation

This macro converts a character *c* to lower case. The behavior of this macro when it is given a value not an upper-case letter is undefined.

Return value

This macro returns a lower-case letter that corresponds to *c*.

Remarks

See also:

toupper

Converts a character to upper case.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>ctype.h</i>	2.x	02/15/98

Syntax

```
long toupper(c)  
long c;
```

Arguments

c Character

Explanation

This macro converts a character *c* to upper case. The behavior of this macro when it is given a value not a lower-case letter is undefined.

Return value

This macro returns an upper-case letter that corresponds to the character *c*.

Remarks

See also:

Chapter 3: Math Library

Table of Contents

Functions	
acos	3-3
asin	3-4
atan	3-5
atan2	3-6
atof	3-7
ceil	3-8
cos	3-9
cosh	3-10
exp	3-11
fabs	3-12
floor	3-13
fmod	3-14
frexp	3-15
hypot	3-16
ldexp	3-17
log	3-18
log10	3-19
modf	3-20
pow	3-21
printf2	3-22
sin	3-23
sinh	3-24
sprintf2	3-25
sqrt	3-26
strtod	3-27
tan	3-28
tanh	3-29

acos

Arccosine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double `acos(x)`

double *x*;

Argument

x Value whose arccosine is to be determined, ranging from -1 to 1

Explanation

Determines the arccosine function of *x*.

Return value

Arccosine function of *x*, ranging from 0 to pi. Error processing is shown as follows:

Table 3-1

Conditions	Return value	Error
<code>fabs(<i>x</i>)>1</code>	0	Domain error

Remarks

See also: `cos` (p. 3-9), `asin` (p. 3-4), `atan` (p. 3-5).

asin

Arcsine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double asin(*x*)

double *x*;

Argument

x Value whose arcsine is to be determined, ranging from -1 to 1.

Explanation

Determines the arcsine function of *x*.

Return value

Arcsine function of *x*, ranging from -pi/2 to pi/2.

Error processing is as follows:

Table 3-2

Conditions	Return value	Error
$\text{fabs}(x) > 1$	0	Domain error

Remarks

See also: sin (p. 3-23), acos (p. 3-3), atan (p. 3-5).

atan

Arctangent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double atan(*x*)

double *x*;

Argument

x Value whose arctangent is to be calculated

Explanation

Determines the arctangent function of *x*.

Return value

Arctangent function of *x*, ranging from $-\pi/2$ to $\pi/2$ radians.

Remarks

See also: tan (p. 3-27), asin (p. 3-4), acos (p. 3-3), atan2 (p. 3-6).

atan2

Arctangent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double atan2(*x*, *y*)

double *x*, *y*;

Arguments

x, *y* Floating-point value

Explanation

Determines the arctangent of x/y .

Return value

Arctangent function of x/y , ranging from $-\pi$ to π .

Remarks

If *x* and *y* are 0, a value of 0 is returned.

See also: atan() (p. 3-5)

atof

Converts a string to a floating-point equivalent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double **atof**(*char *s*)

Arguments

s Pointer to a string

Explanation

Converts a string "s" to its floating-point (double type) equivalent.

Return value

Returns the result from converting input string "s" to a floating point equivalent in double type. When the converted value overflows, either +HUGE_VAL(1.797693134862316e+308) or -HUGE_VAL depending on the sign, will be returned. 0 is returned for underflow case.

Remarks

Error handling is as follows:

Table 3–3

Condition	Returned Value	Error Type
Overflow	+/- HUGE_VAL	Region Error
Underflow	0	Region Error

See also: strtod (p. 3-27).

ceil

Minimum integer not less than x (ceiling function).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double `ceil`(x)

double x ;

Argument

x Floating-point value

Explanation

This function determines the minimum integer (double type) not less than x .

Return value

Minimum integer (double type) not less than x

Remarks

See also: `floor` (p. 3-13).

cos

Cosine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax**double** cos(*x*)**double** *x*;**Argument***x* Angle in radians**Explanation**Determines the cosine function of *x*.**Return value**Cosine function of *x* (cos(*x*))**Remarks****See also:** sin (p. 3-23), tan (p. 3-27), acos (p. 3-3).

cosh

Hyperbolic cosine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double cosh(*x*)

double *x*;

Argument

x Angle in radians

Explanation

Determines the hyperbolic cosine function of *x*.

Return value

Hyperbolic cosine function of *x* (cosh(*x*))

Remarks

See also: sinh (p. 3-24), tanh (p. 3-29).

exp

Exponent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double exp(*x*)

double *x*;

Argument

x Floating-point value

Explanation

This function determines the exponential function of *x*.

Return value

e raised to the *x*-th power (e^{**x})

Remarks

See also: pow (p. 3-21), log (p. 3-18).

fabs

Absolute value (macro).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double fabs(*x*)

double *x*;

Argument

x Floating-point value

Explanation

This function determines an absolute value.

Return value

Absolute value of *x*.

Remarks

This is a macro.

See also:

floor

Maximum integer not more than x (lower function).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double floor(x)

double x ;

Argument

x Floating-point value

Explanation

This function determines the maximum integer (double type) not more than x .

Return value

Maximum integer not more than x (double type)

Remarks

See also: `ceil` (p. 3-6).

fmod

Floating-point remainder resulting from x/y .

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double fmod(x , y)

double x , y ;

Arguments

x Floating-point value

y Floating-point value

Explanation

This function determines the floating-point remainder resulting from x/y .

Return value

Floating-point remainder resulting from x/y .

If y is 0, a value of 0 is returned.

Remarks

The sign of the return value is the same as of x .

See also:

frexp

Resolution into normalized decimal part and the part raised to the second power.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

```
double frexp(x, *n)
double x;
int *n;
```

Arguments

x Floating-point value
n Pointer to a buffer for storing the part raised to the second power

Explanation

This function resolves *x* into a decimal portion normalized at $[1/2, 1)$ and the portion that is raised to the second power. The decimal part is returned, and the part raised to the second power is stored in *n*.

Return value

Normalized decimal part. $[1/2, 1)$.

Remarks

A pair of square brackets `[]` indicates a closed area, while a pair of parentheses `()` indicates an open area.

See also:

hypot

Absolute value of a complex number.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double hypot(*x*, *y*)

double *x*, *y*;

Arguments

x, *y* Floating-point value

Explanation

This function computes the square root of the sum of the squares of *x* and *y*.

Return value

Square root of the sum of (*x***2) and (*y***2).

Remarks

See also:

ldexp

Calculates a real number from a mantissa and an exponent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	6/22/98

Syntax

```
double ldexp(x, n)
double x;
int n;
```

Arguments

x Floating-point value
n Integral exponent

Explanation

This function determines a real number from a mantissa and an exponent.

Return value

Value of *x* multiplied by 2 raised to the *n*th power.

Remarks

See also:

log

Natural logarithm.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double log(*x*)

double *x*;

Argument

x Value subjected to logarithmic operation

Explanation

Determines the natural logarithmic function of *x*.

Return value

Logarithm of *x* (ln(*x*)).

x must be greater than zero. Otherwise, a domain error results. Error processing is as follows.

Table 3–4

Conditions	Return value	Error
$x < 0$	0	Domain error
$x == 0$	1	Range error

Remarks

See also: exp (p. 3-11), log10 (p. 3-19).

log10

Logarithm whose base is 10.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double log10(x)

double x;

Argument

x Value subjected to logarithmic operation

Explanation

This function determines the logarithmic function of *x* whose base is 10.

Return value

Logarithm of *x* whose base is 10 (log10(x))

x must be greater than zero. Otherwise, an error results. Error processing is as follows.

Table 3-5

Conditions	Return value	Error
$x < 0$	0	Domain error
$x == 0$	1	Range error

Remarks

See also: log (p. 3-18).

modf

Separation into integral and fractional parts.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double modf(*x*, **y*)

double *x*, **y*;

Arguments

x Floating-point value

y Pointer to a buffer for storing the integral part

Explanation

This function separates *x* into integral and fractional parts. The integral part is stored in *y*, with the return value being the fractional part.

Return value

Decimal part of *x*

Remarks

The signs of both the integral and decimal parts are the same as *x*.

See also:

pow

x raised to the y -th power.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double pow(x , y)

double x ;

double y ;

Arguments

x Numerical value

y Power

Explanation

This function raises x to the y -th power.

Return value

x raised to the y -th power (x^{**y}).

Error processing is as follows.

Table 3-6

Condition	Return value	Error
$x==0 \ \&\& \ y>0$	0	
$x==0 \ \&\& \ y\leq 0$	1	Domain error
$x<0 \ \&\& \ [y \text{ not an integer}]$	0	Domain error

Remarks

See also: exp (p. 3-11).

printf2

Converts formatted output as standard output stdout (corresponding to floating-point and double-precision arguments).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	6/22/98

Syntax

```
int printf2 (  
const char *fmt[,argument...]  
)
```

Arguments

fmt Pointer to input format character string

Explanation

The conversion directives [f] [e] [E] [g] and [G] can be used.

Stack consumption is greater than with printf.

Return value

Output character length is returned.

See also: `sprintf2`

sin

Sine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double sin(*x*)

double *x*;

Argument

x Angle in radians

Explanation

Determines the sine function of *x*.

Return value

Sine function of *x* (sin(*x*))

Remarks

See also: cos (p. 3-9), tan (p. 3-27), asin (p. 3-4).

sinh

Hyperbolic sine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double sinh(*x*)

double *x*;

Argument

x Angle in radians

Explanation

Determines the hyperbolic sine function of *x*.

Return value

Hyperbolic sine function of *x* (sinh(*x*))

Remarks

See also: cosh (p. 3-10), tanh (p. 3-29).

sprintf2

Format output to an array (corresponding to floating-point and double-precision arguments).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	6/22/98

Syntax

```
int sprintf2 (
char *s,
const char *fmt[,argument...]
)
```

Arguments

s Pointer to storage destination of converted character string
fmt Pointer to input format character string

Explanation

The conversion directives [f] [e] [E] [g] and [G] can be used.

Stack consumption is greater than with `sprintf`.

Return value

Output character length is returned.

See also: `printf2`

sqrt

Square root.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double sqrt(*x*)

double *x*;

Argument

x Non-negative floating-point value

Explanation

This function determines the non-negative square root of *x*.

Return value

Square root of *x*.

Remarks

Error processing is as follows.

Table 3-7

Condition	Return value	Error
<i>x</i> <0	0	Domain error

See also:

strtod

Converts a string to a floating-point equivalent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

```
double strtod (
char *s,
char **endp
)
```

Arguments

s Pointer to a string
endp Holds a pointer to a string that was unable to be converted

Explanation

This function converts a string to a double type floating-point equivalent.

"s" must be one of the following:

[ws][sn][ddd]

- [ws] White space (may be omitted)
- [sn] Sign (may be omitted)
- [ddd] Number string (may be omitted)

Stops converting upon encountering a character that was unable to be converted. If *endp* is not NULL, the pointer to the character in error is set to *endp*.

Return value

Returns the result from converting input string "s" to a floating point in double type. When the converted value overflows, either +HUGE_VAL(1.797693134862316e+308) or -HUGE_VAL according to the sign, will be returned. 0 is returned for underflow case. If no conversion could be performed, 0 is returned.

Remarks

Error handling is as follows:

Table 3–8

Condition	Returned Value	Error Type
Overflow	+/- HUGE_VAL	Area Error
Underflow	0	Area Error

See also: atof (p. 3-7).

tan

Tangent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double **tan**(*x*)

double *x*;

Argument

x Angle in radians

Explanation

Determines the tangent function of *x*.

Return value

Tangent function of *x* (tan(*x*))

Remarks

See also: sin (p. 3-23), cos (p. 3-9), atan (p. 3-5).

tanh

Hyperbolic tangent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	7/31/96

Syntax

double tanh(*x*)

double *x*;

Argument

x Angle in radians

Explanation

Determines the hyperbolic tangent function of *x*.

Return value

Hyperbolic tangent function of *x* (tanh(*x*))

Remarks

See also: sinh (p. 3-24), cosh (p. 3-10).

Chapter 4: Memory Card Library

Table of Contents

Functions

InitCARD	4-3
StartCARD	4-4
StopCARD	4-5
_bu_init	4-6
_card_auto	4-7
_card_chan	4-8
_card_clear	4-9
_card_format	4-10
_card_info	4-11
_card_load	4-12
_card_read	4-13
_card_status	4-14
_card_wait	4-15
_card_write	4-16
_new_card	4-17

InitCARD

Initializes memory card BIOS

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

```
void InitCARD(val)
long val;
```

Arguments

val Indicates sharing with controller
 0: Not shared
 1: Shared

Explanation

Initializes the memory card BIOS and enters an idle state. Specify in *val* whether or not there is sharing with the controller.

When the BIOS is subsequently put into operation by StartCARD(), the low-level interface function that starts _card can be used directly.

The memory card file system uses these interfaces internally, so InitCARD needs to be executed before _bu_init().

There is no effect on the controller.

Return value

None

Remarks

See also: bu_init (p. 4-6).

StartCARD

Starts memory card BIOS.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

void StartCARD(*void*)

Arguments

None

Explanation

Changes the memory card BIOS initialized by InitCARD() to a run state.

Performs ChangeClearPAD(1) internally.

Return value

None

Remarks

See also: InitCARD (p. 4-3), StopCARD (p. 4-5), _bu_init (p. 4-6), ChangeClearPAD (see libapi).

StopCARD

Stops memory card BIOS.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	10/01/97

Syntax

void StopCARD(*void*)

Arguments

None

Explanation

Changes memory card BIOS to an idle state (the same state as that immediately after executing InitCARD())

Also stops the controller. It is necessary to call StartPAD() to start the controller.

Return value

None

Remarks

See also: InitCARD (p. 4-3), StartCARD (p. 4-4), _bu_init (p. 4-6), ChangeClearPAD (see libapi).

`_bu_init`

Initializes memory card file system.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

`void _bu_init(void)`

Arguments

None

Explanation

Initializes the memory card file system.

The initialization routine does not execute automatically, so this function is required to explicitly initialize the file system.

Return value

None

Remarks

See also: InitCARD (p. 4-3), StartCARD (p. 4-4), StopCARD (p. 4-5).

_card_auto

Sets automatic format function.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

```
long _card_auto(val)
long val;
```

Arguments

val Indicates automatic formatting

Explanation

Sets automatic format function.

When 0 is specified in *val*, it is disabled; when 1 is set, it is enabled.

Return value

Previously set automatic format value.

Remarks

This function should be used for testing purposes only.

See also:

_card_chan

Gets a memory card BIOS event.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

long_card_chan(*void*)

Arguments

None

Explanation

Returns the device number of the memory card that just generated an event.

Return value

2-digit hex device number.

Remarks

See also: `card_status` (p. 4-14), `_card_wait` (p. 4-15).

`_card_clear`

Clears unconfirmed flags.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

```
long _card_clear(chan)
long chan;
```

Arguments

chan Port number x 16 + Card number

Explanation

Performs a dummy write to the system management area of the card and clears unconfirmed flags specified in the card.

Port number for Port 1 is zero. Port number for Port 2 is one. Card number is zero when a standard controller is connected. If a multi-tap is connected, then card number may be in the range 0-3.

This function executes asynchronously, so it terminates immediately. Multiplex processing to the same card slot is not performed. That is, multiple `_card_clear` calls to the same multi-tap cannot be processed synchronously. Actual processing termination is communicated by an event. (See table below.)

Table 4-1: Posts an event on completion of processing

Source Descriptor/Event Class	Contents
HwCARD/EvSpiOE	Ends process
HwCARD/EvSpTIMOUT	Card not connected
HwCARD/EvSpNEW	New card detected
HwCARD/EvSpERROR	Error generated
HwCARD/EvSpUNKOWN	Source unknown

Return value

1 if successful processing registration, otherwise 0.

Remarks

See also: `card_info` (p. 4-10).

_card_format

Formats the memory card

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	6/22/98

Syntax

```
long _card_format (
long chan
)
```

Arguments

chan Port number x 16 + Card number
 Port number (A Port:0, B Port:1)
 Card number (usually 0)

Explanation

Formats the memory card.

Synchronous functions between approximately 144 Vsync are blocked.

Does not enter critical section.

Return value

1 if formatting is successful, otherwise 0.

Remarks

See also: `_card_load()`.

_card_info

Gets card status.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

```
long _card_info(chan)
long chan;
```

Arguments

chan Port number x 16 + Card number

Explanation

Tests the connection of the memory card specified in *chan*.

Port number for Port 1 is zero. Port number for Port 2 is one. Card number is zero when a standard controller is connected. If a multi-tap is connected, then card number may be in the range 0-3.

Multiplex processing to the same card slot is not performed. That is, multiple `_card_clear` calls to the same multi-tap cannot be processed synchronously. Actual processing termination is communicated by an event. (See table below.)

Table 4-2: Posts an event on completion of processing

Source Descriptor/Event Class	Description
SwCARD/EvSpIOE	Connected
SwCARD/EvSpTIMOUT	Not connected
SwCARD/EvSpNEW	No writing after connection
SwCARD/EvSpERROR	Generates an error

Return value

1 if successful processing registration, otherwise 0.

This function executes asynchronously, so it terminates immediately.

Remarks

Do not use `_new_card()` to suppress EvSpNEW.

See also: `_card_clear` (p. 4-9), `_new_card` (p. 4-17).

_card_load

Tests logical format

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

long _card_load(*chan*)

long *chan*;

Arguments

chan Port number x 16 + Card number

Explanation

Reads file management information for the card specified by *chan* in the file system in order to get asynchronous access using the I/O management service.

Port number for Port 1 is zero. Port number for Port 2 is one. Card number is zero when a standard controller is connected. If a multi-tap is connected, then card number may be in the range 0-3.

_card_load must be called at least once before you can use open() on a memory card file in O_NOWAIT mode. The function does not have to be reissued unless a card is changed. This function executes asynchronously, so it terminates immediately. Multiplex processing to the same card slot is not performed. That is, multiple _card_clear calls to the same multi-tap cannot be processed synchronously. Actual processing termination is communicated by an event. (See table below.)

Table 4-3: Posts an event on completion of processing

Source Descriptor/ Event Class	Contents
SwCARD/EvSpIOE	Read completed
SwCARD/EvSpTIMOUT	Not connected
SwCARD/EvSpNEW	Uninitialized card
SwCARD/EvSpERROR	Generates an error

Return value

1 if the read is successful, otherwise 0.

Remarks

See also: format (see libcd), card_info (p. 4-10).

_card_read

Reads one block from the memory card.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

```
long _card_read(chan, block, *buf)
long chan;
long block;
long *buf;
```

Arguments

chan Port number x 16 + card number
block Target block number
buf Pointer to 128 byte data buffer

Explanation

Reads 128 bytes of buffer data into *buf* from the target block number (*block*) of the memory card of the specified channel (*chan*).

Port number for Port 1 is zero. Port number for Port 2 is one. Card number is zero when a standard controller is connected. If a multi-tap is connected, then card number may be in the range 0-3.

This function executes asynchronously so it terminates immediately after completion. Multiplex processing to the same card slot is not performed. Actual processing termination is communicated by an event. (See table below.)

Table 4-4: Posts an event on completion of processing

Source Descriptor / Event Class	Contents
HwCARD/EvSpIOE	Ends processing
HwCARD/EvSpTIMOUT	Card not connected
HwCARD/EvSpNEW	New card detected
HwCARD/EvSpERROR	Error generated
HwCARD/EvSpUNKOWN	Source unknown

Return value

1 if successful processing registration, otherwise 0.

Remarks

This function exists within the low-level interface and is one of the special functions used for testing.

See also: `_card_write` (p. 4-16), `open` (see `libapi`), `read` (see `libapi`).

`_card_status`

Gets memory card BIOS status.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

```
long _card_status(drv)
long drv;
```

Arguments

drv Port number

Explanation

Gets the memory card BIOS status of each slot, *drv*. Specify *drv* as 0 for Port 1, 1 for Port 2.

This is a synchronous function.

Return value

If the memory card BIOS is in run state, it can return any of the following values.

Table 4-5

Value	State
0x01	Idle processing
0x02	READ processing
0x04	WRITE processing
0x08	Connection test processing registration
0x11	No registered processing (just prior to EvSpTIMOUT generation)
0x21	No registered processing (just prior to EvSpERROR generation)

Remarks

See also: `card_wait` (p. 4-15), `_card_chan` (p. 4-8).

_card_wait

Waits for memory card BIOS completion

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	10/01/97

Syntax

```
long _card_wait(drv)  
long drv;
```

Arguments

drv Sets slot number

Explanation

Wait until registration processing completes for the *drv* slot. Specify *drv* as 0 for Port 1, 1 for Port 2.

Return value

Always 1.

Remarks

See also: `_card_status` (p. 4-14), `_card_chan` (p. 4-8).

_card_write

Writes to one block of the memory card.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	10/01/97

Syntax

```
long _card_write(chan, block, *buf)
long chan;
long block;
long *buf;
```

Arguments

chan Port number x 16 + card number
block Target block number
buf Pointer to 128-byte data buffer

Explanation

Writes 128 bytes of buffer data pointed to by *buf* to the target block number (*block*) of the memory card of the specified channel (*chan*).

Specifies Port number x 16 + Card number in *chan*. Port 1 is 0, and Port 2 is 1. The card number is normally 0.

This function executes asynchronously, so it terminates immediately. Multiplex processing to the same card slot is not performed. That is, multiple `_card_clear` calls to the same multi-tap cannot be processed synchronously. Actual processing termination is communicated by an event. (See table below.)

Table 4-6: Posts an event on completion of processing

Source Descriptor/Event Class	Contents
HwCARD/EvSpiOE	Ends process
HwCARD/EvSpTIMOUT	Card not connected
HwCARD/EvSpNEW	New card detected
HwCARD/EvSpERROR	Error generated
HwCARD/EvSpUNKOWN	Source unknown

Return value

1 if successful processing registration, otherwise 0.

Remarks

This function exists within the low-level interface and is one of the special functions used for testing only. Do not use this function in your code, it is too low level.

See also: `_card_read` (p. 4-13), `open` (see `libapi`), `write` (see `libapi`).

`_new_card`

Changes settings of unconfirmed flag test.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	7/31/96

Syntax

`void _new_card(void)`

Arguments

None

Explanation

Masks the generation of an EvSpNEW event immediately after `_card_read()` or `_card_write()`.

Terminates immediately even though it is a synchronous function.

Return value

None

Remarks

See also: `_card_clear` (p. 4-9), `_card_read` (p. 4-13), `_card_write` (p. 4-16).

Chapter 5: Extended Memory Card Library

Table of Contents

Functions	
MemCardAccept	5-3
MemCardCallback	5-4
MemCardClose	5-5
MemCardCreateFile	5-6
MemCardDeleteFile	5-7
MemCardEnd	5-8
MemCardExist	5-9
MemCardFormat	5-11
MemCardGetDirent	5-12
MemCardInit	5-13
MemCardOpen	5-14
MemCardReadData	5-15
MemCardReadFile	5-16
MemCardStart	5-18
MemCardStop	5-19
MemCardSync	5-20
MemCardUnformat	5-21
MemCardWriteData	5-22
MemCardWriteFile	5-23

MemCardAccept

Checks Memory card status.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	5/22/97

Syntax

```
long MemCardAccept (
long chan
)
```

Arguments

chan port number + card number
port number(port A: 0x00; port B: 0x10)
card number (normally 0)

Explanation

Connection with the memory card specified by *chan* is tested. The result is returned by the function.

If the card is connected, additional information is obtained.

If the card is new, `_card_clear()` and `_card_load()` are executed, allowing the use of file access functions.

`MemCardAccept()` must be executed before using file access functions such as `MemCardOpen()`.

`MemCardAccept()` does not need to be called again as long as the card is not swapped.

The function itself operates asynchronously so it exits immediately. However, multiple instances of the operation cannot be registered.

The completion of the operation is determined by `MemCardSync()` or an exit callback.

The maximum time required to perform this operation is 76 VSyncs. Approximately 4 VSyncs are needed if a card is not present.

The results are returned with the function or the exit callback.

A new card is detected only once by returning `McErrNewCard`. Subsequent calls will return `McErrNone`.

The function result can have the values shown below.

Value	Macro	Status
0x00	<code>McErrNone</code>	Connected
0x01	<code>McErrCardNotExist</code>	Not connected
0x02	<code>McErrCardInvalid</code>	Bad card
0x03	<code>McErrNewCard</code>	New card (card was replaced)
0x04	<code>McErrNotFormat</code>	Not formatted

Return value

1 is returned if the operation was successfully registered, otherwise 0 is returned.

Remarks

Asynchronous function. Time spent: 4 - 76 VSyncs.

See also: `MemCardOpen()`, `MemCardReadFile()`, `MemCardWriteFile()`.

MemCardCallback

Define exit callback.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	5/22/97

Syntax

```
MemCB MemCardCallback (
MemCB func
)
```

Arguments

func pointer to callback function

Explanation

The callback triggered when an asynchronous function completes is set to the function pointed to by *func*.

No callback is generated if *func* is set to 0.

MemCB uses the following format for callback functions:

```
typedef void (*MemCB)( unsigned long cmds, unsigned long result )
```

cmds the completed asynchronous function (see below)

result the execution result from the asynchronous function

Allowed value for *cmds*:

Value	Macro	Function
0x01	McFuncExist	MemCardExist
0x02	McFuncAccept	MemCardAccept
0x03	McFuncReadFile	MemCardReadFile
0x04	McFuncWriteFile	MemCardWriteFile
0x05	McFuncReadData	MemCardReadData
0x06	McFuncWriteData	MemCardWriteData

See the sections on the respective functions for details of the result value.

Return value

The address of the previously set callback is returned.

Remarks

Asynchronous function. Exits immediately.

See also: MemCardAccept(), MemCardExit(), MemCardReadFile(), MemCardWriteFile(), MemCardReadData(), MemCardWriteData().

MemCardClose

Close file.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	5/22/97

Syntax

```
void MemCardClose(void)
```

Arguments

None

Explanation

The file opened with MemCardOpen() is closed.

Return value

None

Remarks

Asynchronous function. Exits immediately.

See also: MemCardOpen().

MemCardCreateFile

Create a new file in the memory card

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	6/22/98

Syntax

```
long MemCardCreateFile (
long chan,
char *file,
long blocks
)
```

Arguments

chan port number + card number
port number (port A: 0x00, port B: 0x10)
card number (normally 0)

file filename

blocks number of blocks

Explanation

The specified file is created in the memory card.

The block parameter is given in units of 8192 bytes.

Return value

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Card is not connected
0x02	McErrCardInvalid	Communication error generated
0x04	McErrNotFormat	Not formatted
0x06	McErrAlreadyExist	File already exists
0x07	McErrBlockFull	Not enough available blocks
-1	None	A non-synchronous function is active.

Remarks

Synchronous function.

Blocking time: 1 - 4 VSyncs for normal exit. 4 - 76 VSyncs otherwise.

Does not enter critical section.

See also: MemCardOpen(), MemCardWriteFile().

MemCardDeleteFile

Delete file from memory card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	6/22/98

Syntax

```
long MemCardDeleteFile (
long chan,
char *file
)
```

Arguments

chan port number + card number
port number (port A: 0x00, port B: 0x10)
card number (normally 0)
file filename

Explanation

The specified file is deleted from the memory card.

Return value

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Card is not connected
0x02	McErrCardInvalid	Communication error generated
0x04	McErrNotFormat	Not formatted
0x05	McErrFileNotExist	File not found
-1	None	A non-synchronous function is active.

Remarks

Synchronous function.

Blocking time: 1 - 4 VSyncs for normal exit. 4 - 76 VSyncs otherwise.

Does not enter critical section.

See also:

MemCardEnd

Terminates memory card system.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	5/22/97

Syntax

```
void MemCardEnd(void)
```

Arguments

None

Explanation

The memory card system is terminated.

MemCardStop() needs to be executed first if the system was activated from MemCardStart().

Return value

None

Remarks

Synchronous function. Exits immediately.

See also: MemCardInit().

MemCardExist

Get connection status of card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	5/22/97

Syntax

```
long MemCardExist (
long chan
)
```

Arguments

chan port number + card number
 port number (port A: 0x00, port B: 0x10)
 card number (normally 0)

Explanation

The connection status of the memory card specified by *chan* is tested and the result returned by the function.

MemCardExist() requires less time to execute than MemCardAccept() since only the presence of the card is checked.

MemCardAccept() must be used if more detailed card information, such as whether the card is formatted or not, is required.

If cards are swapped, MemCardAccept() must be executed before using file access functions such as MemCardOpen().

The function itself is an asynchronous function and will exit immediately. Multiple instances of the function cannot be registered.

The completion of MemCardAccept() can be determined via MemCardSync() or the exit callback.

The time required by MemCardAccept() is approximately 4 VSyncs.

Results are returned as the result value from MemCardSync() or the exit callback.

A new card will be detected (McErrNewCard) as long as MemCardAccept() is not executed.

The function result can have the values shown below.

Value	Macro	Status
0x00	McErrNone	Connected
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Bad card
0x03	McErrNewCard	New card (card was replaced)

Return value

1 is returned if the command was successfully registered.
 Otherwise, 0 is returned.

Remarks

5-10 Extended Memory Card Library Functions

See also: MemCardAccept().

MemCardFormat

Format memory card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	6/22/98

Syntax

```
long MemCardFormat (
long chan
)
```

Arguments

chan port number + card number
 port number (port A: 0x00, port B: 0x10)
 card number (normally 0)

Explanation

The specified memory card is formatted.

Return value

Value	Macro	Status
0x00	McErrNone	Connected
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Communication error
-1	None	A non-synchronous function is active

Remarks

Synchronous function. Blocking time: approx. 144 VSyncs.

Does not enter critical section.

See also:

MemCardGetDirentry

Get directory information from memory card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	6/22/98

Syntax

```
long MemCardGetDirentry (
long chan,
char *name,
struct DIRENTRY *dir,
long *files,
long ofs,
long max
)
```

Arguments

chan port number + card number
port number (port A: 0x00, port B: 0x10)
card number (normally 0)

name filename to be searched (wildcards can be used)

dir pointer to structure to hold information about matching files

files pointer to buffer to hold number of matching files

ofs offset for entry. Specifies the number of files to skip from the first file that matches before saving to the buffer (0 - 14).

max maximum number of entries to store in the buffer

Explanation

Files matching the filename pattern *name* are found. Data for these files are stored in *dir*, and the total number of matching files is stored in *files*.

The buffer must be prepared by the user application.

Return value

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Card is not connected
0x02	McErrCardInvalid	Bad card
0x04	McErrNotFormat	Not formatted
-1	None	A non-synchronous function is active.

Remarks

Synchronous function.

Blocking time: 1 - 2 VSyncs for normal exit. Otherwise, 4 - 76 VSyncs.

Wildcard characters can be used in the filename pattern: "?" for any single character; "*" for any number of characters. Characters following * are ignored.

See also: DIRENTRY.

MemCardInit

Initialize memory card system.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	5/22/97

Syntax

```
void MemCardInit (
long val
)
```

Arguments

val Use of control routine in ROM (0: do not use, 1: use)

Explanation

The memory card system is initialized.

If the system is subsequently activated with MemCardStart(), libmcrd functions (those beginning with "MemCard") are available.

MemCardInit() cannot be executed twice.

MemCardInit() requires 60 - 70 VSyncs to complete.

val should be set to 0 when using libtap or libgun.

MemCardInit() should be executed after InitPAD, InitGUN(), InitTAP().

Return value

None

Remarks

See also: MemCardEnd(), MemCardStart(), MemCardStop().

MemCardOpen

Open file.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	6/22/98

Syntax

```
long MemCardOpen (
long chan,
char *file,
long flag
)
```

Arguments

chan port number + card number
port number (port A: 0x00, port B: 0x10)
card number (normally 0)

file filename

flag specifies method with which to open
(read only: O_RDONLY, write only: O_WRONLY)

Explanation

The specified memory card file is opened with the method specified by flag.

Once the file is open, MemCardReadData() and MemCardWriteData() can be used.

Methods cannot be combined (O_RDONLY|O_WRONLY).

Multiple files cannot be opened.

Return value

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Card is not connected
0x02	McErrCardInvalid	Bad card
0x04	McErrNotFormat	Not formatted
0x05	McErrFileNotExist	File not found
-1	None	Either another file is already open or a non-synchronous function is active in the background.

Remarks

Synchronous function.

Blocking time: Exits immediately for normal completion.

Otherwise, 4 - 76 VSyncs.

See also: MemCardReadData(), MemCardWriteData(), MemCardClose().

MemCardReadData

Read data from memory card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	10/01/97

Syntax

```
long MemCardReadData (
  unsigned long *adrs,
  long offset,
  long bytes
)
```

Arguments

adrs pointer to destination buffer in main memory
offset offset in bytes from which to read, where the start of the file is defined to be 0
bytes number of bytes to read (multiple of 128)

Explanation

Data is read from the memory card file previously opened in MemCardOpen() and stored in the buffer pointed to by *adrs*.

MemCardReadData() is an asynchronous function and will exit immediately. Multiple instances cannot be registered.

Completion of the operation can be confirmed via MemCardSync() or the exit callback.

The result of the operation is returned via MemCardSync() or the exit callback.

bytes is specified in units of 128. If a number that is not a multiple of 128 is specified, the process is not registered and the operation terminates with a return value of 0.

The function result can have the values shown below.

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Communication error
0x03	McErrNewCard	New card (card swapped)

Return value

1 if operation was registered successfully. Otherwise, 0.

Remarks

Asynchronous function.

Required time: approximately 1 VSync overhead + approximately 130 VSynCs per block (8192 bytes)

See also: MemCardOpen().

MemCardReadFile

Read file from memory card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	10/01/97

Syntax

```
long MemCardReadFile (
long chan,
char *file,
unsigned long *adrs,
long offset,
long byte
)
```

Arguments

chan port number + card number
port number (port A: 0x00, port B: 0x10)
card number (normally 0)

file filename

adrs pointer to destination buffer in main memory

offset offset in bytes from which to read, where the start of the file is defined to be 0

bytes number of bytes to read (multiple of 128)

Explanation

Data is read from the specified memory card file and stored in the buffer pointed to by *adrs*.

MemCardOpen() and MemCardReadData() are executed within MemCardReadFile().

If MemCardOpen() is executed on a file which is already open, an error is generated and the value 0 is returned.

MemCardReadData() is an asynchronous function and will exit immediately. Multiple instances cannot be registered.

Completion of the operation can be confirmed via MemCardSync() or the exit callback.

The result of the operation is returned via MemCardSync() or the exit callback.

bytes is specified in units of 128. If a number that is not a multiple of 128 is specified, the process is not registered and the operation terminates with a return value of 0.

The function result can have the values shown below.

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Communication error
0x03	McErrNewCard	New card (card swapped)
0x05	McErrFileNotExist	File cannot be found

Return value

1 if operation was registered successfully.

If the file was already open, or another asynchronous function was already registered, 0 is returned.

Remarks

Asynchronous function. Required time: approximately 1 VSync overhead + approximately 130 VSynCs per block (8192 bytes)

See also: MemCardOpen().

MemCardStart

Start memory card system.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	5/22/97

Syntax

```
void MemCardStart(void)
```

Arguments

None

Explanation

The memory card system, previously initialized with MemCardInit(), is placed in an active state. Internally, eight events such as HwCARD and SwCARD are opened.

Return value

None

Remarks

Asynchronous Function. Exits immediately.

See also: MemCardInit(), MemCardStop().

MemCardStop

Stop memory card system.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	5/22/97

Syntax

```
void MemCardStop(void)
```

Arguments

None

Explanation

The memory card system activated by MemCardStart() is stopped.

Various events are closed.

Return value

None

Remarks

Asynchronous Function. Exits immediately.

See also: MemCardInit(), MemCardStart().

MemCardSync

Waits for completion of an asynchronous function or checks status.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	6/22/98

Syntax

```
long MemCardSync (
long mode,
long *cmds,
long *result
)
```

Arguments

mode 0: wait for termination of asynchronous function
 1: check current status and return immediately
cmds pointer to the terminated asynchronous function
result pointer to execution results from the asynchronous function

Explanation

Waits for termination of an asynchronous function such as MemCardAccept() and MemCardReadFile().

The operation code corresponding to the terminated asynchronous function is saved in the memory location pointed to by cmds.

The allowed values are shown below.

Value	Macro	Function
0x01	McFuncExist	MemCardExist
0x02	McFuncAccept	MemCardAccept
0x03	McFuncReadFile	MemCardReadFile
0x04	McFuncWriteFile	MemCardWriteFile
0x05	McFuncReadData	MemCardReadData
0x06	McFuncWriteData	MemCardWriteData

Return value

0: Still active
 1: Terminated
 -1: No registered process

Remarks

Synchronous function. Exits immediately if mode is 1.

If mode is 0, the execution time depends on the corresponding asynchronous function.

See also: MemCardAccept(), MemCardExit(), MemCardReadFile(), MemCardWriteFile(), MemCardReadData(), MemCardWriteData().

MemCardUnFormat

Uninitializes a memory card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.3	6/22/98

Syntax

```
long MemCardUnFormat (
long chan,
)
```

Arguments

chan port number + card number
 port number (port A: 0x00, port B: 0x10)
 card number (default 0)

Explanation

Puts memory card in uninitialized (unformatted) state.

MemCardUnFormat() is a debugging function that can be used to create an unformatted card. This function should only be used for testing memory card initialization during program debugging. It should not be used in an actual title.

Return value

The following three values are returned.

- 1: Completed successfully
- 0: Error
- 1: Could not be executed because of an asynchronous function running in the background.

Remarks

Synchronous function

See also: MemCardFormat()

MemCardWriteData

Write data to memory card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	10/01/97

Syntax

```
long MemCardWriteData (
unsigned long *adrs,
long offset,
long byte
)
```

Arguments

adrs pointer to destination buffer in main memory
offset offset in bytes from which to read, where the start of the file is defined to be 0
bytes number of bytes to read (multiple of 128)

Explanation

Data is written from the buffer pointed to by *adrs* to the memory card file previously opened with `MemCardOpen()`.

`MemCardWriteData()` is an asynchronous function and will exit immediately. Multiple instances cannot be registered.

Completion of the operation can be confirmed via `MemCardSync()` or the exit callback.

The result of the operation is returned via `MemCardSync()` or the exit callback.

bytes is specified in units of 128. If a number that is not a multiple of 128 is specified, the process is not registered and the operation terminates with a return value of 0.

The function result can have the values shown below.

Value	Macro	Status
0x00	<code>McErrNone</code>	Normal exit
0x01	<code>McErrCardNotExist</code>	Not connected
0x02	<code>McErrCardInvalid</code>	Communication error
0x03	<code>McErrNewCard</code>	New card (card swapped)

Return value

1 is returned if the operation was registered successfully.

Otherwise 0 is returned.

Remarks

Asynchronous function.

Required time: Approximately 1 VSync overhead + 130 VSynCs per block (8192 bytes)

See also: `MemCardOpen()`.

MemCardWriteFile

Write file to memory card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	10/01/97

Syntax

```
long MemCardWriteFile (
long chan,
char *file,
unsigned long *adrs,
long offset,
long bytes
)
```

Arguments

chan port number + card number
port number (port A: 0x00, port B: 0x10)
card number (normally 0)

file filename

adrs pointer to destination buffer in main memory

offset offset in bytes from which to read, where the start of the file is defined to be 0

bytes number of bytes to read (multiple of 128)

Explanation

Data is written from the buffer pointed to by *adrs* to the specified memory card. If the file is a new file, it must be created beforehand with `MemCardCreateFile()`.

`MemCardOpen()` and `MemCardWriteData()` are executed within `MemCardWriteFile()`.

If `MemCardOpen()` is executed on a file which is already open, an error is generated and the value 0 is returned.

`MemCardWriteData()` is an asynchronous function and will exit immediately. Multiple instances cannot be registered.

Completion of the operation can be confirmed via `MemCardSync()` or the exit callback.

The result of the operation is returned via `MemCardSync()` or the exit callback.

bytes is specified in units of 128. If a number that is not a multiple of 128 is specified, the process is not registered and the operation terminates with a return value of 0.

The function result can have the values shown below.

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Communication error
0x03	McErrNewCard	New card (card swapped)
0x05	McErrFileNotExist	File not found

Return value

1 if operation was registered successfully.

If the file was already open, or another asynchronous function was already registered, 0 is returned.

Remarks

Required time: Approximately 1 VSync overhead + 130 VSyns per block (8192 bytes)

See also: MemCardCreateFile().

Chapter 6: Data Compression Library

Table of Contents

Structures	
DECDCCTENV	6-3
ENCSPUENV	6-4
Functions	
DecDCTBufSize	6-5
DecDCTGetEnv	6-6
DecDCTin	6-7
DecDCTinCallback	6-8
DecDCTinSync	6-9
DecDCTout	6-10
DecDCToutCallBack	6-11
DecDCToutSync	6-12
DecDCTPutEnv	6-13
DecDCTReset	6-14
DecDCTvlc	6-15
DecDCTvlc2	6-16
DecDCTvlcBuild	6-17
DecDCTvlcSize	6-18
DecDCTvlcSize2	6-19
EncSPU	6-20

DECDCTENV

Quantization tables and environment data used during MDEC decoding process.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.5	7/31/96

Structure

```
typedef struct {
    u_char iq_y[64];
    u_char iq_c[64];
    short dct[64];
} DECDCTENV;
```

Members

<i>iq_y</i>	Brightness component quantization table
<i>iq_c</i>	Chrominance component quantization table
<i>dct</i>	System reserved

Explanation

This structure contains the tables used during the reverse-quantization step of the MDEC decoding process. The default values used by the system are:

$$\begin{bmatrix} \mathbf{iq}_y \\ 2 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\ 16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\ 19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\ 22 & 22 & 26 & 27 & 29 & 34 & 37 & 40 \\ 22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\ 26 & 27 & 29 & 32 & 35 & 40 & 48 & 58 \\ 26 & 27 & 29 & 34 & 38 & 46 & 56 & 69 \\ 27 & 29 & 35 & 38 & 46 & 56 & 69 & 83 \end{bmatrix} \times \frac{1}{16}$$

Remarks

The values in the *iq_y* and *iq_c* tables are sorted in a diagonal zig-zag scanning order.

ENCSPUENV

SPU encode environment attribute structure.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.6	6/22/98

Structure

```
typedef struct {
    short *src;
    short *dest;
    short *workt;
    long size;
    long loop_start;
    char loop;
    char byte_swap;
    char proceed;
    char pad4;
} ENCSPUENV;
```

Members

<i>src</i>	16-bit PCM data address
<i>dest</i>	PlayStation original waveform data
<i>work</i>	Work area when encode processing
<i>size</i>	16-bit PCM data size(in bytes)
<i>loop_start</i>	PCM data loop start point(in bytes)
<i>loop</i>	Loop waveform generation specification ENCSPU_ENCODE_LOOP: Generate loop waveform data ENCSPU_ENCODE_NO_LOOP: Generate non-loop waveform data
<i>byte_swap</i>	PCM data endian specification ENCSPU_ENCODE_ENDIAN_BIG: 16-bit big endian ENCSPU_ENCODE_ENDIAN_LITTLE: 16-bit little endian
<i>proceed</i>	Whole/Divided encoding specification ENCSPU_ENCODE_WHOLE Whole encoding ENCSPU_ENCODE_START Start divided encoding ENCSPU_ENCODE_CONTINUE Continue divided encoding ENCSPU_ENCODE_END End divided encoding
<i>pad4</i>	System reserved

Explanation

This structure is used to specify the SPU encode environment attributes for EncSPU() function.

Remarks

When 0 is specified for "loop", "loop_start" will be ignored.

DecDCTBufSize

Obtains the size of the run-level compressed DCT data.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	4/21/98

Syntax

```
long DecDCTBufSize(  
u_long *bs  
)
```

Arguments

bs Pointer to bitstream

Explanation

This function returns the uncompressed length of the data contained in the Huffman-encoded bitstream pointed to by the *bs* parameter. It does not perform the actual decoding.

When using DecDcTvlc()/DecDCTvlc2() to perform decoding, a 1 word header buffer to be added to the size obtained by this function must be reserved in advance.

Return value

Length of uncompressed data in long words (i.e. returns 1000 for a 4000-byte length).

Remarks

See also:

DecDCTGetEnv

Obtain the current quantization tables and environment data used during MDEC image decoding.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.5	7/31/96

Syntax

```
DECDCTENV *DecDCTGetEnv (
DECDCTENV *env
)
```

Argument

env Pointer to decoding environment

Explanation

This function returns the current decoding environment to *env*.

Return value

Top address of *env*.

Remarks

See also:

DecDCTin

Begin decoding of RLE-encoded MDEC image data.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	7/31/96

Syntax

```
void DecDCTin(*runlevel, mode)
unsigned long *runlevel;
long mode;
```

Arguments

runlevel Pointer to input runlevel
mode Decode mode

Explanation

Begins decoding the RLE-encoded MDEC image data at the address specified by *runlevel*. A maximum of 128k may be decoded at a time. The resulting image data is retrieved by the DecDCTout() function.

The image depth and transparency is controlled by the *mode* parameter:

Table 6–1

Bit 0	Output mode
0	16-bit direct color
1	24-bit direct color
Bit 1	STP
0	0
1	1

The depth of the output pixels is specified by bit 0; either 24-bit or 16-bit can be selected. If it is 16-bit mode, bit 15 of the pixel (STP bit), can be specified by *mode* bit 1.

Return value

None

Remarks

The image data produced is raw pixel data without any header information of any kind. The width and height of the image produced is not maintained. It is the responsibility of the application or a higher level structure (such as the STR format) to maintain such information.

Data decoded from a single DecDCTin() call may be read using multiple DecDCTout() calls, or the data created by multiple DecDCTin() calls may be read using a single DecDCTout() call.

The DecDCTin() function is non-blocking. To detect when execution of the primitive list is complete, use the DecDCTinSync() function or install a callback routine with the DecDCTinCallback() function. If a DecDCTin() call is executed before a previous one has finished, the transmission will be blocked until the previous operation is complete.

See also:

DecDCTinCallback

Installs a callback routine to be called at termination of MDEC transmission.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	7/31/96

Syntax

```
long DecDCTinCallback(*func)
void (*func)();
```

Arguments

func Pointer to callback function address

Explanation

This function installs the user-defined callback routine specified by *func*. This routine will be called when the data transmission initiated by a DecDCTin() call has been completed. If *func* is 0, any previous callback routine is disabled.

Return value

A pointer to a previously set callback function.

Remarks

Inside the callback, subsequent transmission termination interrupts are masked. Therefore, the callback routine should return as soon as possible. Also note that although the specified function is called during an interrupt, it is not an interrupt handler. It should be written as normal subroutine that will be called by the main interrupt handler.

See also:

DecDCTinSync

Detects DecDCTin() termination.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	6/22/98

Syntax

```
long DecDCTinSync(mode)
long mode;
```

Arguments

mode Mode

Explanation

Detects termination of DecDCTin(). *Mode* values are as follows:

Table 6–2

Value	Description
0	Blocks until termination
1	Performs only status notification

Return value

Image processing subsystem status. 1 is returned if transmission is in process and 0 if transmission is not being performed.

Remarks

Synchronization with the DecDCTinSync() function must be performed after reading the appropriate volume of decode data with the DecDCTout() function. When calling this function without using the DecDCTout() function after the DecDCTin() function, a time out will occur and MDEC will be reset.

See also:

DecDCTout

Receives decoded data from the image processing subsystem.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	7/31/96

Syntax

```
void DecDCTout(*cell, size)
unsigned long *cell;
long size;
```

Arguments

cell Pointer to decoded image data
size Received data size (long word)

Explanation

The RLE-encoded MDEC image data previously specified in a DecDCTin() call is decoded and stored in the buffer specified by the *cell* parameter. The amount of data to be transferred is specified in long words by the *size* parameter (i.e. *size*=1000 to transfer 4000 bytes of data). Multiple calls to DecDCTout() may be made to retrieve image data.

You must specify a *size* value that is the same as or smaller than the available decoded data. If there is more data available than is read by one DecDCTout() call, then additional calls must be made to avoid MDEC transmission deadlocks.

The decoded image is output one 16 x 16 macroblock at a time. The *size* specified must be a multiple of the total macroblock size for the current decoding mode. If decoding to 16-bit, a macroblock is 128 words. If decoding to 24-bit, the macroblock length is 192 words.

Return value

None

Remarks

The DecDCTout() function is non-blocking. To detect when execution of the primitive list is complete, use the DecDCToutSync() function or install a callback routine with the DecDCToutCallback() function. If a DecDCTout() call is executed before a previous one has finished, the transmission will be blocked until the previous operation is complete.

See also:

DecDCToutCallback

Installs a callback routine to be called at termination of MDEC transmission.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	7/31/96

Syntax

```
long DecDCToutCallback(*func)
long (*func)();
```

Arguments

func Pointer to callback function address

Explanation

This function installs the user-defined callback routine specified by *func*. This routine will be called when the data transmission initiated by a DecDCTout() call has been completed. If *func* is 0, and previous callback routine is disabled.

Return value

A pointer to a previously set callback function.

Remarks

Inside the callback, subsequent transmission termination interrupts are masked. Therefore, the callback routine should return as soon as possible. Also note that although the specified function is called during an interrupt, it is not an interrupt handler. It should be written as a normal subroutine that will be called by the main interrupt handler.

See also:

DecDCToutSync

Detects termination of DecDCTout().

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	7/31/96

Syntax

```
long DecDCToutSync(mode)
long mode;
```

Arguments

mode Mode

Explanation

Detects termination of DecDCTout(). Mode values are as follows:

Table 6–3

Value	Description
0	blocks until termination
1	performs only status notification

Return value

Image processing subsystem status. 1 is returned if reception is in progress and 0 if reception is not being performed.

Remarks

See also:

DecDCTPutEnv

Set image-processing-subsystem environment.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.5	7/31/96

Syntax

```
DECDCCTENV *DecDCTPutEnv (  
DECDCCTENV *env  
)
```

Argument

env Pointer to decoding environment

Explanation

This function sets the quantization tables and environment data used during the reverse-quantization step of the MDEC decoding process.

Return value

Top address of *env*.

Remarks

See also:

DecDCTReset

Initializes image processing subsystem.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	7/31/96

Syntax

```
void DecDCTReset(mode)
long mode;
```

Arguments

mode Reset mode

Explanation

This function resets the image processing subsystem. Values that can be specified for *mode* are as follows:

Table 6–4

Value	Content
0	Initializes all internal states
1	Discontinues only current decoding; does not affect internal states

Return value

None

Remarks

Processing time is longer for mode0 than for mode1 because internal tables are initialized.

See also:

DecDCTvlc

Decodes Huffman-compressed MDEC image data.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	4/21/98

Syntax

```
int DecDCTvlc
u_long *bs,
u_long *buf
)
```

Arguments

bs Input bitstream
buf Output runlevel

Explanation

This function builds the run-level intermediate format under *buf* by decoding the bit stream *bs* using the *table*. When runlevel exceeds the value specified in DecDCTvlcSize(), DecDCTTvlc() interrupts processing once and returns control to the application.

The interrupted VLC decode can be restarted by executing DecDCTvlc (0,0).

With *buf*, the 1 word area added to the header buffer in DecDCTBufSize() must be reserved in advance.

Return value

- 0 Decoding for all bit stream is successfully completed.
- 1 Returned with some bit stream left non-decoded.
- 1 Decode failed.

Remarks

This is a blocking function.

This function is only the first stage of decoding an MDEC image. The Huffman-encoded bitstream must always be decoded using DecDCTvlc() before DecDCTIn() is executed.

A partial result run level cannot be provided as DecDCTIn() input.

In the future it will be possible to replace this function with DecDCTvlc2.

See also:

DecDCTvlc2

Decodes VLC.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.7	4/21/98

Syntax

```
int DecDCTvlc2 (
u_long *bs,
u_long *buf,
DECDCTTAB table
)
```

Arguments

bs Input bit stream
buf Output run level
table VLC table

Explanation

This function builds the run-level intermediate format under *buf* by decoding the bit stream *bs* using the *table*. When the run level exceeds the value specified in DecDCTvlcSize2(), DecDCTvlc2() will be suspended and control will be returned to the application. The suspended VLC decoding process can be restarted by executing DecDCTvlc2(0, 0, table). With *buf*, the 1 word area added to the header buffer in DecDCTBufSize() must be reserved in advance.

Return value

0 Decoding for all bit stream is successfully completed.
1 Returned with some bit stream left non-decoded.
-1 Decode failed.

Remarks

This is a blocking function.

This function is only the first stage of decoding an MDEC image. The Huffman-encoded bitstream must always be decoded using DecDCTvlc() before DecDCTIn() is executed.

A partial result run level cannot be provided as DecDCTIn() input.

The VLC table should be decoded in advance using DecDCTBuild.

See also:

DecDCTvlcBuild

Builds the VLC table.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.7	4/21/98

Syntax

```
void DecDCTvlcBuild (
  u_short *table
)
```

Arguments

table VLC Buffer

Explanation

This function builds the VLC table that will be used for DecDCTvlc2(). The size of the VLC table to be built can be obtained beforehand using sizeof (DECDCTTAB). Refer to libpress.h for the DECDCTTAB structure.

Return value

None

Remarks

The VLC table is held in a compressed (4KB) format and only when a movie is playing is it released to the work area and used in its decompressed form (64 KB).

Although this does not result in a reduction in the memory when a movie is playing, when a movie is not in use, the unnecessary table area can be reduced.

See also: DecDCTvlc2()

DecDCTvlcSize

Sets the maximum amount of data returned by a single call to the DecDCTvlc() function.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.2	7/31/96

Syntax

```
int DecDCTvlcSize (
int size
)
```

Arguments

size Maximum value of a decoded runlevel (long word)

Explanation

This *size* parameter for the DecDCTvlcSize() function specifies the maximum number of long words that may be returned at once by the DecDCTvlc() function. Subsequent calls to the DecDCTvlc() function will halt after decoding the specified number of long words. If *size* is set to zero, the DecDCTvlc() will decode the entire bitstream regardless of length.

This allows your program to make multiple calls to DecDCTvlc() to decode a bitstream in chunks using a smaller buffer size.

Return value

Previously set buffer *size*.

Example:

```
/* Decoding the first VLC_SIZE word in VLC */
DecDCTvlcSize (VLC_SIZE);
isvlcLeft = DecDCTvlc (next, dec.vlcbuf[dec.vlcid]);
/* Waiting for data to be completed */
do {
    /* Decoding the remaining VLC_SIZE words in VLC */
    if (isvlcLeft) {
        isvlcLeft = DecDCTvlc (0, 0);
        FntPrint ("%d, ", VSync (1));
    }
    /* Application code is here */
} while (isvlcLeft || isEndOfFrame == 0);
isEndOfFrame = 0;
```

Remarks

This is a block function. A bitstream must be converted to run- levels by DecDCTvlc() before executing DecDCTin().

See also:

DecDCTvlcSize2

Sets maximum size of single VLC decoding process.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.7	2/13/97

Syntax

```
int DecDCTvlcSize2 (
  int size
)
```

Arguments

size Maximum value of a decoded runlevel (long word)

Explanation

This function gives the value representing a maximum size of bit stream that can be decoded per decoding process. DecDCTvlc2() will suspend decoding process when decoding the first block after number of words specified by *size*. In case of *size* 0, decoding process will not be suspended. Default value is 0.

Return value

Maximum run level set immediate before.

Remarks

Since this is a blocking function, the bit stream must be converted to a run level by DecDCTvlc2() before executing DecDCTin().

See also:

EncSPU

Encodes 16-bit PCM data into PlayStation original waveform format.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.6	6/22/98

Syntax

```
long EncSPU (
ENCSPUENV *es_env
)
```

Arguments

es_env SPU encode environment attribute structure

Explanation

This function encodes the PCM data specified in a member "src" of the SPU encode environment attribute structure, "es_env" into the PlayStation original waveform data (VAG, without header information) and returns the encoded data in a member "dest".

If *es_env.loop_start* is not a multiple of 56 (28 samples), the loop start point is set to a lower value so it is a multiple of 56.

Specify the user area address for both members "src" and "dest" of the SPU encode environment attribute structure, "es_env".

Divided encoding can be done by specifying an attribute to a member "proceed" of the SPU encode environment structure, "es_env".

If *es_env.size* is not a multiple of 56 (28 samples), the data is padded with zeroes until it is. This causes the generated waveform to be discontinuous, and to maintain continuity, perform a divided encode on the data with *en_env.size* equal to a multiple of 56. When *es_env.proceed* is set to ENCSPU_ENCODE_WHOLE, the waveform is padded with zeroes to make *es_env.size* a multiple of 56 and waveform encoding is performed all at once.

When using scratchpad as the workspace, specify *es_env.work* as the scratchpad address. 168 bytes will be used from the specified address. If *es_env.work* is set to NULL, the automatic variables will be used internally.

Return value

The data size of the encoded waveform (VAG) is returned.

ENC_ENCODE_ERROR is returned when an encoding error occurs.

Remarks

See also:

Chapter 7: Basic Graphics Library

Table of Contents

Structures

DISPENV	7-5
DRAWENV	7-6
DR_AREA	7-9
DR_ENV	7-10
DR_LOAD	7-11
DR_MODE	7-12
DR_MOVE	7-13
DR_OFFSET	7-14
DR_STP	7-15
DR_TPAGE	7-16
DR_TWIN	7-17
LINE_F2, LINE_F3, LINE_F4	7-18
LINE_G2, LINE_G3, LINE_G4	7-19
POLY_F3, POLY_F4	7-21
POLY_FT3, POLY_FT4	7-23
POLY_G3, POLY_G4	7-25
POLY_GT3, POLY_GT4	7-27
RECT	7-29
RECT32	7-30
SPRT	7-31
SPRT_8, SPRT_16	7-32
TILE	7-33
TILE_1, TILE_8, TILE_16	7-34
TIM_IMAGE	7-35
TMD_PRIM	7-36

Functions

AddPrim, addPrim	7-37
AddPrims, addPrims	7-38
BreakDraw	7-39
CatPrim, catPrim	7-40
CheckPrim	7-41
ClearImage	7-42
ClearImage2	7-43
ClearOTag	7-44
ClearOTagR	7-45
ContinueDraw	7-46
DrawPrim	7-47
DrawOTag	7-48
DrawOTag2	7-49
DrawOTagEnv	7-50
DrawOTagIO	7-51
DrawSync	7-52
DrawSyncCallback	7-53
DumpClut, dumpClut	7-54
DumpDispEnv	7-55
DumpDrawEnv	7-56
DumpOTag	7-57
DumpTPage, dumpTPage	7-58
FntFlush	7-59
FntLoad	7-60
FntOpen	7-61
FntPrint	7-62
GetClut, getClut	7-63

GetDispEnv	7-64
GetDrawArea	7-65
GetDrawEnv	7-66
GetDrawMode	7-67
GetDrawOffset	7-68
GetGraphDebug	7-69
GetODE	7-70
GetTexWindow	7-71
GetTimSize	7-72
GetTPage, getTPage	7-73
IsEndPrim, isendprim	7-74
IsIdleGPU	7-75
KanjiFntClose	7-76
KanjiFntFlush	7-77
KanjiFntOpen	7-78
KanjiFntPrint	7-79
Krom2Tim	7-80
LoadClut	7-81
LoadClut2	7-82
LoadImage	7-83
LoadImage2	7-84
LoadTPage	7-85
MargePrim	7-86
MoveImage	7-87
MoveImage2	7-88
NextPrim, nextPrim	7-89
OpenTIM	7-90
OpenTMD	7-91
PutDispEnv	7-92
PutDrawEnv	7-93
ReadTIM	7-94
ReadTMD	7-95
ResetGraph	7-96
SetDefDispEnv	7-97
SetDefDrawEnv	7-98
SetDispMask	7-99
SetDrawArea	7-100
SetDrawEnv	7-101
SetDrawLoad	7-102
SetDrawMode	7-103
SetDrawMove	7-104
SetDrawOffset	7-105
SetDrawStp	7-106
SetDrawTPage, setDrawTPage	7-107
SetDumpFnt	7-108
SetGraphDebug	7-109
SetLineF2, SetLineF3, SetLineF4;	7-109
setLineF2, setLineF3, setLineF4	7-110
SetLineG2, SetLineG3, SetLineG4;	7-111
setLineG2, setLineG3, setLineG4	7-111
SetPolyF3, SetPolyF4;	7-112
setPolyF3, setPolyF4	7-112
SetPolyFT3, SetPolyFT4;	7-113
setPolyFT3, setPolyFT4	7-113
SetPolyG3, SetPolyG4;	7-114
setPolyG3, setPolyG4	7-114

SetPolyGT3, SetPolyGT4;	7-115
setPolyGT3, setPolyGT4	7-115
SetSemiTrans, setSemiTrans	7-116
SetShadeTex, setShadeTex	7-117
SetSprt, SetSprt8, SetSprt16;	7-118
setSprt, setSprt8, setSprt16	7-118
SetTexWindow	7-120
SetTile, SetTile1, SetTile8, SetTile16;	7-121
setTile, setTile1, setTile8, setTile16	7-121
setVWH	7-123
setXYWH	7-124
StoreImage	7-125
StoreImage2	7-126
TermPrim, termPrim	7-127
VSync	7-128
VSyncCallback	7-129
Macros	
addVector	7-130
applyVector	7-131
copyVector	7-132
dumpMatrix	7-133
dumpRECT	7-134
dumpVector	7-135
dumpWH...	7-136
SetClut	7-137
setRECT	7-138
setRGB0, setRGB1, setRGB2, setRGB3	7-139
SetTPage	7-140
setUV0, setUV3, setUV4	7-141
setUVWH	7-142
setVector	7-143
setWH	7-144
setXY0, setXY2, setXY3, setXY4	7-145
setXYWH	7-146

DISPENV

Display environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Structure

```
struct DISPENV {
    RECT disp;
    RECT screen;
    unsigned char isinter;
    unsigned char isrgb24;
    unsigned char pad0, pad1;
};
```

Members

disp This is the display area within the frame buffer. Specify the width of the area as one of the following: 256, 320, 384, 512, 640. Specify the area height as 240 or 480.

screen Output screen display area. The screen area is calculated without regard to the value of *disp*, using the standard monitor screen upper left-hand point y (0, 0) and lower right-hand point y (256, 240).

isinter This is the interlace mode flag
 0: non-interlace
 1: interlace

isrgb24 This is the 24-bit mode flag
 0: 16-bit mode
 1: 24-bit mode

pad Reserved by system

Explanation

Specifies display parameters for screen display mode, frame buffer display value, and so on.

Remarks

See also:

DRAWENV

Drawing environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	06/22/98

Structure

```
struct DRAWENV {
    RECT clip;
    short ofs[2];
    RECT tw;
    unsigned short tpage;
    unsigned char dtd;
    unsigned char dfe;
    unsigned char isbg;
    unsigned char r0, g0, b0;
    DR_ENV dr_env;
};
```

Members

<i>clip</i>	Drawing area. Drawing is restricted to a short area specified by clip. Drawing is not performed outside the clipping area. (See Remarks 1, below.)
<i>ofs</i>	Offset. Drawing commands use the added values of (ofs[0], ofs[1]) as an address and draw in the frame buffer. (See Remarks 2.)
<i>tw</i>	Texture window. The short area texture pattern restricted by the texture page <i>tw</i> is used repeatedly.
<i>tpage</i>	Initial value of texture page
<i>dtd</i>	Dithering processing flag 0: off 1: on
<i>dfe</i>	Drawing to display area flag 0: drawing to display area is blocked 1: drawing to display area is permitted
<i>isbg</i>	Drawing area clear flag 0: off 1: on Does not clear drawing area when drawing environment is set. Paints entire clip area with brightness values (<i>r0</i> , <i>g0</i> , <i>b0</i>) when drawing environment is set.
<i>r0, g0, b0</i>	Background color. Valid only when <i>isbg</i> is 1.
<i>dr env</i>	System reserved

Explanation

This sets basic drawing parameters, such as drawing offset and drawing clip area.

GPU carries out the shading internal processes with R, G, B at 8 bits each and when writing to the frame buffer, each is reduced to 5 bits. When *dtd* is ON, a 4x4 dither matrix is used and is processed as follows:

$$i = 8 \text{ bit brightness value} + 1/2 * D - 4$$

D = Dither matrix [X%4][Y%4]

4x4 Dither Matrix

0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

5 bit brightness value = 1 >> 3

Remarks

- *1 Graphics can be actually drawn in an area (0, 0) - (1023, 511) in the graphic space.
- *2 The offset value and the address after the addition of the offset are wrapped around at (-1024, -1024) - (1023, 1023).
- *3 The values which may be specified for the texture window are restricted to the following combinations:

Table 7-1

<i>tw.w, tw.x</i>						
<i>tw.w</i>	0 (=256) 128	8	16	32	64	
<i>tw.x</i>	0 of 128	Multiple of 8	Multiple of 16	Multiple of 32	Multiple of 64	Multiple
<i>tw.h, tw.y</i>						
<i>tw.h</i>	0 (=256) 128	8	16	32	64	
<i>tw.y</i>	0 of 128	Multiple of 8	Multiple of 16	Multiple of 32	Multiple of 64	Multiple

See also:

DR_AREA

Drawing area change primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Structure

```
struct DR_AREA {
    unsigned long *tag;
    unsigned long code[2];
};
```

Members

tag Pointer to the next primitive in primitive list
code New drawing area information specified by SetDrawArea() function

Explanation

The DR_AREA primitive modifies the drawing area of the current drawing environment while a primitive list is being drawn. Use the SetDrawArea() function to set the contents of this primitive.

Remarks

See also:

DR_ENV

Drawing environment modification primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Structure

```
struct DR_ENV {
    unsigned long *tag;
    unsigned long code[15];
};
```

Members

tag Pointer to the next primitive in primitive list
code New drawing environment information specified by SetDrawEnv() function

Explanation

The DR_ENV primitive changes the drawing environment while a primitive list is being drawn. Use the SetDrawEnv() function to specify the new DRAWENV parameters to be used.

Remarks

This function affects only the drawing environment, not the display environment (see DISPENV for that). The entire drawing environment may be changed using this primitive. See the DRAWENV structure definition for more details. See also the DR_MODE primitive, which sets a subset of the drawing environment.

See also:

DR_LOAD

Load Image primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.4	10/01/97

Structure

```
typedef struct {
    unsigned long *tag;
    unsigned long code[3];
    unsigned long p[13];
} DR_LOAD;
```

Members

tag Pointer to next primitive (reserved)
code Primitive ID
p Transfer data

Explanation

DR_LOAD transfers data below array *p* to the frame buffer. As with LoadImage() semitransparent /transparent color control is not performed. Also, there is no dependence on the drawing environment.

Maximum data transfer amount is 12 words (24 pixels).

Remarks

See also:

DR_MODE

Drawing mode modification primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    unsigned long *tag;
    unsigned long code[2];
} DR_MODE;
```

Members

tag Pointer to the next primitive in primitive list
code New drawing environment information as specified by SetDrawMode() function

Explanation

The DR_MODE primitive changes the texture page, texture window, dithering flag, and drawing flag parameters of the current drawing environment while a primitive list is being drawn. See the *tpage*, *tw*, *dtd*, and *dfe* fields of the DRAWENV structure for more information. Use the SetDrawMode() function to specify the parameters to be used.

Remarks

See also:

DR_MOVE

Rectangle domain copy primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.2	2/13/97

Structure

```
typedef struct {
    u_long tag;
    u_long code[5];
} DR_MOVE;
```

Members

tag Hook to the next primitive (reserved)
code Primitive ID

Explanation

DR_MOVE performs rectangle domain transference. High speed is the same as MoveImage().

Unlike the 16-bit SPRT primitive, semitransparent/transparent color control is not carried out. Also, it is not dependent on the drawing environment.

Remarks

See also:

DR_OFFSET

Drawing offset modification primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    unsigned long *tag;
    unsigned long code[2];
} DR_OFFSET;
```

Members

tag Pointer to the next primitive in primitive list
code New drawing offset information specified by SetDrawOffset() function

Explanation

The DR_OFFSET primitive changes the drawing offset parameters of the current drawing environment while a primitive list is being drawn. See the *ofs* field of the DRAWENV structure for more information. Use the SetDrawOffset() function to specify the parameters to be used.

Remarks

See also:

DR_STP

STP bit updated primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	10/01/97

Structure

```
typedef struct DR_STP {
    u_long tag;
    u_long code[2];
} DR_STP;
```

Members

tag Pointer to the next primitive in primitive list (reserved)
code Primitive ID

Explanation

The DR_AREA primitive updates the STP bit during drawing. Use the SetDrawStp() function to set the contents of this primitive.

Remarks

See also:

DR_TPAGE

Texture page change primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.5	7/31/96

Structure

```
typedef struct {
    u_long *tag;
    u_long code[2];
} DR_TPAGE;
```

Members

tag Pointer to the next primitive in primitive list
code New texture page information specified by SetDrawTPage() function

Explanation

The DR_TPAGE primitive changes the texture page parameter of the current drawing environment while a primitive list is being drawn. See the *tpage* field of the DRAWENV structure for more information. Use the SetDrawTPage() function to specify the parameters to be used.

Remarks

See also:

DR_TWIN

Texture window change primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    unsigned long *tag;
    unsigned long code[2];
} DR_TWIN;
```

Members

tag Pointer to the next primitive in primitive list
code New texture window information specified by SetDrawTexWindow() function

Explanation

The DR_TWIN primitive changes the texture window of the current drawing environment while a primitive list is being drawn. See the *tw* field of the DRAWENV structure for more information. Use the SetDrawTexWindow() function to specify the parameters to be used.

Remarks

See also:

LINE_F2, LINE_F3, LINE_F4

One flat-shaded non-connecting line/ Two flat-shaded connected lines/ Three flat-shaded connected lines.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Structure

```

struct LINE_F2 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    short x1, y1;
};

struct LINE_F3 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    short x1, y1;
    short x2, y2;
    unsigned long pad;
};

struct LINE_F4 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    short x1, y1;
    short x2, y2;
    short x3, y3;
    unsigned long pad;
};

```

Member

<i>tag</i>	Pointer to the next primitive (reserved)
<i>code</i>	Primitive ID
<i>r0, g0, b0</i>	RGB color specified by straight line
<i>x*, y*</i>	Coordinate of vertices forming straight line
<i>pad</i>	Reserved

Explanation

LINE_F2 draws a non-connecting line linking $(x0, y0)$ - $(x1, y1)$ with the RGB color specified by $(r0, g0, b0)$.

LINE_F3 draws 2 connecting lines linking $(x0, y0)$ - $(x1, y1)$ - $(x2, y2)$ with the RGB color specified by $(r0, g0, b0)$.

LINE_F4 draws 3 connecting lines linking $(x0, y0)$ - $(x1, y1)$ - $(x2, y2)$ - $(x3, y3)$, with the RGB color specified by $(r0, g0, b0)$.

Remarks

See also:

LINE_G2, LINE_G3, LINE_G4

One Gouraud-shaded non-connecting line/ Two Gouraud-shaded connected lines/ Three Gouraud-shaded connected lines

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Structure

```
struct LINE_G2 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char r1, g1, b1, p1;
    short x1, y1;
};
```

```
struct LINE_G3 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char r1, g1, b1, p1;
    short x1, y1;
    unsigned char r2, g2, b2, p2;
    short x2, y2;
    unsigned long pad;
};
```

```
struct LINE_G4 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char r1, g1, b1, p1;
    short x1, y1;
    unsigned char r2, g2, b2, p2;
    short x2, y2;
    unsigned char r3, g3, b3, p3;
    short x3, y3;
    unsigned long pad;
};
```

Members

<i>tag</i>	Pointer to the next primitive
<i>r0, g0, b0</i>	RGB color values
<i>code</i>	Primitive ID (reserved)
<i>x0, y0</i>	Vertex coordinates
<i>r1, g1, b1</i>	RGB color values
<i>p1</i>	Primitive ID (reserved)
<i>x1, y1</i>	Vertex coordinates
<i>r2, g2, b2</i>	RGB color values
<i>p2</i>	Primitive ID (reserved)
<i>x2, y2</i>	Vertex coordinates
<i>r3, g3, b3</i>	RGB color values
<i>p3</i>	Primitive ID (reserved)
<i>x3, y3</i>	Vertex coordinates
<i>pad</i>	Reserved

Explanation

LINE_G2 draws non-connecting lines linking $(x0, y0) - (x1, y1)$ in such a way that their vertices have the RGB color specified by $(r0, g0, b0) - (r1, g1, b1)$, and perform Gouraud shading at the same time.

LINE_G3 draws the connecting lines linking $(x0, y0) - (x1, y1) - (x2, y2)$ in such a way that their vertices have the RGB color specified by $(r0, g0, b0) - (r1, g1, b1) - (r2, g2, b2)$, and perform Gouraud shading at the same time.

LINE_G4 draws connecting lines linking $(x0, y0) - (x1, y1) - (x2, y2) - (x3, y3)$ in such a way that their vertices have the RGB color specified by $(r0, g0, b0) - (r1, g1, b1) - (r2, g2, b2) - (r3, g3, b3)$ and perform Gouraud shading at the same time.

Remarks

See also:

POLY_F3, POLY_F4

Flat-shaded, non-textured mapped triangle/ Flat-shaded, not-textured mapped quad.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Structure

```

struct POLY_F3 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    short x1, y1;
    short x2, y2;
};

struct POLY_F4 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    short x1, y1;
    short x2, y2;
    short x3, y3;
};

```

Members

<i>tag</i>	Pointer to the next primitive
<i>r0, g0, b0</i>	RGB color values
<i>code</i>	Primitive ID (reserved)
<i>x0, y0</i>	Vertex coordinates
<i>x1, y1</i>	Vertex coordinates
<i>x2, y2</i>	Vertex coordinates
<i>x3, y3</i>	Vertex coordinates

Explanation

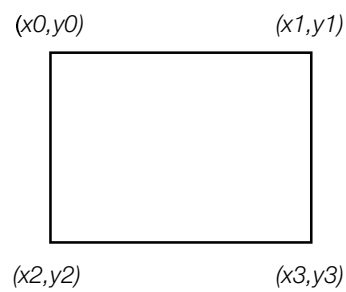
POLY_F3 paints the area demarcated by $(x0, y0) - (x1, y1) - (x2, y2)$ using RGB color specified by $(r0, g0, b0)$.

POLY_F4 paints the area demarcated by $(x0, y0) - (x1, y1) - (x3, y3) - (x2, y2)$ using RGB color specified by $(r0, g0, b0)$.

The address where a picture is actually drawn is equivalent to the value of $x0$ - $x3$ to which the offset value specified by the drawing environment is added. What is drawn is clipped according to the clip area (quadrilateral area) specified by the drawing environment.

Again, if the polygon has a width greater than 1024 and a height greater than 512, all of it will be clipped. In the case of a quadrilateral primitive, the corners are specified in the order shown below.

Figure 7-1



Remarks

See also:

POLY_FT3, POLY_FT4

Flat-shaded, texture-mapped triangle/ Flat-shaded, texture-mapped quad.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Structure

```

struct POLY_FT3 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char u0, v0;
    unsigned short clut;
    short x1, y1;
    unsigned char u1, v1;
    unsigned short tpage;
    short x2, y2;
    unsigned char u2, v2;
    unsigned short pad1;
};

struct POLY_FT4 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char u0, v0;
    unsigned short clut;
    short x1, y1;
    unsigned char u1, v1;
    unsigned short tpage;
    short x2, y2;
    unsigned char u2, v2;
    unsigned short pad1;
    short x3, y3;
    unsigned char u3, v3;
    unsigned short pad2;
};

```

Members

<i>tag</i>	Pointer to the next primitive
<i>r0, g0, b0</i>	RGB color values
<i>code</i>	Primitive ID (reserved)
<i>x0, y0</i>	Vertex coordinates
<i>u0, v0</i>	Texture coordinates
<i>clut</i>	CLUT ID (color-look-up table for 4-bit/8-bit mode only)
<i>x1, y1</i>	Vertex coordinates
<i>u1, v1</i>	Texture coordinates
<i>tpage</i>	Texture page ID
<i>x2, y2</i>	Vertex coordinates
<i>u2, v2</i>	Texture coordinates
<i>pad1</i>	Reserved by the system
<i>x3, y3</i>	Vertex coordinates
<i>u3, v3</i>	Texture coordinates
<i>pad2</i>	Reserved by the system

Explanation

POLY_FT3 draws an area demarcated by $(x0, y0) - (x1, y1) - (x2, y2)$ while mapping the area demarcated by $(u0, v0) - (u1, v1) - (u2, v2)$ in the texture pattern on the texture page *tpage*.

POLY_FT4 draws an area demarcated by $(x0, y0) - (x1, y1) - (x3, y3) - (x2, y2)$ while mapping the area demarcated by $(u0, v0) - (u1, v1) - (u3, v3) - (u2, v2)$ in the texture pattern on the texture page *tpage*.

The actual brightness value for drawn graphics are obtained by multiplying the RGB color values from the texture pattern by the RGB color values given by *r0*, *g0*, *b0*.

The texture coordinates are the coordinates (0 to 255) inside the texture page which correspond to the vertices of the triangle to be drawn. if the texture mode is 4-bit or 8-bit, the texture coordinates and the actual frame buffer address will not be 1-to-1.

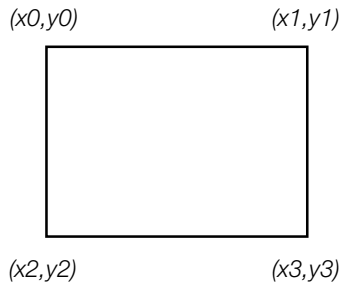
Texture page ID is given to *tpage*. Using the GetTPage() function, the texture page ID is obtained from the address (x, y) of the buffer frame where the texture page is located.

A texture using CLUT gives CLUT ID to be set in *clut*. Using the GetClut() function, CLUT ID is obtained from the address (x, y) of the frame buffer where CLUT is located.

The size of the texture page which can be used by one drawing command is 256 x 256. One primitive can only use one texture page.

In the case of a quadrilateral primitive, the corners are specified in the order shown below. The same applies to designation of (u, v) for a texture map rectangle, and (r, g, b) for a Gouraud shaded rectangle.

Figure 7-2

**Remarks**

See also:

POLY_G3, POLY_G4

Gouraud-shaded, non-textured mapped triangle/ Gourard-shaded, non-textured mapped quad.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Structure

```
struct POLY_G3 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char r1, g1, b1, pad1;
    short x1, y1;
    unsigned char r2, g2, b2, pad2;
    short x2, y2;
};
```

```
struct POLY_G4 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char r1, g1, b1, pad1;
    short x1, y1;
    unsigned char r2, g2, b2, pad2;
    short x2, y2;
    unsigned char r3, g3, b3, pad3;
    short x3, y3;
};
```

Members

<i>tag</i>	Pointer to the next primitive
<i>r0, g0, b0</i>	RGB color values
<i>code</i>	Primitive ID (reserved)
<i>x0, y0</i>	Vertex coordinates
<i>r1, g1, b1</i>	RGB color values
<i>pad1</i>	Reserved by the system
<i>x1, y1</i>	Vertex coordinates
<i>r2, g2, b2</i>	RGB color values
<i>pad2</i>	Reserved by the system
<i>x2, y2</i>	Vertex coordinates
<i>r3, g3, b3</i>	RGB color values
<i>pad3</i>	Reserved by the system
<i>x3, y3</i>	Vertex coordinates

Explanation

When drawing while performing Gouraud shading, POLY_G3 paints the area demarcated by $(x0, y0) - (x1, y1) - (x2, y2)$ so that vertex RGB color value may be set to $(r0, g0, b0) - (r1, g1, b1) - (r2, g2, b2)$.

When drawing while performing Gouraud shading, POLY_G4 paints the area demarcated by $(x0, y0) - (x1, y1) - (x3, y3) - (x2, y2)$ so that vertex RGB color value may be set to $(r0, g0, b0) - (r1, g1, b1) - (r3, g3, b3) - (r2, g2, b2)$.

The brightness of triangle-internal pixels is calculated by performing linear interpolation of the RGB color values of the three vertices. (Gouraud shading).

Remarks

See also:

POLY_GT3, POLY_GT4

Gouraud-shaded, texture-mapped triangle/ Gouraud-shaded, texture-mapped quad.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Structure

```
struct POLY_GT3 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char u0, v0;
    unsigned short clut;
    unsigned char r1, g1, b1, p1;
    short x1, y1;
    unsigned char u1, v1;
    unsigned short tpage;
    unsigned char r2, g2, b2, p2;
    short x2, y2;
    unsigned char u2, v2;
    unsigned short pad2;
};
```

```
struct POLY_GT4 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char u0, v0;
    unsigned short clut;
    unsigned char r1, g1, b1, p1;
    short x1, y1;
    unsigned char u1, v1;
    unsigned short tpage;
    unsigned char r2, g2, b2, p2;
    short x2, y2;
    unsigned char u2, v2;
    unsigned short pad2;
    unsigned char r3, g3, b3, p3;
    short x3, y3;
    unsigned char u3, v3;
    unsigned short pad3;
};
```

Members

<i>tag</i>	Pointer to the next primitive
<i>r0, g0, b0</i>	RGB color values
<i>code</i>	Primitive ID (reserved)
<i>x0, y0</i>	Vertex coordinates
<i>u0, v0</i>	Texture coordinates
<i>clut</i>	CLUT ID (color-look-up table for 4-bit/8-bit mode only)
<i>r1, g1, b1</i>	RGB color values
<i>x1, y1</i>	Vertex coordinates
<i>u1, v1</i>	Texture coordinates
<i>tpage</i>	Texture page ID

<i>r2, g2, b2</i>	RGB color values
<i>pad2</i>	Reserved by the system
<i>x2, y2</i>	Vertex coordinates
<i>u2, v2</i>	Texture coordinates
<i>pad3</i>	Reserved by the system
<i>r3, g3, b3</i>	RGB color values
<i>x3, y3</i>	Vertex coordinates
<i>u3, v3</i>	Texture coordinates
<i>p1</i>	Primitive ID (reserved)
<i>p2</i>	Primitive ID (reserved)
<i>p3</i>	Primitive ID (reserved)

Explanation

POLY_GT3 draws a triangle performing texture mapping and Gouraud shading simultaneously.

POLY_GT4 draws a quadrilateral performing texture mapping and Gouraud shading simultaneously.

The actual RGB color values for the picture are equal to the RGB color values obtained from the texture pattern multiplied by the RGB color values calculated by Gouraud shading.

Remarks

See also:

RECT

Frame buffer rectangular area.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Structure

```
struct RECT {
    short x, y;
    short w, h;
};
```

Members

x, y Top left coordinates of the rectangular area
w, h Width and height of the rectangular area

Explanation

This structure is used by several library functions to specify a rectangular area of the frame buffer. For these functions, neither negative values, nor values exceeding the size of the frame buffer (1024x512) may be specified.

Remarks

See also:

RECT32

Rectangular area (32 bit)

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Structure

```
typedef struct {
    int x, y;
    int w, h;
} RECT32;
```

Members

x, y Top left coordinates of the rectangular area
w, h Width and height of the rectangular area

Explanation

This structure is used by several library functions to specify a rectangular area of the frame buffer. For these functions, neither negative values, nor values exceeding the size of the frame buffer (1024x512) may be specified.

Remarks

See also:

SPRT

Sprite of any desired size.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	2/13/97

Structure

```
struct SPRT {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char u0, v0;
    unsigned short clut;
    short w, h;
};
```

Members

<i>tag</i>	Pointer to next primitive (reserved)
<i>r0, g0, b0</i>	RGB color values for sprite
<i>code</i>	Primitive code (reserved)
<i>x0, y0</i>	Position of sprite (top left coordinate)
<i>u0, v0</i>	Position of sprite texture within the texture page (top left coordinate). <i>u0</i> should be an even number.
<i>clut</i>	CLUT ID used (for 4-bit/8-bit mode only)
<i>w, h</i>	Width and height of sprite. <i>w</i> is an even number

Explanation

This draws a texture-mapped rectangular area. Drawing speed for a SPRT primitive is faster than for a POLY_FT4.

Remarks

Only even numbers can be specified for *u0* and *w*.

Because the SPRT primitive has no *tpage* parameter, the texture page of the current drawing environment is used. Note that you can change the texture page by inserting a DR_TPAGE or DR_MODE primitive into the primitive list to be executed before your SPRT primitive.

See also:

SPRT_8, SPRT_16

8 x 8 fixed size, texture-mapped sprite / 16 x 16 fixed size, texture-mapped sprite.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	2/13/97

Structure

```

struct SPRT_16 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char u0, v0;
    unsigned short clut;
};

struct SPRT_8 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    unsigned char u0, v0;
    unsigned short clut;
};

```

Members

<i>tag</i>	Pointer to next primitive (reserved)
<i>r0, g0, b0</i>	RGB color values for sprite
<i>code</i>	Primitive code (reserved)
<i>x0, y0</i>	Position of sprite (top left coordinate)
<i>u0, v0</i>	Position of sprite texture within the texture page (top left coordinate). <i>u0</i> should be an even number.
<i>clut</i>	CLUT ID used (for 4-bit/8-bit mode only)

Explanation

This primitive draws a sprite with a fixed size of 8 x 8 or 16 x 16. The same result can be obtained if 8 and 16 are designated as the w and h members for the SPRT structure.

Remarks

See also:

TILE

Tile Sprite of any desired size.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	2/13/97

Structure

```
struct TILE {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
    short w, h;
};
```

Members

<i>tag</i>	Pointer to next primitive (reserved)
<i>r0, g0, b0</i>	RGB color values for sprite
<i>code</i>	Primitive code (reserved)
<i>x0, y0</i>	Position of sprite (top left coordinate)
<i>w, h</i>	Width and height of sprite. <i>w</i> is an even number

Explanation

The rectangular area is drawn with the specified RGB color value (r0, g0, b0). No texture mapping or shading is done. This is faster than the POLY_F4 primitive.

Remarks

See also:

TILE_1, TILE_8, TILE_16

1 x 1 fixed-size tile sprite / 8 x 8 fixed-size tile sprite / 16 x 16 fixed-size tile sprite.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	2/13/97

Structure

```

struct TILE_16 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
};

struct TILE_8 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
};

struct TILE_1 {
    unsigned long *tag;
    unsigned char r0, g0, b0, code;
    short x0, y0;
};

```

Members

<i>tag</i>	Pointer to next primitive (reserved)
<i>r0, g0, b0</i>	RGB color values for sprite
<i>code</i>	Primitive code (reserved)
<i>x0, y0</i>	Position of sprite (top left coordinate)

Explanation

These primitives are fixed-size versions of the TILE primitive. The rectangular area is drawn with the specified RGB color value (r0, g0, b0). No texture mapping or shading is done. These are faster than the POLY_F4 primitive.

Remarks

See also:

TIM_IMAGE

TIM format image data header.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Structure

```
typedef struct {
    unsigned long mode;
    RECT *crect;
    unsigned long *caddr;
    RECT *prect;
    unsigned long *paddr;
} TIM_IMAGE;
```

Members

mode Pixel mode
crect Pointer to destination rectangle in VRAM for CLUT data
caddr Pointer to address of CLUT data in main memory
prect Pointer to destination rectangle in VRAM for texture image data
paddr Pointer to address of texture image data in main memory

Explanation

TIM data header information acquired by the ReadTIM() function.

Remarks

crect and *caddr* are assigned a value of zero for TIM having no CLUT.

See also:

TMD_PRIM

TMD format model data header.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Structure

```
typedef struct {
    unsigned long id;
    unsigned char r0, g0, b0, p0;
    unsigned char r1, g1, b1, p1;
    unsigned char r2, g2, b2, p2;
    unsigned char r3, g3, b3, p3;
    unsigned short tpage, clut;
    unsigned char u0, v0, u1, v1;
    unsigned char u2, v2, u3, v3;
    SVECTOR x0, x1, x2, x3;
    SVECTOR n0, n1, n2, n3;
    SVECTOR *v_ofs;
    SVECTOR *n_ofs;
    unsigned short vert0, vert1;
    unsigned short vert2, vert3;
    unsigned short norm0, norm1;
    unsigned short norm2, norm3;
} TMD_PRIM;
```

Members

<i>id</i>	TMD primitive ID
<i>r0, g0, b0,...r3, g3, b3</i>	RGB color values of the vertices of a primitive
<i>clut</i>	CLUT ID used by a primitive
<i>tpage</i>	Texture page used by a primitive
<i>u0, v0, u1, v1..u3, v3</i>	Texture coordinates of the vertices of a primitive
<i>x0, x1, x2, x3</i>	Three-dimensional coordinates of a primitive
<i>n0, n1, n2, n3</i>	Normal coordinates of a primitive
<i>v_ofs</i>	Pointer to start coordinates of a vertex array
<i>n_ofs</i>	Pointer to start coordinates of a normal array
<i>vert0, vert1..vert3</i>	Offset to a vertex array
<i>norm0, no x rm1..norm3</i>	Offset to a vertex array

Explanation

Information on primitives constituting a TMD object. The information is acquired using the ReadTMD() function. *x0, x1, x3, n0, n1, n3* are used for an independent vertex model. *v_ofs, n_ofs* and *vert0,..vert3, norm0...norm3* are used for a common vertex model.

Remarks

Some members have no meaning depending on the TMD primitive type.

See also:

AddPrim, addPrim

Registers the primitive to be drawn to the OT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void AddPrim (
void *ot,
void *p
)
```

```
addPrim(ot, p)
```

Arguments

ot OT entry
p Start address of primitive to be registered

Explanation

This function registers a primitive beginning with the address **p* to the OT entry **ot* in OT table. *ot* is an ordering table or pointer to another primitive.

addPrim() is a macro version of AddPrim().

Return value

None

Remarks

A primitive may only be added to a primitive list once in the same frame. Attempting to add it multiple times within the same frame will result in a corrupted list.

See also:

AddPrims, addPrims

Collectively registers primitives to be drawn to the OT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void AddPrims (
void *ot,
void *p0,
void *p1
)
```

```
addPrims(ot, p0, p1)
```

Arguments

ot OT entry
p0 Start primitive address of primitive list
p1 End primitive address of primitive list

Explanation

Registers primitives beginning with **ps* address the **ot* entry in the OT.

The primitive list is a list of the multiple primitives connected by AddPrim() or a list created by the local ordering table.

addPrims() is a macro version of AddPrims().

Return value

None

Remarks

See also:

BreakDraw

Interrupts drawing.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.4	3/26/98

Syntax

u_long *BreakDraw(*void*)

Arguments

None

Explanation

When this function is issued during drawing, the drawing of the current polygon is interrupted after termination. Because the next polygon drawing entry is returned, redrawing is possible if the DrawOTag() return value is issued in the argument.

Return value

Next polygon drawing entry.

However, during a DMA transfer outside the OT (such as LoadImage, etc.) 0xffffffff is returned.

Remarks

See also:

CatPrim, catPrim

Concatenates a primitive list.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void CatPrim (
void *p0,
void *p1
)
```

```
catPrim(p0, p1)
```

Arguments

p0, p1 Pointer to start addresses of primitive to be concatenated

Explanation

This function links the primitive *p1* to the primitive *p0*.

catPrim() is a macro version of CatPrim().

Return value

Start address of *p0*.

Remarks

AddPrim() adds a primitive to a primitive list. CatPrim() simply concatenates two primitives.

See also:

CheckPrim

Checks the validity of the specified primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
long CheckPrim(*s, *p)
char *s;
unsigned long *p;
```

Arguments

s Pointer to optimal character string
p Pointer to primitive start address

Explanation

This function checks the validity of the primitive. If the primitive is found to be invalid, it prints a message with the contents of the *s* parameter followed by the type code and length of the primitive. The primitive is not modified in any case.

Return value

Returns 0 for valid primitive. Returns -1 for an invalid primitive.

Remarks

See also:

ClearImage

Clears Frame Buffer at high speed.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	5/22/97

Syntax

```
int ClearImage(*rect, r, g, b)
RECT *rect;
unsigned char r, g, b;
```

Arguments

rect Pointer to rectangular area to be cleared
r, g, b Pixel values to be used for clearing

Explanation

Clears a rectangular area inside the Frame Buffer specified by *rect* at RGB color values indicated by (*r*, *g*, *b*).

Return value

Number in the queue

Remarks

Because this is a non-blocking function, the end of actual transfer must be detected using DrawSync(). The drawing area will not be affected by the drawing environment (clip/offset).

See also:

ClearImage2

Clears Frame Buffer at high speed (interlace).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	5/22/97

Syntax

```
int ClearImage2(*rect, r, g, b)
RECT *rect;
unsigned char r, g, b;
```

Arguments

rect Pointer to rectangular area to be cleared
r, g, b Pixel values to be used for clearing

Explanation

Clears a rectangular area inside the Frame Buffer specified by *rect* at the RGB color values indicated by (*r*, *g*, *b*).

Although ClearImage() only clears the field on one side when in interlace mode, ClearImage2() clears both fields.

Return value

Number in the queue

Remarks

Because this is a non-blocking function, the end of actual transfer must be detected using DrawSync(). The drawing area will not be affected by the drawing environment (clip/offset).

See also: ClearImage()

ClearOTag

Initializes an array to a linked list for use as an ordering table.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
unsigned long *ClearOTag(*ot, n)
unsigned long *ot;
int n;
```

Arguments

ot OT starting pointer
n Number of entries in OT

Explanation

This function walks the array specified by the *ot* parameter and sets each element to be a pointer to the following element, except the last. The *n* parameter specifies how many entries are present in the array. The last element of the array is set to a pointer to a special terminator value which the PlayStation uses to recognize the end of a primitive list.

Return value

None

Remarks

When you want to execute the OT initialized by “ClearOTag()”, execute “DrawOTag (ot)”.

See also:

ClearOTagR

Initializes an array to a linked list for use as an ordering table.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void ClearOTagR(*ot, n)
unsigned long *ot;
long n;
```

Arguments

ot Head pointer of OT
n Number of entries in OT

Explanation

This function walks the array specified by the *ot* parameter and sets each element to be a pointer to the previous element, except the first. The *n* parameter specifies how many entries are present in the array. The first element of the array is set to a pointer to a special terminator value which the PlayStation uses to recognize the end of a primitive list.

Return value

None

Remarks

When you want to execute the OT initialized by "ClearOTagR()", execute "DrawOTag (ot+n-1)".

See also:

ContinueDraw

Continues to draw the OT interrupted by BreakDraw

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	5/22/97

Syntax

```
void ContinueDraw (
  unsigned long *inst_ot,
  unsigned long *cont_ot
)
```

Arguments

inst_ot Address of interrupted OT
cont_ot Address of drawn OT immediately after drawing *inst_ot*

Explanation

This function immediately executes the OT supplied by *inst_ot* without entering it in the queue (*1). When the drawing of *inst_ot* is completed, it then draws *cont_ot*. Since the GPU must be in an immediately executable state, ContinueDraw() must be used in combination with BreakDraw(), etc.

This function is used when you wish to draw a specific OT with certain timing and high priority. In such cases, this can be achieved by using BreakDraw() to interrupt the OT being drawn and by executing the return value as *cont_ot*.

Return value

None

Remarks

When you want to execute the OT initialized by "ClearOTagR()", execute "DrawOTag (ot+n-1)".

See also: BreakDraw()

DrawPrim

Draws primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	6/22/98

Syntax

```
void DrawPrim (
void *p
)
```

Arguments

p Pointer to primitive

Explanation

Executes a primitive which has completed initialization.

Return value

None

Remarks

Slower speed than DrawOTag().

See also:

DrawOTag

Executes a list of GPU primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Syntax

```
void DrawOTag(*ot)
unsigned long *ot;
```

Arguments

ot Pointer to a linked list of GPU primitives

Explanation

This function executes the GPU primitives in the specified link list.

Return value

Remarks

The DrawOTag() function is non-blocking. To detect when execution of the primitive list is complete, use the DrawSync() function or install a callback routine with the DrawSyncCallback() function.

See also:

DrawOTag2

Executes a list of GPU primitives (immediate execution).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	10/01/97

Syntax

```
int DrawOTag2(p)
u_long *p;
```

Arguments

p Pointer to a linked list of GPU primitives

Explanation

This function executes the GPU primitives in the specified link list.

When drawing has been suspended using BreakDraw() and you want to execute a linked list of GPU primitives using DrawOTag(), immediate execution is not possible because of the need for queueing. If immediate execution is desired, you must use DrawOTag2().

Return value

0: Normal completion.

-1: Abnormal completion..

Remarks

When drawing is suspended with BreakDraw() after DrawOTag2() is called, before restarting the drawing with ContinueDraw(), it is necessary to confirm the completion of data transfer using IsIdleGPU(). This is because DrawOTag2() is a non-blocking function.

See also: BreakDraw(), ContinueDraw(), IsIdleGPU(), DrawOTag()

DrawOTagEnv

Executes a list of GPU primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Syntax

```
void DrawOTagEnv (
  u_long *p,
  DRAWENV *env
)
```

Arguments

p OT start pointer
env Drawing environment

Explanation

Sets the basic parameters for the drawing such as drawing offset/drawing clip area and collectively executes the primitives registered on OT.

Return value

None

Remarks

Following the drawing environment specified by DrawOTagEnv, PutDrawEnv () or DrawOTagEnv () will be executed or will be effective until the DR_ENV primitive is executed.

See also:

DrawOTagIO

Sets the drawing environment and draws the primitive registered on OT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Syntax

```
void DrawOTagIO (
  u_long *p
)
```

Arguments

p Pointer to top of OT

Explanation

Collectively executes the primitives registered on OT.

Return value

None

Remarks

Despite the fact that DrawOTagIO does not carry out the same operation as DrawOTag in checking the primitive adjustability, this is executed by means of the CPU. For debugging use.

See also:

DrawSync

Wait for all drawing to terminate.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
long DrawSync(mode)
long mode;
```

Arguments

The values which can be specified for mode are shown below.

Table 7-2

Value	Content
0	Wait for the termination of all non-block functions registered in the queue
1	Find out and return the number of positions in the current queue

Explanation

This function waits for drawing to terminate.

Return value

The Return value is the number of positions in the execution queue.

Remarks

If DrawSync(0) is used, and execution of the primitive list takes an exceptionally long time (approximately longer than 8 Vsync) to complete, a timeout is generated and the GPU is reset. Reasons why this might occur include an exceptionally long primitive list, or one that renders exceptionally large numbers of pixels. Another possibility is that the primitive list has been corrupted in some way. To avoid this, the application can use a loop such as:

```
while (DrawSync(1)) ;
```

See also:

DrawSyncCallback

Defines a callback function to be called when the GPU is finished executing a primitive list.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void DrawSyncCallback(*func)
void (*func)();
```

Arguments

func Pointer to callback function

Explanation

This defines the routine to be used as a callback when drawing is completed. When all requests in the queue have terminated, the function *func* is called. If *func* is set to 0, then any previous callback routine is disabled.

Return value

None

Remarks

Inside the callback, subsequent drawing termination interrupts are masked. Therefore, the callback routine should return as soon as possible. Also note that although the specified function is called during an interrupt, it is not an interrupt handler. It should be written as a normal subroutine that will be called by the main interrupt handler.

See also:

DumpClut, dumpClut

Printing contents of “clut” member of primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void DumpClut (
u_short clut
)
dumpClut(clut)
```

Arguments

clut CLUT ID

Explanation

This function prints the CLUT ID contents.

dumpClut() is a macro version of DumpClut().

Return value

None

Remarks

See also:

DumpDispEnv

Printing contents of display environment Structure.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void DumpDispEnv(*env)
DISPENV *env;
```

Arguments

env Pointer to display environment

Explanation

This function prints the contents of the display environment structure.

Return value

None

Remarks

See also:

DumpDrawEnv

Printing contents of drawing environment Structure.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void DumpDrawEnv(*env)
DRAWENV *env;
```

Arguments

env Pointer to drawing environment

Explanation

This function prints the contents of the drawing environment structure.

Return value

None

Remarks

See also:

DumpOTag

Prints the primitives registered in OT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void DumpOTag(*ot)
unsigned long *ot;
```

Arguments

ot OT starting pointer

Explanation

This function prints the code field of the primitives registered in the OT.

Return value

None

Remarks

See also:

DumpTPage, dumpTPage

Prints the contents of “tpage” member of primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void DumpTPage (
u_short tpage
)
```

```
dumpTPage(tpage)
```

Arguments

tpage Texture page ID

Explanation

This function prints the contents of the texture page ID.

dumpTPage () is a macro version of DumpTPage ().

Return value

None

Remarks

See also:

FntFlush

Draws contents of print stream.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
unsigned long *FntFlush(id)
long id;
```

Arguments

id Print stream ID

Explanation

This function draws the contents of the print stream into the frame buffer. It initializes and then draws a sprite primitive list corresponding to the characters specified in the print stream.

When *id* is set to -1, the print stream ID which was set in SetDumpFnt() is used (0 if print stream ID was not set).

Return value

The return value is the starting pointer of the primitive list used to perform the drawing.

Remarks

After the drawing has been done, the print stream contents are also flushed.

See also:

FntLoad

Transmits font pattern.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

void FntLoad(*tx*, *ty*)

long *tx*, *ty*

Arguments

tx, *ty* Font pattern frame buffer address

Explanation

This function transmits the built-in text font used for debugging text output to the frame buffer. It loads the basic font pattern (4-bit, 256x128) and initializes all the print streams.

Return value

None

Remarks

FntLoad() must always be executed before FntOpen() and FntFlush(). The font area must not clash with the frame buffer area used by the application. Font data is located at the upper left of the texture page for FntFlush(). Font data is treated as a RECT (0,0,32,32) area consisting of 128 characters, each 128 x 32. As this is similar to the texture page area, *tx* is restricted to a multiple of 64 and *ty* is restricted to a multiple of 256.

Loads the Clut to location (*tx*, *ty*+128).

See also:

FntOpen

Opens a print stream for printing.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

long FntOpen(*x, y, w, h, isbg, n*)

long *x, y*;

long *w, h*;

long *isbg*;

long *n*;

Arguments

x, y Display start location

w, h Display area

isbg Automatic clearing of background

0: Clear background to (0, 0, 0) when display is performed

1: Do not clear background to (0, 0, 0) when display is performed

n Maximum number of characters

Explanation

This function opens the stream for on-screen printing. After this, character strings up to *n* characters long can be drawn in the (*x, y*)- (*x+w, y+h*) rectangular area of the frame buffer, using FntPrint(). If "1" is specified for *isbg*, the background is cleared when a character string is drawn.

Return value

The return value is the stream ID.

Remarks

Up to 8 streams can be opened at once. However, once a stream is opened, it cannot be closed until the next time FntLoad() is called.

n specifies the maximum number of characters. Up to 1024 characters can be specified together in 8 streams.

See also:

FntPrint

Prints the specified string to an open print stream, using the same arguments and formatting parameters as the C library printf() function.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
long FntPrint(id, *format, [arg]...)
```

```
long id;
```

```
char *format;
```

Arguments

id Print stream ID

format Pointer to print format

Explanation

This function sends the string *format* to the specified print stream using the same interface as the fprintf() standard C library function.

Return value

The return value is the number of characters in the stream.

Remarks

The character string is not actually displayed until FntFlush() has been executed.

See also:

GetClut, getClut

Calculating the value of the “CLUT” member in a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
u_short GetClut (
int x,
int y
)
getClut(x, y)
```

Arguments

x, y Frame buffer address of CLUT

Explanation

This function calculates and returns the texture CLUT ID.

getClut() is a macro version of getClut().

Return value

CLUT ID

Remarks

The CLUT address is limited to multiples of 16 in the x direction.

See also:

GetDispEnv

Gets the current display environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

DISPENV *GetDispEnv(*env)

DISPENV *env;

Arguments

env Pointer to display environment start address

Explanation

This function stores the current display environment in the address specified by *env*.

Return value

The return value is a pointer to the display environment obtained by the function.

Remarks

See also:

GetDrawArea

Gets data for the current draw area.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

```
void GetDrawArea (
DR_AREA *p
)
```

Arguments

p starting address for DR_AREA primitive

Explanation

Reads GPU's current draw area settings into *p*.

Return value

None

Remarks

p must be initialized beforehand using SetDrawArea().

See also: SetDrawArea()

GetDrawEnv

Gets the current drawing environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
DRAWENV *GetDrawEnv(*env)
```

```
DRAWENV *env;
```

Arguments

env Pointer to drawing environment start address

Explanation

This function stores the current drawing environment in the address specified by *env*.

Return value

The return value is a pointer to the drawing environment obtained.

Remarks

See also:

GetDrawMode

Gets current draw mode data

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

```
void GetDrawMode (
DR_MODE *p
)
```

Arguments

p Starting address for DR_MODE primitives

Explanation

Reads GPU's current draw mode settings into p.

Return value

None

Remarks

p must be initialized beforehand with SetDrawMode().

See also: SetDrawMode()

GetDrawOffset

Gets the current draw offset.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

```
void GetDrawOffset (  
DR_OFFSET *p  
)
```

Arguments

p starting address for DR_OFFSET primitive

Explanation

Reads GPU's current draw offset settings into *p*.

Return value

None

Remarks

p must be initialized beforehand with SetDrawOffset().

See also: SetDrawOffset ()

GetGraphDebug

Gets present debug level.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Syntax

```
int GetGraphDebug(void)
```

Arguments

None

Explanation

Gets graphics system debug level.

Return value

Present debug level value.

Remarks

See also:

GetODE

Gets field currently being drawn.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	2/13/97

Syntax

```
int GetODE (
void
)
```

Arguments

None

Explanation

Gets field currently being drawn.

Return value

Current drawing field

0: VRAM even address being drawn

1: VRAM odd address being drawn

Remarks

See also:

GetTexWindow

Gets current texture window data.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

```
void GetTexWindow (
DR_TWAIN *p
)
```

Arguments

p Starting address for DR_TWAIN primitives

Explanation

Reads GPU's current texture window settings into *p*.

Return value

None

Remarks

p must be initialized beforehand with SetTexWindow().

See also: SetTexWindow...()

GetTimSize

Calculates the size of the Tim data domain returned by Krom2Tim ().

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Syntax

```
int GetTimSize (
u_char *sjis
)
```

Arguments

sjis Pointer to sjis character string

Explanation

Calculates size of the Tim data domain returned by Krom2Tim (). This size domain is maintained in malloc () and is designated Krom2Tim ().

Return value

Size of Tim data domain returned by Krom2Tim ().

Remarks

See also:

GetTPage, getTPage

Calculates the value of the member “tpage” in a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
u_short GetTPage (
```

```
int tp,
```

```
int abr,
```

```
int x,
```

```
int y
```

```
)
```

```
getTPage(tp, abr, x, y)
```

Arguments

tp Texture mode

0: 4bitCLUT

1: 8bitCLUT

2: 16bitDirect

abr Semitransparency rate

0: 0.5 x Back + 0.5 x Forward

1: 1.0 x Back + 1.0 x Forward

2: 1.0 x Back - 1.0 x Forward

3: 1.0 x Back + 0.25 x Forward

x, y Texture page address

Explanation

This function calculates the texture page ID, and returns it.

getTPage() is a macro version of GetTPage().

Return value

Texture page ID.

Remarks

The semitransparent rate is also effective for polygons on which texture mapping is not performed.

The texture page address is limited to a multiple of 64 in the X direction and a multiple of 256 in the Y direction.

See also:

IsEndPrim, isendprim

Decides the final ending primitive of the list.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
int IsEndPrim (
void *p
)
isendprim(p)
```

Argument

p Primitive start address

Explanation

Decides if the end of the primitive list is *p*.

isendprim() is a macro version of IsEndPrim().

Return value

Returns 1 in final end case and returns 0 in non-final end case.

Remarks

See also:

IsIdleGPU

Checks if the drawing once suspended by BreakDraw was completed.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.6	10/23/96

Syntax

```
int IsIdleGPU (
  int maxcount
)
```

Argument

maxcount Count value

Explanation

Although drawing is suspended by BreakDraw, GPU will not stop until the drawing is completed. Thus this function checks if the drawing suspended by BreakDraw has been completed or not. If GPU will not be an idle state within the time given by maxcount, -1 will be returned.

Return value

0: GPU is in idle state. -1: GPU is in drawing state.

Remarks

See also:

KanjiFntClose

Closes the printstream.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	6/22/98

Syntax

```
void KanjiFntClose(void)
```

Argument

None

Explanation

This function closes all the streams currently open and are used by KanjiFntPrint() and initialize the state.

Return value

None

Remarks

Since KanjiFntClose() only initializes the internal state, this function operates even when there is no stream opened at the invocation of the function.

See also:

KanjiFntFlush

Draws contents of the specified Kanji print stream.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	6/22/98

Syntax

```
unsigned long *KanjiFntFlush(id)
int id;
```

Argument

id Print stream ID

Explanation

This function draws the contents of the Kanji print stream into the frame buffer. It initializes and then draws a sprite primitive list corresponding to the characters specified in the print stream.

Return value

Start pointer of a primitive list used for drawing

Remarks

The contents of a print stream are also flushed after the end of drawing.

To internally reserve the transfer buffer on the stack, the stack uses approximately 72K.

See also:

KanjiFntOpen

Opens a print stream for printing.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	02/15/98

Syntax

```
int KanjiFntOpen(x, y, w, h, dx, dy, cx, cy, isbg, n)
```

```
int x, y, w, h, dx, dy, cx, cy, isbg, n;
```

Arguments

x, y Position of starting display
w, h Display area
dx, dy Kanji font pattern frame buffer address
cx, cy Kanji clut frame buffer address
isbg Automatic background clear
 0: Does not clear the background to (0, 0, 0) during display
 1: Clears the background to (0, 0, 0) during display
n Maximum number of characters

Explanation

This function opens a stream for open screen print. Then, the KanjiFntPrint() function can be used to render a character string composed of up to *n* characters in the rectangular area of (*x, y*) and (*x+w, y+h*) on the frame buffer. With *isbg* assigned a value of one, the background is cleared when a character string is rendered.

Return value

Stream ID.

Remarks

Up to eight streams can be opened at a time. The opened stream cannot be closed until the KanjiFntLoad() function is called. The kanji font area must not interfere in the frame buffer area used for applications.

See also:

KanjiFntPrint

Prints the specified string, in SJIS ZENKAKU format, to an open print stream, using the same arguments and formatting parameters as the C library printf() function.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	7/31/96

Syntax

```
int KanjiFntPrint(id, *format, [arg]...)
int id;
char *format;
```

Arguments

id Print stream ID
format Pointer to print format

Explanation

Send SJIS ZENKAKU string using printf() interface.

Return value

Number of characters within the stream.

Remarks

KANJI code must be the SJIS. Although both ZENKAKU and HANKAKU characters can be mixed in the string, a HANKAKU character will be converted to ZENKAKU when it is drawn. HANKAKU KANA characters are not supported. Actual drawing of the string will be done at execution of KanjiFntFlush(). When there is ~p in the string format, all the characters after ~p will be drawn in half-pitch.

See also:

Krom2Tim

Converts SJIS character string to 4 bits clut Tim data.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	6/22/98

Syntax

```
int Krom2Tim (
u_char *sjis;
u_long *taddr;
int dx;
int dy;
int cdx;
int cdy;
u_int fg;
u_int bg;
)
```

Arguments

<i>sjis</i>	SJIS character string
<i>taddr</i>	Tim area for storing data
<i>dx, dy</i>	Pixel data x,y coordinates on VRAM
<i>cdx, cdy</i>	Clut data x,y coordinates on VRAM
<i>fg, bg</i>	Character color and bg color

Explanation

Converts SJIS character string to 4 bits clut TIM data and returns to *taddr*.

Return value

When an abnormal code is given, -1 is returned.

Remarks

The size area returned by *GetTimSize* must be secured in advance.

The Kanji code must be SJIS. Full-width and half-width characters can be mixed within the character string, but when they are displayed, they will all be converted to full-width characters. Half-width characters are not supported.

To internally reserve the transfer buffer on the stack, the stack uses approximately 72K.

See also:

LoadClut

Loads texture CLUT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
unsigned short LoadClut(*col, x, y)
unsigned long *col;
long x, y;
```

Arguments

col Pointer to CLUT data start address
x, y Destination coordinates in frame buffer

Explanation

This function loads 256 entries of texture color data (CLUT) from main memory address *col* into the frame buffer to keep consistent with the syntax of the function definition.

Return value

The Return value is the CLUT ID for the loaded CLUT.

Remarks

256 palette entries are always transmitted, even in 4-bit mode.

See also:

LoadClut2

Loads CLUT for 16 colors.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	2/13/97

Syntax

```
u_short LoadClut2 (
u_long *clut,
int x,
int y
)
```

Arguments

clut Texture color start address
x, y Destination frame buffer address

Explanation

This function loads texture color data (CLUT) in the frame buffer (x,y) and calculates the ID of the loaded texture CLUT.

Return value

CLUT ID for loaded CLUT.

Remarks

LoadClut2() transmits 16 palette entries whereas LoadClut() transmits 256 palette entries.

LoadClut2() internally invokes LoadImage().

See also:

LoadImage

Transfers data to a frame buffer.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
int LoadImage(*recp, *p)
RECT *recp;
unsigned long *p;
```

Arguments

recp Pointer to destination rectangular area
p Pointer to main memory address of source of transmission

Explanation

This function transfers the contents of memory from the address *p* to the rectangular area in the frame buffer specified by *recp*.

Return value

Number in the queue.

Remarks

Because LoadImage() is a non-block function, the transmission termination has to be detected by "DrawSync()".

The transfer areas at the source and destination are not affected by the drawing environment (clip, offset). The destination area must be located within a drawable area (0, 0) - (1023, 511). See the description of the DR_LOAD primitive.

See also:

LoadImage2

Transfers data to a frame buffer (immediate execution).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	10/01/97

Syntax

```
int LoadImage2(rectp, p)
RECT *rect;
u_long *p;
```

Arguments

rect Pointer to destination rectangular area
p Pointer to main memory address of source of transfer

Explanation

This function immediately transfers the contents of memory beginning at the address pointed to by *p* to the rectangular area in the frame buffer specified by *rect*, *without queueing*.

When drawing has been suspended using BreakDraw() and you want to transfer data to the frame buffer using LoadImage(), immediate execution is not possible because of the need for queueing. If immediate execution is desired, you must use LoadImage2().

Return value

0: Normal completion.
-1: Abnormal completion..

Remarks

The drawing area (clip offset) does not affect the transfer area.

The destination area must be located within a drawable area (0, 0) - (1023, 511).

When drawing is suspended with BreakDraw() after LoadImage2() is called, before restarting the drawing with ContinueDraw(), it is necessary to confirm the completion of data transfer using IsIdleGPU(). This is because LoadImage2() is a non-blocking function.

See also: BreakDraw(), ContinueDraw(), IsIdleGPU(), LoadImage()

LoadTPage

Loads a texture page.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
unsigned short LoadTPage(*pix, tp, abr, x, y, w, h)
unsigned long *pix;
int tp, abr, x, y, w, h;
```

Arguments

pix Pointer to texture pattern start address
tp Transfer texture type
abr Semitransparency rate
x, y Destination frame buffer address
w, h Texture pattern size

Explanation

This function loads a texture pattern from the memory area starting at the address *pix* into the frame buffer area starting at the address (*x, y*), and calculates the texture page ID for the loaded texture pattern.

Return value

Texture page ID for the loaded texture pattern.

Remarks

The texture pattern size is not the actual size of the transfer area in the frame buffer. The texture pattern size is net in pixels.

LoadTPage() starts from within LoadImage().

See also:

MergePrim

Unites number primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	10/01/97

Syntax

```
int MergePrim (
void *p0,
void *p1
)
```

Arguments

p0 Primitives that are connected
p1 Primitives that connect

Explanation

Links primitive *p0* to primitive *p1*. All following linked primitives are, as usual, able to process AddPrim ().

Return value

If successful, returns 0, in cases of failure returns -1.

Remarks

p0 and *p1* describe continuous regions of memory. *p1* must be the rear address.

The combined primitive size of *p0* and *p1* must be less than 15 words. Within this size, any number of connections is possible.

See also:

MovelImage

Transfers data between two locations within the frame buffer.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
int MovelImage(*rect, x, y)
RECT *rect;
int x, y;
```

Arguments

rect Pointer to source rectangular area
x, y Top left corner of the destination rectangle

Explanation

The rectangular area of the frame buffer specified by *rect* is transmitted to the rectangular area of the same size which starts at (*x, y*).

The content at the source is preserved. If the source and destination areas are the same, normal operation is not guaranteed.

Return value

Number in the queue.

Remarks

Because MovelImage() is a non-block function, the termination of the transmission has to be detected by DrawSync().

The transfer areas at the source and destination are not affected by the drawing environment (clip, offset). The destination area must be located within a drawable area (0, 0) - (1023, 511). See also the description of the DR_MOVE primitive.

See also:

MovelImage2

Transfers data between two locations within the frame buffer (immediate execution).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	10/01/97

Syntax

```
int MovelImage2(*rect, x, y)
RECT *rect;
int x, y;
```

Arguments

rect Pointer to source rectangular area
x, y Top left corner of the destination rectangle

Explanation

The rectangular area of the frame buffer specified by *rect* is transferred to the rectangular area of the same size starting at (*x*, *y*).

The contents of the source rectangle are preserved. If the source and destination areas are the same, normal operation is not guaranteed.

When drawing is suspended with BreakDraw() and you want to move data within the frame buffer using MovelImage(), immediate execution is not possible because of the need for queueing. If immediate execution is desired, you must use MovelImage2().

Return value

0: Normal completion.

-1: Abnormal completion..

Remarks

The source and destination transfer areas are not affected by the drawing environment (clip, offset). The source and destination areas must be located within a drawable area (0, 0) - (1023, 511). See also the description of the DR_MOVE primitive.

When drawing is suspended with BreakDraw() after MovelImage2() is called, before restarting the drawing with ContinueDraw(), it is necessary to confirm the completion of data transfer using IsIdleGPU(). This is because MovelImage2() is a non-blocking function.

See also: BreakDraw(), ContinueDraw(), IsIdleGPU(), MovelImage()

NextPrim, nextPrim

Returns pointer to next primitive in primitive list.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void *NextPrim (
void *p
)
nextPrim(p)
```

Arguments

p Pointer to start address of a primitive

Explanation

This function returns a pointer to the next primitive in a primitive list.

nextPrim () is a macro version of NextPrim().

Return value

Pointer to the next primitive.

Remarks

See also:

OpenTIM

Opens TIM data.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
long OpenTIM(*addr)
unsigned long *addr;
```

Arguments

addr Pointer to main memory address to which the TIM has been loaded

Explanation

This function opens a TIM in main memory. The information in the opened TIM can then be read using the ReadTIM() function.

Return value

If it succeeds, "0" is returned. Any other value indicates failure.

Remarks

Only one TIM can be opened at a time. An opened TIM is not closed until the next time OpenTIM() is called.

See also:

OpenTMD

Opens TMD data.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
long OpenTMD(*tmd, obj_no)
unsigned long *tmd;
long obj_no;
```

Arguments

tmd Pointer to main memory address to which TMD has been loaded
obj_no Object number

Explanation

This function opens the TMD of the object specified by the *obj_no* parameter. The information in the opened TMD can then be read using the ReadTMD() function.

Return value

Returns the number of polygons comprising the object as a positive integer. Returns a negative number if it fails.

Remarks

Calling OpenTMD() closes any previously opened TMD.

See also:

PutDispEnv

Sets the display environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

DISPENV *PutDispEnv(*env)

DISPENV *env;

Arguments

env Pointer to display environment start address

Explanation

This function sets a display environment according to information specified by *env*. The display environment is executed as soon as the function is called.

Return value

This is a pointer to the display environment which has been set. (If the setting failed, the Return value is "0".)

Remarks

See also:

PutDrawEnv

Sets the drawing environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
DRAWENV *PutDrawEnv(*env)
DRAWENV *env;
```

Arguments

env Pointer to drawing environment start address

Explanation

Basic drawing parameters such as the drawing offset and the drawing clip area should be set in accordance with the setting specified in *env*.

Return value

This is a pointer to the drawing environment which has been set. (If setting failed, the Return value is "0".)

Remarks

The drawing environment specified using "PutDrawEnv()" is effective until the next time "PutDrawEnv()" is executed, or until the "DR_ENV" primitive is executed.

See also:

ReadTIM

Produces TIM header.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

TIM_IMAGE *ReadTIM(timing*)**

TIM_IMAGE **timing*;

Arguments

timing TIM_IMAGE AS structure pointer

Explanation

The ReadTIM() function sets the members of the TIM_IMAGE structure pointed to by *timing* according to the data specified by the most recent OpenTIM() function.

Return value

Returns the value of *timing* if succesful; returns 0 if it fails.

Remarks

See also:

ReadTMD

Reads contents of TMD primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
TMD_PRIM *ReadTMD(*tmdprim)
TMD_PRIM *tmdprim;
```

Arguments

tmdprim Pointer to printer for TMD-PRIM structure

Explanation

The ReadTMD() function sets the members of the TMD_PRIM structure pointed to by *tmdprim* according to the data specified by the most recent OpenTMD() function.

Return value

Returns *tmdprim* if successful; 0 if fails.

Remarks

Note that the TMD_PRIM structure includes fields that are not used for all types of objects. ReadTIM() copies only those fields that are valid for the current object.

See also:

ResetGraph

Initializes drawing engine.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
int ResetGraph(mode)
int mode;
```

Arguments

mode Reset mode

Explanation

This function resets the graphic system in mode specified by *mode*. Possible setting of more are listed below.

Table 7-3

Mode	Operation
0	Complete reset. The drawing environment and display environment are initialized.
1	Cancels the current drawing and flushes the command buffer.
3	Initializes the drawing engine while preserving the current display environment (i.e. the screen is not cleared or the screen mode changed).

Return value

None

Remarks

See also:

SetDefDispEnv

Sets display environment structure members and screen display area.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	6/22/98

Syntax

```
DISPENV *SetDispEnv(*disp, x, y, w, h)
DISPENV *disp;
int x, y;
int w, h;
```

Arguments

disp Pointer to display environment
x, y Upper left corner of display area
w, h Width and height of the display area

Explanation

This function sets the members of a DISPENV (display environment) structure. The new display area is specified using the coordinates within the frame buffer of the top left corner, along with the width and height, of the desired rectangle.

Table 7-4

Member	Content	Value
disp	Display area	(x, y, w, h)
screen	Screen display area	(0, 0)-(256, 240)
isinter	Interlace flag	0
isrgb24	24-bit mode flag	0

Return value

The return value is the starting pointer of the display environment which has been set.

Remarks

This function does not actually change the display environment. It merely sets the members of the specified structure as desired. Use the PutDispEnv() function with this structure to change the actual environment.

See also:

SetDefDrawEnv

Set standard drawing environment structure.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
DRAWENV *SetDefDrawEnv(*env, x, y, w, h)
```

```
DRAWENV *env;
```

```
int x, y, w, h;
```

Arguments

env Pointer to drawing environment
x, *y* Upper left corner of drawing area
w, *h* Width and height of drawing area

Explanation

This function sets the drawing area members of a DRAWENV (drawing environment) structure. The new drawing area is specified using the coordinates within the frame buffer of the top left corner, along with the width and height, of the desired rectangle.

Table 7-5

Member	Content	Value
clip	Drawing area	(x, y, w, h)
ofs[2]	Drawing offset	(x, y)
tw	Texture window	(0, 0, 0, 0)
tpage	Texture page (tp, abr, tx, ty)	(0, 0, 640, 0)
dtd	Dither processing flag	1 (ON)
dfe	Permission flag for drawing	1 (drawing on display area is inhibited)
isbg	Draw area clear flag	0 (clear: OFF)
r0, g0, b0	Background color	(0, 0, 0)

Return value

The return value is the starting pointer of the drawing environment which has been set.

Remarks

This function does not actually change the drawing environment. It merely sets the members of the specified structure as desired. Use the PutDrawEnv() function with this structure to change the actual environment.

See also:

SetDispMask

Sets and cancels display mask.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void SetDispMask(mask)
int mask;
```

Arguments

mask Display mask

Explanation

This function puts display mask into the status specified by *mask*. Any of the following can be designated as *mask*:

Table 7-6

Mask	Operation
0	Not displayed on screen
1	Displayed on screen

Return value

None

Remarks

See also:

SetDrawArea

Initializes the content of drawing area setting primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void SetDrawArea(*p, *r)
DR_AREA *p;
RECT *r;
```

Arguments

p Pointer to drawing area setting primitive
r Pointer to drawing area

Explanation

Initializes a DR_AREA primitive. By using AddPrim() to insert a DR_AREA primitive into your primitive list, it is possible to change part of your drawing environment in the middle of drawing.

Return value

None

Remarks

See also:

SetDrawEnv

Initializes the content of the drawing environment change primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void SetDrawEnv(*dr_env, *env)
DR_ENV *dr_env;
DRAWENV *env;
```

Arguments

dr_env Pointer to drawing environment change primitive
env Pointer to drawing environment structure in which the drawing environment is described

Explanation

Initializes a DR_ENV primitive using the values contained in a DRAWENV structure. By using AddPrim() to insert a DR_ENV primitive into your primitive list, it is possible to change part of your drawing environment in the middle of drawing.

Return value

None

Remarks

The DR_ENV primitive uses the same information as the DRAWENV structure, but the data format is different and the DRAWENV structure cannot be used as a primitive. When the DR_ENV primitive is executed, the previous drawing environment settings are destroyed.

See also:

SetDrawLoad

Initializes the content of the LoadImage primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	6/22/98

Syntax

```
void SetDrawLoad (
DR_LOAD *p,
RECT *rect
)
```

Arguments

p Destination rectangular area primitive
rect Destination rectangular area

Explanation

This function initializes the destination rectangular area primitive. By registering the initialized primitive in OT using AddPrim(), the rectangular area can be transferred just as in the LoadImage() function.

Maximum data transfer amount is 12 words (24 pixels).

Return value

None

Remarks

See also:

SetDrawMode

Initializes the content of a drawing mode primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetDrawMode(*p, dfe, dtd, tpage, *tw)
DR_MODE *p;
int dfe, dtd, tpage;
RECT *tw;
```

Arguments

p Pointer to drawing mode primitive
dfe Flag for drawing to a display area 0: OFF, 1: ON
dtd Dither processing flag: 0: OFF, 1: ON
tpage Texture page
tw Pointer to texture window

Explanation

Initializes a DR_MODE primitive using the specified values. By using AddPrim() to insert a DR_MODE primitive into your primitive list, it is possible to change part of your drawing environment in the middle of drawing.

If *tw* is 0, the texture window is not changed.

See the table below for allowable values for the *dfe* and *dtd* parameters.

Table 7-7

<i>dfe</i>	Action
0	No drawing in display area
1	Drawing in display area
<i>dtd</i>	Action
0	Dither processing not performance
1	Dither processing performance

Return value

None

Remarks

See also:

SetDrawMove

Initializes the contents of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetDrawMove (
DR_MOVE *p,
RECT *rect,
int x,y
)
```

Arguments

p Pointer to rectangular area copy primitive
rect Rectangular area to be transferred
x,y Upper left edge of the rectangular area transfer destination

Explanation

Initializes the rectangular area copy primitive. After the primitive is initialized, it is entered in the OT using AddPrim(). Therefore, it can perform the same processing (copy of rectangular area) as MoveImage().

Return value

None

Remarks

See also:

SetDrawOffset

Initializes the content of drawing offset setting primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void SetDrawOffset(*p, *ofs)
DR_OFFSET *p;
u_short *ofs;
```

Arguments

p Pointer to drawing offset setting primitive
ofs Pointer to drawing offset

Explanation

Initializes a DR_OFFSET primitive using the specified values. By using AddPrim() to insert a DR_OFFSET primitive into your primitive list, it is possible to change part of your drawing environment in the middle of drawing.

Return value

None

Remarks

See also:

SetDrawStp

Initializes the STP bit update primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	10/01/97

Syntax

```
void SetDrawStp(*p, *r)
DR_STP *p;
int pbw;
```

Arguments

p Pointer to primitive
r STP bit update flag (0: STP bit OFF, 1: STP bit ON)

Explanation

Initializes the DR_STP primitive pointed to by *p*.

When *pbw* = 0, normal drawing is performed. When *pbw* = 1, drawing is performed with the STP bit set (STP is a 16-bit object).

Return value

None

Remarks

See also:

SetDrawTPage, setDrawTPage

Initializes the contents of texture page change primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetDrawTPage (
DR_TPAGE *p,
int dfe,
int dtd,
int tpage
)
```

```
setDrawTPage(p, dfe, dtd, tpage)
```

Arguments

<i>p</i>	Texture page setting primitive
<i>dfe</i>	Flag for drawing to a display area 0: no drawing in display area 1: drawing in display area
<i>dtd</i>	Dither processing flag 0: dither processing not performed 1: dither processing performed
<i>tpage</i>	Texture page

Explanation

This function initializes the texture page change primitive. By registering the initialized primitive in OT using AddPrim(), the texture page can be changed while drawing.

setDrawTPage() is a macro version of SetDrawTPage().

Return value

None

Remarks

See also:

SetDumpFnt

Defines stream for onscreen dump.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void SetDumpFnt(id)
long id;
```

Arguments

id Print stream ID

Explanation

This function sets the print stream for debug printing. The output of the debug printing functions can then be carried out in relation to the stream specified in *id*.

Return value

None

Remarks

The actual display is executed by the FntFlush() function.

See also:

SetGraphDebug

Sets debugging level.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

```
void SetGraphDebug(level)
int level;
```

Arguments

level Debugging level

Explanation

Set a debugging level for the graphics system. Any of the following can be designated as *level*:

Table 7–8

Level	Operation
0	No checks are performed. (Highest speed mode)
1	Checks coordinating registered and drawn primitives.
2	Registered and drawn primitives are dumped.

Return value

The previously set debug level.

Remarks

See also:

SetLineF2, SetLineF3, SetLineF4; setLineF2, setLineF3, setLineF4

Initializes flat unconnected straight line drawing primitive.

Initializes flat connected 2-straight line drawing primitive.

Initializes flat connected 3-straight line drawing primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetLineF2 (  
  LINE_F2 *p  
)
```

setLineF2 (*p*)

```
void SetLineF3 (  
  LINE_F3 *p  
)
```

setLineF3 (*p*)

```
void SetLineF4 (  
  LINE_F4 *p  
)
```

setLineF4 (*p*)

Arguments

p Pointer to primitive start address

Explanation

These functions initialize the primitives specified by *p*.

setLineF2() is a macro version of SetLineF2().

setLineF3() is a macro version of SetLineF3().

setLineF4() is a macro version of SetLineF4().

Return value

None

Remarks

See also:

SetLineG2, SetLineG3, SetLineG4; setLineG2, setLineG3, setLineG4

Initializes gouraud unconnected straight line drawing primitive.

Initializes gouraud connected 2-straight line drawing primitive.

Initializes gouraud connected 3-straight line drawing primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetLineG2 (
LINE_G2 *p
)
```

```
setLineG2(p)
```

```
void SetLineG3 (
LINE_G3 *p
)
```

```
setLineG3(p)
```

```
void SetLineG4 (
LINE_G4 *p
)
```

```
setLineG4(p)
```

Arguments

p Pointer to primitive start address

Explanation

These functions initialize the primitives specified by *p*.

setLineG2() is a macro version of SetLineG2().

setLineG3() is a macro version of SetLineG3().

setLineG4() is a macro version of SetLineG4().

Return value

None

Remarks

See also:

SetPolyF3, SetPolyF4; setPolyF3, setPolyF4

Initializes flat shaded triangle primitive.

Initializes flat shaded quadrangle primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetPolyF3 (
  POLY_F3 *p
)
```

```
setPolyF3(p)
```

```
void SetPolyF4 (
  POLY_F4 *p
)
```

```
setPolyF4(p)
```

Arguments

p Pointer to primitive start address

Explanation

These functions initialize the primitive specified by *p*.

setPolyF3() is a macro version of SetPolyF3().

setPolyF4() is a macro version of SetPolyF4().

Return value

None

Remarks

See also:

SetPolyFT3, SetPolyFT4; setPolyFT3, setPolyFT4

Initializes flat textured triangle primitive.

Initializes flat textured quadrangle primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetPolyFT3 (
  POLY_FT3 *p
)
```

```
setPolyFT3(p)
```

```
void SetPolyFT4 (
  POLY_FT4 *p
)
```

```
setPolyFT4(p)
```

Arguments

p Pointer to primitive start address

Explanation

These functions initialize the primitive specified by *p*.

setPolyFT3() is a macro version of SetPolyFT3().

setPolyFT4() is a macro version of SetPolyFT4().

Return value

None

Remarks

See also:

SetPolyG3, SetPolyG4; setPolyG3, setPolyG4

Initializes gouraud shaded triangle primitive.

Initializes gouraud shaded quadrangle primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetPolyG3 (
    POLY_G3 *p
)
```

```
setPolyG3(p)
```

```
void SetPolyG4 (
    POLY_G4 *p
)
```

```
setPolyG4(p)
```

Arguments

p Pointer to primitive start address

Explanation

These functions initialize the primitive specified by *p*.

setPolyG3() is a macro version of SetPolyG3().

setPolyG4() is a macro version of SetPolyG4().

Return value

None

Remarks

See also:

SetPolyGT3, SetPolyGT4; setPolyGT3, setPolyGT4

Initializes gouraud textured triangle primitive.
Initializes gouraud textured quadrangle primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetPolyGT3 (
  POLY_GT3 *p
)
```

```
setPolyGT3(p)
```

```
void SetPolyGT4 (
  POLY_GT4 *p
)
```

```
setPolyGT4(p)
```

Arguments

p Pointer to primitive start address

Explanation

These functions initialize the primitive specified by *p*.

setPolyGT3() is a macro version of SetPolyGT3().

setPolyGT4() is a macro version of SetPolyGT4().

Return value

None

Remarks

See also:

SetSemiTrans, setSemiTrans

Sets the semitransparent attribute of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetSemiTrans (  
void *p,  
int abe  
)
```

setSemiTrans(*p*, *abe*)

Arguments

p Pointer to primitive start address
abe Semitransparent flag
 0: semitransparent OFF
 1: semitransparent ON

Explanation

This function sets the semitransparent attribute of the primitive specified by *p* to the value specified by the *abe* parameter. If semitransparent mode is enabled, then semitransparent pixels are drawn as specified by the table below.

Table 7-9

Primitive	Pixels subjected to semitransparent processing
POLY_FT3/POLY_FT4	Pixels for which the topmost bit of the corresponding texture pixel is "1"
POLY_GT3/POLY_GT4	Pixels for which the topmost bit of the corresponding texture pixel is "1"
SPRT/SPRT_8/SPRT_16	Pixels for which the topmost bit of the corresponding texture pixel is "1"
Other drawing primitives	All Pixels

setSemiTrans() is a macro version of SetSemiTrans().

Return value

None

Remarks

Semitransparent pixels are calculated from the foreground pixels *Pf* and background pixels *Pb* as follows:

$$P = F \times Pf + B \times Pb$$

The rate (F, B) of semitransparency is designated by the member *tpage* in the primitive. Drawing speed is reduced because semitransparency requires reading of background brightness values. Therefore, do not draw primitives with semitransparent mode turned on unless they are to be displayed that way.

See also:

SetShadeTex, setShadeTex

Inhibiting shading function.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetShadeTex (
void *p,
int tge
)
```

setShadeTex(*p*, *tge*)

Arguments

p Pointer to primitive start address
tge Unshaded flag
 0: Shading is performed
 1: Shading is not performed

Explanation

This function sets the shading attribute of the primitive pointed to by *p* to the value specified by the *tge* parameter.

When texture and shading are both ON, each pixel in the polygon is calculated as shown below from the pixel value "T" of the corresponding texture pattern, and the brightness value "L" correspondong to the pixel value "T".

$$P = (T \& 0xf8 * L \& 0xf8) / 128$$

if (*p* > 255) *p* = 255
 if (*p* < 0) *p* = 0

When "L" = 128, the brightness value of the texture pattern is drawn as it is. If the value results in an overflow, the pixel value is clipped to 255.

When *tge* = 1, the brightness value is not divided, and the texture pattern value is used, as it is, as the pixel value.

T, L are only effective for the upper 5 bits. The lower 3 bits are discarded.

setShadeTex() is a macro version of SetShadeTex().

Return value

None

Remarks

This function cannot be used for primitives other than "POLY_FT3", "POLY_FT4", "SPRT", "SPRT_8", and "SPRT_16".

Although the texture number of colors is saved at intensity level of 32 when using ShadeTex, the shading brightness value is decreased from an intensity level of 256 to 32.

See also:

**SetSprt, SetSprt8, SetSprt16;
setSprt, setSprt8, setSprt16**

Initialize a SPRT primitive.
Initialize a SPRT8 primitive.
Initialize a SPRT16 primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

void SetSprt (
SPRT **p*
)

setSprt(*p*)

void SetSprt8 (
SPRT_8 **p*
)

setSprt8(*p*)

void SetSprt16 (
SPRT_16 **p*
)

setSprt16(*p*)

Arguments

p Pointer to primitive start address

Explanation

These functions initialize the primitives specified by *p*. Details are given below.

Table 7–10

Function name	Sprite size	Primitive
SetSprt8	8 x 8	SPRT_8
SetSprt16	16 x 16	SPRT_16
SetSprt	Can be set at will using values of members h, w. (0 < h < 255, 0 < w < 255)	SPRT

setSprt() is a macro version of SetSprt().
setSprt8() is a macro version of SetSprt8().
setSprt16() is a macro version of SetSprt16().

Return value

None

Remarks

The SPRT... primitives are faster than POLY_FT4. TILE is also faster than POLY_F4.

See also:

SetTexWindow

Initializes the content of a texture window primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	3/26/98

Syntax

```
void SetTexWindow(*p, *tw)
DR_TWIN *p;
RECT *tw;
)
```

setTexWindow (*p*, *tw*)

Arguments

p Pointer to texture window primitive
tw Pointer to texture window

Explanation

Initializes a DR_TWIN primitive using the specified values. By using AddPrim() to insert a DR_TWIN primitive into your primitive list, it is possible to change the current texture window in the middle of drawing.

setTexWindow() is the macro version of SetTexWindow().

Return value

None

Remarks

See also:

SetTile, SetTile1, SetTile8, SetTile16; setTile, setTile1, setTile8, setTile16

Initialize a TILE primitive.

Initialize a TILE1 primitive.

Initialize a TILE8 primitive.

Initialize a TILE16 primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void SetTile (  
TILE *p  
)
```

```
setTile(p)
```

```
void SetTile (  
TILE_1 *p  
)
```

```
setTile1(p)
```

```
void SetTile (  
TILE_8 *p  
)
```

```
setTile8(p)
```

```
void SetTile (  
TILE_16 *p  
)
```

```
setTile16(p)
```

Arguments

p Pointer to primitive start address

Explanation

These functions initialize the primitives specified by *p*. Details are given below.

Table 7-11

Function name	Tile size	Primitive size
SetTile1	1 x 1	TILE_1
SetTile8	8 x 8	TILE_8
SetTile16	16 x 16	TILE_16
SetTile	Can be set at will using values of members h, w. (0 < h < 255, 0 < w < 255)	TILE

setTile() is a macro version of SetTile().
setTile1() is a macro version of SetTile1().
setTile8() is a macro version of SetTile8().
setTile16() is a macro version of SetTile16().

Return value

None

Remarks

The SPRT... primitives are faster than POLY_FT4. TILE is also faster than POLY_F4.

See also:

setVWH

Sets the UV members of the 4-point designated primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

setVWH(**p*, *u0*, *v0*, *w*, *h*)

Arguments

p Primitive pointer
u0, *v0* Left top point of primitive texture
w, *h* Width and height of primitive texture

Explanation

Designates the (u0, v0) - (u0 + w, v0 + h) on the diagonal line containing each coordinate of the rectangle as the (u0, v0). . (u3, v3) members of the primitive.

Return value

None

Remarks

setVWH is a macro, so there is no dependence on the primitive model.

Cannot be used in the sprite primitive.

See also:

setXYWH

Sets the XY members of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

setXYWH(**p*, *x0*, *y0*, *w*, *h*)

Arguments

p Primitive pointer
x0, *y0* Upper left corner of primitive
w, *h* Width and height of primitive

Explanation

This macro sets the *x0*, *y0*, *x1*, *y1*, *x2*, *y2*, *x3*, and *y3* fields of a primitive structure to represent the corners of the rectangle specified by the input parameters.

Return value

None

Remarks

This is a C preprocessor macro and can be used with any primitive or structure with the appropriate fields.

See also:

StoreImage

Transfers image data from the frame buffer to main memory.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
int StoreImage(*recp, *p)
RECT *recp;
unsigned long *p;
```

Arguments

recp Pointer to destination rectangular area
p Pointer to main memory address of destination of transmission

Explanation

This function transfers the rectangular area of the frame buffer specified by *recp* to the main memory storage location starting at the address specified by the *p* parameter.

Return value

Number in the queue.

Remarks

Because StoreImage() is a non-blocking function, use the DrawSync() function to determine when the operation has completed.

The transfer areas at the source and destination are not affected by the drawing environment (clip, offset). The source area must be located within a drawable area (0, 0) - (1023, 511).\

See also:

StoreImage2

Transfers data from a frame buffer (immediate execution).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	10/01/97

Syntax

```
int StoreImage2(*rect, *p)
RECT *rect;
u_long *p;
```

Arguments

rect Pointer to source rectangular area
p Pointer to destination main memory address

Explanation

This function immediately transfers the contents of the rectangular area in the frame buffer pointed to by *rect* to the main memory location starting with the address pointed to by *p*, without queueing. *p* must be on a word boundary.

When drawing is suspended with `BreakDraw()`, and you want to transfer data from the frame buffer using `StoreImage()`, immediate execution is not possible because of the need for queueing. If immediate execution is desired, you must use `StoreImage2()`.

Return value

0: Normal completion.
 -1: Abnormal completion..

Remarks

The drawing area (clip offset) does not affect the transfer area.

The transfer area must be located within a drawable area (0, 0) - (1023, 511).

When drawing is suspended with `BreakDraw()` after `StoreImage2()` is called, before restarting the drawing with `ContinueDraw()`, it is necessary to confirm the completion of data transfer using `IsIdleGPU()`. This is because `StoreImage2()` is a non-blocking function.

See also: `BreakDraw()`, `ContinueDraw()`, `IsIdleGPU()`, `StoreImage()`

TermPrim, termPrim

Terminates a primitive list

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

```
void TermPrim (
void *p
)
```

```
termPrim(p)
```

Arguments

p Pointer to start address of a primitive

Explanation

This function sets the tag pointer of the primitive specified by *p* to point at a special terminator value that will signal the end of the list when it is executed. Any primitives already pointed to by *p* will be removed from the list.

termPrim () is a macro version of TermPrim().

Return value

None

Remarks

See also:

VSync

Waits for the next vertical blank, or returns the vertical blank counter value.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	2.x	7/31/96

Syntax

```
int VSync(mode)
int mode;
```

Arguments

mode Mode

Explanation

This function waits for vertical blank using the method specified by the *mode* parameter, as defined below.

Table 7–12

Mode	Operation
0	Blocks until vertical sync is generated
1	Returns time elapsed from the point VSync() processing is last completed when mode=1 or n in horizontal sync units
n (n>1)	Blocks from the point VSync() processing is last completed when mode=1 or n until n number of vertical syncs are generated.
-n (n>0)	Returns absolute time after program boot in vertical sync interval units.

Return value

Mode value is as listed below.

Table 7–13

Mode	Return value
mode>=0	Time elapsed from the point that Vsync() processing is last completed when mode=1 or n (horizontal blanking units)
mode<0	Time elapsed after program boot (vertical blanking units)

Remarks

The Vsync() function may generate a timeout if long blocking periods are specified. To prevent deadlocks, rather than using Vsync() to block for an especially long time (say more than 4 vertical blank periods), have your program poll VSync(-1) in a loop instead.

See also:

VSynCallback

Defines a function to be executed during each vertical blank period.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	2.x	7/31/96

Syntax

```
void VSynCallback(*func)
void (*func)();
```

Arguments

func Pointer to callback function

Explanation

Specifies that the routine at address *func* should be executed at the start of the vertical blank interrupt. If *func* is 0, then any previous callback routine is disabled.

Return value

None

Remarks

Subsequent interrupts will be masked inside *func*. Therefore, it is necessary to return quickly after performing necessary processes using *func*.

Although the specified function is called during an interrupt, it is not the actual interrupt handler. It should be written as a normal subroutine that will be called by the main interrupt handler.

See also:

addVector

Adds vectors.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

addVector(*v0*, *v1*)

Arguments

v0, *v1* Pointers to vectors

Explanation

This macro adds *v1* to the vector *v0*, and stores the result in *v0*.

Return value

None

Remarks

`addVector()` is a macro, so there is no dependence on the vector type.

See also:

applyVector

Adds vectors.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.4	7/31/96

Syntax

applyVector(*v, x, y, z, op)

Arguments

v Pointer to vector
 x,y,z Coordinate value
 op Operator

Explanation

Performing the operation specified on vector v, x, y, z and op

appVector (v, 2, 4, 8, +=)

is equivalent to:

v->vx += 2, v->vy += 4, v->vz += 8

Return value

None

Remarks

applyVector is a macro, so there is no dependence on the vector model.

See also:

copyVector

Copies vectors.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

copyVector(*v0*, *v1*)

Arguments

v0, *v1* Vector pointer

Explanation

Copies vector *v1* to *v0*.

Return value

None.

Remarks

copyVector is a macro, so there is no dependence on the vector type.

See also:

dumpMatrix

Displays matrix contents.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

dumpMatrix (*x*)

Arguments

x pointer to matrix

Explanation

Displays the contents of the matrix pointed to by *x*.

If SetDumpFnt() is called, the output is displayed to the screen. Otherwise, the output is sent to stdout.

Return value

None

See also: SetDumpFnt()

dumpRECT

Displays contents of RECT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

dumpRECT (*r*)

Arguments

r pointer to RECT

Explanation

Displays the contents of the RECT structure pointed to by *r*.

If SetDumpFnt() is called, the output is displayed to the screen. Otherwise, the output is sent to stdout.

Return value

None.

Remarks

Since this is a macro, it does not depend on the vector type.

See also: SetDumpFnt()

dumpVector

Displays contents of vector

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

`dumpVector (str, v)`

Arguments

`str` string to be displayed

`v` pointer to vector

Explanation

Displays the contents of the vector pointed to by `v`.

If `SetDumpFnt()` is called, the output is displayed to the screen. Otherwise, the output is sent to `stdout`.

Return value

None.

See also: `SetDumpFnt()`

dumpWH...

Displays contents of primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

dump... (*p*)

Arguments

p pointer to primitive

Explanation

Displays the contents of the primitive pointed to by *p*.

The following macros are defined in the current *libgpu.h* for displaying primitives.

Macro name	Description
dumpWH	Displays w, h of primitive.
dumpXY0	Displays x0, y0 of primitive.
dumpXY2	Displays x0, x1, y0, y1 of primitive.
dumpXY3	Displays x0-x2, y0-y2 of primitive.
dumpXY4	Displays x0-x3, y0-y3 of primitive.
dumpUV0	Displays u0, v0 of primitive.
dumpUV3	Displays u0-u2, v0-v2 of primitive.
dumpUV3	Displays u0-u3, v0-v3 of primitive.
dumpRGB0	Displays r0, g0, b0 of primitive.
dumpRGB1	Displays r0, r1, g0, g1, b0, b1 of primitive.
dumpRGB2	Displays r0-r2, g0-g2, b0-b2 of primitive.
dumpRGB3	Displays r0-r3, g0-g3, b0-b3 of primitive.

If `SetDumpFnt()` is called, the output is displayed to the screen. Otherwise, the output is sent to `stdout`.

Return value

None.

Remarks

Since these are macros, they do not depend on the primitive type.

See also: `SetDumpFnt()`

SetClut

Sets CLUT for primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

setClut (*p*, *x*, *y*)

Arguments

p pointer to primitive
x, *y* CLUT frame buffer address

Explanation

The texture CLUT ID specified by the parameters is set in the clut member of primitive *p*.

Return value

None.

Remarks

Since this is a macro, it does not depend on the primitive type.
 For the CLUT address, the *x* component must be a multiple of 16.

See also: SetDumpFnt()

setRECT

Set rectangular area.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

setRECT(*r*, *x*, *y*, *w*, *h*)

Arguments

r Pointer to RECT structure
x, *y* Upper left point of rectangular area
w, *h* Size of rectangular area

Explanation

Sets the *x*, *y*, *w*, and *h* values of the RECT structure *r*.

Return value

None

Remarks

See also:

setRGB0, setRGB1, setRGB2, setRGB3

Initialize *r0*, *g0*, and *b0* fields of a primitive.

Initialize *r1*, *g1*, and *b1* fields of a primitive.

Initialize *r2*, *g2*, and *b2* fields of a primitive.

Initialize *r3*, *g3*, and *b3* fields of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

setRGB0(*p*, *r0*, *g0*, *b0*)

setRGB1(*p*, *r1*, *g1*, *b1*)

setRGB2(*p*, *r2*, *g2*, *b2*)

setRGB3(*p*, *r3*, *g3*, *b3*)

Arguments

p Primitive pointer

r, *g*, *b* RGB members of primitive

Explanation

These macros set the values for the RGB members of the primitive *p*.

Return value

None

Remarks

These are macros, so there is no dependence on the primitive type.

See also:

SetTPage

Sets texture page for primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

setTPage (*p*, *tp*, *abr*, *x*, *y*)

Arguments

- p* pointer to primitive
- tp* texture mode
 0: 4bitCLUT
 1: 8bitCLUT
 2: 16bitDirect
- abr* semi-transparency rate
 0: 0.5 x Back + 0.5 x Forward
 1: 1.0 x Back + 1.0 x Forward
 2: 1.0 x Back - 1.0 x Forward
 3: 1.0 x Back + 0.25 x Forward
- x*, *y* texture page address

Explanation

The texture page specified by the parameters is set in the tpage member of primitive p.

Return value

None.

Remarks

Since this is a macro, it does not depend on the primitive type.

See also:

setUV0, setUV3, setUV4

Set the *u0* and *v0* parameters of a primitive.

Set the *u3* and *v3* parameters of a primitive.

Set the *u4* and *v4* parameters of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

setUV0(**p*, *u0*, *v0*)

setUV3(**p*, *u0*, *v0*, *u1*, *v1*, *u2*, *v2*)

setUV4(**p*, *u0*, *v0*, *u1*, *v1*, *u2*, *v2*, *u3*, *v3*)

Arguments

p Primitive pointer

u, *v* UV members of primitive

Explanation

These macros set the values of the appropriate UV fields of the primitive *p*.

Return value

None

Remarks

These are C preprocessor macros and can be used with any primitive or structure with the appropriate fields.

See also:

setUVWH

Sets the UV members of a primitive structure.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

setUVWH(**p*, *u0*, *v0*, *w*, *h*)

Arguments

p Primitive pointer.
u0, *v0* Upper left corner of primitive texture
w, *h* Width and height of primitive texture

Explanation

This macro sets the *u0*, *v0*, *u1*, *v1*, *u2*, *v2*, *u3*, and *v3* fields of a primitive structure to represent the corners of the rectangle specified by the input parameters.

Return value

None

Remarks

This is a C preprocessor macro and can be used with any primitive or structure with the appropriate fields.

See also:

setVector

Setting a vector value.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

setVector(*v, x, y, z)

Arguments

v Pointer to a vector
x, y, z Coordinate values

Explanation

Sets the (x, y, z) value for VECTOR/SVECTOR.

Return value

None

Remarks

setVector() is not dependent on vector format because it is a macro instruction.

Operation differs between:

- a) setVector ((SVECTOR*)v, x, y, z)
- b) setVector ((VECTOR *)v, x, y, z)

See also:

setWH

Sets a value to the w,h members of primitive *p*.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	10/01/97

Syntax

setWH(**p*, *w*, *h*)

Arguments

p Primitive pointer.
w, *h* Width and height of primitive texture

Explanation

Specifies the w,h of primitives having w,h members.

Return value

None

Remarks

Since this is a macro, it does not depend on the primitive type. It cannot be used with primitives which do not have w,h members.

See also:

setXY0, setXY2, setXY3, setXY4

Set the $x0$ and $y0$ parameters of a primitive.

Set the $x2$ and $y2$ parameters of a primitive.

Set the $x3$ and $y3$ parameters of a primitive.

Set the $x4$ and $y4$ parameters of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	7/31/96

Syntax

setXY0(* p , $x0$, $y0$)

setXY2(* p , $x0$, $y0$, $x1$, $y1$)

setXY3(* p , $x0$, $y0$, $x1$, $y1$, $x2$, $y2$)

setXY4(* p , $x0$, $y0$, $x1$, $y1$, $x2$, $y2$, $x3$, $y3$)

Arguments

p Primitive pointer

x, y XY members of primitive

Explanation

These macros set the values for the XY members of the primitive.

Return value

None

Remarks

These are macros, so there is no dependence on the primitive type.

Remarks

See also:

setXYWH

Set XY member for 4-point primitives

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	6/22/98

Syntax

setXYWH *p, x0, y0, w, h*)

Arguments

p pointer to primitive

x0, y0 upper left point of primitive

w, h width and height of primitive

Explanation

The coordinates of the rectangle defined by (*x0*, *y0*)-(*x0* + *w*, *y0* + *h*) in a diagonal, are set in members (*x0*,*y0*)..*(x3, y3)* of the primitive.

Return value

None

Remarks

Since this is a macro, it does not depend on the primitive type.

See also:

Chapter 8: Basic Geometry Library

Table of Contents

Structures	
CRVECTOR3	8-5
CRVECTOR4	8-6
CVECTOR	8-7
DIVPOLYGON3	8-8
DIVPOLYGON4	8-9
DVECTOR	8-10
EVECTOR	8-11
MATRIX	8-12
POL3	8-14
POL4	8-15
QMESH	8-16
RVECTOR	8-17
SPOL	8-18
SVECTOR	8-19
TMESH	8-20
VECTOR	8-21
Functions	
ApplyMatrix	8-22
ApplyMatrixLV	8-23
ApplyMatrixSV	8-24
ApplyRotMatrix	8-25
ApplyRotMatrixLV	8-26
ApplyTransposeMatrixLV	8-27
AverageZ3	8-28
AverageZ4	8-29
catan	8-30
ccos	8-31
Clip3F	8-32
Clip3FP	8-33
Clip3FT	8-34
Clip3FTP	8-35
Clip3G	8-36
Clip3GP	8-37
Clip3GT	8-38
Clip3GTP	8-39
Clip4F	8-40
Clip4FP	8-41
Clip4FT	8-42
Clip4FTP	8-43
Clip4G	8-44
Clip4GP	8-45
Clip4GT	8-46
Clip4GTP	8-47
cln	8-48
ColorCol	8-49
ColorDpq	8-50
ColorMatCol	8-51
ColorMatDpq	8-52
CompMatrix	8-53
CompMatrixLV	8-54
csin	8-55
csqrt	8-56
DivideF3	8-57

DivideF4	8-58
DivideFT3	8-59
DivideFT4	8-60
DivideG3	8-61
DivideG4	8-62
DivideGT3	8-63
DivideGT4	8-64
DpqColor	8-65
DpqColor3	8-66
DpqColorLight	8-67
EigenMatrix	8-68
gteMIMefunc	8-69
InitClip	8-70
InitGeom	8-71
Intpl	8-72
InvSquareRoot	8-73
IsldMatrix	8-74
LightColor	8-75
LoadAverage0	8-76
LoadAverage12	8-77
LoadAverageByte	8-78
LoadAverageCol	8-79
LoadAverageShort0	8-80
LoadAverageShort12	8-81
LocalLight	8-82
Lzc	8-83
MatrixNormal	8-84
MatrixNormal_0	8-85
MatrixNormal_1	8-86
MatrixNormal_2	8-87
MulMatrix	8-88
MulMatrix0	8-89
MulMatrix2	8-90
MulRotMatrix	8-91
MulRotMatrix0	8-92
NormalClip	8-93
NormalColor	8-94
NormalColor_nom	8-95
NormalColor3	8-96
NormalColor3_nom	8-97
NormalColorCol	8-98
NormalColorCol_nom	8-99
NormalColorCol3	8-100
NormalColorCol3_nom	8-101
NormalColorDpq	8-102
NormalColorDpq_nom	8-103
NormalColorDpq3	8-104
NormalColorDpq3_nom	8-105
otz2p	8-106
OuterProduct0	8-107
OuterProduct12	8-108
p2otz	8-109
pers_map	8-110
PhongLine	8-111
PopMatrix	8-112
PushMatrix	8-113
ratan2	8-114

rcos	8-115
RCpolyF3	8-116
RCpolyF4	8-117
RCpolyFT3	8-118
RCpolyFT4	8-119
RCpolyG3	8-120
RCpolyG4	8-121
RCpolyGT3	8-122
RCpolyGT4	8-123
ReadColorMatrix	8-124
ReadGeomOffset	8-125
ReadGeomScreen	8-126
ReadLightMatrix	8-127
ReadRGBfifo	8-128
ReadRotMatrix	8-129
ReadSXSyfifo	8-130
ReadSZfifo3	8-131
ReadSZfifo4	8-132
RotAverage3	8-133
RotAverage3_nom	8-134
RotAverage4	8-135
RotAverageNclip3	8-136
RotAverageNclip3_nom	8-137
RotAverageNclip4	8-138
RotAverageNclipColorCol3	8-139
RotAverageNclipColorCol3_nom	8-141
RotAverageNclipColorDpq3	8-143
RotAverageNclipColorDpq3_nom	8-145
RotColorDpq	8-147
RotColorDpq_nom	8-148
RotColorDpq3	8-149
RotColorDpq3_nom	8-150
RotColorMatDpq	8-151
RotMatrix...	8-153
RotMatrix_gte	8-155
RotMatrixC	8-156
RotMatrixX	8-157
RotMatrixY	8-158
RotMatrixYXZ_gte	8-159
RotMatrixZ	8-160
RotMatrixZYX_gte	8-161
RotMeshH	8-162
RotMeshPrimQ_T	8-163
RotMeshPrimR_...	8-165
RotMeshPrimS_...	8-167
RotNclip3	8-169
RotNclip3_nom	8-170
RotNclip4	8-171
RotPMD_...	8-172
RotPMD_SV_...	8-173
RotRMD_...	8-174
RotRMD_SV_...	8-176
RotSMD_...	8-178
RotSMD_SV_...	8-180
RotTrans	8-182
RotTrans_nom	8-183
RotTransPers	8-184

RotTransPers_nom	8-185
RotTransPers3	8-186
RotTransPers3_nom	8-187
RotTransPers3N	8-188
RotTransPers4	8-189
RotTransPers4_nom	8-190
RotTransPersN	8-191
RotTransSV	8-192
rsin	8-193
ScaleMatrix	8-194
ScaleMatrixL	8-195
SetBackColor	8-196
SetColorMatrix	8-197
SetFarColor	8-198
SetFogFar	8-199
SetFogNear	8-200
SetFogNearFar	8-201
SetGeomOffset	8-202
SetGeomScreen	8-203
SetLightMatrix	8-204
SetMulMatrix	8-205
SetMulRotMatrix	8-206
SetRGBcd	8-207
SetRotMatrix	8-208
SetTransMatrix	8-209
Square0	8-210
Square12	8-211
SquareRoot0	8-212
SquareRoot12	8-213
SquareSL0	8-214
SquareSL12	8-215
SquareSS0	8-216
SquareSS12	8-217
SubPol3	8-218
SubPol4	8-220
TransMatrix	8-222
TransposeMatrix	8-223
TransRotPers	8-224
TransRotPers3	8-225
TransRot_32	8-226
VectorNormal	8-227
VectorNormalS	8-228
VectorNormalSS	8-229

CRVECTOR3

Triangular recursive vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    RVECTOR r01, r12, r20;
    RVECTOR *r0, *r1, *r2;
    unsigned long *rtn;
} CRVECTOR3;
```

Members

r01, *r12*, *r20* Division vertex vector data
r0, *r1*, *r2* Pointer to division vector data
rtn Pointer to return address for assembler

Explanation

Remarks

See also:

CRVECTOR4

Quadrilateral recursive vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    RVECTOR r01, r02, r31, r32, rc;
    RVECTOR *r0, *r1, *r2, *r3;
    unsigned long *rtn;
} CRVECTOR4;
```

Members

<i>r01</i> , <i>r02</i> , <i>r31</i> , <i>r32</i> , <i>rc</i>	Division vertex vector data
<i>r0</i> , <i>r1</i> , <i>r2</i> , <i>r3</i>	Pointer to division vertex vector data
<i>rtn</i>	Pointer to return address for assembler

Explanation

Remarks

See also:

CVECTOR

Character vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Structure

```
typedef struct {
    unsigned char r, g, b, cd;
};
```

Members

r, g, b Color palette
cd GPU code

Explanation

Remarks

See also:

DIVPOLYGON3

Triangular division buffer.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    unsigned long ndiv;
    unsigned long pih, piv;
    unsigned short clut, tpage;
    CVECTOR rgbc;
    unsigned long *ot;
    RVECTOR r0, r1, r2;
    CRVECTOR3 cr[5];
} DIVPOLYGON3;
```

Members

<i>ndiv</i>	Number of divisions
<i>pih, piv</i>	Clip area specification (display screen resolution)
<i>clut</i>	CLUT
<i>tpage</i>	Texture page
<i>rgbc</i>	Code + RGB color
<i>ot</i>	Pointer to OT
<i>r0, r1, r2</i>	Division vertex vector data
<i>cr</i>	Triangular recursive vector data

Explanation

Remarks

See also:

DIVPOLYGON4

Quadrilateral recursive vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    unsigned long ndiv;
    unsigned long pih, piv;
    unsigned short clut, tpage;
    CVECTOR rgbc;
    unsigned long *ot;
    RVECTOR r0, r1, r2, r3;
    CRVECTOR4 cr[5];
} DIVPOLYGON4;
```

Members

<i>ndiv</i>	Number of divisions
<i>pih, piv</i>	Clip area specification (display screen's resolution)
<i>clut</i>	CLUT
<i>tpage</i>	Texture page
<i>rgbc</i>	Code + RGB color
<i>ot</i>	Pointer to OT
<i>r0, r1, r2, r3</i>	Division vertex vector data
<i>cr</i>	Quadrilateral recursive vector data

Explanation

Remarks

See also:

DVECTOR

2D vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    short vx, vy;
} DVECTOR;
```

Members

vx, *vy* Vector coordinates

Explanation

Remarks

See also:

EVECTOR

Clip vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    SVECTOR v;
    VECTOR sxyz;
    DVECTOR sxy;
    CVECTOR rgb;
    short txuv, pad;
    long chx, chy;
} EVECTOR;
```

Members

<i>v</i>	Local object 3D vertex
<i>sxyz</i>	Screen 3D vertex
<i>sxy</i>	Screen 2D vertex
<i>rgb</i>	Color palette
<i>txuv, pad</i>	Texture mapping data
<i>chx, chy</i>	Clip area data

Explanation

Remarks

See also:

MATRIX

Matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Structure

```
struct MATRIX {
    short m[3][3];
    long t[3];
};
```

Members

m 3 x 3 matrix coefficient value
t Parallel transfer volume

Explanation

Specifies each component on the MATRIX *m*[*i*][*j*]. Specifies the transfer volume after conversion on the MATRIX *t* [*i*]. Pay attention to the differing word lengths on *m* and *t*.

The GTE essentially performs the following multiply and accumulate calculations from the MATRIX structure.

- a) RotTrans system function (function group which does not perform coordinate conversion). Performs only basic matrix calculations and vector addition.

MATRIX *m*

SVECTOR *xi*

SVECTOR *xo*

$$\begin{bmatrix} xo.vx \\ xo.vy \\ xo.vz \end{bmatrix} = \begin{bmatrix} m.m[0][0] & m.m[0][1] & m.m[0][2] \\ m.m[1][0] & m.m[1][1] & m.m[1][2] \\ m.m[2][0] & m.m[2][1] & m.m[2][2] \end{bmatrix} \begin{bmatrix} xi.vx \\ xi.vy \\ xi.vz \end{bmatrix} + \begin{bmatrix} m.t[0] \\ m.t[1] \\ m.t[2] \end{bmatrix}$$

- b) RotTransPers system function (function group which performs coordinate conversion). In addition to the (a) calculation, perspective conversion (division by *z*) is performed at the same time.

MATRIX *m*

SVECTOR *xi*

SVECTOR *xo*

SVECTOR *x2*

long *h*

$$\begin{bmatrix} xo.vx \\ xo.vy \\ xo.vz \end{bmatrix} = \begin{bmatrix} m.m[0][0] & m.m[0][1] & m.m[0][2] \\ m.m[1][0] & m.m[1][1] & m.m[1][2] \\ m.m[2][0] & m.m[2][1] & m.m[2][2] \end{bmatrix} \begin{bmatrix} xi.vx \\ xi.vy \\ xi.vz \end{bmatrix} + \begin{bmatrix} m.t[0] \\ m.t[1] \\ m.t[2] \end{bmatrix}$$

$$x2.vx = (h * xo.vx) / xo.vz$$

$$x2.vy = (h * xo.vy) / xo.vz$$

Remarks

See also:

POL3

Triangle polygon.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Structure

```
struct POL3 {
    short sxy[3][2];
    short sz[3][2];
    short uv[3][2];
    short rgb[3][3];
    short code;
};
```

Members

<i>sxy</i>	Screen coordinates
<i>sz</i>	Screen coordinates
<i>uv</i>	Texture coordinates
<i>rgb</i>	RGB value
<i>code</i>	Code

Table 8-1

Code	Values
F3	1
TF3	2
G3	3
TG3	4

Explanation

Remarks

See also:

POL4

Four-sided polygon.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Structure

```
struct POL4 {
    short sxy[4][2];
    short sz[4][2];
    short uv[4][2];
    short rgb[4][3];
    short code;
};
```

Members

sxy Screen coordinates
sz Screen coordinates
uv Texture coordinates
rgb RGB value
code Code

Table 8-2

Code	Values
F4	5
TF4	6
G4	7
TG4	8

Explanation

Remarks

See also:

QMESH

Quadrilateral mesh

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.3	6/22/98

Structure

```
typedef struct {
    SVECTOR *v;
    SVECTOR *n;
    SVECTOR *u;
    CVECTOR *c;
    u_long lenv;
    u_long lenh;
}QMESH;
```

Members

v	pointer to shared vertex coordinates array
n	pointer to shared normal array
u	pointer to shared texture coordinates array
c	pointer to shared color data array
lenv	vertex length (horizontal)
lenh	vertex length (vertical)

RVECTOR

Division vertex vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    SVECTOR v;
    unsigned char uv[2];
    unsigned short pad;
    CVECTOR c;
    DVECTOR sxy;
    unsigned long sz;
} RVECTOR;
```

Members

<i>v</i>	Local object 3D vertex
<i>uv</i>	Texture mapping data
<i>c</i>	Vertex color palette
<i>sxy</i>	Screen 2D vertex
<i>sz</i>	Clip Z-data

Explanation

Remarks

See also:

SPOL

Vertex information.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Structure

```
struct SPOL {
    short xy[3];
    short uv[2];
    short rgb[3];
};
```

Members

xy XY coordinates
uv UV coordinates
rgb RGB value

Explanation

Remarks

See also:

SVECTOR

Short vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Structure

```
struct SVECTOR {
    short vx, vy;
    short vz, pad;
};
```

Members

<i>vx</i> , <i>vy</i> , <i>vz</i>	Vector coordinates
<i>pad</i>	System reserved

Explanation

Remarks

See also:

TMESH

Triangle mesh.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Structure

```
struct TMESH {
    SVECTOR *v;
    SVECTOR *n;
    SVECTOR *u;
    CVECTOR *c;
    unsigned long len;
};
```

Members

v Pointer to vertex string
n Pointer to normal string
u Pointer to texture string
c Pointer to RGB string
len Mesh length

Explanation

Remarks

See also:

VECTOR

Vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Structure

```
struct VECTOR {
    long vx, vy, vz, pad;
};
```

Members

<i>vx, vy, vz</i>	Vector coordinates
<i>pad</i>	System reserved

Explanation

Remarks

See also:

ApplyMatrix

Multiply a vector by a matrix. The vector is in effect rotated and then translated.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
VECTOR *ApplyMatrix(*m, *v0, *v1)
MATRIX *m;
SVECTOR *v0;
VECTOR *v1;
```

Arguments

m Pointer to matrix to be multiplied (input)
v0 Pointer to short vector (input)
v1 Pointer to vector (output)

Explanation

This function multiplies the matrix *m* by the short vector *v0* beginning with the rightmost end. The result is saved in the vector *v1*.

The argument format is as follows:

m -> *m* [*i*] [*j*] : (1, 3, 12)

v0 -> *vx*, *vy*, *vz* : (1, 15, 0)

v1 -> *vx*, *vy*, *vz* : (1, 31, 0)

Return value

This function returns *v1*.

Remarks

The function destroys the constant rotation matrix.

See also:

ApplyMatrixLV

Multiply a vector by a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
VECTOR *ApplyMatrixLV(*m, *v0, *v1)
MATRIX *m;
VECTOR *v0, *v1;
```

Arguments

m Pointer to matrix to be multiplied (input)
v0 Pointer to vector (input)
v1 Pointer to vector (output)

Explanation

This function destroys the rotation matrix.

This function multiplies matrix *m* by vector *v0* beginning from the rightmost end. The result is saved in vector *v1*.

```
m -> m [i] [j]        : (1, 3, 12)
v0 -> vx, vy, vz      : (1, 31, 0)
v1 -> vx, vy, vz      : (1, 31, 0)
```

Return value

v1

Remarks

This function destroys the rotation matrix.

ApplyMatrixLV is a 16 x 32 bit multiplier which uses the GTE.

See also:

ApplyMatrixSV

Multiply a vector by a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

SVECTOR ***ApplyMatrixSV**(**m*, **v0*, **v1*)

MATRIX **m*;

SVECTOR **v0*, **v1*;

Arguments

m Pointer to matrix to be multiplied (input)

v0 Pointer to short vector (input)

v1 Pointer to short vector (output)

Explanation

This function multiplies matrix *m* by short vector *v0* beginning at the rightmost end. The result is saved in the short vector *v1*.

m -> *m* [*i*] [*j*] : (1, 3, 12)

v0 -> *vx*, *vy*, *vz* : (1, 15, 0)

v1 -> *vx*, *vy*, *vz* : (1, 15, 0)

Return value

v1

Remarks

This function destroys the rotation matrix.

See also:

ApplyRotMatrix

Multiply a vector by a constant rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
VECTOR *ApplyRotMatrix(*v0, *v1)
SVECTOR *v0;
VECTOR *v1;
```

Arguments

v0 Pointer to short vector (input)
v1 Pointer to vector (output)

Explanation

This function multiplies a constant rotation matrix by short vector *v0* beginning at the rightmost end. The result is saved in vector *v1*.

v0 -> vx, vy, vz : (1, 15, 0)

v1 -> vx, vy, vz : (1, 31, 0)

Return value

v1

Remarks

See also:

ApplyRotMatrixLV

Multiplies a vector by a constant rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	10/23/96

Syntax

```
VECTOR *ApplyRotMatrixLV(*v0, *v1)
VECTOR *v0;
VECTOR *v1;
```

Arguments

v0 Pointer to long vector (input)
v1 Pointer to vector (output)

Explanation

This function multiplies a constant rotation matrix by long vector *v0* beginning at the rightmost end. The result is saved in vector *v1*.

v0 -> vx, vy, vz : (1, 31, 0)

v1 -> vx, vy, vz : (1, 31, 0)

Return value

v1

Remarks

See also:

ApplyTransposeMatrixLV

Multiplies a vector by a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
VECTOR *ApplyTransposeMatrixLV(*m, *v0, *v1)
MATRIX *m;
VECTOR *v0;
VECTOR *v1;
```

Arguments

m Pointer to matrix to be multiplied
v0 Pointer to vector (input)
v1 Pointer to vector (output)

Explantation

Return value

Remarks

See also:

AverageZ3

Average of three values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

long AverageZ3(sz0, sz1, sz2)

long sz0, sz1, sz2;

Arguments

sz0, sz1, sz2 Input values

Explanation

This function calculates an average of three values sz0, sz1, and sz2.

The argument format is as follows:

sz0, sz1, sz2 : (0, 16, 0)

Return value : (0, 16, 0)

Return value

Average of 1/4 of three values sz0, sz1, and sz2.

Remarks

See also:

AverageZ4

Average of four values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long AverageZ4(sz0, sz1, sz2, sz3)
long sz0, sz1, sz2, sz3;
```

Arguments

sz0, *sz1*, *sz2*, *sz3* Input values

Explanation

This function calculates an average of four values *sz0*, *sz1*, *sz2*, and *sz3*.

The argument format is as follows:

sz0, *sz1*, *sz2*, *sz3* : (0, 16, 0)

Return value : (0, 16, 0)

Return value

Average of 1/4 of four values *sz0*, *sz1*, *sz2*, and *sz3*.

Remarks

See also:

catan

Computes the arctangent of angle(*a*) within 180 degrees.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
int catan(a)
```

```
int a;
```

Arguments

a Value

Explanation

This function uses PlayStation format (where 4096 = 360 degrees = 2 π) to find the arctan (between -90 and +90 degrees, $-\pi/2 \dots \pi/2$) of *a*.

The argument format is as follows:

a : (1, 19, 12)

Return value : (1, 19, 12)

Return value

atan (*a*)

Remarks

See also:

ccos

Computes the cosine of angle a .

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
int ccos(a)
int a;
```

Arguments

a Angle (in PlayStation format)

Explanation

Finds the cosine function of the angle (in PlayStation format) ($4096 = 360 \text{ degrees} = 2 \text{ pi}$) using fixed point math (where $4096 = 1.0$).

The speed and size of `rcos()` and `ccos()` differ as shown below.

	Speed	Size
rcos	fast	large
ccos	slow	small

The argument format is as follows:

a : PlayStation format ($4096 = 360 \text{ degrees} = 2 \text{ pi}$)

Return value : (1, 19, 12)

Return value

$\cos(a)$

Remarks

See also: `rcos()`

Clip3F

Three-vertex clipping (without perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip3F(*v0, *v1, *v2, **evmx)
SVECTOR *v0, *v1, *v2;
EVECTOR **evmx;
```

Arguments

v0, *v1*, *v2* Pointer to vertex coordinate vector (input)
evmx Pointer arrays for clip vector data (20,output)

Explanation

This function clips six surfaces of a triangle having vertices *v0*, *v1*, and *v2*, and defined by InitClip(), and angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

evmx[*i*] -> *v* Local Object 3D Vertex
evmx[*i*] -> *sxyz* Screen 3D Vertex
evmx[*i*] -> *chx* $chx = vz \times (hw/2)/h$
evmx[*i*] -> *chy* $chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip3FP

Three-vertex (triangle) clipping (with perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip3FP(*v0, *v1, *v2, **evmx)
SVECTOR *v0, *v1, *v2;
EVECTOR **evmx;
```

Arguments

v0, v1, v2 Pointer to vertex coordinate vector (input)
evmx Pointer arrays (for clip vector data (20, output))

Explanation

This function clips six surfaces of a triangle having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

evmx[i] -> *v* Local Object 3D Vertex
evmx[i] -> *sxyz* Screen 3D Vertex
evmx[i] -> *sxyz.pad* FOG effect interpolation value (p)
evmx[i] -> *sxy* Screen 2D Vertex
evmx[i] -> *chx* $chx = vz \times (hw/2)/h$
evmx[i] -> *chy* $chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip3FT

Three-vertex clipping (without perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip3FT(*v0, *v1, *v2, *uv0, *uv1, *uv2, **evmx)
SVECTOR *v0, *v1, *v1;
short *uv0, *uv1, *uv2;
EVECTOR **evmx;
```

Arguments

v0, v1, v2 Pointer to vertex coordinate vector (input)
uv0, uv1, uv2 Pointer to texture coordinate vector (input)
evmx Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a triangle having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

evmx[i] -> *v* Local Object 3D Vertex
evmx[i] -> *sxyz* Screen 3D Vertex
evmx[i] -> *txuv* Texture Mapping Vertex
evmx[i] -> *chx* $chx = vz \times (hw/2)/h$
evmx[i] -> *chy* $chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip3FTP

Three-vertex clipping (with perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip3FTP(*v0, *v1, *v2, *uv0, *uv1, *uv2, **evmx)
SVECTOR *v0, *v1, *v2;
short *uv0, *uv1, *uv2;
EVECTOR **evmx;
```

Arguments

v0, v1, v2 Pointer to vertex coordinate vector (input)
uv0, uv1, uv2 Pointer to texture coordinate vector (input)
evmx Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a triangle having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

evmx[i] -> *v* Object (Local) 3D Vertex
evmx[i] -> *sxyz* Screen 3D Vertex
evmx[i] -> *sxyz.pad* FOG effect interpolation value (p)
evmx[i] -> *sxy* Screen 2D Vertex
evmx[i] -> *txuv* Texture Mapping Data
evmx[i] -> *chx* $chx = vz \times (hw/2)/h$
evmx[i] -> *chy* $chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip3G

Three-vertex clipping (without perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip3G(*v0, *v1, *v2, *rgb0, *rgb1, *rgb2, **evmx)
SVECTOR *v0, *v1, *v2;
CVECTOR *rgb0, *rgb1, *rgb2;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vector (input)
<i>rgb0, rgb1, rgb2</i>	Pointer to vertex color data (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a triangle having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

<i>evmx[i] -> v</i>	Local Object 3D Vertex
<i>evmx[i] -> xyz</i>	Screen 3D Vertex
<i>evmx[i] -> rgb</i>	Vertex Color Data
<i>evmx[i] -> chx</i>	$chx = vz \times (hw/2)/h$
<i>evmx[i] -> chy</i>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip3GP

Three-vertex clipping (with perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip3GP(*v0, *v1, *v2, *rgb0, *rgb1, *rgb2, **evmx)
SVECTOR *v0, *v1, *v2;
CVECTOR *rgb0, *rgb1, *rgb2;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vector (input)
<i>rgb0, rgb1, rgb2</i>	Pointer to vertex color data (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a triangle having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

<i>evmx[i] -> v</i>	Local Object3D Vertex
<i>evmx[i] -> xyz</i>	Screen 3D Vertex
<i>evmx[i] -> xyz.pad</i>	FOG effect interpolation value (p)
<i>evmx[i] -> sxy</i>	Screen 2D Vertex
<i>evmx[i] -> rgb</i>	Vertex Color Data
<i>evmx[i] -> chx</i>	$chx = vz \times (hw/2)/h$
<i>evmx[i] -> chy</i>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip3GT

Three-vertex clipping (without perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip3GT(*v0, *v1, *v2, *uv0, *uv1, *uv2, *rgb0, *rgb1, *rgb2, **evmx)
SVECTOR *v0, *v1, *v2;
short *uv0, *uv1, *uv2;
CVECTOR *rgb0, *rgb1, *rgb2;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vector (input)
<i>uv0, uv1, uv2</i>	Pointer to texture coordinate vector (input)
<i>rgb0, rgb1, rgb2</i>	Pointer to vertex color data (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a triangle having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

<i>evmx[i] -> v</i>	Local Object3D Vertex
<i>evmx[i] -> xyz</i>	Screen 3D Vertex
<i>evmx[i] -> rgb</i>	Vertex Color Data
<i>evmx[i] -> txuv</i>	Texture Mapping Data
<i>evmx[i] -> chx</i>	$chx = vz \times (hw/2)/h$
<i>evmx[i] -> chy</i>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip3GTP

Three-vertex clipping (with perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip3GTP(*v0, *v1, *v2, *uv0, *uv1, *uv2, *rgb0, *rgb1, *rgb2, **evmx)
SVECTOR *v0, *v1, *v2;
short *uv0, *uv1, *uv2;
CVECTOR *rgb0, *rgb1, *rgb2;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vector (input)
<i>uv0, uv1, uv2</i>	Pointer to texture coordinate vector (input)
<i>rgb0, rgb1, rgb2</i>	Pointer to vertex color data (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a triangle having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

<i>evmx[i] -> v</i>	Local Object 3D Vertex
<i>evmx[i] -> xyz</i>	Screen 3D Vertex
<i>evmx[i] -> xyz.pad</i>	Fog effect interpolation value (p)
<i>evmx[i] -> xy</i>	Screen 2D Vertex
<i>evmx[i] -> rgb</i>	Vertex Color Data
<i>evmx[i] -> txuv</i>	Texture Mapping Data
<i>evmx[i] -> chx</i>	$chx = vz \times (hw/2)/h$
<i>evmx[i] -> chy</i>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip4F

Four-vertex clipping (without perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip4F(*v0, *v1, *v2, *v3, **evmx)
SVECTOR *v0, *v1, *v2, *v3;
EVECTOR **evmx;
```

Arguments

v0, v1, v2, v3 Pointer to vertex coordinate vector (input)
evmx Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a quadrilateral (linked triangle) having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

evmx[i] -> *v* Local Object 3D Vertex
evmx[i] -> *sxyz* Screen 3D Vertex
evmx[i] -> *chx* $chx = vz \times (hw/2)/h$
evmx[i] -> *chy* $chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip4FP

Four-vertex clipping (with perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip4FP(*v0, *v1, *v2, *v3, **evmx)
SVECTOR *v0, *v1, *v2, *v3;
EVECTOR **evmx;
```

Arguments

v0, v1, v2, v3 Pointer to vertex coordinate vector (input)
evmx Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a quadrilateral (linked triangle) having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

evmx[i] -> *v* Local Object 3D Vertex
evmx[i] -> *sxyz* Screen 3D Vertex
evmx[i] -> *sxyz.pad* FOG effect interpolation value (p)
evmx[i] -> *sxy* Screen 2D Vertex
evmx[i] -> *chx* $chx = vz \times (hw/2)/h$
evmx[i] -> *chy* $chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip4FT

Four-vertex clipping (without perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip4FT(*v0, *v1, *v2, *v3, *uv0, *uv1, *uv2, *uv3, **evmx)
SVECTOR *v0, *v1, *v2, *v3;
short *uv0, *uv1, *uv2, *uv3;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vector (input)
<i>uv0, uv1, uv2, uv3</i>	Pointer to texture coordinate vector (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a quadrilateral (linked triangle) having vertices *v0*, *v1*, and *v2*, and defined by *InitClip()*. Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

<i>evmx[i]</i> -> <i>v</i>	Local Object 3D Vertex
<i>evmx[i]</i> -> <i>sxyz</i>	Screen 3D Vertex
<i>evmx[i]</i> -> <i>txuv</i>	Texture Mapping Data
<i>evmx[i]</i> -> <i>chx</i>	$chx = vz \times (hw/2)/h$
<i>evmx[i]</i> -> <i>chy</i>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip4FTP

Four-vertex clipping (with perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	5/22/97

Syntax

```
long Clip4FTP(*v0, *v1, *v2, *v3, *uv0, *uv1, *uv2, *uv3, **evmx)
SVECTOR *v0, *v1, *v2, *v3;
short *uv0, *uv1, *uv2, *uv3;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vector (input)
<i>uv0, uv1, uv2, uv3</i>	Pointer to texture coordinate vector (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a quadrilateral (linked triangle) having vertices *v0, v1, v2* and *v3* and defined by `InitClip()`. It outputs clipped polygon information with the clipping vector information pointer array and the number of vertices

Effective output clip vector data:

<code>evmx[i] -> v</code>	Local Object 3D Vertex
<code>evmx[i] -> xyz</code>	Screen 3D Vertex
<code>evmx[i] -> xyz.pad</code>	Interpolation value (p) FOG effect
<code>evmx[i] -> xy</code>	Screen 2D Vertex
<code>evmx[i] -> txuv</code>	Texture Mapping Data
<code>evmx[i] -> chx</code>	$chx = vz \times (hw/2)/h$
<code>evmx[i] -> chy</code>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip4G

Four-vertex clipping (without perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip4G(*v0, *v1, *v2, *v3, *rgb0, *rgb1, *rgb2, *rgb3, **evmx)
SVECTOR *v0, *v1, *v2, *v3;
CVECTOR *rgb0, *rgb1, *rgb2, *rgb3;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vector (input)
<i>rgb0, rgb1, rgb2, rgb3</i>	Pointer to vertex color data (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a quadrilateral (linked triangle) having vertices *v0*, *v1*, and *v2*, and defined by *InitClip()*. Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

<i>evmx[i] -> v</i>	Local Object 3D Vertex
<i>evmx[i] -> xyz</i>	Screen 3D Vertex
<i>evmx[i] -> rgb</i>	Vertex Color Data
<i>evmx[i] -> chx</i>	$chx = vz \times (hw/2)/h$
<i>evmx[i] -> chy</i>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip4GP

Four-vertex clipping (with perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip4GP(*v0, *v1, *v2, *v3, *rgb0, *rgb1, *rgb2, *rgb3, **evmx)
SVECTOR *v0, *v1, *v2, *v3;
CVECTOR *rgb0, *rgb1, *rgb2, *rgb3;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vector (input)
<i>rgb0, rgb1, rgb2, rgb3</i>	Pointer to vertex color data (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a quadrilateral (linked triangle) having vertices *v0*, *v1*, and *v2*, and defined by `InitClip()`. Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

<i>evmx[i] -> v</i>	Local Object 3D Vertex
<i>evmx[i] -> xyz</i>	Screen 3D Vertex
<i>evmx[i] -> xyz.pad</i>	interpolation value (p) for FOG effect
<i>evmx[i] -> sxy</i>	Screen 2D Vertex
<i>evmx[i] -> rgb</i>	Vertex Color Data
<i>evmx[i] -> chx</i>	$chx = vz \times (hw/2)/h$
<i>evmx[i] -> chy</i>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip4GT

Four-vertex clipping (without perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip4GT(*v0, *v1, *v2, *v3, *uv0, *uv1, *uv2, *uv3, *rgb0, *rgb1, *rgb2, *rgb3, **evmx)
SVECTOR *v0, *v1, *v2, *v3;
short *uv0, *uv1, *uv2, *uv3;
CVECTOR *rgb0, *rgb1, *rgb2, *rgb3;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vector (input)
<i>uv0, uv1, uv2, uv3</i>	Pointer to texture coordinate vector (input)
<i>rgb0, rgb1, rgb2, rgb3</i>	Pointer to vertex color data (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a quadrilateral (linked triangle) having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

<i>evmx[i] -> v</i>	Local Object 3D Vertex
<i>evmx[i] -> xyz</i>	Screen 3D Vertex
<i>evmx[i] -> rgb</i>	Vertex Color Data
<i>evmx[i] -> txuv</i>	Texture Mapping Data
<i>evmx[i] -> chx</i>	$chx = vz \times (hw/2)/h$
<i>evmx[i] -> chy</i>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

Clip4GTP

Four-vertex clipping (with perspective transformation).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long Clip4GTP(*v0, *v1, *v2, *v3, *uv0, *uv1, *uv2, *uv3, *rgb0, *rgb1, *rgb2, *rgb3, **evmx)
SVECTOR *v0, *v1, *v2, *v3;
short *uv0, *uv1, *uv2, *uv3;
CVECTOR *rgb0, *rgb1, *rgb2, *rgb3;
EVECTOR **evmx;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vector (input)
<i>uv0, uv1, uv2, uv3</i>	Pointer to texture coordinate vector (input)
<i>rgb0, rgb1, rgb2, rgb3</i>	Pointer to vertex color data (input)
<i>evmx</i>	Pointer arrays for clip vector data (20, output)

Explanation

This function clips six surfaces of a quadrilateral (linked triangle) having vertices *v0*, *v1*, and *v2*, and defined by InitClip(). Angle information is stored in *evmx*. The output number of vertices is returned.

Effective output clip vector data:

<i>evmx[i] -> v</i>	Local Object 3D Vertex
<i>evmx[i] -> xyz</i>	Screen 3D Vertex
<i>evmx[i] -> xyz.pad</i>	Fog effect interpolation value (p)
<i>evmx[i] -> sxy</i>	Screen 2D Vertex
<i>evmx[i] -> rgb</i>	Vertex Color Data
<i>evmx[i] -> txuv</i>	Texture Mapping Data
<i>evmx[i] -> chx</i>	$chx = vz \times (hw/2)/h$
<i>evmx[i] -> chy</i>	$chy = vz \times (vw/2)/h$

This function reserves the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

n: output number of vertices

Remarks

See also:

cln

C logarithm function.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
int cln(a)
```

```
int a;
```

Arguments

a Value

Explanation

This function uses fixed point math (where 4096 = 1.0) to find the fixed point natural logarithm.

Argument format is as follows:

a : (1, 19, 12)

Return value : (1, 19, 12)

Return value

ln (*a*)

Remarks

See also:

ColorCol

Finds a local color from a local light vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void ColorCol(*v0, *v1, *v2)
VECTOR *v0;
CVECTOR *v1;
CVECTOR *v2;
```

Arguments

v0 Pointer to local light vector (input)
v1 Pointer to primary color vector (input)
v2 Pointer to color vector (output)

Explanation

This function calculates the following:

$LC = BK + LCM \times v0$

$v2 = v1 \times LC$ (product of multiplication)

The argument format is as follows:

v0 -> vx, vy, vz : (1, 19, 12)

v1 -> r, g, b : (0, 8, 0)

v2 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

ColorDpq

Finds a local color from a local light vector, and performs depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void ColorDpq(*v0, *v1, p, *v2)
VECTOR *v0;
CVECTOR *v1;
long p;
CVECTOR *v2;
```

Arguments

v0 Pointer to local light vector (input)
v1 Pointer to primary color vector (input)
p Interpolation value (input)
v2 Pointer to color vector (output)

Explanation

This function calculates the following:

$$LC = BK + LCM \times v0$$

$$v2 = (1-p) \times v1 \times LC + p \times FC$$

where $v1 \times LC$ is the product of separate multiplication.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 19, 12)
v1 -> r, g, b : (0, 8, 0)
p : (0, 20, 12)
v2 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

ColorMatCol

Finds a color.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	5/22/97

Syntax

```
void ColorMatCol(*v0, *v1, *v2, matc)
SVECTOR *v0;
CVECTOR *v1;
CVECTOR *v2;
long matc;
```

Arguments

v0 Pointer to normal vector (input)
v1 Pointer to primary color vector (input)
v2 Pointer to color vector (output)
matc Material (output)

Explanation

This function performs the following calculations:

$LLV = LLM \times v0$

$LLV = LLV^{(2^{matc})}$

$LC = BK + LCM \times LLV$

$v2 = v1 \times LC$ (separate multiplications)

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)
v1 -> r, g, b : (0, 8, 0)
v2 -> r, g, b : (0, 8, 0)
matc : (0, 32, 0)

Return value

None

Remarks

See also:

ColorMatDpq

Finds a color and performs depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	5/22/97

Syntax

```
void ColorMatDpq(*v0, *v1, p, *v2, matc)
SVECTOR *v0;
CVECTOR *v1;
long p;
CVECTOR *v2;
long matc;
```

Arguments

v0 Pointer to normal vector (input)
p Interpolation value (output)
v2 Pointer to color vector (output)
matc Material (output)

Explanation

This function performs the following calculations:

$$LLV = LLM \times v0$$

$$LLV = LLV^{(2^{matc})}$$

$$LC = BK + LCM \times LLV$$

$$v2 = (1-p) \times v1 \times LC + p \times FC.$$

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)
v1 -> r, g, b : (0, 8, 0)
p : (0, 20, 12)
v2 -> r, g, b : (0, 8, 0)
matc : (0, 32, 0)

Return value

None

Remarks

See also:

CompMatrix

Make a composite coordinate transformation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
MATRIX *CompMatrix(*m0, *m1, *m2)
MATRIX *m0, *m1, *m2;
```

Arguments

m0, m1 Pointer to matrix (input)
m2 Pointer to matrix (output)

Explanation

This function makes a composite coordinate transformation matrix that includes parallel translation.

$$[m2 \rightarrow m] = [m0 \rightarrow m] \times [m1 \rightarrow m]$$

$$(m2 \rightarrow t) = [m0 \rightarrow m] \times (m1 \rightarrow t) + (m0 \rightarrow t)$$

However, the values of the elements of $m1 \rightarrow t$ should be in the range $(-2^{15}, 2^{15})$.

Argument format

m0 -> m [i] [j] : (1, 3, 12)

m0 -> t [i] : (1, 31, 0)

m1 -> m [i] [j] : (1, 3, 12)

m1 -> t [i] : (1, 15, 0)

m2 -> m [i] [j] : (1, 3, 12)

m2 -> t [i] : (1, 31, 0)

Return value

m2

Remarks

This function destroys a constant rotation matrix.

See also:

CompMatrixLV

Make a composite coordinate transformation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.6	10/23/96

Syntax

```
MATRIX *CompMatrix(*m0, *m1, *m2)
MATRIX *m0;
MATRIX *m1;
MATRIX *m2;
```

Arguments

m0, m1 Pointer to matrix (input)
m2 Pointer to matrix (output)

Explanation

This function makes a composite coordinate transformation matrix that includes parallel translation.

$[m2 \rightarrow m] = [m0 \rightarrow m] * [m1 \rightarrow m]$

$(m2 \rightarrow t) = [m0 \rightarrow m] * (m1 \rightarrow t) + (m0 \rightarrow t)$

Argument format

m0 -> m [i] [j] : (1, 3, 12)

m0 -> t [i] : (1, 31, 0)

m1 -> m [i] [j] : (1, 3, 12)

m1 -> t [i] : (1, 31, 0)

m2 -> m [i] [j] : (1, 3, 12)

m2 -> t [i] : (1, 31, 0)

Return value

m2

Remarks

This function destroys a rotation matrix.

See also:

csin

C sine function.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	10/01/97

Syntax

int csin(*a*)

int *a*;

Arguments

a Angle (in PlayStation format)

Explanation

Find the sine function of the angle (in PlayStation format) ($4096 = 360 \text{ degrees} = 2\pi$) using fixed point math (where $4096 = 1.0$).

The speed and size of `rsin()` and `csin()` differ as shown below.

	Speed	Size
rsin	fast	large
csin	slow	small

The argument format is as follows:

a : PlayStation format ($4096 = 360 \text{ degrees} = 2\pi$)

Return value : (1, 19, 12)

Return value

$\sin(a)$

Remarks

See also: `rsin()`

csqrt

C square root function.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
int csqrt(a)
```

```
int a;
```

Arguments

a Value

Explanation

This function uses fixed point math (where 4096 = 1.0) to find the fixed point square root.

This function is the same as the SquareRoot12 function except that it requires a smaller table memory area.

The argument format is as follows:

a : (1, 19, 12)

Return value : (1, 19, 12)

Return value

sqrt (*a*)

Remarks

See also:

DivideF3

Division of flat triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *DivideF3(*v0, *v1, *v2, *rgbc, *s, *ot, *divp)
SVECTOR *v0, *v1, *v2;
CVECTOR *rgbc;
POLY_F3 *s;
u_long *ot;
DIVPOLYGON3 *divp;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vectors (input)
<i>rgbc</i>	Pointer to color vector + code (input)
<i>s</i>	Pointer to GPU packet buffer address
<i>ot</i>	Pointer to OT entry
<i>divp</i>	Pointer to division work area (input)

Explanation

This is a flat triangle division program. It divides a flat triangle (POLY_F3) indicated by the vertex coordinate vectors and color vector based on the *divp* -> ndiv value, and registers the result to OT.

The *divp* -> ndiv values and division format are shown below:

Table 8-3

ndiv value	Processing
1	2x2 division
2	4x4 division
3	8x8 division
4	16 x 16 division
5	32 x 32 division

Return value

Updated GPU packet buffer address.

Remarks

You must set *divp* -> ndiv (number of divisions) and *divp* -> pih.piv (display screen (clipping) resolution) beforehand.

See also:

DivideF4

Division of flat quadrilateral.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *DivideF4(*v0, *v1, *v2, *v3, *rgbc, *s, *ot, *divp)
SVECTOR *v0, *v1, *v2, *v3;
CVECTOR *rgbc;
POLY_F4 *s;
u_long *ot;
DIVPOLYGON4 *divp;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vectors (input)
<i>rgbc</i>	Pointer to color vector + code (input)
<i>s</i>	Pointer to GPU packet buffer address
<i>ot</i>	Pointer to OT entry
<i>divp</i>	Pointer to division work area (input)

Explanation

This is a flat quadrilateral division program. It divides a flat quadrilateral (POLY_F4) indicated by the vertex coordinate vectors and color vector based on the *divp* -> *ndiv* value and registers the result to OT.

The *divp* -> *ndiv* values and division format are shown below:

Table 8-4

ndiv value	Processing
1	2x2 division
2	4x4 division
3	8x8 division
4	16 x 16 division
5	32 x 32 division

Return value

Updated GPU packet buffer address.

Remarks

You must set *divp* -> *ndiv* (number of divisions) and *divp* -> *pih.piv* (display screen (clipping) resolution) beforehand.

See also:

DivideFT3

Division of flat textured triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *DivideFT3(*v0, *v1, *v2, *uv0, *uv1, *uv2, *rgbc, *s, *ot, *divp)
SVECTOR *v0, *v1, *v2;
u_long *uv0, *uv1, *uv2;
CVECTOR *rgbc;
POLY_FT3 *s;
u_long *ot;
DIVPOLYGON3 *divp;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vectors (input)
<i>uv0, uv1, uv2</i>	Pointer to texture coordinate vector (input) v0+clut, uv1:uv1+tpage (uv0)
<i>rgbc</i>	Pointer to color vector +code (input)
<i>s</i>	Pointer to GPU packet buffer address
<i>ot</i>	Pointer to OT entry
<i>divp</i>	Pointer to division work area (input)

Explanation

This is the flat textured triangle division program. It divides a flat textured triangle (POLY_FT3) indicated by the vertex coordinate vectors, texture coordinate vector, and color vector based on the *divp* -> ndiv value and registers the result to OT.

The *divp* -> ndiv values and division format are shown below:

Table 8-5

ndiv value	processing
1	2x2 division
2	4x4 division
3	8x8 division
4	16 x 16 division
5	32 x 32 division

Return value

Updated GPU packet buffer address.

Remarks

You must set *divp* -> ndiv (number of divisions) and *divp* -> pih.piv (display screen (clipping) resolution) beforehand.

See also:

DivideFT4

Division of flat textured quadrilateral.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *DivideFT4(*v0, *v1, *v2, *v3, *uv0, *uv1, *uv2, *uv3, *rgbc, *s, *ot, *divp)
SVECTOR *v0, *v1, *v2, *v3;
u_long *uv0, *uv1, *uv2, *uv3;
CVECTOR *rgbc;
POLY_FT4 *s;
u_long *ot;
DIVPOLYGON4 *divp;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vectors (input)
<i>uv0, uv1, uv2, uv3</i>	Pointer to texture coordinate vector (input) uv0:uv0+clut, uv1:uv1+tpage
<i>rgbc</i>	Pointer to color vector + code (input)
<i>s</i>	Pointer to GPU packet buffer address
<i>ot</i>	Pointer to OT entry
<i>divp</i>	Pointer to division work area (input)

Explanation

This is the flat textured quadrilateral division program. It divides a flat textured quadrilateral (POLY_FT4) indicated by the vertex coordinate vectors, texture coordinate vector, and color vector based on the *divp* -> ndiv value and registers the result to OT.

The *divp* -> ndiv values and division format are shown below:

Table 8-6

ndiv value	Processing
1	2x2 division
2	4x4 division
3	8x8 division
4	16 x 16 division
5	32 x 32 division

Return value

Updated GPU packet buffer address.

Remarks

You must set *divp* -> ndiv (number of divisions) and *divp* -> pih.piv (display screen (clipping) resolution) beforehand.

See also:

DivideG3

Division of Gouraud triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *DivideG3(*v0, *v1, *v2, *rgb0, *rgb1, *rgb2, *s, *ot, *divp)
SVECTOR *v0, *v1, *v2;
CVECTOR *rgb0, *rgb1, *rgb2;
POLY_G3 *s;
u_long *ot;
DIVPOLYGON3 *divp;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vectors (input)
<i>rgb0, rgb1, rgb2</i>	Pointer to color vector (input) rgb0:rgb0+code
<i>s</i>	Pointer to GPU packet buffer address
<i>ot</i>	Pointer to OT entry
<i>divp</i>	Pointer to division work area (input)

Explanation

This is a Gouraud-shaded triangle division program. It divides a Gouraud-shaded (POLY_G3) triangle indicated by the vertex coordinate vectors and color vector based on the *divp* -> ndiv value and registers the result to OT.

The *divp* -> ndiv values and division format are shown below:

Table 8-7

ndiv value	processing
1	2x2 division
2	4x4 division
3	8x8 division
4	16 x 16 division
5	32 x 32 division

Return value

Updated GPU packet buffer address.

Remarks

You must set *divp* -> ndiv (number of divisions) and *divp* -> pih.piv (display screen (clipping) resolution) beforehand.

See also:

DivideG4

Division of Gouraud-shaded quadrilateral.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *DivideG4(*v0, *v1, *v2, *v3, *rgb0, *rgb1, *rgb2, *rgb3, *s, *ot, *divp)
SVECTOR *v0, *v1, *v2, *v3;
CVECTOR *rgb0, *rgb1, *rgb2, *rgb3;
POLY_G4 *s;
u_long *ot;
DIVPOLYGON4 *divp;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vectors (input)
<i>rgb0, rgb1, rgb2, rgb3</i>	Pointer to color vector (input) rgb0:rgb0+code
<i>s</i>	Pointer to GPU packet buffer address
<i>ot</i>	Pointer to OT entry
<i>divp</i>	Pointer to division work area (input)

Explanation

This is the Gouraud-shaded quadrilateral division program. It divides a Gouraud-shaded quadrilateral (POLY_G4) indicated by the vertex coordinate vectors and color vector based on the *divp* -> ndiv value and registers the result to OT.

The *divp* -> ndiv values and division format are shown below:

Table 8-8

ndiv value	Processing
1	2x2 division
2	4x4 division
3	8x8 division
4	16 x 16 division
5	32 x 32 division

Return value

Updated GPU packet buffer address.

Remarks

You must set *divp* -> ndiv (number of divisions) and *divp* -> pih.piv (display screen (clipping) resolution) beforehand.

See also:

DivideGT3

Division of Gouraud-shaded, textured triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *DivideGT3(*v0, *v1, *v2, *uv0, *uv1, *uv2, *rgb0, *rgb1, *rgb2, *s, *ot, *divp)
SVECTOR *v0, *v1, *v2;
u_long *uv0, *uv1, *uv2;
CVECTOR *rgb0, *rgb1, *rgb2;
POLY_GT3 *s;
u_long *ot;
DIVPOLYGON3 *divp;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vectors (input)
<i>uv0, uv1, uv2</i>	Pointer to texture coordinate vector (input) uv0:uv0+clut, uv1:uv1+tpage
<i>rgb0, rgb1, rgb2</i>	Pointer to color vector (input) rgb0:rgb0+code
<i>s</i>	Pointer to GPU packet buffer address
<i>ot</i>	Pointer to OT entry
<i>divp</i>	Pointer to division work area (input)

Explanation

This is the Gouraud-shaded textured triangle division program. It divides a Gouraud-shaded textured triangle (POLY_GT3) indicated by the vertex coordinate vectors, texture coordinate vector, and color vector based on the *divp* -> ndiv value and registers the result to OT.

The *divp* -> ndiv values and division format are shown below:

Table 8-9

ndiv value	Processing
1	2x2 division
2	4x4 division
3	8x8 division
4	16 x 16 division
5	32 x 32 division

Return value

Updated GPU packet buffer address.

Remarks

You must set *divp* -> ndiv (number of divisions) and *divp* -> pih.piv (display screen (clipping) resolution) beforehand.

See also:

DivideGT4

Division of Gouraud-shaded textured quadrilateral.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *DivideGT4(*v0, *v1, *v2, *v3, *uv0, *uv1, *uv2, *uv3, *rgb0, *rgb1, *rgb2, *rgb3, *s, *ot, *divp)
SVECTOR *v0, *v1, *v2, *v3;
u_long *uv0, *uv1, *uv2, *uv3;
CVECTOR *rgb0, *rgb1, *rgb2, *rgb3;
POLY_GT4 *s;
u_long *ot;
DIVPOLYGON4 *divp;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vertex coordinate vectors (input)
<i>uv0, uv1, uv2, uv3</i>	Pointer to texture coordinate vector (input) uv0:uv0+clut, uv1:uv1+tpage
<i>rgb0, rgb1, rgb2, rgb3</i>	Pointer to color vector (input) rgb0:rgb0+code
<i>s</i>	Pointer to GPU packet buffer address
<i>ot</i>	Pointer to OT entry
<i>divp</i>	Pointer to division work area (input)

Explanation

This is the Gouraud-shaded textured quadrilateral division program. It divides a Gouraud-shaded textured quadrilateral (POLY_GT4) indicated by the vertex coordinate vectors, texture coordinate vector, and color vector based on the *divp* -> ndiv value and registers the result to OT.

The *divp* -> ndiv values and division format are shown below:

Table 8-10

ndiv value	Processing
1	2x2 division
2	4x4 division
3	8x8 division
4	16 x 16 division
5	32 x 32 division

Return value

Updated GPU packet buffer address.

Remarks

You must set *divp* -> ndiv (number of divisions) and *divp* -> pih.piv (display screen (clipping) resolution) beforehand.

See also:

DpqColor

Interpolation of a primary color vector and far color.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void DpqColor(*v0, p, *v1)
CVECTOR *v0;
long p;
CVECTOR *v1;
```

Arguments

v0 Pointer to primary color vector (input)
p Interpolation value (input)
v1 Pointer to primary color vector (output)

Explanation

This function calculates $v1 = (1-p) \times v0 + p \times FC$.

The argument format is as follows:

v0 -> r, g, b : (0, 8, 0)
p : (0, 20, 12)
v1 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

DpqColor3

Interpolation of three primary color vectors and far color.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void DpqColor3(*v0, *v1, *v2, p, *v3, *v4, *v5)
CVECTOR *v0, *v1, *v2;
long p;
CVECTOR *v3, *v4, *v5;
```

Arguments

v0, v1, v2 Pointer to primary color vectors (input)
p Interpolation value (input)
v3, v4, v5 Pointer to color vectors (output)

Explanation

This function calculates:

$$v3 = (1-p) \times v0 + p \times FC$$

$$v4 = (1-p) \times v1 + p \times FC$$

$$v5 = (1-p) \times v2 + p \times FC.$$

The argument format is follows:

v0, v1, v2 -> r, g, b : (0, 8, 0)

p : (0, 20, 12)

v3, v4, v5 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

DpqColorLight

Interpolation of the product from the multiplication of a local color vector by primary color vector, and far color.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void DpqColorLight(*v0, *v1, p, *v2)
SVECTOR *v0;
CVECTOR *v1;
long p;
CVECTOR *v2;
```

Arguments

v0 Pointer to local color vector (input)
v1 Pointer to primary color vector (input)
p Interpolation value (input)
v2 Pointer to color vector (output)

Explanation

This function calculates $v2 = (1-p) \times (v1 \times v0) + p \times FC$.

where $v1 \times v0$ is a separate multiplication product.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)

v1 -> r, g, b : (0, 8, 0)

p : (0, 20, 12)

v2 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

EigenMatrix

Obtains the eigen matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
void EigenMatrix (
MATRIX *m,
MATRIX *t
)
```

Arguments

m Input: rotation matrix

t Output: eigen matrix

Explanation

The eigen matrix (ordered eigen vectors) corresponding to the input rotation matrix, *m*, is output to the matrix *t*.

The operation performed is shown below.

$$[t]^{-1} \times [m] \times [t]^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

The argument format is as follows:

m -> m [i] [j] : (1, 3, 12)

t -> m [i] [j] : (1, 3, 12)

Return value

None

Remarks

See also:

gteMIMefunc

Adding a vertex data array to a differential data array multiplied by a coefficient.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void gteMIMefunc(*otp, *dfp, n, p)
SVECTOR *otp;
SVECTOR *dfp;
long n;
long p;
```

Arguments

otp Pointer to a vertex array
dfp Pointer to a differential array
n Number of vertex (differential) data
p Weight (control) coefficient: (1, 19, 12)

Explanation

Executes calculation of multiple interpolations using vertex data array and difference data array. The argument format is as follows.

p : (1, 19, 12)

otp, *dfp* optional

It operates at high speed in a similar way to the program given in the example below.

```
void gteMIMefunc (otp, dfp, n, p)
SVECTOR *otp, *dfp;
long n, p;
{
    int i;
    for (i = 0; i < n; i++) {
        (otp+i)->x += ((int) ((dfp+i)->x) x p) >> 12;
        (otp+i)->y += ((int) ((dfp+i)->y) x p) >> 12;
        (otp+i)->z += ((int) ((dfp+i)->z) x p) >> 12;
    }
}
```

Return value

None

Remarks

See also:

InitClip

Initialize clipping parameter.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
void InitClip(*evbfad, hw, vw, h, near, far)
EVECTOR *evbfad;
long hw, vw;
long h;
long near, far;
```

Arguments

evbfad Pointer to addresses of (16) clip vector data arrays
hw, vw *hw*: Window width, *vw*: Window height
h Projection distance from view point to screen
near, far *near*: NearClip position, *far*: FarClip position

Explanation

This function sets parameters used for clipping.

The clip vector data array *evbfad* reserves 16 data arrays (176 words or 704 bytes).

Return value

None

Remarks

See also:

InitGeom

Initialization of geometry transformation engine.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void InitGeom(void)
```

Arguments

None

Explanation

This function initializes GTE. It is called when the basic geometry library is used.

Return value

None

Remarks

See also:

Intpl

Interpolation of a vector and far color.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void Intpl(*v0, p, *v1)
SVECTOR *v0;
long p;
CVECTOR *v1;
```

Arguments

v0 Pointer to vector (input)
p Interpolation value (input)
v1 Pointer to vector (output)

Explanation

This function calculates $v1 = (1-p) \times v0 + p \times FC$.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)

p : (0, 20, 12)

v1 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

InvSquareRoot

1/square root.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void InvSquareRoot(a, *b, *c)
```

```
long a;
```

```
long *b;
```

```
long *c;
```

Arguments

- a* Value
- b* Pointer to address where a mantissa will be stored
- c* Pointer to address where an exponent will be stored

Explanation

The function returns 1/square root of a value *a*.

The argument format is as follows:

a : (0, 32, 0)

b : (0, 20, 12)

c : (0, 32, 0)

Return value

None

Remarks

See also:

IsIdMatrix

Judges distance from unit matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
void IsIdMatrix (
  MATRIX *m
)
```

Arguments

m Input matrix

Explanation

This function compares the input matrix *m* with the unit matrix. If the elements of matrix *m* are less than 20 away from the unit matrix, the function returns the value 1.

The argument format is as follows:

m -> *m* [i] [j] : (1, 3, 12)

Return value

1 if the matrix is the unit matrix, else 0.

Remarks

See also:

LightColor

Coordinate transformation using the local color matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void LightColor(*v0, *v1)
SVECTOR *v0;
VECTOR *v1;
```

Arguments

v0 Pointer to vector (input)
v1 Pointer to vector (output)

Explanation

This function calculates $v1 = LCM \times v0$. A limiter works on negative components of *v1* when 0 is reached. The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)

v1 -> vx, vy, vz : (0, 20, 12)

Return value

None

Remarks

See also:

LoadAverage0

Weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
void LoadAverage0(*v0, *v1, p0, p1, *v2)
VECTOR *v0, *v1;
long p0, p1;
VECTOR *v2;
```

Arguments

v0, v1 Pointer to vectors (input)
p0, p1 Weights (input)
v2 Pointer to vector (output)

Explanation

This function returns the weighted average of two vectors *v0* and *v1* in *v2* using weights of *p0* and *p1*.

The argument format is as follows:

v0, v1 -> vx, vy, vz : (1, 31, 0)
p0, p1 : (1, 15, 0)
v2 -> vx, vy, vz : (1, 31, 0)

Return value

None

Remarks

See also:

LoadAverage12

Weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void LoadAverage12(*v0, *v1, p0, p1, *v2)
VECTOR *v0, *v1;
long p0, p1;
VECTOR *v2;
```

Arguments

v0, v1 Pointer to vectors (input)
p0, p1 Weights (input)
v2 Pointer to vector (output)

Explanation

This function finds the weighted average of two vectors *v0* and *v1* using weights of *p0* and *p1* after division by 4096 (1 in fixed point format) the results are returned in *v2*.

The argument format is as follows:

v0, v1 -> vx, vy, vz : (1, 31, 0)
p0, p1 : (1, 3, 12)
v2 -> vx, vy, vz : (1, 31, 0)

Return value

None

Remarks

See also:

LoadAverageByte

Find weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
void LoadAverageByte(v0, v1, p0, p1, v2)
unsigned char v0[2], v1[2];
long p0, p1;
unsigned char v2[2];
```

Arguments

v0, *v1* Vector (input)
p0, *p1* Weights (input)
v2 Vector (output)

Explanation

This function finds the weighted average of two vectors *v0* and *v1* using weights *p0* and *p1*. The result is returned in *v2* after division by 4096.

The argument format is as follows:

v0[*i*], *v1*[*i*] : (0, 8, 0)
p0, *p1* : (1, 3, 12)
v2[*i*] : (0, 8, 0)

Return value

None

Remarks

See also:

LoadAverageCol

Find weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
void LoadAverageCol(v0, v1, p0, p1, v2)
unsigned char v0[3], v1[3];
long p0, p1;
unsigned char v2[3];
```

Arguments

v0, v1 Vectors (input)
p0, p1 Weights (input)
v2 Vector (output)

Explanation

This function finds the weighted average of two vectors *v0* and *v1* using weights *p0* and *p1*. The result is returned in *v2* after division by 4096.

The argument format is as follows:

v0[i], v1[i] : (0, 8, 0)
p0, p1 : (1, 3, 12)
v2[i] : (0, 8, 0)

Return value

None

Remarks

See also:

LoadAverageShort0

Weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void LoadAverageShort0(*v0, *v1, p0, p1, *v2)
SVECTOR *v0, *v1;
long p0, p1;
SVECTOR *v2;
```

Arguments

v0, *v1* Pointer to vectors (input)
p0, *p1* Weights (input)
v2 Pointer to vector (output)

Explanation

This function returns the weighted average of two vectors *v0* and *v1* in *v2* using weights of *p0* and *p1*.

The argument format is as follows:

v0, *v1* -> vx, vy, vz : (1, 15, 0)
p0, *p1* : (1, 15, 0)
v2 -> vx, vy, vz : (1, 30, 0)

Return value

None

Remarks

See also:

LoadAverageShort12

Weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void LoadAverageShort12(*v0, *v1, p0, p1, *v2)
SVECTOR *v0, *v1;
long p0, p1;
SVECTOR *v2;
```

Arguments

v0, v1 Pointer to vectors (input)
p0, p1 Weights (input)
v2 Pointer to vector (output)

Explanation

This function finds the weighted average of two vectors *v0* and *v1* using weights of *p0* and *p1* after division by 4096 (1 in fixed point format) the results are returned to *v2*.

The argument format is as follows:

v0, v1 -> vx, vy, vz : (1, 15, 0)
p0, p1 : (1, 3, 12)
v2 -> vx, vy, vz : (1, 15, 0)

Return value

None

Remarks

See also:

LocalLight

Coordinate transformation using the local light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void LocalLight(*v0, *v1)
SVECTOR *v0;
VECTOR *v1;
```

Arguments

v0 Pointer to vector (input)
v1 Pointer to vector (output)

Explanation

This function calculates $v1 = LLM * v0$. A limiter works on negative components of *v1* when 0 is reached. The argument format is as follows:

v0 -> vx, vy, vz: :(1, 3, 12)

v1 -> vx, vy, vz: :(0, 20, 12)

Return value

None

Remarks

See also:

Lzc

Returning a leading zero count (LZC).

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
long Lzc(data)
long data;
```

Arguments

data Value

Explanation

This function calculates the Leading Zero Count given by *data*.

The argument format is as follows:

data : (1, 31, 0)

Return value

Returns the value of LZC.

Remarks

See also:

MatrixNormal

Normalize a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
void MatrixNormal(*m, *n)
```

```
MATRIX *m;
```

```
MATRIX *n;
```

Arguments

m Pointer to matrix (input)

n Pointer to matrix (output)

Explanation

This function orthonormalizes a distorted rotation matrix.

(Do not use `m[2][0]`, `m[2][1]`, `m[2][2]`.)

The argument format is as follows:

`m -> m [i] [j] : (1, 3, 12)`

`n -> m [i] [j] : (1, 3, 12)`

Return value

Remarks

See also:

MatrixNormal_0

Orthonormalizes a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.6	10/23/96

Syntax

```
void MatrixNormal_0(*m, *n)
MATRIX *m;
MATRIX *n;
```

Arguments

m Pointer to matrix (input)
n Pointer to matrix (output)

Explanation

This function orthonormalizes a distorted rotation matrix.

(Do not use `m[2][0]`, `m[2][1]`, `m[2][2]`.)

The argument format is as follows:

`m -> m [i] [j] : (1, 3, 12)`
`n -> m [i] [j] : (1, 3, 12)`

Return value

None

Remarks

See also:

MatrixNormal_1

Normalizes a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
void MatrixNormal_1(*m, *n)
```

```
MATRIX *m;
```

```
MATRIX *n;
```

Arguments

m Pointer to matrix (input)

n Pointer to matrix (output)

Explanation

This function orthonormalizes a distorted rotation matrix.

(Do not use `m[0][0]`, `m[0][1]`, `m[0][2]`.)

The argument format is as follows:

`m` -> `m [i] [j] : (1, 3, 12)`

`n` -> `m [i] [j] : (1, 3, 12)`

Return value

Remarks

See also:

MatrixNormal_2

Normalizes a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
void MatrixNormal_2(*m, *n)
MATRIX *m;
MATRIX *n;
```

Arguments

m Pointer to matrix (input)
n Pointer to matrix (output)

Explanation

This function orthonormalizes a distorted rotation matrix.

(Do not use `m[1][0]`, `m[1][1]`, `m[1][2]`.)

The argument format is as follows:

`m -> m [i] [j] : (1, 3, 12)`
`n -> m [i] [j] : (1, 3, 12)`

Return value

Remarks

See also:

MulMatrix

Multiplication of two matrices.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

MATRIX *MulMatrix(**m0*, **m1*)

MATRIX **m0*, **m1*;

Arguments

m0, *m1* Pointer to input/output matrices

Explanation

This function multiplies two matrices. The result is saved in *m0*.

The argument format is as follows:

m0, *m1* -> m [i] [j] : (1, 3, 12)

Return value

This function returns *m0*.

Remarks

The function destroys the constant rotation matrix.

See also:

MulMatrix0

Multiplication of two matrices.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
MATRIX *MulMatrix0(*m0, *m1, *m2)
MATRIX *m0, *m1;
MATRIX *m2;
```

Arguments

m0, m1 Pointer to input matrices
m2 Pointer to output matrix

Explanation

This function multiplies two matrices *m0* and *m1*.

The argument format is as follows:

m0, m1, m2 -> m [i] [j] : (1, 3, 12)

Return value

This function returns *m2*.

Remarks

The function destroys the constant rotation matrix.

See also:

MulMatrix2

Multiplication of two matrices.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

MATRIX *MulMatrix2(*m0, *m1)

MATRIX *m0, *m1;

Arguments

m0, m1 Pointer to input/output matrices

Explanation

This function multiplies two matrices. The result is saved in *m1*.

The argument format is as follows:

m0, m1 -> m [i] [j] : (1, 3, 12)

Return value

This function returns *m1*.

Remarks

The function destroys the constant rotation matrix.

See also:

MulRotMatrix

Multiply a constant rotation matrix by a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
MATRIX *MulRotMatrix(*m0)
MATRIX *m0;
```

Arguments

m0 Pointer to input/output matrix

Explanation

This function multiplies a constant rotation matrix by a matrix. It stores the value in *m0*.

The argument format is as follows:

m0, m1 -> m [i] [j] : (1, 3, 12)

Return value

Returns *m0*.

Remarks

See also:

MulRotMatrix0

Multiply a constant rotation matrix by a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

MATRIX *MulRotMatrix0(**m0*, **m1*)

MATRIX **m0*;

MATRIX **m1*;

Arguments

m0 Pointer to input matrix

m1 Pointer to output matrix

Explanation

This function multiplies a constant rotation matrix by matrix *m0*. The result is saved in *m1*.

The argument format is as follows:

m0, *m1* -> m [i] [j] : (1, 3, 12)

Return value

Returns *m1*.

Remarks

See also:

NormalClip

Outer product of three points.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	02/15/98

Syntax

long NormalClip(*sxy0*, *sxy1*, *sxy2*)

long *sxy0*, *sxy1*, *sxy2*;

Arguments

sxy0, *sxy1*, *sxy2* Input : value (sx: LS16bit, sy: MS16bit)

Explanation

This function returns the outer product for a triangle formed by three points (*sx0*, *sy0*), (*sx1*, *sy1*), and (*sx2*, *sy2*).

When viewed from the direction of the viewpoint (Z axis negative) the value is positive when the triangle is righthanded.

(However, the X axis positive faces right and the Y axis positive faces down.)

The argument format is as follows:

sxy0, *sxy1*, *sxy2* : (1, 15, 0), (1, 15, 0)

Return value

| *sx1-sx0*, *sy1-sy0* |

| *sx2-sx0*, *sy2-sy0* |

Remarks

See also:

NormalColor

Finds a local color from a normal vector and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColor(*v0, *v1)
SVECTOR *v0;
CVECTOR *v1;
```

Arguments

v0 Pointer to normal vector (input)
v1 Pointer to color vector (output)

Explanation

$LLV = LLM \times v0$

$v1 = BK + LCM \times LLV$

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)

v1 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

NormalColor_nom

Finds a local color from a normal vector, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColor_nom
SVECTOR *v0;
```

Arguments

v0 Pointer to normal vector (input)

Explanation

$LLV = LLM \times v0$

$v1 = BK + LCM \times LLV$ (See remarks below)

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)

Return value

None

Remarks

Read (*v1*->r, *v1*->g, *v1*->b) in the read_rgb2 macro.

The operation result(s) must be retrieved from the GTE. For further information see the Inline Reference documentation.

See also:

NormalColor3

Finds three local colors from three normal vectors and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColor3(*v0, *v1, *v2, *v3, *v4, *v5)
SVECTOR *v0, *v1, *v2;
CVECTOR *v3, *v4, *v5;
```

Arguments

v0, v1, v2 Pointer to normal vectors (input)
v3, v4, v5 Pointer to color vectors (output)

Explanation

$(LLV0, LLV1, LLV2) = LLM \times (v0, v1, v2)$

$(v3, v4, v5) = BK + LCM \times (LLV0, LLV1, LLV2)$

The argument format is as follows:

v0, v1, v2 -> vx, vy, vz : (1, 3, 12)

v3, v4, v5 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

NormalColor3_nom

Finds three local colors from three normal vectors, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColor3_nom
```

```
SVECTOR *v0, *v1, *v2;
```

Arguments

v0, *v1*, *v2* Pointer to normal vectors (input)

Explanation

$(LLV0, LLV1, LLV2) = LLM \times (v0, v1, v2)$

$(v3, v4, v5) = BK + LCM \times (LLV0, LLV1, LLV2)$ (See remarks below)

The argument format is as follows:

v0, *v1*, *v2* -> *vx*, *vy*, *vz* : (1, 3, 12)

Return value

None

Remarks

Read (*v3*, *v4* and *v5*) in the `read_rgb_fifo` macro. For further information, refer to the Inline Reference documentation.

See also:

NormalColorCol

Finds a local color from a normal vector and returns the operation result

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColorCol(*v0, *v1, *v2)
SVECTOR *v0;
CVECTOR *v1;
CVECTOR *v2;
```

Arguments

v0 Pointer to normal vector (input)
v1 Pointer to primary color vector (input)
v2 Pointer to color vector (output)

Explanation

$LLV = LLM \times v0$

$LC = BK + LCM \times LLV$

$v2 = v1 \times LC$

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)

v1 -> r, g, b : (0, 8, 0)

v2 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

NormalColorCol_nom

Finds a local color from a normal vector, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColorCol_nom
```

```
SVECTOR *v0;
```

```
CVECTOR *v1;
```

Arguments

v0 Pointer to normal vector (input)

v1 Pointer to primary color vector (input)

Explanation

$LLV = LLM \times v0$

$LC = BK + LCM \times LLV$

$v2 = v1 \times LC$ (See remarks below)

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)

v1 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

For (*v2*->r,*v2*->g,*v2*->b) read the read_rgb2 macro.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

NormalColorCol3

Finds a local color from three normal vectors and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColorCol3(*v0, *v1, *v2, *v3, *v4, *v5, *v6)
SVECTOR *v0, *v1, *v2;
CVECTOR *v3;
CVECTOR *v4, *v5, *v6;
```

Arguments

v0, *v1*, *v2* Pointer to normal vectors (input)
v3 Pointer to primary color vector (input)
v4, *v5*, *v6* Pointer to color vectors (output)

Explanation

$(LLV0, LLV1, WL2) = LLM \times (v0, v1, v2)$

$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$

$(v4, v5, v6) = v3 \times (LC0, LC1, LC2)$

The argument format is as follows:

v0, *v1*, *v2* -> vx, vy, vz : (1, 3, 12)

v3 -> r, g, b : (0, 8, 0)

v4, *v5*, *v6* -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

NormalColorCol3_nom

Finds a local color from three normal vectors, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColorCol3_nom
```

```
SVECTOR *v0, *v1, *v2;
```

```
CVECTOR *v3;
```

Arguments

v0, *v1*, *v2* Pointer to normal vectors (input)
v3 Pointer to primary color vector (input)

Explanation

$(LLV0, LLV1, WL2) = LLM \times (v0, v1, v2)$

$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$

$(v4, v5, v6) = v3 \times (LC0, LC1, LC2)$ (See remarks below)

The argument format is as follows:

v0, *v1*, *v2* -> *vx*, *vy*, *vz* : (1, 3, 12)

v3 -> *r*, *g*, *b* : (0, 8, 0)

Return value

None

Remarks

Read (*v4*,*v5*,*v6*) in the `read_rgb_fifo` macro.

The operation result(s) must be retrieved from the GTE.

For further information, see the Inline Reference documentation.

See also:

NormalColorDpq

Finds a local color from a normal vector, performs depth cueing and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColorDpq(*v0, *v1, p, *v2)
SVECTOR *v0;
CVECTOR *v1;
long p;
CVECTOR *v2;
```

Arguments

v0 Pointer to normal vector (input)
v1 Pointer to primary color vector (input)
p Interpolation value (input)
v2 Pointer to color vector (output)

Explanation

$LLV = LLM \times v0$

$LC = BK + LCM \times LLV$

$v2 = (1 - p) \times v1 \times LC + p \times FC$

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)

v1 -> r, g, b : (0, 8, 0)

p : (0, 20, 12)

v2 -> r, g, b : (0, 8, 0)

Return value

None

Remarks

See also:

NormalColorDpq_nom

Finds a local color from a normal vector and performs depth cueing, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColorDpq_nom
```

```
SVECTOR *v0;
```

```
CVECTOR *v1;
```

```
long p;
```

Arguments

v0 Pointer to normal vector (input)

v1 Pointer to primary color vector (input)

p Interpolation value (input)

Explanation

$LLV = LLM \times v0$

$LC = BK + LCM \times LLV$

$v2 = (1 - p) \times v1 \times LC + p \times FC$ (See remarks below)

The argument format is as follows:

v0 -> vx, vy, vz : (1, 3, 12)

v1 -> r, g, b : (0, 8, 0)

p : (0, 20, 12)

Return value

None

Remarks

For (v2->r,v2->g,v2->b) read the read_rgb2 macro.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

NormalColorDpq3

Finds local color from three normal vectors, performs depth cueing and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColorDpq3(*v0, *v1, *v2, *v3, p, *v4, *v5, *v6)
SVECTOR *v0, *v1, *v2;
CVECTOR *v3;
long p;
CVECTOR *v4, *v5, *v6;
```

Arguments

v0, v1, v2 Pointer to normal vectors (input)
v3 Pointer to primary color vector (input)
p Interpolation value (input)
v4, v5, v6 Pointer to color vectors (output)

Explanation

$(LLV0, LLV1, LLV2) = LLM \times (v0, v1, v2)$

$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$

$(v4, v5, v6) = (1 - p) \times v3 \times (LC0, LC1, LC2) + p \times FC$

The argument format is as follows:

v0, v1, v2 -> vx, vy, vz : (1, 3, 12)
v3 -> r, g, b : (0, 8, 0)
p : (0, 20, 12)
v4, v5, v6 -> r, g, b) : (0, 8, 0)

Return value

None

Remarks

See also:

NormalColorDpq3_nom

Finds local color from three normal vectors and performs depth cueing, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void NormalColorDpq3_nom
```

```
SVECTOR *v0, *v1, *v2;
```

```
CVECTOR *v3;
```

```
long p;
```

Explanation

$(LLV0, LLV1, LLV2) = LLM \times (v0, v1, v2)$

$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$

$(v4, v5, v6) = (1 - p) \times v3 \times (LC0, LC1, LC2) + p \times FC$ (See remarks below)

The argument format is as follows:

$v0, v1, v2 \rightarrow vx, vy, vz$: (1, 3, 12)

$v3 \rightarrow r, g, b$: (0, 8, 0)

p : (0, 20, 12)

Return value

None

Remarks

Read (v4,v5,v6) in the read_rgb_fifo macro.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

otz2p

Get depth cueing interpolation value p from OTZ value.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
long otz2p (
long otz,
long projection
)
```

Arguments

<i>otz</i>	OTZ
<i>projection</i>	Distance between visual point and screen

Explanation

Get the approximate depth cueing interpolation value p from sz, the z element of the screen coordinates. sz is sz/4 *otz* value.

Return value

Depth cueing interpolation value p (0: 0%, 4096 : 100%).

Remarks

Depending on the fog setting, errors can increase and the results are not necessarily the same as the RotTransPers system function.

See also:

OuterProduct0

Outer product of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void OuterProduct0(*v0, *v1, *v2)
VECTOR *v0, *v1;
VECTOR *v2;
```

Arguments

v0, *v1* Pointer to vectors (input)
v2 Pointer to vector (output)

Explanation

This function returns the outer product vector of two vectors *v0* and *v1* to *v2*.

The argument format is as follows:

v0, *v1* -> vx, vy, vz : (1, 31, 0)

v2 -> vx, vy, vz : (1, 31, 0)

Return value

None

Remarks

See also:

OuterProduct12

Outer product of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void OuterProduct12(*v0, *v1, *v2)
VECTOR *v0, *v1;
VECTOR *v2;
```

Arguments

v0, *v1* Pointer to vectors (input)
v2 Pointer to vector (output)

Explanation

This function returns the outer product vector of two vectors, *v0* and *v1*, to *v2*.

The argument format is as follows:

v0, *v1*, *v2* -> vx, vy, vz : (1, 19, 12)

Return value

None

Remarks

See also:

p2otz

Get otz from depth cueing interpolation value.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
long p2otz (
long p,
long projection
)
```

Arguments

p Can be 0 to 4096
projection Distance between visual point and screen

Explanation

Gets the z element of the screen coordinates or sz/4 *otz* value from the depth cueing interpolation value *p*.

Return value

OTZ value.

Remarks

Depending on the fog setting, errors can increase and the results are not necessarily the same as the RotTransPers system functions.

otz when P=0 or p=4096 is not theoretically decided as identification, but with this function a convenient value is returned.

See also:

pers_map

Perspective conversion texture mapping.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
void pers_map (
int abuf,
SVECTOR **vertex,
int tex[4][2],
u_short *dtext
)
```

Arguments

abuf ID of displayed buffer
vertex 3 dimensional coordinates of 4 vertices
tex Texture address of 4 vertices
dtext Pointer to texture storage location converted to direct color

Explanation

Performs texture mapping with no distortion.

Return value

None

Remarks

Flat texture, with no light source calculations only.

The 4 vertices are only square, rectangle and parallelogram locations.

Z sort by OT is not possible.

See also:

PhongLine

Phong shading.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
void PhongLine (
int istat_x,
int iend_x,
int p,
int q,
u_short *pixx,
int fs,
int ft,
int i4,
int det
)
```

Arguments

isstat_x X coordinate of starting point
iend_x X coordinate of finishing point
p Differential X coordinate of *fs* value
q Differential caused by X coordinate of *ft* value
pixx Pixel pointer
fs Interpolation coefficient at start point
ft Interpolation coefficient at start point
i4 (Line number) %4 due to dithering
det Queue method of edge queue

Explanation

Performs one line Phong shading.

Return value

None

Remarks

Refer to sample program (sample/graphics/phong)

See also:

PopMatrix

Resets a constant rotation matrix from a stack.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

void PopMatrix(*void*)

Arguments

None

Explanation

This function resets a constant rotation matrix from a stack.

Return value

None

Remarks

See also:

PushMatrix

Saving a constant rotation matrix in a stack.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

void PushMatrix(*void*)

Arguments

None

Explanation

This function saves a constant rotation matrix on a stack. The stack has 20 slots.

Return value

None

Remarks

See also:

atan2

Arctan.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

long atan2(*x*, *y*)

long *x*, *y*;

Arguments

x, *y* Value

Explanation

This function uses PlayStation format (4096 = 360 degrees = 2pai) to finish the x/y arctan function (-180 degrees and +180 degrees, -pai...pai).

The argument format is as follows:

Return value : (1, 19, 12)

Return value

atan2 (*x*, *y*)

Remarks

The return value is incorrect if either *x* or *y* is -2147483648 (0x80000000 = long negative maximum value).

See also:

rcos

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	10/01/97

Syntax

```
int rcos(a)
int a;
```

Arguments

a Angle (in PlayStation format)

Explanation

Finds the cosine function of the angle (in PlayStation format) ($4096 = 360 \text{ degrees} = 2\pi$) using fixed-point math (where $4096=1.0$).

The speed and size of `rcos()` and `ccos()` differ as shown below.

	Speed	Size
rcos	fast	large
ccos	slow	small

The argument format is as follows:

a : PlayStation format ($4096 = 360 \text{ degrees} = 2\pi$)

Return value : (1, 19, 12)

Return value

`cos (a)`

Remarks

See also: `ccos()`

RCpolyF3

Division of flat triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *RCpolyF3(*s, *divp)
POLY_F3 *s;
DIVPOLYGON3 *divp;
```

Arguments

s Pointer to GPU packet buffer address
divp Pointer to division work area

Explanation

This is a recursive function for division of flat triangles (POLY_F3). In order to use it, you must set the data below in the *divp* work area:

u_long <i>ndiv</i>	Number of divisions
u_long <i>pih, piv</i>	Display screen resolution (for clipping)
u_short <i>clut, tpage</i>	CBA & TSB
CVECTOR <i>rgbc</i>	Color vector (+code)
u_long <i>*ot</i>	OT entry
RVECTOR <i>r0, r1, r2</i>	Division vertex vector data
CRVECTOR3 <i>cr[5]</i>	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of CRVECTOR3 (*cr[5]*) to the value of the vertex vector data of RVECTORs *r0*, *r1*, and *r2*.

Note: See DIVPOLYGON3 for a full description of *divp*.

Return value

Updated GPU packet buffer address

Remarks

See also: DIVPOLYGON3 (p. 8-8).

RCpolyF4

Division of flat quadrilateral.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *RCpolyF4(*s, *divp)
POLY_F4 *s;
DIVPOLYGON4 *divp;
```

Arguments

s Pointer to GPU packet buffer address
divp Pointer to division work area

Explanation

This is a recursive function for division of flat quadrilaterals (POLY_F4). In order to use it, you must set the data below in the *divp* work area:

u_long <i>ndiv</i>	Number of divisions
u_long <i>pih, piv</i>	Display screen resolution (for clipping)
u_short <i>clut, tpage</i>	CBA & TSB
CVECTOR <i>rgbc</i>	Color vector (+code)
u_long * <i>ot</i>	OT entry
RVECTOR <i>r0, r1, r2, r3</i>	Division vertex vector data
CRVECTOR4 <i>cr[5]</i>	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of CRVECTOR4 (*cr[5]*) to the value of the vertex vector data of RVECTORs *r0*, *r1*, *r2* and *r3*.

Note: See DIVPOLYGON4 for a full description of *divp*.

Return value

Updated GPU packet buffer address.

Remarks

See also: DIVPOLYGON4 (p. 8-9).

RCpolyFT3

Division of flat, textured triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *RCpolyFT3(*s, *divp)
POLY_FT3 *s;
DIVPOLYGON3 *divp;
```

Arguments

s Pointer to GPU packet buffer address
divp Pointer to division work area

Explanation

This is a recursive function for division of flat , textured triangles (POLY_FT3). In order to use it, you must set the data below in the *divp* work area:

u_long <i>ndiv</i>	: Number of divisions
u_long <i>pih, piv</i>	Display screen resolution (for clipping)
u_short <i>clut, tpage</i>	CBA & TSB
CVECTOR <i>rgbc</i>	Color vector (+code)
u_long <i>*ot</i>	OT entry
RVECTOR <i>r0, r1, r2</i>	Division vertex vector data
CRVECTOR3 <i>cr[5]</i>	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of CRVECTOR3 (*cr[5]*) to the value of the vertex vector data of RVECTORs *r0*, *r1*, and *r2*.

Note: See DIVPOLYGON3 for a full description of *divp*.

Return value

Updated GPU packet buffer address.

Remarks

See also: DIVPOLYGON3 (p. 8-8).

RCpolyFT4

Division of flat, textured quadrilateral.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *RCpolyFT4(*s, *divp)
POLY_FT4 *s;
DIVPOLYGON4 *divp;
```

Arguments

s Pointer to GPU packet buffer address
divp Pointer to division work area

Explanation

This is a recursive function for division of flat, textured quadrilaterals (POLY_FT4). In order to use it, you must set the data below in the *divp* work area:

u_long <i>ndiv</i>	Number of divisions
u_long <i>pih</i> , <i>piv</i>	Display screen resolution (for clipping)
u_short <i>clut</i> , <i>tpage</i>	CBA & TSB
CVECTOR <i>rgbc</i>	Color vector (+code)
u_long * <i>ot</i>	OT entry
RVECTOR <i>r0</i> , <i>r1</i> , <i>r2</i> , <i>r3</i>	Division vertex vector data
CRVECTOR4 <i>cr</i> [5]	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of CRVECTOR4 (*cr*[5]) to the value of the vertex vector data of RVECTORs *r0*, *r1*, *r2*, and *r3*.

Note: See DIVPOLYGON4 for a full description of *divp*.

Return value

Updated GPU packet buffer address.

Remarks

See also: DIVPOLYGON4 (p. 8-9).

RCpolyG3

Division of Gouraud-shaded triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *RCpolyG3(*s, *divp)
POLY_G3 *s;
DIVPOLYGON3 *divp;
```

Arguments

s Pointer to GPU packet buffer address
divp Pointer to division work area

Explanation

This is a recursive function for division of Gourard-shaded triangles (POLY_G3). In order to use it, you must set the data below in the *divp* work area:

u_long <i>ndiv</i>	Number of divisions
u_long <i>pih, piv</i>	Display screen resolution (for clipping)
u_short <i>clut, tpage</i>	CBA & TSB
CVECTOR <i>rgbc</i>	Color vector (+code)
u_long <i>*ot</i>	OT entry
RVECTOR <i>r0, r1, r2</i>	Division vertex vector data
CRVECTOR3 <i>cr[5]</i>	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of CRVECTOR3 (*cr[5]*) to the value of the vertex vector data of RVECTORs *r0*, *r1*, and *r2*.

Note: See DIVPOLYGON3 for a full description of *divp*.

Return value

Updated GPU packet buffer address.

Remarks

See also: DIVPOLYGON3 (p. 8-8).

RCpolyG4

Division of Gouraud-shaded quadrilateral.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *RCpolyG4(*s, *divp)
POLY_G4 *s;
DIVPOLYGON4 *divp;
```

Arguments

s Pointer to GPU packet buffer address
divp Pointer to division work area

Explanation

This is a recursive function for division of Gouraud-shaded quadrilaterals (POLY_G4). In order to use it, you must set the data below in the *divp* work area:

u_long <i>ndiv</i>	Number of divisions
u_long <i>pih, piv</i>	Display screen resolution (for clipping)
u_short <i>clut, tpage</i>	CBA & TSB
CVECTOR <i>rgbc</i>	Color vector (+code)
u_long * <i>ot</i>	OT entry
RVECTOR <i>r0, r1, r2, r3</i>	Division vertex vector data
CRVECTOR4 <i>cr[5]</i> ;	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of CRVECTOR4 (*cr[5]*) to the value of the vertex vector data of RVECTORs *r0*, *r1*, *r2* and *r3*.

Note: See DIVPOLYGON4 for a full description of *divp*.

Return value

Updated GPU packet buffer address.

Remarks

See also: DIVPOLYGON4 (p. 8-9).

RCpolyGT3

Division of Gouraud-shaded, textured triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *RCpolyGT3(*s, *divp)
POLY_GT3 *s;
DIVPOLYGON3 *divp;
```

Arguments

s Pointer to GPU packet buffer address
divp Pointer to division work area

Explanation

This is a recursive function for division of Gourard-shaded, textured triangles (POLY_GT3). In order to use it, you must set the data below in the *divp* work area:

u_long <i>ndiv</i>	Number of divisions
u_long <i>pih, piv</i>	Display screen resolution (for clipping)
u_short <i>clut, tpage</i>	CBA & TSB
CVECTOR <i>rgbc</i>	Color vector (+code)
u_long <i>*ot</i>	OT entry
RVECTOR <i>r0, r1, r2</i>	Division vertex vector data
CRVECTOR3 <i>cr[5]</i>	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of CRVECTOR3 (*cr[5]*) to the value of the vertex vector data of RVECTORs *r0*, *r1*, and *r2*.

Note: See DIVPOLYGON3 for a full description of *divp*.

Return value

Updated GPU packet buffer address.

Remarks

See also: DIVPOLYGON3 (p. 8-8).

RCpolyGT4

Division of Gouraud-shaded, textured quadrilateral.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
u_long *RCpolyGT4(*s, *divp)
POLY_GT4 *s;
DIVPOLYGON4 *divp;
```

Arguments

s Pointer to GPU packet buffer address
divp Pointer to division work area

Explanation

This is a recursive function for division of Gouraud-shaded, textured quadrilaterals (POLY_GT4). In order to use it, you must set the data below in the *divp* work area:

u_long <i>ndiv</i>	Number of divisions
u_long <i>pih</i> , <i>piv</i>	Display screen resolution (for clipping)
u_short <i>clut</i> , <i>tpage</i>	CBA & TSB
CVECTOR <i>rgbc</i>	Color vector (+code)
u_long <i>*ot</i>	OT entry
RVECTOR <i>r0</i> , <i>r1</i> , <i>r2</i> , <i>r3</i>	Division vertex vector data
CRVECTOR4 <i>cr</i> [5]	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of CRVECTOR4 (*cr*[5]) to the value of the vertex vector data of RVECTORs *r0*, *r1*, *r2*, and *r3*.

Note: See DIVPOLYGON4 for a full description of *divp*.

Return value

Updated GPU packet buffer address.

Remarks

See also: DIVPOLYGON4 (p. 8-9).

ReadColorMatrix

Reading a local color matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void ReadColorMatrix(*m)
MATRIX *m;
```

Arguments

m Pointer to matrix (input)

Explanation

This function reads the current local color matrix, and saves it in *m*.

The argument format is as follows:

m -> m [i] [j] : (1, 3, 12)

Return value

None

Remarks

See also:

ReadGeomOffset

Read GTE offset value.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
void ReadGeomOffset(*ofx, *ofy)
long *ofx, *ofy;
```

Arguments

ofx Pointer to offset X coordinate
ofy Pointer to offset Y coordinate

Explanation

This function reads the GTE offset value.

The argument format is as follows:

ofx, *ofy* : (0, 32, 0)

Return value

None

Remarks

See also:

ReadGeomScreen

Read distance from view point to screen.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

long ReadGeomScreen(*void*)

Arguments

None

Explanation

This function reads the distance h from the view point (eye) to the screen.

Return value

h value

Remarks

See also:

ReadLightMatrix

Reading a local light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void ReadLightMatrix(*m)
MATRIX *m;
```

Arguments

m Pointer to matrix (input)

Explanation

This function reads the current local light matrix, and saves it in *m*.

The argument format is as follows:

m -> m [i] [j] : (1, 3, 12)

Return value

None

Remarks

See also:

ReadRGBfifo

Reading RGBcd values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void ReadRGBfifo(*v0, *v1, *v2)
CVECTOR *v0, *v1, *v2;
```

Arguments

v0, *v1*, *v2* Pointer to vectors (output)

Explanation

This function stores the RGBcd0, RGBcd1, and RGBcd2 values in *v0*, *v1*, and *v2*.

The argument format is as follows:

v0, *v1*, *v2* -> r, g, b, cd: (0, 8, 0)

Return value

None

Remarks

See also:

ReadRotMatrix

Reads a constant rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void ReadRotMatrix(*m)
MATRIX *m;
```

Arguments

m Pointer to matrix (output)

Explanation

This function reads the current rotation matrix, and saves it in *m*.

The argument format is as follows:

m -> m [i] [j] : (1, 3, 12)

m -> t [i] : (1, 31, 0)

Return value

None

Remarks

See also:

ReadSXSyfifo

Reads SXSy values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void ReadSXSyfifo(*sxy0, *sxy1, *sxy2)
long *sxy0, *sxy1, *sxy2;
```

Arguments

sxy0, *sxy1*, *sxy2* Pointer to addresses where SZ values are stored

Explanation

This function stores the *sx0*, *sy0*, *sx1*, *sy1*, *sx2*, and *sy2* values in *sxy0*, *sxy1*, and *sxy2*.

The argument format is as follows:

(*sxy0*, *sxy1*, *sxy2*) : (1, 15, 0)

Return value

None

Remarks

See also:

ReadSZfifo3

Reads SZ values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void ReadSZfifo3(*sz0, *sz1, *sz2)
long *sz0, *sz1, *sz2;
```

Arguments

sz0, *sz1*, *sz2* Pointer to addresses where SZ values are stored

Explanation

This function stores the *sz0*, *sz1*, and *sz2* values in *sz0*, *sz1*, and *sz2*.

The argument format is as follows:

(*sz0*, *sz1*, *sz2*) : (0, 16, 0)

Return value

None

Remarks

See also:

ReadSZfifo4

Reads SZ values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void ReadSZfifo4(*szx, *sz0, *sz1, *sz2)
```

```
long *szx, *sz0, *sz1, *sz2;
```

Arguments

szx, sz0, sz1, sz2 Pointer to addresses where SZ values are stored

Explanation

This function stores the *szx, sz0, sz1, and sz2* values in *szx, sz0, sz1, and sz2*.

The argument format is as follows:

(*szx, sz0, sz1, sz2*) : (0, 16, 0)

Return value

None

Remarks

See also:

RotAverage3

Performs coordinate transformation for 3 points and perspective transformation, finds an interpolation value and an average of Z values for depth cueing and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotAverage3(*v0, *v1, *v2, *sxy0, *sxy1, *sxy2, *p, *flag)
SVECTOR *v0, *v1, *v2;
long *sxy0, *sxy1, *sxy2;
long *p;
long *flag;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vectors (input)
<i>sxy0, sxy1, sxy2</i>	Pointer to address where the coordinates will be stored
<i>p</i>	Pointer to address where the interpolation value will be stored
<i>flag</i>	Pointer to address where a flag will be stored

Explanation

A coordinate transformation of three points *v0, v1, v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sxy0, sxy1, and sxy2* are returned. An interpolation value for depth cueing on *v2* to *p* is also returned. The return value becomes the average of three screen coordinate Z values.

The argument format is as follows:

<i>v0, v1, v2</i>	-> vx, vy, vz : (1, 15, 0)
<i>sxy0, sxy1, sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>flag</i>	: (0, 32, 0)

Return value

OTZ value

Remarks

See also:

RotAverage3_nom

Performs coordinate transformation for 3 points and perspective transformation, finds an interpolation value and an average of Z values for depth cueing, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

long RotAverage3_nom

SVECTOR *v0, *v1, *v2

Arguments

v0, v1, v2 Pointer to vectors (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, and sz2, the interpolation value p for depth cueing corresponding to v2, and an average of Z values (otz) for the three screen coordinates in GTE's internal register.

The argument format and internal data format are as follows:

v0, v1, v2 -> vx, vy, vz : (1, 15, 0)
 sxy0, sxy0, sxy0 : (1, 15, 0), (1, 15, 0) (0, 16, 0)
 sxy1, sxy1, sxy1 : (1, 15, 0), (1, 15, 0) (0, 16, 0)
 sxy2, sxy2, sxy2 : (1, 15, 0), (1, 15, 0) (0, 16, 0)
 p : (0, 20, 12)
 flag : (0, 32, 0)

Return value

None

Remarks

(sz0,sz1,sz2) is read by macro read_sz_fifo3, \bullet ((sx0,sy0),(sx1,sy1),(sx2,sy2)) is read by macro read_sxsy_fifo3, \bullet p is read by macro read_p and otz is read by macro read_otz. flag is returned in v0.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotAverage4

Perform coordinate transformation for 3 points and perspective transformation, and find an interpolation value and an average of Z values for depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
long RotAverage4(*v0, *v1, *v2, *v3, *sxy0, *sxy1, *sxy2, *sxy3, *p, *flag)
SVECTOR *v0, *v1, *v2, *v3;
long *sxy0, *sxy1, *sxy2, *sxy3;
long *p;
long *flag;
```

Arguments

<i>v0</i> , <i>v1</i> , <i>v2</i> , <i>v3</i>	Pointer to vectors (input)
<i>sxy0</i> , <i>sxy1</i> , <i>sxy2</i> , <i>sxy3</i>	Pointer to address where the coordinates will be stored
<i>p</i>	Pointer to address where the interpolation value will be stored
<i>flag</i>	Pointer to address where a flag will be stored

Explanation

A coordinate transformation of four points *v0*, *v1*, *v2* and *v3* is performed using a rotation matrix. Next a perspective transformation is performed and four screen coordinates *sxy0*, *sxy1*, *sxy2*, and *sxy3* are returned. An interpolation value for depth cueing on *v2* to *p* is also returned.

The argument format is as follows:

<i>v0</i> , <i>v1</i> , <i>v2</i> , <i>v3</i> -> vx, vy, vz	: (1, 15, 0)
<i>sxy0</i> , <i>sxy1</i> , <i>sxy2</i> , <i>sxy3</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>flag</i>	: (0, 32, 0)

Return value

1/4 (OTZ value) average of four screen coordinate Z values.

Remarks

See also:

RotAverageNclip3

Performs coordinate transformation and perspective transformation for three points, finds an interpolation value, average of Z values, and outer product, and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotAverageNclip3(*v0, *v1, *v2, *sxy0, *sxy1, *sxy2, *p, *otz, *flag)
SVECTOR *v0, *v1, *v2;
long *sxy0, *sxy1, *sxy2;
long *p;
long *otz;
long *flag;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vectors (input)
<i>sxy0, sxy1, sxy2</i>	Pointer to address where coordinates will be stored
<i>p</i>	Pointer to address where an interpolation value will be stored
<i>otz</i>	Pointer to address where an OTZ value will be stored
<i>flag</i>	Pointer to address where a flag will be stored

Explanation

A coordinate transformation of three points *v0, v1, v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sxy0, sxy1, and sxy2* are returned. An interpolation value for depth cueing on *v2* to *p* is also returned. Finally, we also receive 1/4 of the Z value of the screen coordinates for *v2* to *otz*.

The argument format is as follows:

<i>v0, v1, v2 -> vx, vy, vz</i>	: (1, 15, 0)
<i>sxy0, sxy1, sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>otz</i>	: (0, 32, 0)
<i>flag</i>	: (0, 32, 0)

Return value

Outer product of (sx0, sy0), (sx1, sy1), (sx2, sy2).

Remarks

When the return value is negative, SX, SY, etc. are incorrect. When SX and SY are required, use RotAverage3().

See also:

RotAverageNclip3_nom

Performs coordinate transformation and perspective transformation for three points, finds an interpolation value, average of Z values, and outer product, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

long RotAverageNclip3_nom

SVECTOR *v0, *v1, *v2;

Arguments

v0, v1, v2 Pointer to vectors (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, and sz2, the interpolation value p for depth cueing corresponding to v2, an average of Z values (otz) for the three screen coordinates, and an outer product value (opz) for (sx0,sy0), (sx1,sy1), and (sx2,sy2) in GTE's internal register.

The argument format and data format are as follows:

v0, v1, v2 -> vx, vy, vz	: (1, 15, 0)
sy0, sx0, sz0	: (1, 15, 0), (1, 15, 0), (0, 16, 0)
sx1, sy1, sz1	: (1, 15, 0), (1, 15, 0) (0, 16, 0)
sx2, sy2, sz2	: (1, 15, 0), (1, 15, 0) (0, 16, 0)
p	: (0, 20, 12)
otz	: (0, 32, 0)
flag	: (0, 32, 0)

Return value

None

Remarks

(sz0,sz1,sz2) is read by macro `_sz_fifo3`, ((sx0,sy0),(sx1,sy1),(sx2,sy2)) is read by macro `_sxsy_fifo3`, p is read by macro `read_p`, otz is read by macro `read_otz`, and opz is read by macro `read_opz`.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotAverageNclip4

Performs a coordinate transformation and perspective transformation for four points; finds an interpolation value, average of Z values, and outer product.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
long RotAverageNclip4(*v0, *v1, *v2, *v3, *sxy0, *sxy1, *sxy2, *sxy3, *p, *otz, *flag)
SVECTOR *v0, *v1, *v2, *v3;
long *sxy0, *sxy1, *sxy2, *sxy3;
long *p;
long *otz;
long *flag;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vectors (input)
<i>sxy0, sxy1, sxy2, sxy3</i>	Pointer to address where coordinates will be stored
<i>p</i>	Pointer to address where an interpolation value will be stored
<i>otz</i>	Pointer to address where an OTZ value will be stored
<i>flag</i>	Pointer to address where a flag will be stored

Explanation

A coordinate transformation of four points *v0*, *v1*, *v2*, and *v3* is performed using a rotation matrix. Next a perspective transformation is performed and four screen coordinates *sxy0*, *sxy1*, *sxy2* and *sxy3* are returned. An interpolation value for depth cueing on *v2* to *p* is also returned. Finally, we also receive 1/4 of the Z value of the screen coordinates for *v2* to *otz*.

The argument format is as follows:

<i>v0, v1, v2, v3</i> -> vx, vy, vz	: (1, 15, 0)
<i>sxy0, sxy1, sxy2, sxy3</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>otz</i>	: (0, 32, 0)
<i>flag</i>	: (0, 32, 0)

Return value

Outer product of (sx0, sy0), (sx1, sy1), (sx2, sy2).

Remarks

When the return value is negative, SX, SY, etc., are incorrect. When SX and SY are required, use RotAverage4().

See also:

RotAverageNclipColorCol3

Performs a coordinate transformation for three points, perspective transformation, finds a color and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotAverageNclipColorCol3(*v0, *v1, *v2, *v3, *v4, *v5, *v6, *sxy0, *sxy1, *sxy2, *v7, *v8, *v9, *otz,
*flag)
SVECTOR *v0, *v1, *v2;
SVECTOR *v3, *v4, *v5;
CVECTOR *v6;
long *sxy0, *sxy1, *sxy2;
CVECTOR *v7, *v8, *v9;
long *otz;
long *flag;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vectors (input)
<i>v3, v4, v5</i>	Pointer to normal vectors (input)
<i>v6</i>	Pointer to primary color vector (input)
<i>sxy0, sxy1, sxy2</i>	Pointer to address where coordinate values will be stored
<i>v7, v8, v9</i>	Pointer to color vectors (output)
<i>otz</i>	Pointer to address where an OTZ value will be stored
<i>flag</i>	Pointer to address where a flag will be stored

Explanation

A coordinate transformation of three points *v0, v1, v2* is performed using a rotation matrix. Next a perspective transformation is performed and four screen coordinates *sxy0, sxy1, sxy2* are returned. The remaining values are calculated as follows:

$$(LLV0, LLV1, LLV2) = LLM \times (v3, v4, v5)$$

$$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$$

$$(v7, v8, v9) = v6 \times (LC0, LC1, LC2)$$

(separate multiplications)

The function also returns an average of Z values of three screen coordinates to *otz*.

The argument format is as follows:

<i>v0, v1, v2</i> -> vx, vy, vz	: (1, 15, 0)
<i>v3, v4, v5</i> -> vx, vy, vz	: (1, 3, 12)
<i>v6</i> -> r, g, b	: (0, 8, 0)
<i>sxy0, sxy1, sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>v7, v8, v9</i> -> r, g, b	: (0, 8, 0)
<i>otz</i>	: (0, 32, 0)
<i>flag</i>	: (0, 32, 0)

Return value

Outer product of (sx0, sy0), (sx1, sy1), (sx2, sy2)

Remarks

When the return value is negative, SX, SY, etc., are incorrect.

See also:

RotAverageNclipColorCol3_nom

Performs a coordinate transformation for three points, perspective transformation, finds a color, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

long RotAverageNclipColorCol3_nom

SVECTOR *v0, *v1, *v2;

SVECTOR *v3, *v4, *v5;

CVECTOR *v6;

Arguments

v0, v1, v2 Pointer to vectors (input)
v3, v4, v5 Pointer to normal vectors (input)
v6 Pointer to primary color vector (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, and sz2, an average of Z values (otz) for the three screen coordinates, and an outer product value (opz) for (sx0,sy0), (sx1,sy1), and (sx2,sy2) in GTE's internal register.

$(LLV0, LLV1, LLV2) = LLM \times (v3, v4, v5)$

$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$

$(v7, v8, v9) = v6 \times (LC0, LC1, LC2)$

The argument format is as follows:

v0, v1, v2 -> vx, vy, vz : (1, 15, 0)

v3, v4, v5 -> vx, vy, vz : (1, 3, 12)

sx0, sy0,sz0 : (1,15,0), (1,15,0), (0,16,0)

sx1, sy1,sz1 : (1,15,0), (1,15,0), (0,16,0)

sx2, sy2,sz2 : (1,15,0), (1,15,0), (0,16,0)

v6 -> r, g, b : (0, 8, 0)

v7, v8, v9 -> r,g,b: (0,8,0)

otz: (0,32,0)

flag: (0,32,0)

Return value

None

Remarks

(sz0,sz1,sz2) is read by macro read_sz_fifo3, ((sx0,sy0),(sx1,sy1),(sx2,sy2)) is read by macro read_sxsy_fifo3, ((r0,g0,b0), (r1,g1,b1), (r2,g2,b2)) is read by macro read_rgb_fifo, p is read by macro read_p, otz is read by macro read_otz and opz is read by macro read_opz. flag is returned in v0.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotAverageNclipColorDpq3

Performs coordinate transformation for three points, perspective transformation, depth cueing and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotAverageNclipColorDpq3(*v0, *v1, *v2, *v3, *v4, *v5, *v6, *sxy0, *sxy1, *sxy2, *v7, *v8, *v9, *otz,
*flag)
SVECTOR *v0, *v1, *v2;
SVECTOR *v3, *v4, *v5;
CVECTOR *v6;
long *sxy0, *sxy1, *sxy2;
CVECTOR *v7, *v8, *v9;
long *otz;
long *flag;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vectors (input)
<i>v3, v4, v5</i>	Pointer to normal vectors (input)
<i>v6</i>	Pointer to primary color vector (input)
<i>sxy0, sxy1, sxy2</i>	Pointer to address where coordinate values will be stored
<i>v7, v8, v9</i>	Pointer to color vectors (output)
<i>otz</i>	Pointer to address where an OTZ value will be stored
<i>flag</i>	Pointer to address where a flag will be stored

Explanation

A coordinate transformation of three points *v0, v1, v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sxy0, sxy1, and sxy2* are returned. The function uses the interpolation value *p* for depth cueing; *p* is found by the following calculations:

$$(LLV0, LLV1, LLV2) = LLM \times (v3, v4, v5)$$

$$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$$

$$(v7, v8, v9) = (1-p) \times v6 \times (LC0, LC1, LC2) + p \times FC$$

where $v6 \times (LC0, LC1, LC2)$ indicates a separate multiplication.

The function also returns an average of the Z values of the three screen coordinates to *otz*.

The argument format is as follows:

<i>v0, v1, v2</i> -> vx, vy, vz	: (1, 15, 0)
<i>v3, v4, v5</i> -> vx, vy, vz	: (1, 3, 12)
<i>v6</i> -> r, g, b	: (0, 8, 0)
<i>sxy0, sxy1, sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>v7, v8, v9</i> -> r, g, b	: (0, 8, 0)
<i>otz</i>	: (0, 32, 0)
<i>flag</i>	: (0, 32, 0)

Return value

Outer product of (sx0, sy0), (sx1, sy1), (sx2, sy2)

Remarks

When the return value is negative, SX, SY, etc. are incorrect.

See also:

RotAverageNclipColorDpq3_nom

Performs coordinate transformation for three points, perspective transformation, and depth cueing, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

long RotAverageNclipColorDpq3_nom

SVECTOR *v0, *v1, *v2;

SVECTOR *v3, *v4, *v5;

CVECTOR *v6;

Arguments

v0, v1, v2 Pointer to vectors (input)
v3, v4, v5 Pointer to normal vectors (input)
v6 Pointer to primary color vector (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, and sz2, an average of Z values (otz) for the three screen coordinates, and an outer product value (opz) for (sx0,sy0), (sx1,sy1), and (sx2,sy2) in GTE's internal register. The interpolation value p is used in the calculation below for the desired depth cueing.

$$(LLV0, LLV1, LLV2) = LLM \times (v3, v4, v5)$$

$$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$$

$$(v7, v8, v9) = (1-p) \times v6 \times (LC0, LC1, LC2) + p \times FC$$

The argument and internal data format is as follows:

v0, v1, v2 -> vx, vy, vz : (1, 15, 0)

v3, v4, v5 -> vx, vy, vz : (1, 3, 12)

v6 -> r, g, b : (0, 8, 0)

sx0, sy0, sz0 : (1,15,0), (1,15,0), (0,16,0)

sx1, sy1, sz1: (1,15,0), (1,15,0), (0,16,0)

sx2, sy2, sz2: (1,15,0), (1,15,0), (0,16,0)

v7,v8,v9 -> r,g,b: (0,8,0)

otz: (0,32,0)

flag: (0,32,0)

Return value

None

Remarks

(sz0,sz1,sz2) is read by macro read_sz_fifo3,((sx0,sy0),(sx1,sy1),(sx2,sy2)) is read by macro read_sxsy_fifo3, ((r0,g0,b0), (r1,g1,b1), (r2,g2,b2)) is read by macro read_rgb_fifo, p is read by macro read_p, otz is read by macro read_otz and opz is read by macro read_opz. flag is returned in v0.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotColorDpq

Performs coordinate transformation for one point, perspective transformation, depth cueing and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotColorDpq(*v0, *v1, *v2, *sxy, *v3, *flag)
SVECTOR *v0;
SVECTOR *v1;
CVECTOR *v2;
long *sxy;
CVECTOR *v3;
long *flag;
```

Arguments

v0 Pointer to vector (input)
v1 Pointer to normal vector (input)
v2 Pointer to primary color vector (input)
sxy Pointer to address where coordinate values will be stored
v3 Pointer to color vector (output)
flag Pointer to address where a flag will be stored

Explanation

A coordinate transformation for the point *v0* is performed using a rotation matrix. Next a perspective transformation is performed and the screen coordinate *sxy* is returned. The function uses the interpolation value *p* for depth cueing, which is found by the following calculations:

$$LLV = LLM \times v1$$

$$LC = BK + LCM \times LLV$$

$$v3 = (1-p) \times v2 \times LC + p \times FC$$

where $v2 \times LC$ indicates a separate multiplication.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 15, 0)
v1 -> vx, vy, vz : (1, 3, 12)
v2 -> r, g, b : (0, 8, 0)
sxy : (1, 15, 0), (1, 15, 0)
v3 -> r, g, b : (0, 8, 0)
flag : (0, 32, 0)

Return value

1/4 of the Z component sz of the screen coordinates

Remarks

See also:

RotColorDpq_nom

Performs coordinate transformation for one point, perspective transformation, and depth cueing, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

long RotColorDpq_nom

SVECTOR *v0;

SVECTOR *v1;

CVECTOR *v2;

Arguments

v0 Pointer to vector (input)

v1 Pointer to normal vector (input)

v2 Pointer to primary color vector (input)

Explanation

A coordinate transformation for the point *v0* is performed using a rotation matrix. Next a perspective transformation is performed and the screen coordinates (sx,sy,sz) are stored in the GTE internal register. The function uses the interpolation value *p* for depth cueing and stores the obtained color vector *v3* in the internal register which is found by the following calculations:

$$LLV = LLM \times v1$$

$$LC = BK + LCM \times LLV$$

$$v3 = (1-p) \times v2 \times LC + p \times FC$$

where $v2 \times LC$ indicates a separate multiplication.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 15, 0)

v1 -> vx, vy, vz : (1, 3, 12)

sx,sy,sz : (1,15,0), (1,15,0), (0,16,0)

v2 -> r, g, b : (0, 8, 0)

v3 -> r, g, b : (0, 8, 0)

flag : (0, 32, 0)

Return value

None

Remarks

sz is read by macro read_sz2, (sx,sy) is read by macro read_sxsy2, p is read by macro read_p and v3 is read by macro read_rgb2. flag is returned in register v0.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotColorDpq3

Performs coordinate transformation for three points, perspective transformation, depth cueing and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotColorDpq3(*v0, *v1, *v2, *v3, *v4, *v5, *v6, *sxy0, *sxy1, *sxy2, *v7, *v8, *v9, *flag)
SVECTOR *v0, *v1, *v2;
SVECTOR *v3, *v4, *v5;
CVECTOR *v6;
long *sxy0, *sxy1, *sxy2;
CVECTOR *v7, *v8, *v9;
long *flag;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vectors (input)
<i>v3, v4, v5</i>	Pointer to normal vectors (input)
<i>v6</i>	Pointer to primary color vector (input)
<i>sxy0, sxy1, sxy2</i>	Pointer to address where coordinate values will be stored
<i>v7, v8, v9</i>	Pointer to color vectors (output)
<i>flag</i>	Pointer to address where a flag will be stored

Explanation

A coordinate transformation of three points *v0, v1, v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sxy0, sxy1, and sxy2* are returned. The function uses the interpolation value *p* for depth cueing, which is found by the following calculations:

$$LLV0, LLV1, LLV2 = LLM \times (v3, v4, v5)$$

$$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$$

$$(v7, v8, v9) = (1-p) \times v6 \times (LC0, LC1, LC2) + p \times FC$$

where $v6 \times (LC0, LC1, LC2)$ indicates a separate multiplication.

The argument format is as follows:

<i>v0, v1, v2</i> -> vx, vy, vz	: (1, 15, 0)
<i>v3, v4, v5</i> -> vx, vy, vz	: (1, 3, 12)
<i>v6</i> -> r, g, b	: (0, 8, 0)
<i>sxy0, sxy1, sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>v7, v8, v9</i> -> r, g, b	: (0, 8, 0)
<i>flag</i>	: (0, 32, 0)

Return value

1/4 of the Z component sz of the screen coordinates.

Remarks

See also:

RotColorDpq3_nom

Performs coordinate transformation for three points, perspective transformation, and depth cueing, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

long RotColorDpq3_nom

SVECTOR *v0, *v1, *v2;

SVECTOR *v3, *v4, *v5;

CVECTOR *v6;

Arguments

v0, v1, v2	Pointer to vectors (input)
v3, v4, v5	Pointer to normal vectors (input)
v6	Pointer to primary color vector (input)

Explanation

A coordinate transformation of three points *v0*, *v1*, *v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sx0*, *sy0*, *sz0*, *sx1*, *sy1*, *sz1*, *sx2*, *sy2* and *sz2* are stored in the GTE internal register. The function uses the interpolation value *p* for depth cueing and stores the obtained color vector in the internal register which is found by the following calculations:

$$LLV0, LLV1, LLV2 = LLM \times (v3, v4, v5)$$

$$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$$

$$(v7, v8, v9) = (1-p) \times v6 \times (LC0, LC1, LC2) + p \times FC$$

The argument format is as follows:

v0, v1, v2 -> vx, vy, vz	: (1, 15, 0)
v3, v4, v5 -> vx, vy, vz	: (1, 3, 12)
v6 -> r, g, b	: (0, 8, 0)
sxy0, sxy1, sxy2	: (1, 15, 0), (1, 15, 0)
v7, v8, v9 -> r, g, b	: (0, 8, 0)
flag	: (0, 32, 0)

Return value

None

Remarks

(*sz0*, *sz1*, *sz2*) is read by macro `read_sz_fifo3`, ((*sx0*, *sy0*), (*sx2*, *sy2*), (*sx2*, *sy2*)) is read by macro `read_sxsy_fifo3`, *p* is read by macro `read_p` and *v7*, *v8*, *v9* is read by macro `read_rgb_fifo`. *flag* is returned in register *v0*.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotColorMatDpq

Coordinate transformation, perspective transformation, and depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
long RotColorMatDpq(*v0, *v1, *v2, *sxy, *v3, matc, flag)
SVECTOR *v0;
SVECTOR *v1;
SVECTOR *v2;
long *sxy;
CVECTOR *v3;
long matc;
long flag;
```

Arguments

v0 Pointer to vector (input)
v1 Pointer to normal vector (input)
v2 Pointer to primary color vector (input)
sxy Pointer to address where coordinate values will be stored
v3 Pointer to color vector (output)
matc Material (input)
flag Address where a flag will be stored

Explanation

A coordinate transformation for the point *v0* is performed using a rotation matrix. Next a perspective transformation is performed and the coordinate *sxy* is returned. The function uses the interpolation value *p*, found by the following calculations, for depth cueing.

$$LLV = LLM \times v1$$

$$LLV = LLV^{(2^{matc})}$$

$$LC = BK + LCM \times LLV$$

$$v3 = (1-p) \times v2 \times LC + p \times FC$$

where $v2 \times LC$ indicates a separate multiplication.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 15, 0)
v1 -> vx, vy, vz : (1, 3, 12)
v2 -> r, g, b : (0, 8, 0)
sxy : (1, 15, 0), (1, 15, 0)
v3 -> r, g, b : (0, 8, 0)
matc : (0, 32, 0)
flag : (0, 32, 0)

Return value

1/4 of the Z component sz of screen coordinates.

Remarks

See also:

RotMatrix...

Determines rotation matrix from a rotation angle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.3	6/22/98

Syntax

```

MATRIX *RotMatrix (
SVECTOR *r,
MATRIX *m
)
MATRIX *RotMatrixXZY (
SVECTOR *r,
MATRIX *m
)
MATRIX *RotMatrixYXZ (
SVECTOR *r,
MATRIX *m
)
MATRIX *RotMatrixYZX (
SVECTOR *r,
MATRIX *m
)
MATRIX *RotMatrixZXY (
SVECTOR *r,
MATRIX *m??)
MATRIX *RotMatrixZYX (
SVECTOR *r,
MATRIX *m
)

```

Arguments

r Input: rotation angle
m Output: rotation matrix

Explanation

Matrix m is set to a rotation matrix according to the rotation angle (r[0],r[1],r[2]).

A rotation angle value of 4096 is equivalent to 360 degrees. A matrix element value of 4096 is equivalent to 1.0.

When matrix is:

```

c0 = cos(r[0]),   s0 = sin(r[0])
c1 = cos(r[1]),   s1 = sin(r[1])
c2 = cos(r[2]),   s2 = sin(r[2])

```

$$mX = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c0 & -s0 \\ 0 & s0 & c0 \end{bmatrix} \quad mY = \begin{bmatrix} c1 & 0 & s1 \\ 0 & 1 & 0 \\ -s1 & 0 & c1 \end{bmatrix} \quad mZ = \begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

it is the result of the following product:

Function name	Matrix calculation formula	Rotation sequence
RotMatrix	$mX \times mY \times mZ$	Z axis -> Y axis -> X axis
RotMatrixXZY	$mX \times mZ \times mY$	Y axis -> Z axis -> X axis
RotMatrixYXZ	$mY \times mX \times mZ$	Z axis -> X axis -> Y axis
RotMatrixYZX	$mY \times mZ \times mX$	X axis -> Z axis -> Y axis
RotMatrixZXY	$mZ \times mX \times mY$	Y axis -> X axis -> Z axis
RotMatrixZYX	$mZ \times mY \times mX$	X axis -> Y axis -> Z axis

In GTE coordinate conversion functions (such as RotTransPers), the vector is applied from the right side. For example, with RotMatrix, the rotation is performed in the following sequence: Z axis, Y axis, X axis.

Parameter format:

$m \rightarrow m[i][j] : (1,3,12)$

$r \rightarrow vx,vy,vz : (1,3,12)$ (where 360 degrees is 1.0)

Return value

m

Remarks

RotMatrixC is slow and results in small tables.

See also:

RotMatrix_gte

Finds a rotation matrix from a rotation angle. Approximately 2 X faster than RotMatrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
MATRIX *RotMatrix_gte (
SVECTOR *r,
MATRIX *m
)
```

Arguments

r Pointer to rotation angle (input)
m Pointer to rotation matrix (output)

Explanation

This function generates a rotation queue from the rotation angle (*r*[0], *r*[1], *r*[2]) in matrix *m*. A value of 4096 represents 360 degrees; and in matrices, 4096 represents 1.0.

The matrix is obtained by doing the following multiplication. In a coordinate conversion function (such as RotTransPers) for GTE, a vector is multiplied beginning with the rightmost end. So, it is rotated around the Z-, Y-, and X-axes.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c0 & -s0 \\ 0 & s0 & c0 \end{bmatrix} \times \begin{bmatrix} c1 & 0 & s1 \\ 0 & 1 & 0 \\ -s1 & 0 & c1 \end{bmatrix} \times \begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

However,

c0=cos (*r*[0]), *s0*=sin (*r*[0])
c1=cos (*r*[1]), *s1*=sin (*r*[1])
c2=cos (*r*[2]), *s2*=sin (*r*[2])

The argument format is as follows:

m -> *m* [*i*] [*j*] : (1, 3, 12)
r -> *vx*, *vy*, *vz* : (1, 3, 12) (where 1.0 stands for 360 degrees)

Return value

m

Remarks

RotMatrix_gte is approximately 2 X faster than RotMatrix but the result is different by 2/4096 or less.
RotMatrix uses the same sincos table.

See also:

RotMatrixC

Finds a rotation matrix from a rotation angle. Same as RotMatrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	10/01/97

Syntax

```
MATRIX *RotMatrixC (
SVECTOR *r,
MATRIX *m
)
```

Arguments

r Pointer to rotation angle (input)
m Pointer to rotation matrix (output)

Explanation

Same as RotMatrix()

Return value

This function returns *m*.

Remarks

By using RotMatricC, a small table and slower speed will result.

See also:

RotMatrixX

Finds a rotation matrix around the X axis.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	10/01/97

Syntax

```
MATRIX *RotMatrixX (
long r,
MATRIX *m
)
```

Arguments

- r* Rotation angle (input)
- m* Pointer to rotation matrix (output)

Explanation

This function generates a rotation queue in matrix *m* as the product of a rotation matrix around the X axis at rotation angle *r*. A value of 4096 represents a rotation angle of 360 degrees and as a matrix element, 4096 represents 1.0.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} \times m$$

However,
c = cos (*r*), *s* = sin (*r*)

The argument format is as follows:

- m* -> m [*i*] [*j*] : (1, 3, 12)
- r* : (1, 3, 12) (where 1.0 stands for 360 degrees)

Return value

m

Remarks

See also:

RotMatrixY

Find a rotation matrix around the Y axis.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	10/01/97

Syntax

```
MATRIX *RotMatrixY (  
long r,  
MATRIX *m  
)
```

Arguments

r Rotation angle (input)
m Pointer to rotation matrix (input/output)

Explanation

This function generates a rotation queue in matrix *m* as the product of a rotation matrix around the Y axis at rotation angle *r*. A value of 4096 represents a rotation angle of 360 degrees and as a matrix element, 4096 represents 1.0.

$$\begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix} \times m$$

However,

$$c = \cos (r), \quad s = \sin (r)$$

The argument format is as follows:

m -> *m* [*i*] [*j*] : (1, 3, 12)

r : (1, 3, 12) (where 1.0 stands for 360 degrees)

Return value

m

Remarks

See also:

RotMatrixYXZ_gte

Finds a rotation matrix from a rotation angle. Approximately 2 X faster than RotMatrixYXZ.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
MATRIX *RotMatrixYXZ_gte (
SVECTOR *r,
MATRIX *m
)
```

Arguments

r Pointer to rotation angle (input)
m Pointer to rotation matrix (output)

Explanation

This function generates a rotation queue in matrix *m* from the rotation angle (*r*[0], *r*[1], *r*[2]). A value of 4096 represents a rotation angle of 360 degrees, and as a matrix element, 4096 represents 1.0.

The matrix is found by performing the following multiplication. In GTE's coordinate transformation functions (such as RotTransPers()) a vector is multiplied beginning with the rightmost end. This produces rotation around the Z axis, Y axis, and X axis.

$$\begin{bmatrix} c1 & 0 & s1 \\ 0 & 1 & 0 \\ -s1 & 0 & c1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & c0 & -s0 \\ 0 & s0 & c0 \end{bmatrix} \times \begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

However,

$c0 = \cos(r[0]), \quad s0 = \sin(r[0])$

$c1 = \cos(r[1]), \quad s1 = \sin(r[1])$

$c2 = \cos(r[2]), \quad s2 = \sin(r[2])$

The argument format is as follows:

m -> *m* [i] [j] : (1, 3, 12)

r -> vx, vy, vz : (1, 3, 12) (where 1.0 stands for 360 degrees)

Return value

m

Remarks

RotMatrixYXZ_gte is approximately 2 X faster than RotMatrixYXZ but the result is different by 2/4096 or less.

RotMatrixYXZ uses the same sincos table.

See also:

RotMatrixZ

Finds a rotation matrix around the Z axis.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	10/01/97

Syntax

```
MATRIX *RotMatrixZ (  
long r,  
MATRIX *m  
)
```

Arguments

r Rotation angle input
m Pointer to rotation matrix output

Explanation

This function generates a rotation queue in matrix *m* as the product of a rotation matrix around the Z axis at rotation angle *r*. A value of 4096 represents a rotation angle of 360 degrees and as a matrix element, 4096 represents 1.0.

$$\begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \times m$$

However,

$$c = \cos (r), \quad s = \sin (r)$$

The argument format is as follows:

m -> *m* [*i*] [*j*] : (1, 3, 12)

r : (1, 3, 12) (where 1.0 stands for 360 degrees)

Return value

This function returns *m*.

Remarks

See also:

RotMatrixZYX_gte

Find a rotation matrix around the z, y, and x axis. Approximately 2 X faster than RotMatrixZYX.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
MATRIX *RotMatrixZYX_gte (
SVECTOR *r,
MATRIX *m
)
```

Arguments

r Rotation angle (input)
m Pointer to rotation matrix (output)

Explanation

This function generates a rotation queue from the rotation angle (*r*[0], *r*[1], *r*[2]) in matrix *m*. A value of 4096 represents 360 degrees; and in matrices, 4096 represents 1.0.

The matrix is obtained by doing the following multiplication. In a coordinate conversion function (such as RotTransPers) for GTE, a vector is multiplied beginning with the rightmost end. So, it is rotated around the X axis, Y axis, and Z axis.

$$\begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} c1 & 0 & s1 \\ 0 & 1 & 0 \\ -s1 & 0 & c1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & c0 & -s0 \\ 0 & s0 & c0 \end{bmatrix}$$

However,

$c0 = \cos (r[0]), \quad s0 = \sin (r[0])$

$c1 = \cos (r[1]), \quad s1 = \sin (r[1])$

$c2 = \cos (r[2]), \quad s2 = \sin (r[2])$

The argument format is as follows:

m -> *m* [i] [j] : (1, 3, 12)

r -> vx, vy, vz : (1, 3, 12) (where 1.0 stands for 360 degrees)

Return value

m

Remarks

RotMatrixZYX_gte is approximately 2 X faster than RotMatrixZYX but the result is different by 2/4096 or less.

RotMatrixZYX uses the same sincos table.

See also:

RotMeshH

Performs coordinate transformation and perspective transformation.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
void RotMeshH(*Yheight, *Vo, *sz, *flag, Xoffset, Zoffset, m, n, *base)
short *Yheight;
DVECTOR *Vo;
unsigned short *sz;
unsigned short *flag;
short Xoffset, Zoffset;
short m, n;
DVECTOR *base;
```

Arguments

<i>Yheight</i>	Pointer to vertex Y coordinate (input)
<i>Vo</i>	Pointer to screen coordinate (output)
<i>sz</i>	Pointer to SZ value (output)
<i>flag</i>	Pointer to flag (output)
<i>Xoffset, Zoffset</i>	Offsets for X and Z (input)
<i>m, n</i>	Number of vertices (input)
<i>base</i>	Pointer to base address

Explanation

This function performs coordinate transformation and perspective transformation for the number of quadrilateral mesh vertices indicated by m x n.

Vo, sz and flag are not scalar quantities but represent m x n meshes. In other words, this function returns various vertex parameters such as DVECTOR vo[n][m], u_short sz[n][m] and u_short flag[n][m].

Arguments and internal data format are as follows:

<i>Yheight</i>	: (1, 15, 0)
<i>Vo</i> -> vx, vy	: (1, 15, 0)
<i>sz</i>	: (0, 16, 0)
<i>flag</i>	: (0, 16, 0)
<i>Xoffset, Zoffset</i>	: (1, 15, 0)
<i>m, n</i>	: (1, 15, 0)
<i>base</i>	: (1, 15, 0)

Return value

None

Remarks

The flag must normally be set between bit 27 and bit 12 of the 32-bit flag.

See also:

RotMeshPrimQ_T

Two-dimensional mesh.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
void RotMeshPrimQ_T (
  TMERH *msh,
  POLY_FT4 *prim,
  u_long *ot,
  u_long otlen,
  long dpq,
  long backc,
  SCLIP *sclip,
  LINE_BUF *line_sxy
)
```

Arguments

<i>msh</i>	Pointer to mesh model data
<i>prim</i>	Pointer to GPU packet that should be created
<i>ot</i>	Pointer to ordering table
<i>otlen</i>	Ordering table length
<i>dpq</i>	Decides whether depth cueing will be done
<i>backc</i>	Decides whether back clip will be done
<i>sclip</i>	Pointer to screen clip area
<i>line_sxy</i>	Pointer to one line buffer for internal processing

Explanation

Perform coordinate conversion, perspective conversion, normal line clip, clipping by screen coordinates (x, y, z) and linking to OT of the following two dimensional mesh (qmesh) data.

The H direction vertex number must be a multiple of 3 (msh -> lenh = 3 x n).

```

1-----2-----3
|         |         |
4-----5-----6
|         |         |
7-----8-----9
```

Write texture as is (fog gathers, but do not calculate light source). Set the texture coordinates.

Return value

None

Remarks

Use the following structure. The line buffer is secured above 1H + 3 vertices). If scratch pad is used as a line buffer it will be faster.

```
typedef struct {
  long sminX
  long smaxS
  long sminY
  long smaxY
  long sminZ
  long smaxZ
} SCLIP;
```

8-164 Basic Geometry Library Functions

```
typedef struct {  
    long sxy  
    long code  
} LINE_BUF;
```

See also:

RotMeshPrimR_...

The round mesh series of functions.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	4.1	10/01/97

Syntax

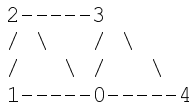
```
void RotMeshPrimR_... (  
  TMESH *msh,  
  POLY_... *prim,  
  u_long *ot,  
  u_long otlen,  
  long dpq,  
  unsigned long backc  
)
```

Arguments

- msh* Pointer to mesh model data
- prim* Pointer to GPU packet that should be created
- ot* Pointer to ordering table
- otlen* Ordering table length
- dpq* Decides whether depth cueing will be done
- backc* Decides whether back clip will be done

Explanation

Perform coordinate conversion, perspective conversion, and linking to *ot* of the following round model mesh (rmesh) data.



The following round mesh functions are supported in libgte:

Function name	Type of <i>prim</i> (2 nd arg)	Description
RotMeshPrimR_F3	POLY_F3	Perform flat shading with one vertex color (light source calculation).
RotMeshPrimR_FC3	POLY_F3	Perform complete painting with one vertex color (no light source calculation).
RotMeshPrimR_FCT3	POLY_FT3	Multiply texture with one vertex color (no light source calculation).
RotMeshPrimR_FT3	POLY_FT3	Multiply the flat-shaded items and the texture with one vertex color (light source calculation).
RotMeshPrimR_G3	POLY_G3	Perform Gouraud shading with vertex color (light source calculation).
RotMeshPrimR_GC3	POLY_G3	Perform complete Gouraud painting with vertex color (no light source calculation).
RotMeshPrimR_GCT3	POLY_GT3	Multiply the Gouraud completely painted items and the texture with vertex color (no light source calculation).
RotMeshPrimR_GT3	POLY_GT3	Multiply the Gouraud-shaded items and the texture with vertex color (light source calculation).
RotMeshPrimR_T3	POLY_FT3	Write out texture as-is.

Return value

None

Remarks**See also:**

RotMeshPrimS_...

The strip mesh series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
void RotMeshPrimS_... (
  TMESH *msh,
  POLY_... *prim,
  u_long *ot,
  u_long otlen,
  long dpq,
  unsigned long backc
)
```

Arguments

msh Pointer to mesh model data
prim Pointer to GPU packet that should be created
ot Pointer to ordering table
otlen Ordering table length
dpq Decides whether depth cueing will be done
backc Decides whether back clip will be done

Explanation

Perform coordinate conversion, perspective conversion, and linking to *ot* of the following strip model mesh (rmesh) data.

```

1-----3-----5
 / \   / \   /
/   \ /   \ /
0-----2-----4
```

The following strip mesh functions are supported in libgte:

Function name	Type of <i>prim</i> (2 nd arg)	Description
RotMeshPrimS_F3	POLY_F3	Perform flat shading with one vertex color (light source calc.).
RotMeshPrimS_FC3	POLY_F3	Perform complete painting with one vertex color (no light source calculation).
RotMeshPrimS_FCT3	POLY_FT3	Multiply texture with one vertex color (no light source calc).
RotMeshPrimS_FT3	POLY_FT3	Multiply the flat-shaded items and the texture with 1 vertex color (light source calculation).
RotMeshPrimS_G3	POLY_G3	Perform Gouraud shading with vertex color (light source calc).
RotMeshPrimS_GC3	POLY_G3	Perform complete Gouraud painting with vertex color (no light source calculation).
RotMeshPrimS_GCT3	POLY_GT3	Multiply the Gouraud completely painted items and the texture with vertex color (no light source calculation).
RotMeshPrimS_GT3	POLY_GT3	Multiply the Gouraud-shaded items and the texture with vertex color (light source calculation).
RotMeshPrimS_T3	POLY_FT3	Write out texture as-is.

Return value

None

Remarks

See also:

RotNclip3

Performs coordinate transformation and perspective transformation for three points, finds an interpolation value and outer product for depth cueing and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotNclip3(*v0, *v1, *v2, *sxy0, *sxy1, *sxy2, *p, *otz, *flag)
SVECTOR *v0, *v1, *v2;
long *sxy0, *sxy1, *sxy2;
long *p;
long *otz;
long *flag;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vectors (input)
<i>sxy0, sxy1, sxy2</i>	Pointer to address where coordinates will be stored
<i>p</i>	Pointer to address where an interpolation value will be stored
<i>otz</i>	Pointer to address where an OTZ value will be stored
<i>flag</i>	Pointer to address where a flag will be stored

Explanation

A coordinate transformation of three points *v0, v1, v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sx0, sx1*, and *sx2* are returned. An interpolation value for depth cueing on *v2* to *p* is also returned. Finally, we also receive 1/4 of the Z value of the screen coordinates for *v2* to *otz*.

The argument format is as follows:

<i>v0, v1, v2 -> vx, vy, vz</i>	: (1, 15, 0)
<i>sxy0, sxy1, sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>otz</i>	: (0, 32, 0)
<i>flag</i>	: (0, 32, 0)

Return value

Outer product of (*sx0, sy0*), (*sx1, sy1*), (*sx2, sy2*)

Remarks

When the return value is negative, SX, SY, etc. are incorrect. When SX and SY are needed, use RotTransPer3().

See also:

RotNclip3_nom

Performs coordinate transformation and perspective transformation for three points, finds an interpolation value and outer product for depth cueing, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

long RotNclip3_nom

SVECTOR *v0, *v1, *v2

Arguments

v0, v1, v2 Pointer to vectors (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, and sz2, the interpolation value p for depth cueing corresponding to v2, and an outer product value (opz) for (sx0,sy0), (sx1,sy1), and (sx2,sy2) in GTE's internal register.

The argument format is as follows:

```
v0, v1, v2 -> vx, vy, vz      : (1, 15, 0)
sx0, sy0, sz0                  : (1, 15, 0), (1, 15, 0), (0,16,0)
sx1, sy1, sz1                  : (1, 15, 0), (1, 15, 0), (0,16,0)
sx2,sy2, sz2                   : (1, 15, 0), (1, 15, 0), (0,16,0)
p                              : (0, 20, 12)
otz                            : (0, 32, 0)
flag                           : (0, 32, 0)
```

Return value

None

Remarks

(sz0,sz1,sz2) is read by macro read_sz_fifo3, ((sx0,sy0), (sx1,sy1), (sx2,sy2)) is read by macro read_sxsy_fifo3, p is read by macro read_p and opz is read by macro read_opz. flag is returned in v0.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotNclip4

Perform coordinate transformation and perspective transformation for four points, and find an interpolation value and outer product for depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
long RotNclip4(*v0, *v1, *v2, *v3, *sxy0, *sxy1, *sxy2, *sxy3, *p, *otz, *flag)
SVECTOR *v0, *v1, *v2, *v3;
long *sxy0, *sxy1, *sxy2, *sxy3;
long *p;
long *otz;
long *flag;
```

Arguments

<i>v0, v1, v2, v3</i>	Pointer to vectors (input)
<i>sxy0, sxy1, sxy2, sxy3</i>	Pointer to address where coordinates will be stored
<i>p</i>	Pointer to address where an interpolation value will be stored
<i>otz</i>	Pointer to address where an OTZ value will be stored
<i>flag</i>	Pointer to address where a flag will be stored

Explanation

A coordinate transformation of four points *v0, v1, v2, v3* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sx0, sx1, sx2, and sx3* are returned. An interpolation value for depth cueing on *v2* to *p* is also returned. Finally, we also receive 1/4 of the Z value of the screen coordinates for *v2* to *otz*.

The argument format is as follows:

<i>v0, v1, v2, v3</i> -> vx, vy, vz	: (1, 15, 0)
<i>sxy0, sxy1, sxy2, sxy3</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>otz</i>	: (0, 32, 0)
<i>flag</i>	: (0, 32, 0)

Return Value

Outer product of (*sx0, sy0*), (*sx1, sy1*), (*sx2, sy2*)

Remarks

When the return value is negative, SX, SY, etc. are incorrect. When SX and SY are required, use RotTransPer4().

See also:

RotPMD_...

The independent vertex PMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
void RotPMD_...(*pa, *ot, otlen, id, backc)
long *pa;
unsigned long *ot;
int otlen, id, backc;
```

Arguments

pa Pointer to starting address of PRIMITIVE Gp
ot Pointer to starting address of OT
otlen Length of OT (number of bits)
id Double buffer ID
backc Normal line clipping ON/OFF (0: ON)

Explanation

These functions perform coordinate transformations and perspective transformations on all three and four-side polygons included in the independent vertex PRIMITIVE Gp, then complete the GPU packet and link it to OT.

Only polygons with an SZ value within the range $[h/2, 2^{16}]$ may be linked.

The following independent vertex PMD functions are supported in libgte:

Function Name	Description
RotPMD_F3	Flat triangle
RotPMD_F4	Flat quadrilateral
RotPMD_FT3	Flat textured triangle
RotPMD_FT4	Flat textured quadrilateral
RotPMD_G3	Gouraud triangle
RotPMD_G4	Gouraud quadrilateral
RotPMD_GT3	Gouraud textured triangle
RotPMD_GT4	Gouraud textured quadrilateral

Return value

None

Remarks

An error may occur when placing model data (PRIMITIVEGp) on the scratch pad.

See also:

RotPMD_SV_...

The shared vertex PMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
void RotPMD_SV_...(*pa, *va, *ot, otlen, id, backc)
long *pa;
long *va;
u_long *ot;
int otlen;
int id;
int backc;
```

Arguments

pa Pointer to starting address of PRIMITIVE Gp
va Pointer to starting address of VERTEX Gp
ot Pointer to starting address of OT
otlen Length of OT (number of bits)
id Double buffer ID
backc Normal line clipping ON/OFF (0: ON)

Explanation

These functions perform coordinate transformations and perspective transformations on all three-sided and four-sided polygons included in the shared vertex PRIMITIVE Gp, then complete the GPU packet and link it to OT.

Only polygons with an SZ value within the range $[h/2, 2^{16}]$ may be linked.

The following shared vertex PMD functions are supported in libgte:

Function Name	Description
RotPMD_SV_F3	Flat triangle
RotPMD_SV_F4	Flat quadrilateral
RotPMD_SV_FT3	Flat textured triangle
RotPMD_SV_FT4	Flat textured quadrilateral
RotPMD_SV_G3	Gouraud triangle
RotPMD_SV_G4	Gouraud quadrilateral
RotPMD_SV_GT3	Gouraud textured triangle
RotPMD_SV_GT4	Gouraud textured quadrilateral

Return value

None

Remarks

See also:

RotRMD_...

The independent vertex RMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
void RotRMD_...(*pa, *ot, otlen, id, sclip, hclip, vclip, nclipmode)
```

```
long *pa;
```

```
u_long *ot;
```

```
int otlen;
```

```
int id;
```

```
int sclip;
```

```
int hclip;
```

```
int vclip;
```

```
int nclipmode;
```

Arguments

pa Pointer to starting address of PRIMITIVE Gp

ot Pointer to starting address of OT

otlen Length of OT (number of bits)

id Double buffer ID

sclip Screen clip ON/OFF (ON=1)

hclip H direction clip ([0,hclip]=display)

vclip V direction clip ([0,vclip]=display)

nclipmode Near Z clip mode (0=0,SCRZ/2=1)

Explanation

These functions perform coordinate transformations and perspective transformations on all three and four-sided polygons included in independent vertex type PRIMITIVE Gp, then create GPU packets and link them to OT.

If sclip=0, all polygons are displayed.

If sclip=1, only polygons having at least one vertex that is included in the square ([0,hclip],[0,vclip]) are displayed.

If nclipmode=0, polygons are far&near clipped by sz=[0,2¹⁶].

If nclipmode=1, polygons are far&near clipped by sz=[h,2¹⁶] (h=distance of eye to screen).

No polygons are backface clipped.

The following independent vertex RMD functions are supported in libgte:

Function Name	Description
RotRMD_F3	Flat triangle
RotRMD_F4	Flat quadrilateral
RotRMD_FT3	Flat textured triangle
RotRMD_FT4	Flat textured quadrilateral
RotRMD_G3	Gouraud triangle
RotRMD_G4	Gouraud quadrilateral
RotRMD_GT3	Gouraud textured triangle
RotRMD_GT4	Gouraud textured quadrilateral

Return value

None

Remarks

An error may occur when placing model data (PRIMITIVEGp) on the scratch pad.

See also:

RotRMD_SV_...

The shared vertex RMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
void RotRMD_SV_...(*pa, *ot, otlen, id, sclip, hclip, vclip, nclipmode)
```

```
long *pa;
u_long *ot;
int otlen;
int id;
int sclip;
int hclip;
int vclip;
int nclipmode;
```

Arguments

<i>pa</i>	Pointer to starting address of PRIMITIVE Gp
<i>ot</i>	Pointer to starting address of OT
<i>otlen</i>	Length of OT (number of bits)
<i>id</i>	Double buffer ID
<i>sclip</i>	Screen clip ON/OFF (ON=1)
<i>hclip</i>	H direction clip ([0,hclip]=display)
<i>vclip</i>	V direction clip ([0,vclip]=display)
<i>nclipmode</i>	Near Z clip mode (0=0,SCRZ/2=1)

Explanation

These functions perform coordinate transformations and perspective transformations on all three and four-sided polygons included in shared vertex type PRIMITIVE Gp, then create GPU packets and link them to OT.

If sclip=0, all polygons are displayed.

If sclip=1, only polygons having at least one vertex that is included in the square ([0,hclip],[0,vclip]) are displayed.

If nclipmode=0, polygons are far&near clipped by $sz=[0,2^{16}]$.

If nclipmode=1, polygons are far&near clipped by $sz=[h,2^{16}]$ (h=distance of eye to screen).

No polygons are backface clipped.

The following shared vertex RMD functions are supported in libgte:

Function Name	Description
RotRMD_SV_F3	Flat triangle
RotRMD_SV_F4	Flat quadrilateral
RotRMD_SV_FT3	Flat textured triangle
RotRMD_SV_FT4	Flat textured quadrilateral
RotRMD_SV_G3	Gouraud triangle
RotRMD_SV_G4	Gouraud quadrilateral
RotRMD_SV_GT3	Gouraud textured triangle
RotRMD_SV_GT4	Gouraud textured quadrilateral

Return value

None

Remarks

See also:

RotSMD_...

The independent vertex SMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
void RotSMD_...(*pa, *ot, otlen, id, sclip, hclip, vclip, nclipmode)
```

```
long *pa;
```

```
u_long *ot;
```

```
int otlen;
```

```
int id;
```

```
int sclip;
```

```
int hclip;
```

```
int vclip;
```

```
int nclipmode;
```

Arguments

<i>pa</i>	Pointer to starting address of PRIMITIVE Gp
<i>ot</i>	Pointer to starting address of OT
<i>otlen</i>	Length of OT (number of bits)
<i>id</i>	Double buffer ID
<i>sclip</i>	Screen clip ON/OFF (ON=1)
<i>hclip</i>	H direction clip ([0,hclip]=display)
<i>vclip</i>	V direction clip ([0,vclip]=display)
<i>nclipmode</i>	Near Z clip mode (0=0,SCRZ/2=1)

Explanation

These functions perform coordinate transformations and perspective transformations on all three and four-sided polygons included in independent vertex type PRIMITIVE Gp, then create GPU packets, and link them to OT.

If sclip=0, all polygons are displayed.

If sclip=1, only polygons with at least one vertex that is included in the square ([0,hclip],[0,vclip]) are displayed.

If nclipmode=0, polygons are far&near clipped by $sz=[0,2^{16}]$.

If nclipmode=1, polygons are far&near clipped by $sz=[h,2^{16}]$ (h =distance of eye to screen).

All polygons are backface clipped.

The following independent vertex SMD functions are supported in libgte:

Function Name	Description
RotSMD_F3	Flat triangle
RotSMD_F4	Flat quadrilateral
RotSMD_FT3	Flat textured triangle
RotSMD_FT4	Flat textured quadrilateral
RotSMD_G3	Gouraud triangle
RotSMD_G4	Gouraud quadrilateral
RotSMD_GT3	Gouraud textured triangle
RotSMD_GT4	Gouraud textured quadrilateral

Return value

None

Remarks

An error may occur when placing model data (PRIMITIVEGp) on the scratch pad.

See also:

RotSMD_SV_...

The shared vertex SMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	10/01/97

Syntax

```
void RotSMD_SV_...(*pa, *ot, otlen, id, sclip, hclip, vclip, nclipmode)
long *pa;
u_long *ot;
int otlen;
int id;
int sclip;
int hclip;
int vclip;
int nclipmode;
```

Arguments

<i>pa</i>	Pointer to starting address of PRIMITIVE Gp
<i>ot</i>	Pointer to starting address of OT
<i>otlen</i>	Length of OT (number of bits)
<i>id</i>	Double buffer ID
<i>sclip</i>	Screen clip ON/OFF (ON=1)
<i>hclip</i>	H direction clip ([0,hclip]=display)
<i>vclip</i>	V direction clip ([0,vclip]=display)
<i>nclipmode</i>	Near Z clip mode (0=0,SCRZ/2=1)

Explanation

These functions perform coordinate transformations and perspective transformations on all three and four-sided polygons included in shared vertex type PRIMITIVE Gp, then create GPU packets, and link them to OT.

If *sclip*=0, all polygons are displayed.

If *sclip*=1, only polygons with at least one vertex that is included in the square ([0,hclip],[0,vclip]) are displayed.

If *nclipmode*=0, polygons are far&near clipped by $sz=[0,2^{16}]$.

If *nclipmode*=1, polygons are far&near clipped by $sz=[h,2^{16}]$ (*h*=distance of eye to screen).

All polygons are backface clipped.

The following shared vertex SMD functions are supported in *libgte*:

Function Name	Description
RotSMD_SV_F3	Flat triangle
RotSMD_SV_F4	Flat quadrilateral
RotSMD_SV_FT3	Flat textured triangle
RotSMD_SV_FT4	Flat textured quadrilateral
RotSMD_SV_G3	Gouraud triangle
RotSMD_SV_G4	Gouraud quadrilateral
RotSMD_SV_GT3	Gouraud textured triangle
RotSMD_SV_GT4	Gouraud textured quadrilateral

Return value

None

Remarks

See also:

RotTrans

Performs coordinate transformation using a rotation matrix and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void RotTrans(*v0, *v1, *flag)
SVECTOR *v0;
VECTOR *v1;
long *flag;
```

Arguments

v0 Pointer to vector (input)
v1 Pointer to vector (output)
flag Pointer to address where a flag is stored

Explanation

This function calculates $v1 = RTM \times v0$.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 15, 0)
v1 -> vx, vy, vz : (1, 31, 0)
flag : (0, 32, 0)

Return value

None

Remarks

See also:

RotTrans_nom

Performs coordinate transformation using a rotation matrix, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
void RotTrans_nom (
SVECTOR *v0
```

Arguments

v0 Pointer to vector (input)

Explanation

This function calculates $v1 = RTM \times v0$.

The argument and internal data format is as follows:

v0 -> vx, vy, vz : (1, 15, 0)

v1 -> vx, vy, vz : (1, 31, 0)

flag : (0, 32, 0)

Return value

None

Remarks

(*v1*->vx,*v1*->vy,*v1*->vz) is read by macro read_mt and *flag* is read by macro read_flag.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotTransPers

Performs coordinate and perspective transformation for one vertice and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotTransPers(*v0, *sxy, *p, *flag)
SVECTOR *v0;
long *sxy;
long *p;
long *flag;
```

Arguments

v0 Pointer to vertex coordinate vector (input)
sxy Pointer to address where the screen coordinates are stored
p Pointer to address where the interpolated value is stored
flag Pointer to address where a flag is stored

Explanation

After converting the coordinate vector *v0* with a rotation matrix, the function performs perspective transformation, and returns screen coordinates *sx*, *sy*. It also returns an interpolated value for depth cueing in *p*.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 15, 0)
sxy : (1, 15, 0), (1, 15, 0)
p : (0, 20, 12)
flag : (0, 32, 0)

Return value

1/4 of the screen coordinate Z component *sz*.

Remarks

See also:

RotTransPers_nom

Performs coordinate and perspective transformation for one vertice, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotTransPers_nom
SVECTOR *v0
```

Arguments

v0 Pointer to vertex coordinate vector (input)

Explanation

After converting the coordinate vector *v0* with a rotation matrix, the function performs perspective transformation, and stores screen coordinates *sx*, *sy*, *sz* and the interpolated value *p* for depth cueing in the GTE internal register.

The argument and internal data format is as follows:

```
v0 -> vx, vy, vz      : (1, 15, 0)
sx                    : (1, 15, 0)
sy                    : (1, 15, 0)
sz                    : (0, 16, 0)
p                     : (0, 20, 12)
flag                  : (0, 32, 0)
```

Return value

None

Remarks

sz is read by macro `read_sz2`, (*sx*,*sy*) is read by macro `read_sxsy2`, *p* is read by macro `read_p` and *flag* is read by macro `read_flag`.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotTransPers3

Performs coordinate transformation of three vertices, perspective transformation and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotTransPers3(*v0, *v1, *v2, *sxy0, *sxy1, *sxy2, *p, *flag)
SVECTOR *v0, *v1, *v2;
long *sxy0, *sxy1, *sxy2;
long *p;
long *flag;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vectors
<i>sxy0, sxy1, sxy2</i>	Pointer to addresses where the screen coordinates are stored
<i>p</i>	Pointer to address where the interpolated value is stored
<i>flag</i>	Pointer to address where a flag is stored

Explanation

After transforming the three coordinate vectors *v0*, *v1*, and *v2* using a rotation matrix, the function performs perspective transformation, and returns three screen coordinates *sxy0*, *sxy1*, and *sxy2*. It also returns to *p* an interpolated value for depth cueing corresponding to *v2*. The argument format is as follows:

<i>v0, v1, v2</i> -> vx, vy, vz	: (1, 15, 0)
<i>sxy0, sxy1, sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>flag</i>	: (0, 32, 0)

Return value

1/4 of the screen coordinate Z component sz corresponding to *v2*.

Remarks

See also:

RotTransPers3_nom

Performs coordinate transformation of three vertices and perspective transformation, but does not return the result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

long RotTransPers3_nom

SVECTOR *v0, *v1, *v2

Arguments

v0, v1, v2 Pointer to vertex coordinate vectors

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, sz2 and the interpolation value p for depth cueing corresponding to v2 in GTE's internal register.

The argument and internal data format is as follows:

```
v0, v1, v2 -> vx, vy, vz : (1, 15, 0)
sx0, sy0, sz0      : (1, 15, 0), (0, 16, 0)
sx1, sy1, sz1      : (1, 15, 0), (0, 16, 0)
sx2, sy2, sz2      : (1, 15, 0), (0, 16, 0)
p                  : (0, 20, 12)
flag               : (0, 32, 0)
```

Return value

None

Remarks

(sz0,sz1,sz2) is read by macro read_sz_fifo3, ((sx0,sy0), (sx1,sy1),(sx2,sy2) is read by macro read_sxsy_fifo3, p is read by macro read_p and flag is read by macro read_flag.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotTransPers3N

Performs coordinate transformation, perspective transformation.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
void RotTransPers3N(*v0, *v1, *sz, *flag, n)
SVECTOR *v0;
DVECTOR *v1;
unsigned short *sz;
unsigned short *flag;
long n;
```

Arguments

v0 Pointer to vertex coordinate vector (input)
v1 Pointer to vertex coordinate vector (output)
sz Pointer to SZ value (output)
flag Pointer to flag (output)
n Number of vertices (output)

Explanation

This function executes the RotTransPers3() function for the number of triangles specified by *n*.

Arguments and internal data formats are as follows:

v0 -> vx, vy, vz : (1, 15, 0)
v1 -> vx, vy : (1, 15, 0)
sz : (0, 16, 0)
flag : (0, 16, 0)

Return value

None

Remarks

The flag must normally be set between bits 27 and 12 of the 32-bit flag.

See also:

RotTransPers4

Performs coordinate transformation and perspective transformation for 4 vertices and returns the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotTransPers4(*v0, *v1, *v2, *v3, *sxy0, *sxy1, *sxy2, *sxy3, *p, *flag)
SVECTOR *v0, *v1, *v2, *v3;
long *sxy0, *sxy1, *sxy2, *sxy3;
long *p;
long *flag;
```

Arguments

<i>v0</i> , <i>v1</i> , <i>v2</i> , <i>v3</i>	Pointer to vectors (input)
<i>sxy0</i> , <i>sxy1</i> , <i>sxy2</i> , <i>sxy3</i>	Pointer to addresses where the screen coordinates are stored
<i>p</i>	Pointer to address where the interpolated value is stored
<i>flag</i>	Pointer to address where the flag is stored

Explanation

This function executes the RotTransPers3() function for the number of triangles specified by n.

Arguments and internal data formats are as follows:

<i>v0</i> -> vx, vy, vz	: (1, 15, 0)
<i>v1</i> -> vx, vy	: (1, 15, 0)
<i>sz</i>	: (0, 16, 0)
<i>flag</i>	: (0, 16, 0)

Return value

None

Remarks

The flag must normally be set between bits 27 and 12 of the 32-bit flag.

See also:

RotTransPers4_nom

Performs coordinate transformation and perspective transformation for 4 vertices, but does not return the operation result.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
long RotTransPers4_nom
SVECTOR *v0, *v1, *v2, *v3
```

Arguments

v0, v1, v2, v3 Pointer to vectors (input)

Explanation

After performing coordinate transformation for local coordinate vectors *v0*, *v1*, *v2* and *v3* using a rotation matrix, this function performs perspective transformation and stores three screen coordinates (*sz0*), (*sz1*,*sy1*,*sz1*), (*sz2*,*sy2*,*sz2*),(*sz3*,*sy3*,*sz3*), and the interpolation value *p* for depth cueing corresponding to *v3* in GTE's internal register.

The argument and internal data format is as follows:

v0,v1,v2-> *vx, vy, v* : (1, 15, 0)

sz0,sy0,sz0 : (1,15,0), (1,15,0), (0,16,0)

sz1,sy1,sz1 : (1,15,0), (1,15,0), (0,16,0)

sz2,sy2,sz2 : (1,15,0), (1,15,0), (0,16,0)

sz3,sy3,sz3 : (1,15,0), (1,15,0), (0,16,0)

p : (0,20,12)

flag : (0,32,0)

Return value

flag

Remarks

(*sz0,sz1,sz2,sz3*) is read by macro *read_sz_fifo4*, ((*sz1,sy1*),(*sz2,sy2*),(*sz3,sy3*)) is read by macro *read_sxsy_fifo3* and *p* is read by macro *read_p*. (*sz0,sy0*) is returned in register *v1*. *flag* is returned in register *v0*.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

See also:

RotTransPersN

Perform coordinate transformation and perspective transformation.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
void RotTransPersN(*v0, *v1, *sz, *p, *flag, n)
SVECTOR *v0;
DVECTOR *v1;
unsigned short *sz;
unsigned short *p;
unsigned short *flag;
long n;
```

Arguments

v0 Pointer to vertex coordinate vector (input)
v1 Pointer to vertex coordinate vector (output)
sz Pointer to SZ value (output)
p Pointer to interpolation value (output)
flag Pointer to flag (output)
n Number of vertices (output)

Explanation

This function performs the RotTransPers() function for the number of vertices specified by *n*.

The arguments and internal data formats are as follows:

v0 -> vx, vy, vz : (1, 15, 0)
v1 -> vx, vy : (1, 15, 0)
sz : (0, 16, 0)
flag : (0, 16, 0)

Return value

None

Remarks

The flag must normally be set between bits 27 and 12 of the 32-bit flag.

See also:

RotTransSV

Performs coordinate translation with rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
void RotTransSV (
SVECTOR *v0,
SVECTOR *v1,
long *flag
)
```

Arguments

v0 Pointer to input: vector
v1 Pointer to output: vector
flag Pointer to output: flag

Explanation

RotTrans output short vector edition

$v1 = RTM \times v0$

Argument format:

v0→*vx*,*vy*,*vz* : (1,15,0)

v1→*vx*,*vy*,*vz* : (1,15,0)

flag : (0,32,0)

Return value

None

Remarks

See also:

rsin

Sine.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	10/01/97

Syntax

```
int rsin(a)
```

```
int a;
```

Arguments

a Angle (in PlayStation format)

Explanation

Finds the sine function of the angle (in PlayStation format) ($4096 = 360 \text{ degrees} = 2\pi$) using fixed-point math (where $4096=1.0$).

The speed and size of `rsin()` and `csin()` differ as shown below.

	Speed	Size
rsin	fast	large
csin	slow	small

The argument format is as follows:

a : PlayStation format ($4096 = 360 \text{ degrees} = 2\pi$)

Return value : (1, 19, 12)

Return value

$\sin(a)$

Remarks

See also: `csin()`

ScaleMatrix

Scales a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	6/22/98

Syntax

```
MATRIX *ScaleMatrix (  
MATRIX *m,  
VECTOR *v  
(
```

Arguments

m Pointer to matrix (output)
v Pointer to scale vector (input)

Explanation

This function scales *m* by *v*. The components of *v* are fixed point decimals in which 1.0 represents 4096.

If:

$$m = \begin{bmatrix} a00 & a01 & a02 \\ a10 & a11 & a12 \\ a20 & a21 & a22 \end{bmatrix}, \quad v = [sx, sy, sz]$$

Then:

$$m = \begin{bmatrix} a00 \times sx & a01 \times sy & a02 \times sz \\ a10 \times sx & a11 \times sy & a12 \times sz \\ a20 \times sx & a21 \times sy & a22 \times sz \end{bmatrix}$$

The argument format is as follows:

m -> *m* [*i*] [*j*] : (1, 19, 12)

v -> *vx*, *vy*, *vz* : (1, 19, 12)

Return value

m

Remarks

See also:

ScaleMatrixL

Scales a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	6/22/98

Syntax

```
MATRIX *ScaleMatrixL (  
MATRIX *m,  
VECTOR *v  
)
```

Arguments

m Pointer to matrix (output)
v Pointer to scale vector (input)

Explanation

This function scales matrix *m* by *v*. The elements of *v* are fixed point numbers in which 4096 represents a value of 1.0.

If:

$$m = \begin{bmatrix} a00 & a01 & a02 \\ a10 & a11 & a12 \\ a20 & a21 & a22 \end{bmatrix}, \quad v = [sx \ sy \ sz]$$

Then:

$$m = \begin{bmatrix} a00 \times sx & a01 \times sx & a02 \times sx \\ a10 \times sy & a11 \times sy & a12 \times sy \\ a20 \times sz & a21 \times sz & a22 \times sz \end{bmatrix}$$

The argument format is as follows:

m -> m [i] [j] : (1, 19, 12)

v -> vx, vy, vz : (1, 19, 12)

Return value

m

Remarks

See also:

SetBackColor

Sets back color vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetBackColor(rbk, gbk, bbk)
long rbk, gbk, bbk;
```

Arguments

rbk, *gbk*, *bbk* Vectors (input)

Explanation

This function sets the back color vectors (*rbk*, *gbk*, *bbk*). Color values are in the range 0 to 255.

The argument format is as follows:

(*rbk*, *gbk*, *bbk*) : (0, 32, 0)

Return value

None

Remarks

See also:

SetColorMatrix

Sets a local color matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetColorMatrix(*m)
MATRIX *m;
```

Arguments

m Pointer to matrix (input)

Explanation

This function sets a local color matrix specified by *m*. The argument format is as follows:

m -> *m* [i] [j] : (1, 3, 12)

Return value

None

Remarks

See also:

SetFarColor

Sets far color vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetFarColor(rfc, gfc, bfc)
```

```
long rfc, gfc, bfc;
```

Arguments

rfc, *gfc*, *bfc* Vectors (input)

Explanation

This function sets the far color vectors (*rfc*, *gfc*, *bfc*). Color values are in the range 0 to 255. The argument format is as follows:

(*rfc*, *gfc*, *bfc*) : (0, 32, 0)

Return value

None

Remarks

See also:

SetFogFar

Sets a fog parameter.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetFogFar(a, h)
long a, h;
```

Arguments

a Z value
h Distance

Explanation

When the distance between the visual point and screen is *h*, *a* defines the Z value at which the fog is 100%. A Z value which makes fog 0% is automatically set to $0.2 \times a$. *a* should satisfy $0 < a < 65536$.

The argument format is as follows:

a : (0, 32, 0)

h : (0, 32, 0)

Return value

None

Remarks

See also:

SetFogNear

Sets a fog parameter.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetFogNear(a, h)
long a, h;
```

Arguments

a Z value
h Distance

Explanation

When the distance between the visual point and screen is *h*, *a* defines the Z value at which the fog is 0%. A Z value which makes fog 100% is automatically set to $5 \times a$. *a* should satisfy $0 < a < 65536 \times 0.2$.

The argument format is as follows:

a : (0, 32, 0)

h : (0, 32, 0)

Return value

None

Remarks

See also:

SetFogNearFar

Sets the fog parameters.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	6/22/98

Syntax

```
void SetFogNearFar (
long a,
long b,
long h
)
```

Arguments

a Z value with fog at 0%
b Z value with fog at 100%
h Distance between visual point and screen

Explanation

When the distance between the visual point and screen is *h*, the Z value with fog at 0% is set as *a*.

The Z value with fog at 100% is set as *b*.

$0 < a, b < 65536$

$(b - a) >= 100$

The actual value set to the DQA and DQB GTE register is calculated using the method below:

$K = -a * b / (b - a)$

$c = (b << 12) / (b - a)$

$DQA = (K / h) << 8$

$DQB = c << 12$

However, since the DQA register is 16 bit, a limit of 16 bits is set.

Argument format:

a : (0, 32, 0)

b : (0, 32, 0)

h : (0, 32, 0)

Return value

None

Remarks

See also:

SetGeomOffset

Sets offset values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetGeomOffset(ofx, ofy)
```

```
long ofx, ofy;
```

Arguments

ofx, *ofy* Offset input values

Explanation

This function sets the offset values (*ofx*, *ofy*).

The argument format is as follows:

ofx, *ofy* : (1, 31, 0)

Return value

None

Remarks

See also:

SetGeomScreen

Sets the projection.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetGeomScreen(h)
long h;
```

Arguments

h Distance

Explanation

This function sets the distance *h* (projection) from a visual point (the eye) to the screen.

The argument format is as follows:

h : (0, 32, 0)

Return value

None

Remarks

See also:

SetLightMatrix

Sets a local light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetLightMatrix(*m)
MATRIX *m;
```

Arguments

m Pointer to matrix (input)

Explanation

This function sets a local light matrix specified by *m*.

The argument format is as follows:

m -> m [i] [j] : (1, 3, 12)

Return value

None

Remarks

See also:

SetMulMatrix

Multiplies two matrices and sets one rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
MATRIX *SetMulMatrix(*m0, *m1)
MATRIX *m0, *m1;
```

Arguments

m0, m1 Pointer to input matrices

Explanation

Multiplies two matrices and stores that value in one constant rotation matrix. The argument format is as follows:

m0, m1 -> m [i] [j] : (1, 3, 12)

Return value

Returns *m0*.

Remarks

See also:

SetMulRotMatrix

Multiplies constant rotation matrix by a matrix and sets one constant rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.6	10/23/96

Syntax

```
MATRIX *SetMulRotMatrix(*m0)
MATRIX *m0;
```

Arguments

m0 Pointer to input matrix

Explanation

This function multiplies constant rotation matrix and a matrix and stores that value in one constant rotation matrix.

The argument format is as follows:

m0 -> m [i] [j] : (1, 3, 12)

Return value

m0

Remarks

See also:

SetRGBcd

Set primary color vector and GPU code.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetRGBcd(*v)
CVECTOR *v;
```

Arguments

v Pointer to color vector and GPU code input

Explanation

This function sets the primary color vector and GPU code *v*.

The argument format is as follows:

v -> r, g, b, cd : (0, 8, 0)

Return value

None

Remarks

See also:

SetRotMatrix

Sets a constant rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetRotMatrix(*m)
MATRIX *m;
```

Arguments

m Pointer to matrix (input)

Explanation

This function sets a 3x3 matrix *m* as a constant rotation matrix.

The argument format is as follows:

m -> *m* [*i*] [*j*] : (1, 3, 12)

Return value

None

Remarks

See also:

SetTransMatrix

Setting a constant parallel transfer vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SetTransMatrix(*m)
MATRIX *m;
```

Arguments

m Pointer to matrix (input)

Explanation

This function sets a constant parallel transfer vector specified by *m*.

The argument format is as follows:

m -> t [i] : (1, 31, 0)

Return value

None

Remarks

See also:

Square0

Squares a vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void Square0(*v0, *v1)
```

```
VECTOR *v0;
```

```
VECTOR *v1;
```

Arguments

v0 Pointer to vector (L1, L2, L3) (input)

v1 Pointer to vector (L1^2, L2^2, L3^2) (output)

Explanation

This function returns a vector, obtained by squaring each term of the vector *v0*, to *v1*.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 31, 0)

v1 -> vx, vy, vz : (1, 31, 0)

Return value

Returns *v1*

Remarks

See also:

Square12

Squares a vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void Square12(*v0, *v1)
VECTOR *v0;
VECTOR *v1;
```

Arguments

v0 Pointer to vector (L1, L2, L3) (input)
v1 Pointer to vector (L1^2, L2^2, L3^2) (output)

Explanation

This function returns a vector, obtained by dividing the square of each term of the vector *v0* by 4096, to *v1*.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 19, 12)

v1 -> vx, vy, vz : (1, 19, 12)

Return value

Returns *v1*

Remarks

See also:

SquareRoot0

Square root.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
long SquareRoot0(a)
```

```
long a;
```

Arguments

a Value

Explanation

This function returns the square root of a value *a*.

The argument format is as follows:

a : (0, 32, 0)

Return value

Returns the square root of *a*

Remarks

See also:

SquareRoot12

Square root.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

long SquareRoot12(*a*)

long *a*;

Arguments

a Value

Explanation

This function returns the square root of a value *a*, which has (0, 20, 12) format, in (0, 20, 12) format.

The argument format is as follows:

a : (0, 20, 12)

Return value

Square root of *a*

Remarks

See also:

SquareSL0

Squares a short vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.0	5/22/97

Syntax

```
VECTOR *SquareSL0 (
SVECTOR *v0,
VECTOR *v1
)
```

Arguments

v0 Input: short vector (L1, L2, L3)
v1 Output: vector (L1^2, L2^2, L3^2)

Explanation

This function returns a vector, obtained by squaring each term of the short vector *v0*, to *v1*.

Argument format:

v0 -> vx, vy, vz : (1, 15, 0)

v1 -> vx, vy, vz : (1, 31, 0)

Return value

v1

Remarks

See also:

SquareSL12

Squares a short vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.0	5/22/97

Syntax

```
VECTOR *SquareSL12 (
SVECTOR *v0,
VECTOR *v1
)
```

Arguments

v0 Input: short vector (L1, L2, L3)
v1 Output: vector (L1^2, L2^2, L3^2)

Explanation

This function returns a vector divided by 4096, obtained by squaring each term of the short vector *v0*, to *v1*.

Argument format:

v0 -> vx, vy, vz : (1, 3,12)

v1 -> vx, vy, vz : (1,19,12)

Return value

v1

Remarks

See also:

SquareSS0

Squares a short vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.0	6/22/98

Syntax

```
SVECTOR *SquareSS0 (
SVECTOR *v0,
SVECTOR *v1
)
```

Arguments

v0 Input: short vector (L1, L2, L3)
v1 Output: vector (L1^2, L2^2, L3^2)

Explanation

This function returns a short vector, obtained by squaring each term of the short vector *v0*, to *v1*.

Argument format:

v0 -> vx,vy,vz : (1,15, 0)

v1 -> vx, vy, vz : (1,15, 0)

Return value

v1

Remarks

See also:

SquareSS12

Square root.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.0	6/22/98

Syntax

```
SVECTOR *SquareSS12 (
SVECTOR *v0,
SVECTOR *v1
)
```

Arguments

v0 Input: short vector (L1, L2, L3)
v1 Output: vector ($L1^2$, $L2^2$, $L3^2$)

Explanation

This function returns a short vector divided by 4096, obtained by squaring each term of the short vector *v0*, to *v1*.

Argument format:

v0 -> vx, vy, vz : (1, 3, 12)

v1 -> vx, vy, vz : (1, 3, 12)

Return value

v1

Remarks

See also:

SubPol3

Subdivides a triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SubPol3(*p, *sp, ndiv)
POL3 *p;
SPOL *sp;
int ndiv;
```

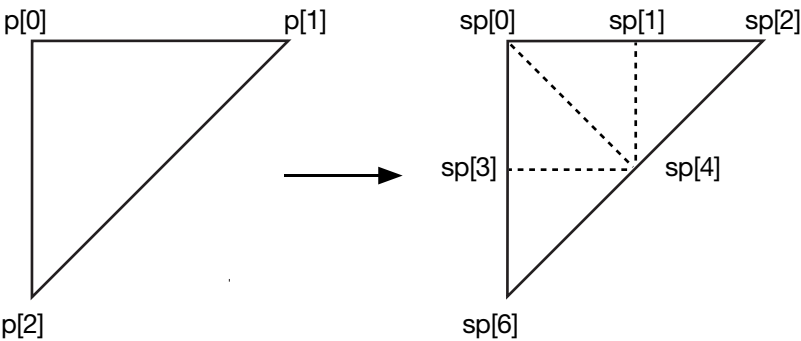
Arguments

- p* Pointer to a 3-vertex polygon
- sp* Pointer to subdivision vertex array
- ndiv* Number after subdivision
 0: None
 1: 2x2 2: 4x4

Explanation

This function subdivides a three-sided polygon *p* by the number 2^{ndiv} , and returns the subdivision vertex coordinates, texture coordinates, and RGB to a triangle in an array indicated by *sp*. See the figure below:

Figure 8-1



The argument format is as follows:

```
p -> sxy : (1, 15, 0), (1, 15, 0)
p -> sz  : (0, 16, 0)
p -> uv  : (1, 15, 0), (1, 15, 0)
p -> rgb : (0, 8, 0), (0, 8, 0), (0, 8, 0)
p -> code : (0, 32, 0)
sp -> xy  : (1, 15, 0), (1, 15, 0)
sp -> uv  : (1, 15, 0), (1, 15, 0)
sp -> rgb : (0, 8, 0), (0, 8, 0), (0, 8, 0)
```

Return value

None

Remarks

See also:

SubPol4

Subdivides a quadrangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void SubPol4(*p, *sp, ndiv)
POL4 *p;
SPOL *sp;
int ndiv;
```

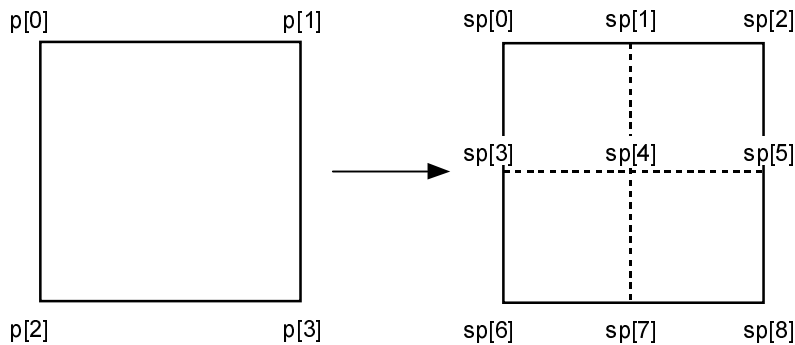
Arguments

- p* Pointer to a 4-vertex polygon
- sp* Pointer to subdivision vertex array
- ndiv* Number after subdivision
- 0: None
- 1: 2x2 2: 4x4

Explanation

This function subdivides a four-sided polygon *p* by the number 2^{**ndiv} , and returns the subdivision vertex coordinates, texture coordinates, and RGB to an array indicated by *sp*. See the figure below:

Figure 8-2



The argument format is as follows:

- p* -> sxy : (1, 15, 0), (1, 15, 0)
- p* -> sz : (0, 16, 0)
- p* -> uv : (1, 15, 0), (1, 15, 0)
- p* -> rgb : (0, 8, 0), (0, 8, 0), (0, 8, 0)
- p* -> code : (0, 32, 0)
- sp* -> xy : (1, 15, 0), (1, 15, 0)
- sp* -> uv : (1, 15, 0), (1, 15, 0)
- sp* -> rgb : (0, 8, 0), (0, 8, 0), (0, 8, 0)

Return value

None

Remarks

See also:

TransMatrix

Sets the amount of parallel transfer.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
MATRIX *TransMatrix(*m, *v)
MATRIX *m;
VECTOR *v;
```

Arguments

m Pointer to matrix (output)
v Pointer to transfer vector (input)

Explanation

This function gives an amount of parallel transfer expressed by *v* to the matrix *m*.

The argument format is as follows:

```
m -> m [i] [j]        : (1, 3, 12)
m -> t [i]             : (1, 31, 0)
v -> vx, vy, vz       : (1, 31, 0)
```

Return value

This function returns *m*.

Remarks

See also:

TransposeMatrix

Transposes a matrix

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

MATRIX *TransposeMatrix(**m0*, **m1*)

MATRIX **m0*, **m1*;

Arguments

m0 Pointer to matrix (input)

m1 Pointer to matrix (output)

Explanation

Transposes matrix *m0* into *m1*.

The argument format is as follows:

m0 -> m [i] [j] : (1, 3, 12)

m1 -> m [i] [j] : (1, 3, 12)

Return value

Returns *m1*

Remarks

See also:

TransRotPers

Inversely performs rotation parallel move of RotTransPers.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
long TransRotPers(*v0, *sxy, *p, *flag)
SVECTOR *v0;
long *sxy;
long *p;
long *flag;
```

Arguments

v0 Pointer to vertex coordinate vector (input)
sxy Pointer to screen coordinate value (output)
p Pointer to interpolation value (output)
flag Pointer to flag (output)

Explanation

Rotates after performing a parallel move of the coordinate vector *v0* with the rotation matrix.

Performs a perspective conversion and then a coordinate conversion and returns screen coordinates *sx*, *sy*.

Also, returns the interpolation value for depth cueing to *p*.

Argument format:

v0 -> vx, vy, vz : (1, 15, 0)
sxy : (1, 15, 0), (1, 15, 0)
p : (0, 20, 12)
flag : (0, 32, 0)

Return value

1/4 of the screen coordinate Z component *sz* corresponding to *v2*.

Remarks

See also:

TransRotPers3

Inversely performs rotation parallel move of RotTransPers3.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
long TransRotPers3(*v0, *v1, *v2, *sxy0, *sxy1, *sxy2, *p, *flag)
SVECTOR *v0, *v1, *v2;
long *sxy0, *sxy1, *sxy2;
long *p;
long *flag;
```

Arguments

<i>v0, v1, v2</i>	Pointer to vertex coordinate vector (input)
<i>sxy0, sxy1, sxy2</i>	Pointer to screen coordinate value (output)
<i>p</i>	Pointer to interpolation value (output)
<i>flag</i>	Pointer to flag (output)

Explanation

Rotates after performing a parallel move of the three coordinate vectors *v0,v1,v2* with the rotation matrix. Performs a perspective conversion and then a coordinate conversion and returns the three screen coordinates *sxy0, sxy1, and sxy2*.

Also, returns the interpolation value for depth cueing compatible with *v2* to *p*.

Also, returns the screen coordinate Z item *sz* 1/4 compatible with *v2* as the return value.

Argument format:

<i>v0, v1, v2n</i> -> <i>vx, vy, vz</i>	: (1, 15, 0)
<i>sxy0, sxy1, sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>flag</i>	: (0, 32, 0)

Return value

1/4 of the screen coordinate Z component *sz* corresponding to *v2*.

Remarks

See also:

TransRot_32

Inversely performs rotation parallel move of RotTrans.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	7/31/96

Syntax

```
void TransRot(*v0, *v1, *flag)
VECTOR *v0;
VECTOR *v1;
long *flag;
```

Arguments

v0 Pointer to vector (input)
v1 Pointer to vector (output)
flag Pointer to flag (output)

Explanation

After adding the 32 bit parallel move volume to *v0*, performs rotation with constant rotation matrix.

Argument format:

v0 -> vx, vy, vz : (1, 31, 0)

v1 -> vx, vy, vz : (1, 31, 0)

flag : (0, 32, 0)

Return value

None

Remarks

See also:

VectorNormal

Normalizes a vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	7/31/96

Syntax

```
void VectorNormal(*v0, *v1)
VECTOR *v0;
VECTOR *v1;
```

Arguments

v0 Pointer to vector (input)
v1 Pointer to vector (output)

Explanation

This function normalizes a vector *v0* and returns the result in *v1*.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 31, 0)

v1 -> vx, vy, vz : (1, 19, 12)

Return value

Sum of squared *v0* elements

Remarks

See also:

VectorNormalS

Normalizes a vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	7/31/96

Syntax

```
long VectorNormalS(*v0, *v1)
```

```
VECTOR *v0;
```

```
SVECTOR *v1;
```

Arguments

v0 Pointer to vector (input)

v1 Pointer to vector (output)

Explanation

This function normalizes a vector *v0* and returns the result in *v1*.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 31, 0)

v1 -> vx, vy, vz : (1, 3, 12)

Return value

Sum of squared *v0* elements

Remarks

The calculation will be incorrect if the sum of the squared elements of *v0* exceeds $2^{31}-1$.

See also:

VectorNormalSS

Normalize a vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	10/01/97

Syntax

```
long VectorNormalSS(*v0, *v1)
SVECTOR *v0;
SVECTOR *v1;
```

Arguments

v0 Pointer to vector (input)
v1 Pointer to vector (output)

Explanation

This function normalizes a vector *v0* and returns the result in *v1*.

The argument format is as follows:

v0 -> vx, vy, vz : (1, 16, 0)

v1 -> vx, vy, vz : (1, 3, 12)

Return value

Sum of squared *v0* elements

Remarks

The calculation will be incorrect if the sum of the squared elements of *v0* exceeds $2^{31}-1$.

See also:

Chapter 9: Extended Graphics Library

Table of Contents

Structures	
GsBG	9-3
GsBOXF	9-5
GsCELL	9-6
GsCOORD2PARAM	9-7
GsCOORDINATE2	9-8
GsDOBJ2	9-9
GsDOBJ3	9-11
GsDOBJ5	9-12
GsFOGPARAM	9-14
GsF_LIGHT	9-15
GsGLINE	9-16
GsIMAGE	9-17
GsLINE	9-18
GsMAP	9-19
GsOBJTABLE2	9-20
GsOT	9-21
GsOT_TAG	9-22
GsRVIEW2	9-23
GsSPRITE	9-24
GsVIEW2	9-27
TMD_STRUCT	9-28
_GsFCALL	9-29
_GsPOSITION	9-32
Functions	
dmyGsPrst...	9-33
dmyGsTMD...	9-34
GsA4div...	9-35
GsClearDispArea	9-38
GsClearOt	9-39
GsClearVcount	9-40
GsCutOt	9-41
GsDefDispBuff	9-42
GsDefDispBuff2	9-43
GsDrawOt	9-44
GsDrawOtIO	9-45
GsGetActiveBuffer	9-46
GsGetLs	9-47
GsGetLw	9-48
GsGetLws	9-49
GsGetTimInfo	9-50
GsGetVcount	9-51
GsGetWorkBase	9-52
GsInit3D	9-53
GsInitCoordinate2	9-54
GsInitFixBg16	9-55
GsInitFixBg32	9-56
GsInitGraph	9-57
GsInitGraph2	9-59
GsInitVcount	9-60
GsLinkObject3	9-61
GsLinkObject4	9-62
GsLinkObject5	9-63
GsMapModelingData	9-64

GsMulCoord0	9-65
GsMulCoord2	9-66
GsMulCoord3	9-67
GsPresetObject	9-68
GsPrst...	9-69
GsScaleScreen	9-72
GsSetAmbient	9-73
GsSetClip	9-74
GsSetClip2	9-75
GsSetClip2D	9-76
GsSetDrawBuffClip	9-77
GsSetDrawBuffOffset	9-78
GsSetFlatLight	9-79
GsSetFogParam	9-80
GsSetLightMatrix	9-81
GsSetLightMatrix2	9-82
GsSetLightMode	9-83
GsSetLsMatrix	9-84
GsSetOffset	9-85
GsSetOrign	9-86
GsSetProjection	9-87
GsSetRefView2	9-88
GsSetRefView2L	9-89
GsSetView2	9-90
GsSetWorkBase	9-91
GsSortBg, GsSortFastBg	9-92
GsSortBoxFill	9-93
GsSortClear	9-94
GsSortFastSpriteB	9-95
GsSortFixBg16	9-96
GsSortFixBg32	9-97
GsSortGLine	9-98
GsSortLine	9-99
GsSortObject3	9-100
GsSortObject4	9-101
GsSortObject4J	9-102
GsSortObject5	9-103
GsSortObject5J	9-105
GsSortOt	9-106
GsSortPoly	9-107
GsSortSprite, GsSortFastSprite, GsSortFlipSprite	9-108
GsSortSpriteB	9-109
GsSwapDispBuffer	9-110
GsTMDdiv...	9-111
GsTMDfast...	9-115
Macros	
GsIncFrame	9-121
GsSetAzwh	9-122
List of External Variables	9-123

GsBG

BG (background surface) handler.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsBG {
    unsigned long attribute;
    short x, y;
    short w, h;
    short scrollx, scrolly;
    unsigned char r, g, b;
    GsMAP *map;
    short mx, my;
    short scalex, scaley;
    long rotate;
};
```

Members

<i>attribute</i>	Attribute
<i>x, y</i>	Top left point display position
<i>w, h</i>	BG display size
<i>scrollx, scrolly</i>	x and y scroll values
<i>r, g, b</i>	Display brightness is set in r, g, b. (Normal brightness is 128.)
<i>map</i>	Pointer to map data
<i>mx, my</i>	Rotation and enlargement central point coordinates
<i>scalex, scaley</i>	Scale values in x and y directions
<i>rotate</i>	Rotation angle (4096 = 1 degree)

Explanation

For *attribute*, see the description in GsSPRITE.

BG (background) draws a large rectangle based on GsMAP data on a combination of small rectangles defined by GsCELL data. There is a GsBg for each BG. The BG may be manipulated via the GsBG structure.

To register a GsBG object in the ordering table, use GsSortBg().

x, y specifies the screen display position.

w, h specifies BG display size in pixels, and is not dependent on cell size or map size.

If the display area is larger than the map, the content of the map is repeatedly displayed. (Tiling function)

scrollx, scrolly specifies offset from the map display position in dots.

r, g, b specifies brightness values for red, green, and blue. The range is 0 to 255. 128 is the brightness of the original pattern; 255 doubles the brightness.

map specifies the starting address of map data with a pointer to GsMAP format map data.

mx, my specify the center of rotation and scaling as relative coordinates. The top left point of the BG is the point of origin. For example, if rotation is around the center of the BG, specify w/2 and h/2.

scalex, scaley specifies enlargement/reduction values in the x and y directions. These values are expressed in units of 4096, which stands for 1.0 (i.e. is the same size as 1.0). You can set these values up to 8 times the original size.

9-4 Extended Graphics Library Structures

rotate specifies a rotation angle around the z-axis ($4096 = 1$ degree).

Remarks

See also:

GsBOXF

Rectangle handler.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsBOXF {
    unsigned long attribute;
    short x, y;
    unsigned short w, h;
    unsigned char r, g, b;
};
```

Members

<i>attribute</i>	Attribute (see GsLINE attributes)
<i>x</i> , <i>y</i>	Display position (top left point)
<i>w</i> , <i>h</i>	Size of rectangle (width, height)
<i>r</i> , <i>g</i> , <i>b</i>	Drawing color

Explanation

GsBOXF is a structure used to draw a rectangle in a single color. To register GsBOXF in the ordering table, the GsSortBoxFill() function is used.

Remarks

See also: GsLINE (p. 9-18).

GsCELL

Cells constituting BG.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsCELL {
    unsigned short u, v;
    unsigned short cba;
    unsigned short flag;
    unsigned short tpage;
};
```

Members

u Offset (X-direction) within the page
v Offset (Y-direction) within the page
cba CLUT ID
flag An option at the time of drawing
tpage Texture page number

Explanation

A rectangular array of GsCell structures is used to describe individual cells that fit together to create a BG. Each individual GsCell structure defines a rectangular portion of the overall BG.

cba is data that displays the position within the frame buffer of a CLUT corresponding to the cell, as follows.

Table 9–1

Bit	Value
Bit 0-5	X position of CLUT/16
Bit 6-15	Y position of CLUT

tpage is a page number that indicates the position of a Sprite pattern within a frame buffer.

The *u* and *v* parameters specify the offset position for the sprite pattern within the texture page defined by *tpage*.

flag specifies option information for performing drawing. The meaning of each bit is as shown below.

Table 9–2

Bit	Value
Bit 0	Vertical flip (0: no flip; 1: flip)
Bit 1	Horizontal flip (0: no flip; 1: flip)

Remarks

See also:

GsCOORD2PARAM

GsCOORDINATE2 parameter format.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
struct GsCOORDINATE2 {
    VECTOR scale;
    SVECTOR rotate;
    VECTOR trans;
} GsCOORD2PARAM;
```

Members

scale Retains coordinate scaling information
rotate Retains coordinate rotation information
trans Retains coordinate parallel shift information

Explanation

This structure is used in order to retain GSCOORDINATE2 SRT information when GsCOORD2PARAM is TOD animation.

Remarks

See also: GsInitObjTable2(), GsCOORDINATE2().

GsCOORDINATE2

Matrix type coordinate system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsCOORDINATE2 {
    unsigned long flag;
    MATRIX coord;
    MATRIX workm;
    GsCOORD2PARAM *param;
    GsCOORDINATE2 *super;
    GsCOORDINATE2 *sub;
};
```

Members

flag Flag indicating whether coord was rewritten
coord Matrix
workm Result of multiplication from this coordinate system to the WORLD coordinate system
param Pointer for scale, rotation, and transfer parameters
super Pointer to superior coordinates
sub Not in current use

Explanation

GsCOORDINATE2 has superior coordinates and is defined by the matrix type *coord*.

workm retains the result of multiplication of matrices performed by the GsGetLw() and GsGetLs() functions in each node of GsCOORDINATE2 using the WORLD coordinates.

flag is referenced to omit calculations for a node for which calculations were already made, during GsGetLw() calculations. 1 means the flag is set; 0 clears the flag. The programmer must clear this flag when he has changed coord. If you neglect to clear it, the GsGetLw() and GsGetLs() functions will fail to execute normally.

param is used for setting coord values with layout tools.

Remarks

param may be freely used if TOD animation is not used.

See also:

GsDOBJ2

Used by the three-dimensional object handler GsCOORDINATE2.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	10/01/97

Structure

```
struct GsDOBJ2 {
    unsigned long attribute;
    GsCOORDINATE2 *coord2;
    unsigned long *tmd;
    unsigned long id;
};
```

Members

<i>attribute</i>	Object attribute (32-bit)
<i>coord2</i>	Pointer to a local coordinate system
<i>tmd</i>	Pointer to model data
<i>id</i>	Reserved by the layout tool

Explanation

There is a GsDOBJ2 for each object of a three dimensional model; GsDOBJ2 structures may be used to manipulate the 3-dimensional model.

Use GsLinkObject4() is to link GsDOBJ2 to TMD file model data. Use GsSortObject4() to register GsDOBJ2 in the ordering table.

The *coord2* parameter is a pointer to a GsCOORDINATE2 structure defining the object's coordinate system. The location, inclination, and size of the object is defined in the matrix in this structure.

tmd contains the starting address of TMD model data stored in memory. *tmd* is calculated and set using GsLinkObject4().

attribute is 32-bit; various display attributes are set here. An explanation of each bit follows.

(a) Bits 0-2: material attenuation (not currently supported)

This sets the relationship between the normal gradient and brightness attenuation when light source calculation is performed. This takes a value of 0-3. With 0 there is no attenuation; the steepest attenuation is with 3. This parameter can be used to display an object's material quality. In general, making the attenuation steep produces a metallic quality.

Note the following points:

- (1) If the material attenuation value is high, calculation takes longer and the processing requires a lot of resources.
- (2) This parameter is invalid in lighting mode unless material ON is set.

(b) Bits 3-5: lighting mode

This sets the light source calculation formula. It takes a value of 0-3. The values are as listed below.

Bit 5, the highest ranking bit, is a switch to validate the lighting mode set by GsSetLightMode().

Table 9-3: Lighting modes

Value	Operation
0	Normal mode without fog or material attenuation. This is the fastest mode and calculation takes least time.
1	Fog only mode. The fog parameter is GsFOGPARAM; set the parameter with GsSetFogParam().
2	Material attenuation only mode. The amount of attenuation is set by the material attenuation bit. Not currently supported.
3	Applies both fog and material attenuation. Not currently supported.

- (c) Bit 6: Light source calculation ON/OFF switch
This bit is used when light source calculation is not performed. When light source calculation is removed, a texture-mapped polygon is displayed in the original texture color. An unmapped polygon is displayed in the model data color.
- (d) Bits 7-27: Reserved, set to zero
- (e) Bits 28-29: Semi-transparency rate
When semi-transparency is set to ON with bit 30, the semi-transparency rate sets the pixel-blending formula. This feature is currently not supported.

Table 9-4: Semi-transparency Rate

Value	Processing
0	Normal semi-transparency processing
1	Pixel addition
2	50% addition
3	Pixel subtraction

- (f) Bit 30: Semi-transparency ON/OFF
This sets semi-transparency ON/OFF.
This bit must be used with the uppermost bit (STP bit) of the texture color field (texture pattern when direct and CLUT color field when indexed) to set semi-transparency. Also, the semi-transparency and non-transparency of each pixel unit may be controlled using this STP bit.
- (g) Bit 31: Display ON/OFF
This turns display ON and OFF.

Remarks

See also:

GsDOBJ3

Used by the three-dimensional object handler PMD FORMAT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Structure

```
struct GsDOBJ3 {
    unsigned long attribute;
    GsCOORDINATE2 *coord2;
    unsigned long *pmd;
    unsigned long *base;
    unsigned long *sv;
    unsigned long id;
};
```

Members

<i>attribute</i>	Object attribute (32-bit)
<i>coord2</i>	Pointer to a local coordinate system
<i>pmd</i>	Pointer to model data (PMD FORMAT)
<i>base</i>	Pointer to object base address
<i>sv</i>	Pointer to shared vertex base address
<i>id</i>	Reserved by the layout tool

Explanation

There is a GsDOBJ3 for each object of a 3-dimensional model; GsDOBJ3 structures may be used to manipulate the 3-dimensional model.

Use GsLinkObject3() to link GsDOBJ3 to PMD file model data.

You can use GsDOBJ3 to access PMD data linked by GsLinkObject3(). Use GsSortObject3() to register GsDOBJ3 in the ordering table.

coord2 is a pointer to a coordinate system unique to an object. The location, inclination, and size of the object is reflected in a matrix set in the coordinate system to point to *coord2*.

pmd retains the starting address of PMD model data stored in memory. *pmd* is calculated and set using GsLinkObject3().

attribute is 32-bit; various display attributes are set here.

Only the attribute shown below is currently available.

- (a) Bits 0-30: Reserved, set to zero
- (b) Bit 31: Display ON/OFF
This turns display ON and OFF.

Remarks

id is not used unless the layout function is used.

See also:

GsDOBJ5

Used by the three-dimensional object handler GsSortObject5.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Structure

```
struct GsDOBJ5 {
    unsigned long attribute;
    GsCOORDINATE2 *coord2;
    unsigned long *tmd;
    unsigned long *packet;
    unsigned long id;
};
```

Members

<i>attribute</i>	Object attribute (32-bit)
<i>coord2</i>	Pointer to local coordinate system
<i>tmd</i>	Pointer to model data
<i>packet</i>	Pointer to preset packet area
<i>id</i>	Reserved by the layout tool

Explanation

There is a GsDOBJ5 for each object of a 3-dimensional model; GsDOBJ5 structures may be used to manipulate the 3-dimensional model.

Use GsLinkObject5() to link GsDOBJ5 to TMD file model data.

You can use GsDOBJ5 to access TMD data linked by GsLinkObject5(). Use GsSortObject5() to register GsDOBJ5 in the ordering table.

coord2 is a pointer to a coordinate system unique to an object. The location, inclination, and size of the object is reflected in a matrix set in the coordinate system to point to *coord2*.

tmd retains the starting address of TMD model data stored in memory. *tmd* is calculated and set using GsLinkObject5().

packet retains the starting address of a preset packet copied into memory. A preset packet is copied by GsPresetObject(), and is set in a GsDOBJ5 packet.

attribute is 32-bit; various display attributes are set here. An explanation of each bit follows.

(a) Bits 0-2: Material attenuation (not currently supported)

This sets the relationship between the normal gradient and brightness attenuation when light source calculation is performed. This takes a value of 0-3. With 0 there is no attenuation; the steepest attenuation is with 3. This parameter can be used to display an object's material quality. In general, making the attenuation steep produces a metallic quality.

Note the following points:

- (1) If the material attenuation value is high, calculation takes longer and the processing requires a lot of resources.
- (2) This parameter is invalid in lighting mode unless material ON is set.

(b) Bits 3-5: Lighting mode

This sets the light source calculation formula. It takes a value of 0-3. The values are as listed below.

Bit 5, the highest ranking bit, is a switch to validate the lighting mode set by GsSetLightMode().

Table 9–5: Lighting Modes

Value	Operation
0	Normal mode without fog or material attenuation. This is the fastest mode and calculation takes least time.
1	Fog only mode. The fog parameter is GsFOGPARAM; set the parameter with GsSetFogParam().
2	Material attenuation only mode. The amount of attenuation is set by the material attenuation bit. Not currently supported.
3	Applies both fog and material attenuation. Not currently supported.

- (c) Bit 6: Light source calculation ON/OFF switch

This bit is used when light source calculation is not performed. When light source calculation is removed, a texture-mapped polygon is displayed in the original texture color. An unmapped polygon is displayed in the model data color.

- (d) Bits 7-30: Reserved, set to zero.

- (e) Bits 31: Display ON/OFF

This turns display ON and OFF.

Remarks

id is not used unless the layout function is used.

See also:

GsFOGPARAM

Fog (depth cue) information.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsFOGPARAM {
    short dqa;
    long dqb;
    unsigned char rfc, gfc, bfc;
};
```

Members

dqa Parameter for the degree of merging due to depth
dqb Parameter for the degree of merging due to depth
 For the meaning of these parameters, see the description of “FOG” in “FUNDAMENTAL GEOMETRY LIBRARY Part 1”.
rfc, gfc, bfc Background colors

Explanation

dqa and *dqb* are background color attenuation coefficients. They can be calculated using the following formulas:

$$DQA = -df \bullet 4096/64/h$$

$$DQB = 1.25 \bullet 4096 \bullet 4096$$

df is the distance where the attenuation coefficient is “1”; that is, the distance from the viewpoint to where the background colors are completely blended.

“h” indicates a projection, or a distance from the visual point to the screen.

Remarks

See also:

GsF_LIGHT

Parallel light source.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	6/22/98

Structure

```
struct GsF_LIGHT {
    int vx, vy, vz;
    unsigned char r, g, b;
};
```

Members

vx, vy, vz Directional vectors for light source
r, g, b Light colors

Explanation

GsF_LIGHT holds parallel light source information, and is set in the system by the GsSetFlatLight() function. Up to three parallel light sources may be set at the same time.

The light source directional vector is specified by *vx, vy, vz*. It is unnecessary for the programmer to perform normalization because the system does it. A polygon whose normal vectors are opposite to these directional vectors is exposed to the strongest light.

Light source colors are set in 8 bits by *r, g, b*.

Remarks

See also:

GsGLINE

Straight line handler with gradation.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.5	7/31/96

Structure

```
struct GsLINE {
    unsigned long attribute;
    short x0, y0;
    short x1, y1;
    unsigned char r0, g0, b0;
    unsigned char r1, g1, b1;
};
```

Members

<i>attribute</i>	Attribute (see GsLINE attributes)
<i>x0</i> , <i>y0</i>	Drawing start point position
<i>x1</i> , <i>y1</i>	Drawing end point position
<i>r0</i> , <i>g0</i> , <i>b0</i>	Drawing colors of start point
<i>r1</i> , <i>g1</i> , <i>b1</i>	Drawing colors of end point

Explanation

GsGLINE is a structure used to draw straight lines with gradation. It is the same as GsLINE except that drawing colors for the starting point and end point may be specified separately.

Remarks

See also:

GsIMAGE

Information on image data composition.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	6/22/98

Structure

```
struct GsIMAGE {
    unsigned long pmode;
    short px, py;
    unsigned short pw, ph;
    unsigned long *pixel;
    short cx, cy;
    unsigned short cw, ch;
    unsigned long *clut;
};
```

Members

pmode Pixel mode
 0: 4-bit CLUT
 1: 8-bit CLUT
 2: 16-bit DIRECT
 3: 24-bit DIRECT
 4: Coexistence of multiple modes

px, py Pixel data storage location within the frame buffer

pw, ph Pixel data width and height

pixel Pointer to pixel data

cx, cy CLUT data storage location within the frame buffer

cw, ch CLUT data width and height

clut Pointer to CLUT data

Explanation

A structure in which TIM format data information is stored by the GsGetTimInfo() function.

Remarks

See also:

GsLINE

Straight line handler.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsLINE {
    unsigned long attribute;
    short x0, y0;
    short x1, y1;
    unsigned char r, g, b;
};
```

Members

attribute Attribute
 Bits 28-29: Semi-transparency rate
 0 50% x Back + 50% x Line
 1 100% x Back + 100% x Line
 2 100% x Back + 50% x Line
 3 100% x Back - 100% x Line
 Bit 30: Semi-transparency ON/OFF
 0: Semi-transparency OFF
 1: Semi-transparency ON
 Bit 31
 0: Displayed
 1: Not displayed
x0, y0 Drawing start point position
x1, y1 Drawing end point position
r, g, b Drawing color

Explanation

GsLINE is a structure for drawing straight lines. Use GsSortLine() to register a GsLINE in the ordering table.

attribute is 32 bits, and sets various attributes for display.

- (a) Bits 0-27: Reserved, set to 0.
- (b) Bits 28-29: Semi-transparency rate
 If semi-transparency is turned on using bit 30, bits 28 and 29 are used to set the pixel blending method.
- (c) Bit 30: Semi-transparency ON/OFF
 This bit turns semi-transparency ON and OFF.
- (d) Bit 31: Display ON/OFF

Remarks

See also:

GsMAP

Map comprising BG.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsMAP {
    unsigned char cellw, cellh;
    unsigned short ncellw, ncellh;
    GsCELL *base;
    unsigned short *index;
};
```

Members

<i>cellw</i> , <i>cellh</i>	Cell size (0 is treated as 256.)
<i>ncellw</i> , <i>ncellh</i>	Size of BG (in cells) (Not displayed if w or h is 0.)
<i>base</i>	Pointer to GsCELL structure array
<i>index</i>	Pointer to cell information

Explanation

GsMAP is map data used to compose BG from GsCELL. Map data are managed by cell index array information.

cellw, *cellh* specify the size of one cell in pixels. Note that one BG is made up of cells of the same size.

ncellw and *ncellh* set the size of the BG map in cells.

base sets the starting address of the GsCELL array.

index sets the starting address of the cell data table. Cell data is a list of index values whose size is equivalent to (*ncellw***ncellh*) for the array specified by *base*. If a cell value is 0xFFFF it indicates a NULL (transparent) cell.

Remarks

See also:

GsOBJTABLE2

Object table information.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsDOBJTABLE2 {
    GsDOBJ2 *top;
    int nobj;
    int maxnobj;
};
```

Members

top Pointer to object array
maxobj Size of object array
nobj Number of valid objects in array

Explanation

When the three-dimensional animation function group is used, a three-dimensional object must be in the array in order to give effect to the object ID number specification. This array is called an object table. GsOBJTABLE2 contains information relating to the object table.

top is a pointer to the GsDOBJ2 array, within which the three-dimensional object managed by ID is created. The GsDOBJ2 array must be allocated prior to object table initialization.

maxobj is the size of array indicated by *top*; its value must be greater than the maximum value of the object handled.

nobj is the number of valid objects within the array.

GsOBJTABLE2 is initialized by GsInitObjTable2().

Remarks

See also:

GsOT

Ordering table header.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	6/22/98

Structure

```
struct GsOT {
    unsigned long length;
    GsOT_TAG *org;
    unsigned long offset;
    unsigned long point;
    GsOT_TAG *tag;
};
```

Members

length Bit length of OT
org Pointer to start address of GsOT_TAG table
offset OT screen coordinate system Z-axis offset
point OT screen coordinate system Z-axis typical value
tag Pointer to GsOT_TAG currently located at the start

Explanation

The GsOT structure describes the header of the ordering table format supported by libgs. This header has pointers to the actual ordering table array, specified by the *org* and *tag* members. These members are initialized using the GsClearOt() function.

The *org* member always points to the start of the ordering table. The *tag* field points to the element within the ordering table at which drawing will take place.

The *length* field indicates the size of the ordering table. It is a value from 1-14 where the actual ordering table size is $2^{**length}$ (i.e. a value of 14 indicates an array of 16384 GsOT_TAG items, while a value of 8 indicates an array of 256 GsOT_TAG items).

length sets the size of the ordering table to values from 1-14. If the value "1" is specified, *org* points to a GsOT_TAG array running from 0-1. If the value "14" is specified, *org* points to a GsOT_TAG array running from 0-16384.

The GsClearOt() function initializes memory from *org* through to the size indicated by *length*. Note that memory will be destroyed if the size of the GsOT_TAG array pointed to by *org* is greater than that specified by *length*.

point is used by the GsSortOt() function in the sorting of ordering tables.

The ordering table Z-axis offset is set by *offset*. For example, if *offset* = 256, the start of the ordering table is Z = 256. (Not yet supported.)

Remarks

length and *org* values should be set first. The other members are set by the GsClearOt() function.

See also: GsClearOt (), GsDrawOt (p. 9-42), GsSortOt (p. 9-105), GsCutOt(9-41).

GsOT_TAG

Ordering table unit.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsOT_TAG {
    unsigned p : 24;
    unsigned char num : 8;
};
```

Members

p Pointer to next item in ordering table list
num Number of words in current GPU packet (i.e. primitive)

Explanation

A libgs ordering table is a linked list of GsOT_TAG structures and various types of GPU primitive structures. The *p* field of a GsOT_TAG structures indicates the least significant 24-bits of a pointer to the next item in the list. A value of 0xFFFFFFFF indicates the end of the list.

The GsOT structure is used by libgs to manage an array of GsOT_TAG items. Allocate an array of GsOT_TAG structures after initializing your GsOT structure.

Remarks

See also:

GsVIEW2

Viewpoint position (Reference type).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsVIEW2 {
    long vpx, vpy, vpz;
    long vrpx, vrpy, vrpz;
    long rz;
    GsCOORDINATE2 *super;
};
```

Members

<i>vpx, vpy, vpz</i>	Viewpoint coordinates
<i>vrpx, vrpy, vrpz</i>	Reference point coordinates
<i>rz</i>	Viewpoint twist
<i>super</i>	Pointer to the coordinate system that sets the viewpoint (GsCOORDINATE2type)

Explanation

GsVIEW2 holds viewpoint information, and is set in libgs by the GsSetRefView2() function.

The viewpoint coordinates in the coordinate system displayed by *super* are set in *vpx, vpy, vpz*.

The reference point coordinates in the coordinate system displayed by *super* are set in *vrpx, vrpy, vrpz*.

When the z axis is a vector from the viewpoint to the reference point, *rz* specifies the screen inclination against the z axis in fixed decimal format, with 4096 set to one degree.

Viewpoint and reference point coordinate systems are set in *super*. As an example of using this function, an airplane cockpit view can be realized simply by setting *super* to the airplane coordinate system.

Remarks

See also:

GsSPRITE

Sprite handler.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	3/25/98

Structure

```
struct GsSPRITE {
    unsigned long attribute;
    short x, y;
    unsigned short w, h;
    unsigned short tpage;
    unsigned char u, v;
    short cx, cy;
    unsigned char r, g, b;
    short mx, my;
    short scalex, scaley;
    long rotate;
};
```

Members

attribute 32 bits
 Bit 6: Brightness adjustment
 0: OFF
 1: ON
 Bit 22: Vertical flip
 0: not flipped
 1: flipped
 Bit 23: Horizontal flip
 0: not flipped
 1: flipped
 Bits 24-25: Sprite pattern bit mode
 0: 4-bit CLUT
 1: 8-bit CLUT
 2: 15-bit Direct
 Bit 27: Rotation, enlargement, and reduction functions
 0: ON
 1: OFF
 Bits 28-29: Semi-transparency rate
 0: 50% x Back + 50% x Sprite
 1: 100% x Back + 100% x Sprite
 2: 100% x Back + 50% x Sprite
 3: 100% x Back - 100% x Sprite
 Bit 30: Semi-transparency ON/OFF
 0: Semi-transparency OFF
 1: Semi-transparency ON
 Bit 31:
 0: Displayed
 1: Not displayed
 NOTE: Bit 26 is not supported as yet.

x, y Display position of the top left point
w, h Width and height of the Sprite (Not displayed if w or h is 0.)
tpage Sprite pattern texture page number
u, v Sprite pattern offset within the page

<i>cx, cy</i>	Sprite CLUT address
<i>r, g, b</i>	Display brightness is set in r, g, b (Normal brightness is 128.)
<i>mx, my</i>	Rotation and enlargement central point coordinates
<i>scalex, scaley</i>	Scale values in x and y directions
<i>rotate</i>	Rotation angle (4096 = 1 degree)

Explanation

GsSPRITE is a structure used to display a Sprite. This structure makes it possible to manipulate each Sprite via its parameters.

To register a GsSPRITE in the ordering table, use GsFlipSprite(), GsSortSprite(), or GsSortFastSprite().

x, y specifies the screen display position. (*mx, my*) specifies the point in the Sprite pattern used as the display position in GsSortSprite(); in GsSortFastSprite(), the point at the top left of the Sprite is used as the display position.

w, h specifies the width and height of the Sprite in pixels.

tpage specifies the texture page number (0-31) of the Sprite pattern.

u, v specifies the offset within the page from the top left point of the Sprite pattern. The range that may be specified is (0, 0) - (255, 255).

cx, cy specifies the starting position of CLUT (color palette) as a VRAM address. (Valid for 4-bit/8-bit mode only)

r, g, b specify the brightness values for red, green, and blue. The range is 0 to 255. 128 is the brightness of the original pattern; 255 doubles the brightness.

mx, my specify the coordinates used as the center of rotation and scaling. The top left point of the Sprite is the point of origin. For example, if rotation is around the center of the Sprite, specify *w/2* and *h/2*.

scalex, scaley specifies enlargement/reduction values in the x and y directions. These values are expressed in units of 4096, which stands for 1.0 (i.e. is the same size as 1.0). You can set these values up to 8 times the original size.

rotate sets rotation around the z-axis according to fixed-decimal format, in which 4096 is 1 degree.

attribute is 32 bits, and sets various attributes for display. An explanation of each bit follows.

(a) Bits 0-5: Reserved, set to zero.

(b) Bit 6: Brightness adjustment ON/OFF switch

This bit sets Sprite pattern pixel colors according to (*r, g, b*) values. If this bit is set to 1, brightness is not adjusted, and (*r, g, b*) values are ignored.

(c) Bits 7-21: Reserved, set to zero.

(d) Bits 22-23: Vertical flipping, horizontal flipping

Sets Sprite pattern flipping display.

(e) Bits 24-25: Color mode

A Sprite pattern has 4-bit mode and 8-bit mode, both of which use the color table, and 15-bit mode, which directly displays colors. These bits are used to select any of these modes.

(f) Bit 26: Reserved, set to zero.

(g) Bit 27: Rotation enlargement/reduction function

This bit turns on or off the Sprite enlargement function. If rotation or enlargement of the Sprite is not needed, this bit should be set to OFF for high speed processing.

GsSortFastSprite() and GsSortFlipSprite() ignore this bit and always set the enlargement function to off.

(h) Bits 28-29: Semi-transparency rate

When semi-transparency is set to ON with bit 30, the semi-transparency rate sets the pixel-blending formula.

Table 9–6: Semi-transparency Rate

Value	Processing
0	Normal semi-transparency processing
1	Pixel addition
2	50% addition
3	Pixel subtraction

- (i) Bit 30: Semi-transparency ON/OFF

This sets semi-transparency ON/OFF.

This bit must be used with the uppermost bit (STP bit) of the texture color field (texture pattern when direct and CLUT color field when indexed) to set semi-transparency,. Also, the semi-transparency and non-transparency of each pixel unit may be controlled using this STP bit.

- (j) Bit 31: Display ON/OFF

This turns display ON and OFF.

Remarks

SPRT primitives are used when neither rotation, enlargement, nor reduction are performed by GsSortSprite(). Consequently, it is necessary to keep track of whether the uv, wh of a texture is odd or even.

GsSortFlipSprite() uses POLY_FT4 unconditionally.

With GsSortSprite(), wh is set to -1 to handle the lower-right texture page problem. Consequently, the texture will be displayed slightly enlarged. (When programming, the lower-right line can be included, since reduction takes place within the function.)

Since GsSortFlipSprite() displays POLY_FT4 at the original scale, the rendering rules dictate that the lower right corner cannot be used. When flipping, to ensure proper display, the lower left line should not be used either.

See also:

GsVIEW2

Viewpoint position (matrix type).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Structure

```
struct GsVIEW2 {
    MATRIX view;
    GsCOORDINATE *super;
};
```

Members

view Matrix used to change from superior coordinates to viewpoint coordinates
super Pointer to the coordinate system that sets viewpoint

Explanation

This sets the viewpoint coordinate system. It specifies the matrix used by view to change from superior coordinates to viewpoint coordinates.

The function that sets GsVIEW2 is GsSetView2().

Remarks

See also:

TMD_STRUCT

TMD data object header.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```

typedef struct {
    unsigned long *vertop;
    unsigned long vern;
    unsigned long *nortop;
    unsigned long norn;
    unsigned long *primtop;
    unsigned long primn;
    unsigned long scale;
} TMD_STRUCT;

```

Members

<i>vertop</i>	VERTEX start address
<i>vern</i>	VERTEX coefficient
<i>nortop</i>	NORMAL start address
<i>norn</i>	NORMAL coefficient
<i>primtop</i>	PRIMITIVE start address
<i>primn</i>	PRIMITIVE coefficient
<i>scale</i>	Scaling factor

Explanation

This is a structure in the OBJ TABLE section within the TMD data. It contains information regarding the pointer which displays where each object is stored.

Remarks

See also:

_GsFCALL

The function table of GsSortObject5J(),GsSortObject4J().

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	6/22/98

Structure

```
struct _GsFCALL {
    PACKET *(*f3[2][3])(), *(*nf3[2])(), *(*g3[2][3])(), *(*ng3[2])();
    PACKET *(*tf3[2][3])(), *(*ntf3[2])(), *(*tg3[2][3])(), *(*ntg3[2])();
    PACKET *(*f4[2][3])(), *(*nf4[2])(), *(*g4[2][3])(), *(*ng4[2])();
    PACKET *(*tf4[2][3])(), *(*ntf4[2])(), *(*tg4[2][3])(), *(*ntg4[2])();
    PACKET *(*f3g[3])(), *(*g3g[3])();
    PACKET *(*f4g[3])(), *(*g4g[3])();
};
```

Members

Each member is a pointer to a low-level function.

<i>f3, g3, tf3, tg3, f4, g4, tf4, tg4</i>	Pointer to polygon types
First matrix: GsDivMODE_DIV/GsDivMode_NDIV	Division/no division
Second matrix: GsLMODE_NORMAL/GsLMODE_FOG/GsLMODE_LOFF	Light source calculation mode
<i>nf3, ng3, ntf3, ntg3, nf4, ng4, ntf4, ntg4</i>	Pointer to polygon types
First matrix: GsDivMODE_DIV/GsDivMode_NDIV	Division/no division
<i>f3g, g3g, f4g, g4g</i>	Gradation polygon type
First arrayGsLMODE_NORMAL/GsLMODE_FOG/GsLMODE_LOFF	Light source calculation mode

Explanation

GsSortObject5(),GsSortObject4() dispatches attributes, pre-set data, etc. and calls low-level functions. There are 64 low-level functions, and a single application is unlikely to use all of them.

You don't need to link GsSortObject5J() and GsSortObject4J() with unnecessary low-level functions, thereby making the code more compact. These functions are compatible with GsSortObject5() and GsSortObject4(), which organize low-level functions as tables.

_GsFCALL is the structure in which the function table is defined.The function table is organized according to polygon type, whether or not division is performed, and the light-source calculation mode.

The relevant functions are linked by entering the pointers of the appropriate low-level functions in each of the elements. It is possible to avoid linking by not including the pointers and not making extern declarations. However, if a function that does not have a pointer is called, a BUS ERROR will be generated.

The example below shows the use of GsSortObject5() with appropriate functions in all the elements. In this example, GsSortObject5J() functions the same as GsSortObject5(). This example is included in comments in the file libgs.h.

```
/* extern and fook only using functions */
extern _GsFCALL GsFCALL5; /* GsSortObject5J Func Table */
jt_init() /* Gs SortObject5J Fook Func */
{
    PACKET *GsPrstF3NL(),*GsPrstF3LFG(),*GsPrstF3L(),*GsPrstNF3();
    PACKET *GsTMDdivF3NL(),*GsTMDdivF3LFG(),*GsTMDdivF3L(),*GsTMDdivNF3();
    PACKET *GsPrstG3NL(),*GsPrstG3LFG(),*GsPrstG3L(),*GsPrstNG3();
    PACKET *GsTMDdivG3NL(),*GsTMDdivG3LFG(),*GsTMDdivG3L(),*GsTMDdivNG3();
    PACKET *GsPrstTF3NL(),*GsPrstTF3LFG(),*GsPrstTF3L(),*GsPrstTNF3();
    PACKET *GsTMDdivTF3NL(),*GsTMDdivTF3LFG(),*GsTMDdivTF3L(),*GsTMDdivTNF3();
}
```

```

PACKET *GsPrstTG3NL(), *GsPrstTG3LFG(), *GsPrstTG3L(), *GsPrstTNG3();
PACKET *GsTMDdivTG3NL(), *GsTMDdivTG3LFG(), *GsTMDdivTG3L(), *GsTMDdivTNG3();
PACKET *GsPrstF4NL(), *GsPrstF4LFG(), *GsPrstF4L(), *GsPrstNF4();
PACKET *GsTMDdivF4NL(), *GsTMDdivF4LFG(), *GsTMDdivF4L(), *GsTMDdivNF4();
PACKET *GsPrstG4NL(), *GsPrstG4LFG(), *GsPrstG4L(), *GsPrstNG4();
PACKET *GsTMDdivG4NL(), *GsTMDdivG4LFG(), *GsTMDdivG4L(), *GsTMDdivNG4();
PACKET *GsPrstTF4NL(), *GsPrstTF4LFG(), *GsPrstTF4L(), *GsPrstTNF4();
PACKET *GsTMDdivTF4NL(), *GsTMDdivTF4LFG(), *GsTMDdivTF4L(), *GsTMDdivTNF4();
PACKET *GsPrstTG4NL(), *GsPrstTG4LFG(), *GsPrstTG4L(), *GsPrstTNG4();
PACKET *GsTMDdivTG4NL(), *GsTMDdivTG4LFG(), *GsTMDdivTG4L(), *GsTMDdivTNG4();
PACKET *GsPrstF3GNL(), *GsPrstF3GLFG(), *GsPrstF3GL();
PACKET *GsPrstG3GNL(), *GsPrstG3GLFG(), *GsPrstG3GL();

/* flat triangle */
GsFCALL5.f3[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstF3L;
GsFCALL5.f3[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstF3LFG;
GsFCALL5.f3[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstF3NL;
GsFCALL5.f3[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivF3L;
GsFCALL5.f3[GsDivMODE_DIV][GsLMODE_FOG]      = GsTMDdivF3LFG;
GsFCALL5.f3[GsDivMODE_DIV][GsLMODE_LOFF]     = GsTMDdivF3NL;
GsFCALL5.nf3[GsDivMODE_NDIV]                  = GsPrstNF3;
GsFCALL5.nf3[GsDivMODE_DIV]                   = GsTMDdivNF3;
/* gour triangle */
GsFCALL5.g3[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstG3L;
GsFCALL5.g3[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstG3LFG;
GsFCALL5.g3[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstG3NL;
GsFCALL5.g3[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivG3L;
GsFCALL5.g3[GsDivMODE_DIV][GsLMODE_FOG]      = GsTMDdivG3LFG;
GsFCALL5.g3[GsDivMODE_DIV][GsLMODE_LOFF]     = GsTMDdivG3NL;
GsFCALL5.ng3[GsDivMODE_NDIV]                  = GsPrstNG3;
GsFCALL5.ng3[GsDivMODE_DIV]                   = GsTMDdivNG3;
/* texture flat triangle */
GsFCALL5.tf3[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstTF3L;
GsFCALL5.tf3[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstTF3LFG;
GsFCALL5.tf3[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstTF3NL;
GsFCALL5.tf3[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivTF3L;
GsFCALL5.tf3[GsDivMODE_DIV][GsLMODE_FOG]      = GsTMDdivTF3LFG;
GsFCALL5.tf3[GsDivMODE_DIV][GsLMODE_LOFF]     = GsTMDdivTF3NL;
GsFCALL5.ntf3[GsDivMODE_NDIV]                  = GsPrstTNF3;
GsFCALL5.ntf3[GsDivMODE_DIV]                   = GsTMDdivTNF3;
/* texture gour triangle */
GsFCALL5.tg3[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstTG3L;
GsFCALL5.tg3[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstTG3LFG;
GsFCALL5.tg3[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstTG3NL;
GsFCALL5.tg3[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivTG3L;
GsFCALL5.tg3[GsDivMODE_DIV][GsLMODE_FOG]      = GsTMDdivTG3LFG;
GsFCALL5.tg3[GsDivMODE_DIV][GsLMODE_LOFF]     = GsTMDdivTG3NL;
GsFCALL5.ntg3[GsDivMODE_NDIV]                  = GsPrstTNG3;
GsFCALL5.ntg3[GsDivMODE_DIV]                   = GsTMDdivTNG3;
/* flat quad */
GsFCALL5.f4[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstF4L;
GsFCALL5.f4[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstF4LFG;
GsFCALL5.f4[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstF4NL;
GsFCALL5.f4[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivF4L;
GsFCALL5.f4[GsDivMODE_DIV][GsLMODE_FOG]      = GsTMDdivF4LFG;
GsFCALL5.f4[GsDivMODE_DIV][GsLMODE_LOFF]     = GsTMDdivF4NL;
GsFCALL5.nf4[GsDivMODE_NDIV]                  = GsPrstNF4;
GsFCALL5.nf4[GsDivMODE_DIV]                   = GsTMDdivNF4;
/* gour quad */
GsFCALL5.g4[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstG4L;
GsFCALL5.g4[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstG4LFG;
GsFCALL5.g4[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstG4NL;
GsFCALL5.g4[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivG4L;
GsFCALL5.g4[GsDivMODE_DIV][GsLMODE_FOG]      = GsTMDdivG4LFG;
GsFCALL5.g4[GsDivMODE_DIV][GsLMODE_LOFF]     = GsTMDdivG4NL;
GsFCALL5.ng4[GsDivMODE_NDIV]                  = GsPrstNG4;
GsFCALL5.ng4[GsDivMODE_DIV]                   = GsTMDdivNG4;
/* texture flat quad */

```

```

GsFCALL5.tf4[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstTF4L;
GsFCALL5.tf4[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstTF4LFG;
GsFCALL5.tf4[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstTF4NL;
GsFCALL5.tf4[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivTF4L;
GsFCALL5.tf4[GsDivMODE_DIV][GsLMODE_FOG]      = GsTMDdivTF4LFG;
GsFCALL5.tf4[GsDivMODE_DIV][GsLMODE_LOFF]     = GsTMDdivTF4NL;
GsFCALL5.ntf4[GsDivMODE_NDIV]                  = GsPrstTNF4;
GsFCALL5.ntf4[GsDivMODE_DIV]                   = GsTMDdivTNF4;
/* texture gour quad */
GsFCALL5.tg4[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstTG4L;
GsFCALL5.tg4[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstTG4LFG;
GsFCALL5.tg4[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstTG4NL;
GsFCALL5.tg4[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivTG4L;
GsFCALL5.tg4[GsDivMODE_DIV][GsLMODE_FOG]      = GsTMDdivTG4LFG;
GsFCALL5.tg4[GsDivMODE_DIV][GsLMODE_LOFF]     = GsTMDdivTG4NL;
GsFCALL5.ntg4[GsDivMODE_NDIV]                  = GsPrstTNG4;
GsFCALL5.ntg4[GsDivMODE_DIV]                   = GsTMDdivTNG4;
/* gradation triangle */
GsFCALL5.f3g[GsLMODE_NORMAL]                  = GsPrstF3GL;
GsFCALL5.f3g[GsLMODE_FOG]                     = GsPrstF3GLFG;
GsFCALL5.f3g[GsLMODE_LOFF]                    = GsPrstF3GNL;
GsFCALL5.g3g[GsLMODE_NORMAL]                  = GsPrstG3GL;
GsFCALL5.g3g[GsLMODE_FOG]                     = GsPrstG3GLFG;
GsFCALL5.g3g[GsLMODE_LOFF]                    = GsPrstG3GNL;
}

```

Remarks

See also:

_GsPOSITION

Two-dimensional offset.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```

struct _GsPOSITION {
    short offx;
    short offy;
};

```

Members

offx Rendering offset (x direction)
offy Rendering offset (y direction)

Explanation

Two dimensional rendering offset.

Remarks

See also:

dmyGsPrst...

Jump Table Insignificant function group (Dummy).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	7/31/96

Syntax

PACKET *dmyGsPrst... ()

Arguments

None

Explanation

When this function is called for the first time, the jump table entry name is printed in standard output. It is used as an insignificant function dummy and is utilized when distinguishing which entry is being called.

Return value

Returns the pointer to the packet.

Remarks

For debugging use.

See also:

dmyGsTMD...

Jump Table Insignificant function group (Dummy).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	7/31/96

Syntax

PACKET *dmyGsTMD... ()

Arguments

None

Explanation

When this function is called for the first time, the jump table entry name is printed in standard output. It is used as an insignificant function dummy and is utilized when distinguishing which entry is being called.

Return value

Returns the pointer to the packet.

Remarks

For debugging use.

See also:

GsA4div...

Low-level functions for GsSortObject4J() (performs automatic division).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	10/01/97

Syntax

```
PACKET *GsA4div... (
  TMD_P_... *op,
  VERT *vp,
  VERT *np,
  PACKET *pk,
  int n,
  int shift,
  GsOT *ot,
  unsigned long *scratch
)
```

```
PACKET *GsA4divN... (
  TMD_P_... *op,
  VERT *vp,
  PACKET *pk,
  int n,
  int shift,
  GsOT *ot,
  unsigned long *scratch
)
```

Arguments

<i>op</i>	Pointer to starting address of TMD data primitives
<i>vp</i>	Pointer to starting address of TMD data vertices TMD
<i>np</i>	Pointer to starting address of TMD data normals
<i>pk</i>	Pointer to top address of GPU packet buffer
<i>n</i>	Number of primitives
<i>shift</i>	OT shift bit
<i>ot</i>	Pointer to GsOT
<i>scratch</i>	Pointer to starting address of unused scratch pad

Explanation

Performs active automatic division based on Z-values, polygon size, etc.

To use these functions, they must be registered in GsFCALL4 as low-order functions, and the number of divisions must be specified in the attributes of GsDOBJ4.

Because each of these functions uses a relatively large amount of code, it would be more efficient to use only the code needed for the polygon types used.

Parameters for division include Z-values, polygon size, and GTE calculation overflow flags. These are set using the GsSetAzwh (az, aw, ah) macro.

The active division algorithm is as follows:

1. Do not divide polygons that are further away than az and that do not cause overflow in GTE calculations.
2. If cases other than 1, perform divisions (go to step 3).
3. If polygon size does not exceed aw, ah, and there is no overflow in GTE calculations, then halt division there.

Otherwise, reduce by 1/2 in the x and y directions, and divide into four sections. Call step 3 recursively. If the maximum value for divisions (the number of divisions in attribute) is reached, then halt division.

For function types which do not operate on normals within the data (e.g. GsA4divN...), light source calculations are not performed so fewer parameters are passed compared to those function types which operate on normals (e.g. GsA4div...),

Low-level functions in libs that support automatic division are shown below.

GsA4div...() [have normals]

Low-level function name	First arg (op) type	Description
GsA4divF3L	TMD_P_F3	Flat triangle (light source calculation)
GsA4divF3LFG	TMD_P_F3	Flat triangle (light source calculation +FOG)
GsA4divF3NL	TMD_P_F3	Flat triangle
GsA4divF4L	TMD_P_F4	Flat quadrilateral (light source calculation)
GsA4divF4LFG	TMD_P_F4	Flat quadrilateral (light source calculation +FOG)
GsA4divF4NL	TMD_P_F4	Flag quadrilateral
GsA4divG3L	TMD_P_G3	Gouraud triangle (light source calculation)
GsA4divG3LFG	TMD_P_G3	Gouraud triangle (light source calculation +FOG)
GsA4divG3NL	TMD_P_G3	Gouraud triangle
GsA4divG4L	TMD_P_G4	Gouraud quadrilateral (light source calculation)
GsA4divG4LFG	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG)
GsA4divG4NL	TMD_P_G4	Gouraud quadrilateral
GsA4divTF3L	TMD_P_TF3	Textured flat triangle (light source calculation)
GsA4divTF3LFG	TMD_P_TF3	Textured flat triangle (light source calculation +FOG)
GsA4divTF3NL	TMD_P_TF3	Textured flat triangle
GsA4divTF4L	TMD_P_TF4	Textured flat quadrilateral (light source calculation)
GsA4divTF4LFG	TMD_P_TF4	Flat quadrilateral (light source calculation +FOG)
GsA4divTF4NL	TMD_P_TF4	Textured flat quadrilateral
GsA4divTF4LM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +mip-map)
GsA4divTF4LFGM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG+mip-map)
GsA4divTF4NLM	TMD_P_TF4	Textured flat quadrilateral (mip-map)
GsA4divTG3L	TMD_P_TG3	Textured Gouraud triangle (light source calculation)
GsA4divTG3LFG	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG)

Low-level function name	First arg (op) type	Description
GsA4divTG3NL	TMD_P_TG3	Textured Gouraud triangle
GsA4divTG4L	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation)
GsA4divTG4LFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG)
GsA4divTG4NL	TMD_P_TG4	Textured Gouraud quadrilateral
GsA4divTG4LM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +mip-map)
GsA4divTG4LFGM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation+FOG+mip-map)
GsA4divTG4NLM	TMD_P_TG4	Textured Gouraud quadrilateral (mip-map)

GsA4divN...() [no normals]

Low-level function name	First arg (op) type	Description
GsA4divNF3	TMD_P_NF3	Flat triangle
GsA4divNF4	TMD_P_NF4	Flat quadrilateral
GsA4divNG3	TMD_P_NG3	Gouraud triangle
GsA4divNG4	TMD_P_NG4	Gouraud quadrilateral
GsA4divTNF3	TMD_P_TNF3	Textured flat triangle
GsA4divTNF4	TMD_P_TNF4	Textured flat quadrilateral
GsA4divTNF4M	TMD_P_TNF4	Textured flat quadrilateral (mip-map)
GsA4divTNG3	TMD_P_TNG3	Textured Gouraud triangle
GsA4divTNG4	TMD_P_TNG4	Textured Gouraud quadrilateral
GsA4divTNG4M	TMD_P_TNG4	Textured Gouraud quadrilateral (mip-map)

Return value

Starting address of unused packet area.

Remarks

GsTMDdiv functions must be registered in GsFCALL4 when using the conventional fixed division method.

See also: GsTMDdiv..., GsSetAzwh (p. 9-122), GsSortObject4J().

GsClearDispArea

Clears screen.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

GsClearDispArea(*r,g,b*)
unsigned char *r,g,b*;

Arguments

r,g,b Background color RGB values

Explanation

The display area is cleared using IO.

Return value

Remarks

Unlike GsSortClear, a clear command is issued when GsClearDispArea() is called.

See also:

GsClearOt

Initializes a libgs ordering table structure.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsClearOt(offset, point, *otp)
unsigned short offset;
unsigned short point;
GsOT *otp;
```

Arguments

offset Ordering table offset value
point Ordering table typical value Z
otp Pointer to ordering table

Explanation

This function initializes the libgs-style ordering table specified by the *otp* parameter. The *length* field of the GsOT structure must be properly set before this function is called. The *offset* parameter specifies the Z-depth value used for the start of the ordering table. The *point* offset represents the Z-depth of the entire ordering table and is used to determine depth priority when linking multiple ordering tables together.

Return value

None

Remarks

See also:

GsClearVcount

Clears vertical retrace counter.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	7/31/96

Syntax

void GsClearVcount(*void*)

Arguments

None

Explanation

This function clears the vertical retrace counter.

Return value

None

Remarks

See also:

GsCutOt

OT separation.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

GsOT *GsCutOt (**ot_src*, **ot_dest*)

GsOt **ot_src*;

GsOt **ot_dest*;

Arguments

ot_src Pointer to old OT

ot_dest Pointer to new OT

Explanation

The GsCutOt() function moves the drawing commands registered in the *ot_src* ordering table to the *ot_dest* ordering table. The *length* and *tag* fields of *ot_src* are reset to zero. The *tag* field of *ot_dest* is updated to point at the drawing command which was at the start of *ot_src*. Afterwards, *ot_dest* can be used to access the ordering table.

Return value

ot_dest starting address.

Remarks

See also:

GsDefDispBuff

Defines double buffers.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	5/22/97

Syntax

```
void GsDefDispBuffer(x0, y0, x1, y1)
    unsigned short x0, y0;
    unsigned short x1, y1;
```

Arguments

x0, y0 Buffer 0 origin point coordinates (top left point)
x1, y1 Buffer 1 origin point coordinates (top left point)

Explanation

This function defines the display areas used for double-buffering.

The *x0* & *y0* parameters specify the coordinates within the frame buffer for buffer #0. The *x1* & *y1* parameters specify the coordinates within the frame buffer for buffer #1. Normally, buffer #0 is located at (0,0) and buffer #1 is located at (0, *yres*), where *yres* is the vertical resolution specified using the `GsInitGraph()` function.

If *x0, y0* and *x1, y1* are specified as the same coordinates, the double buffers are released. However, double-buffer swapping of even-numbered and odd-numbered fields is performed automatically when *x0, y0* and *x1, y1* are specified as the same coordinates in interlace mode.

The `GsSwapDispBuffer()` function is used to swap double buffers. The double buffer is implemented by the GPU/GTE offset. Set the `libgpu` or `libgte` offset with `GsInitGraph()`. When using the `libgpu` offset, coordinate values based on the coordinate system using the upper left point in the double buffer as the origin will be created in the packet (add the offset at the time of drawing, not at the time of packet preparation).

Return value

None

Remarks

See also:

GsDefDispBuff2

Defines double buffers.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	5/22/97

Syntax

```
void GsDefDispBuff2(x0, y0, x1, y1)
    unsigned short x0, y0;
    unsigned short x1, y1;
```

Arguments

x0, y0 Buffer 0 origin point coordinates (top left point)
x1, y1 Buffer 1 origin point coordinates (top left point)

Explanation

This function defines the double buffer.

Differs from GsDefDispBuff only in the modification of internal variables. These modifications are not updated in libgpu and libgte until GsSwapDispBuff() is called.

Settings can be changed in the middle of the program without affecting the screen.

Return value

None

Remarks

This first time this function is used, it should be set with GsDefDispBuff().

See also:

GsDrawOt

Processes GPU commands registered to OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	<i>2.x</i>	<i>02/15/98</i>

Syntax

```
void GsDrawOt (  
  GsOT *ot  
)
```

Arguments

ot Pointer to OT

Explanation

This function starts execution of commands registered in OT, specified by *ot*. Because processing is performed in the background, GsDrawOt() returns immediately.

Return value

None

Remarks

This function does not execute properly when GPU operations are already in progress. Prior to calling GsDrawOT, use either ResetGraph(1) to terminate any ongoing GPU operations or DrawSync() to detect completion of previous operations.

See also:

GsDrawOtIO

Processes GPU commands (I/O version) allocated to OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.5	02/15/98

Syntax

```
void GsDrawOtIO (
  GsOT *ot
)
```

Arguments

ot Pointer to OT

Explanation

Starts the execution of commands registered in OT, indicated by *ot*. Unlike GsDrawOt(), the processing is performed in the foreground; thus this function does not return until drawing is completed.

Return value

None

Remarks

Mainly used for debugging.

See also:

GsGetActiveBuffer

Gets a buffer number during drawing.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
int GsGetActiveBuffer(void)
```

Arguments

None

Explanation

This function gets a double buffer index. Index values are either 0 or 1.

By entering indexes in the external variables, PSDBASEX[] and PSDBASEY[], it is possible to determine the two-dimensional address of the double buffer origin point (top left coordinates) in the frame buffer.

Return value

Index of a double buffer (0 for buffer 0, and 1 for buffer 1)

Remarks

See also:

GsGetLs

Calculating a local screen matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsGetLs(*coord, *m)
GsCOORDINATE2 *coord;
MATRIX *m;
```

Arguments

coord Pointer to local coordinates
m Pointer to matrix

Explanation

This function calculates a local screen perspective transformation matrix from the GsCOORDINATE2 structure pointed to by the *coord* argument and stores the result in the MATRIX structure pointed to by the *m* argument.

For high speed operation, the function retains the result of calculation at each node of the hierarchical coordinate system. When the next GsGetLs() function is called, calculation up to the node to which no changes have been made is omitted. This is controlled by a GsCOORDINATE2 member flag (libgs replaces 1 in flags already calculated by GsCOORDINATE2).

If the contents of a superior node are changed, the effect on a subordinate node is handled by libgs, so it is not necessary to clear the flags of all subordinate nodes of the changed superior node.

Return value

None

Remarks

See also:

GsGetLw

Calculating a local world matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsGetLw(*coord, *m)
GsCOORDINATE2 *coord;
MATRIX *m;
```

Arguments

coord Pointer to local coordinate system
m Pointer to matrix

Explanation

This function calculates a local world perspective transformation matrix from the GsCOORDINATE2 structure pointed to by the *coord* argument and stores the result in the MATRIX structure pointed to by the *m* argument.

For high speed operation, the function retains the result of calculation at each node of the hierarchical coordinate system. When the next GsGetLw() function is called, calculation up to the node to which no changes have been made is omitted. This is controlled by a GsCOORDINATE2 member flag (libgs replaces 1 in flags already calculated by GsCOORDINATE2).

If the contents of a superior node are changed, the effect on a subordinate node is handled by libgs, so it is not necessary to clear the flags of all subordinate nodes of the changed superior node.

Return value

None

Remarks

See also:

GsGetLws

Calculates local world and local screen matrices.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
void GsGetLws (*coord2, *lw, *ls)
GsCOORDINATE2 *coord2;
MATRIX *lw, *ls;
```

Arguments

coord2 Pointer to local coordinates
lw Pointer to matrix that stores the local world coordinates
ls Pointer to matrix that stores the local screen coordinates

Explanation

GsGetLws() calculates local world and local screen coordinates. This function is faster than calling GsGetLw() followed by calling GsGetLs(). Light source calculations are performed at the time of application execution. When you use GsSetLightMatrix(), it is valid because you calculate the LW matrix.

Return value

None

Remarks

See also:

GsGetTimInfo

Finds TIM format header.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsGetTimInfo(*tim, *im)
unsigned long *tim;
GsIMAGE *im;
```

Arguments

tim Pointer to TIM data
im Pointer to an image Structure

Explanation

Fills in the GsIMAGE structure pointed to by the *im* parameter with the appropriate information obtained from the TIM data located at the address specified by the *tim* parameter.

Return value

None

Remarks

See also:

GsGetVcount

Gets the value of the vertical retrace counter.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

long GsGetVcount(*void*)

Arguments

None

Explanation

Obtains the value of the vertical retrace counter.

Return value

Value of the vertical retrace counter.

Remarks

See also:

GsGetWorkBase

Gets address for storing current drawing commands.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

PACKET *GsGetWorkBase(void)

Arguments

None

Explanation

Allocates and returns a pointer to a buffer used for generating a drawing primitive GPU packet.

Return value

Address to prepare the next drawing primitive packet.

Remarks

See also:

GsInit3D

Initializes the graphics system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

void GsInit3D(*void*)

Arguments

None

Explanation

GsIncFrame is a macro called from within GsSwapDispBuff().

It increments the global variable PSDCNT by 1. PSDCNT is 32 bits in length, and restarts at 1 rather than 0 when it overflows.

PSDCNT is used by GsGetLw(), GsGetLs(), GsGetLws() when determining the validity of the matrix cache.

If you are not using GsSwapDispBuff() to swap double buffers, you must call GsIncFrame to swap the buffers when you use GsGetLw(), GsGetLs(), and GsGetLws().

Return value

None

Remarks

See also:

GsInitCoordinate2

Initializes a local coordinate system (for use by GsCOORDINATE2).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsInitCoordinate2(*super, *base)
GsCOORDINATE2 *super;
GsCOORDINATE2 *base;
```

Arguments

super Pointer to a superior coordinate system
base Pointer to a coordinate system (to be initialized)

Explanation

base->coord is indicated in the coordinate system by a single determinant, base->super is indicated with an argument, and both are initialized.

Return value

None

Remarks

See also:

GsInitFixBg16

High-speed BG work area initialization

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
void GsInitFixBg16(*bg, *work)
GsBG *bg;
unsigned long *work;
```

Arguments

bg Pointer to GsBG
work Pointer to work area (primitive area)

Explanation

This function initializes the work area used by the functions GsSortFixBg16() and GsSortFixBg32. The size of the array differs with the screen mode as follows:

size (in long units)=(((ScreenW/CellW+1)•(ScreenH/CellH+1+1)•6+4)•2+2)

ScreenH: screen height in pixels (240/480)

ScreenW: screen width in pixels (256/320/384/512/640)

CellH: cell height (in pixels)

CellW: cell width (in pixels)

Executing GsInitFixBg16()/GsInitFixBg32() once is sufficient; you need not execute it for every frame.

Return value

None

Remarks

See also:

GsInitFixBg32

High-speed BG work area initialization

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
void GsInitFixBg32(*bg, *work)
GsBG *bg;
unsigned long *work;
```

Arguments

bg Pointer to GsBG
work Pointer to work area (primitive area)

Explanation

This function initializes the work area used by the functions GsSortFixBg16() and GsSortFixBg32. The size of the array differs with the screen mode as follows:

size (in long units)=(((ScreenW/CellW+1)•(ScreenH/CellH+1+1)•6+4)•2+2)

ScreenH: screen height in dots (240/480)

ScreenW: screen width in dots (256/320/384/512/640)

CellH: cell height (in pixels)

CellW: cell width (in pixels)

Executing GsInitFixBg16()/GsInitFixBg32() once is sufficient; you need not execute it for every frame.

Return value

None

Remarks

See also:

GsInitGraph

Initializes the graphics system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	5/22/97

Syntax

```
void GsInitGraph(x_res, y_res, int1, dither, vram)
unsigned short x_res;
unsigned short y_res;
unsigned short int1;
unsigned short dither;
unsigned short vram;
```

Arguments

<i>x_res</i>	Horizontal resolution (256/320/384/512/640)
<i>y_res</i>	Vertical resolution (240/480)
<i>int1</i>	Interlace display flag (bit 0)
	0: Non-interlace GsNONINTER
	1: Interlace GSINTER
	Double buffer offset mode (bit 2)
	0: GTE offset GsOFSGTE
	1: GPU offset GsOFSGPU
	GPU Initialize Parameter (bit 4-5)
	0: ResetGraph(0) GsRESET0
	3: ResetGraph(3) GsRESET3
<i>dither</i>	Dithering processing flag
	0: OFF
	1: ON
<i>vram</i>	VRAM mode
	0: 16-bit
	1: 24-bit

Explanation

Resets "libgpu" and initializes "libgs" graphic system. "libgpu" settings is notified by a global variables, GsDISPENV, and GsDRAWENV. Thus the programmer can verify and/or modify "libgpu" by referencing the settings.

Vertical 480 line non-interlace mode is effective only when a VGA monitor is connected. For the vertical 240 line mode, top and bottom 8 lines are almost invisible on the home-use TV monitors. Thus special attentions need to be given. For PAL mode, display position should be shifted down by 24 lines. Double buffer offset mode determines implementation of either GTE or GPU offset mode. When implemented as GPU, the packet does not include offset value and thus is easy to be handled.

For 24 bit mode, only the memory image display is available and thus polygon drawing cannot be done.

Since initialization of the graphic system involves initialization of GsIDMATRIX and GsIDMATRIX2 as well, GsInitGraph() must be called prior to all other Gs library functions for correct operation.

Return value

None

Remarks

See also:

GsInitGraph2

Initializes the graphics system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	5/22/97

Syntax

```
void GsInitGraph2(x_res, y_res, int1, dither, vram)
unsigned short x_res;
unsigned short y_res;
unsigned short int1;
unsigned short dither;
unsigned short vram;
```

Arguments

x_res Horizontal resolution (256/320/384/512/640)
y_res Vertical resolution (240/480)
int1 Interlace display flag (bit 0)
 0: Non-interlace
 1: Interlace
 Double buffer offset mode (bit 2)
 0: GTE offset
 1: GPU offset
dither Dither ON/OFF during drawing
 0: OFF
 1: ON
vram VRAM mode
 0: 16-bit
 1: 24-bit

Explanation

GsInitGraph2 is different from GsInitGraph in that the GPU is not initialized COLD. This function is useful for changing libgs resolution without affecting screen synchronization.

Return value

None

Remarks

Always use GsInitGraph() for the first initialization.

See also:

GsInitVcount

Initializes vertical retrace counter.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsInitVcount(void)
```

Arguments

None

Explanation

This function initializes the vertical retrace counter, and starts it.

Return value

None

Remarks

See also:

GsLinkObject3

Links an object with PMD data (For GsSortObject3).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
void GsLinkObject3(*pmd, *obj_base)
unsigned long *pmd;
GsDOBJ3 *obj_base;
```

Arguments

pmd Pointer to starting address of the PMD data to be linked
obj_base Pointer to array of the object structure to be linked

Explanation

Links GsDOBJ3 object structure to all objects contained in the PMD data, so that the PMD format three-dimensional object modelled can be handled by GsDOBJ3.

Return value

None

Remarks

Unlike GsLinkObject4(), it is not possible to select and link an object included in the PMD data. All objects contained in *pmd* will be linked to the object handler array beginning with *obj_base*.

See also:

GsLinkObject4

Links an object to TMD data (For GsSortObject4).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	5/22/97

Syntax

```
void GsLinkObject4(tmd, *obj_base, n)
unsigned long tmd;
GsDOBJ2 *obj_base;
int n;
```

Arguments

<i>tmd</i>	Starting address of the TMD data to be linked
<i>obj_base</i>	Array of the object structure to be linked
<i>n</i>	Index of the object to be linked

Explanation

Links GsDOBJ2 object structure to the *n*-th object of the TMD data so that the TMD format three-dimensional object modelled can be handled by GsDOBJ2.

Return value

None

Remarks

An object linked using GsLinkObject4() uses GsSortObject4() to create a packet.

See also:

GsLinkObject5

Links an object to TMD data (For GsSortObject5).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	5/22/97

Syntax

```
void GsLinkObject5(tmd, *obj_base, n)
unsigned long tmd;
GsDOBJ5 *obj_base;
int n;
```

Arguments

<i>tmd</i>	Starting address of the TMD data to be linked
<i>obj_base</i>	Array of the object structure to be linked
<i>n</i>	Index of the object to be linked

Explanation

Links GsDOBJ5 object structure to the *n*-th object of the TMD data so that the TMD format three-dimensional object modelled can be handled by GsDOBJ5.

Return value

None

Remarks

See also:

GsMapModelingData

Maps TMD data to real addresses.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsMapModelingData(*p)
unsigned long *p;
```

Arguments

p Pointer to starting address of TMD data

Explanation

TMD data includes various fields which contain the memory addresses of certain pieces of data. However, during the preparation of TMD data, the memory address where the data will be loaded is not yet known. Therefore, address fields in the TMD data are stored as offsets from the start of the data. The GsMapModelingData function changes these offsets into actual addresses after the TMD data has been loaded into memory. This must be done before the TMD data may be used.

Return value

None

Remarks

A flag is set in the TMD data whose offset addresses have been converted into real addresses. So, no side effect occurs even if GsMapModelingData() is called again.

See also:

GsMulCoord0

MATRIX multiplication.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
void GsMulCoord2(*m1, *m2)
MATRIX *m1, *m2;
```

Arguments

m1, *m2* Pointer to matrix

Explanation

This function multiplies MATRIX *m2* by the translation matrix. The results are stored in *m3*.

$m3 = m1 \times m2$

Return value

None

Remarks

See also:

GsMulCoord2

MATRIX multiplication.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsMulCoord2(*m1, *m2)
MATRIX *m1, *m2;
```

Arguments

m1, *m2* Pointer to matrix

Explanation

GsMulCoord2 multiplies the MATRIX *m2* by the translation matrix *m1* and stores the result in *m2*.

$m2 = m1 \times m2$

Return value

None

Remarks

See also:

GsMulCoord3

MATRIX multiplication.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsMulCoord3(*m1, *m2)
MATRIX *m1, *m2;
```

Arguments

m1, *m2* Pointer to matrix

Explanation

GsMulCoord3 multiplies the MATRIX *m2* by the translation matrix *m1* and stores the result in *m2*.

$m1 = m1 \times m2$

Return value

None

Remarks

See also:

GsPresetObject

Creates a preset packet for a GsDOBJ5-type object.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
unsigned long *GsPresetObject(*objp, *addr)
GsDOBJ5 *objp;
unsigned long *addr;
```

Arguments

objp Pointer to the object to be preset
addr Pointer to starting address of the area in which the preset packet is to be prepared.

Explanation

Presetting refers to the advance preparation of polygons of all objects as packets. The areas that need not be rewritten (e.g., U and V of texture) for each frame will not be rewritten, thus ensuring high speed.

The return value of GsPresetObject points to the address next to the last preset address, so when presetting the next object, preserve the return value and pass it as an argument of the next GsPresetObject(). The return value will indicate how large an area must be allocated for the preset area.

A GsDOBJ5 type object pointer is exclusively used for presetting.

Return value

Pointer that indicates the next to the last preset address.

Remarks

See also:

GsPrst...

Low-level functions for GsSortObject5J().

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	10/01/97

Syntax

```
PACKET *GsPrst... (
  TMD_P_... *op,
  VERT *vp,
  VERT *np,
  PACKET *pk,
  int n,
  int shift,
  GsOT *ot,
  unsigned long *scratch
)
```

```
PACKET *GsPrstN... (
  TMD_P_... *op,
  VERT *vp,
  PACKET *pk,
  int n,
  int shift,
  GsOT *ot,
  unsigned long *scratch
)
```

Arguments

<i>op</i>	Pointer to starting address of TMD data primitives
<i>vp</i>	Pointer to starting address of TMD data vertices TMD
<i>np</i>	Pointer to starting address of TMD data normals
<i>pk</i>	Pointer to top address of GPU packet buffer
<i>n</i>	Number of primitives
<i>shift</i>	OT shift bit
<i>ot</i>	Pointer to GsOT
<i>scratch</i>	Pointer to starting address of unused scratch pad

Explanation

These are low-level functions for GsSortObject5J().

To use these functions, they must be registered in GsFCALL5 as low-level functions.

These functions perform coordinate & perspective transformation, backface clipping, and light source calculation for *n* primitives, create the GPU packet in the buffer, and link it into the OT. There must be two preset packets in the buffer per polygon.

For function types which do not operate on normals within the data (e.g. GsPrstN...), light source calculations are not performed so fewer parameters are passed compared to those function types which operate on normals (e.g. GsPrst...),

Low-level functions in libgs that are supported are shown below.

GsPrst...() [have normals]

Low-level function name	First arg (op) type	Description
GsPrstF3L	TMD_P_F3	Flat triangle (light source calculation)
GsPrstF3LFG	TMD_P_F3	Flat triangle (light source calculation +FOG)
GsPrstF3NL	TMD_P_F3	Flat triangle
GsPrstF4L	TMD_P_F4	Flat quadrilateral (light source calculation)
GsPrstF4LFG	TMD_P_F4	Flat quadrilateral (light source calculation +FOG)
GsPrstF4NL	TMD_P_F4	Flag quadrilateral
GsPrstG3L	TMD_P_G3	Gouraud triangle (light source calculation)
GsPrstG3LFG	TMD_P_G3	Gouraud triangle (light source calculation +FOG)
GsPrstG3NL	TMD_P_G3	Gouraud triangle
GsPrstG4L	TMD_P_G4	Gouraud quadrilateral (light source calculation)
GsPrstG4LFG	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG)
GsPrstG4NL	TMD_P_G4	Gouraud quadrilateral
GsPrstTF3L	TMD_P_TF3	Textured flat triangle (light source calculation)
GsPrstTF3LFG	TMD_P_TF3	Textured flat triangle (light source calculation +FOG)
GsPrstTF3NL	TMD_P_TF3	Textured flat triangle
GsPrstTF4L	TMD_P_TF4	Textured flat quadrilateral (light source calculation)
GsPrstTF4LFG	TMD_P_TF4	Flat quadrilateral (light source calculation +FOG)
GsPrstTF4NL	TMD_P_TF4	Textured flat quadrilateral
GsPrstTG3L	TMD_P_TG3	Textured Gouraud triangle (light source calculation)
GsPrstTG3LFG	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG)
GsPrstTG3NL	TMD_P_TG3	Textured Gouraud triangle
GsPrstTG4L	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation)
GsPrstTG4LFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG)
GsPrstTG4NL	TMD_P_TG4	Textured Gouraud quadrilateral
GsPrstF3GL	TMD_P_F3G	Flat gradation triangle (light source calculation)
GsPrstF3GLFG	TMD_P_F3G	Flat gradation triangle (light source calculation +FOG)
GsPrstF3GNL	TMD_P_F3G	Flat gradation triangle
GsPrstG3GL	TMD_P_G3G	Gouraud gradation triangle (light source calculation)

Low-level function name	First arg (op) type	Description
GsPrstG3GLFG	TMD_P_G3G	Gouraud gradation triangle (light source calculation +FOG)
GsPrstG3GNL	TMD_P_G3G	Gouraud gradation triangle

GsPrstN...() [no normals]

Low-level function name	First arg (op) type	Description
GsPrstNF3	TMD_P_NF3	Flat triangle
GsPrstNF4	TMD_P_NF4	Flat quadrilateral
GsPrstNG3	TMD_P_NG3	Gouraud triangle
GsPrstNG4	TMD_P_NG4	Gouraud quadrilateral
GsPrstTNF3	TMD_P_TNF3	Textured flat triangle
GsPrstTNF4	TMD_P_TNF4	Textured flat quadrilateral
GsPrstTNG3	TMD_P_TNG3	Textured Gouraud triangle
GsPrstTNG4	TMD_P_TNG4	Textured Gouraud quadrilateral

Return value

Starting address of unused packet area.

Remarks

With gradation, each vertex of the TMD polygon has a different RGB value.

For high speed operation, libgte contains tuned assembly-level low-level functions.

See also: GsPresetObject(), GsSortObject5J()

GsScaleScreen

Scales the screen coordinate system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	7/31/96

Syntax

```
void GsScaleScreen (
SVECTOR *scale
)
```

Arguments

scale Pointer to the scale factor (12 bit fixed radix point format)
Always set the factor in relation to the original screen coordinate systems set on GsSetView2 () and GsSetRefView2 (). When ONE is inserted into vx, vy or vz it returns to the original.

Explanation

GsScaleScreen () scales the screen coordinate system against the world coordinates.

Unlike the world coordinates which have 32 bits of space, the screen coordinates have only 16 bits. Accordingly, this brings about problems such as FarClip being close.

In order to solve this, GsScaleScreen is provided to scale the screen coordinates and cover a larger area than world.

For example, when specifying ONE/2 to vx, vy or vz, the screen coordinate system is expanded to the equivalent of 17 bits. However, since the precision is 16 bits, the lower 1 bit will be invalid.

Attention must be paid here to make sure that the screen coordinate system which has a different scale is not registered to the OT with the same scale.

For example, in order to register an object calculated with the normal scaling screen coordinate system to the OT which has already registered an object with a 1/2 screen coordinate system scale, it is necessary to shift the excess 1 bit before registering.

When the scaling matrix set by GsScaleScreen to the external variable GsWSMATRIX, and the screen coordinates set by GsSetView2 and GsSetRefView2 to the external variable GsWSMATRIX_ORG are defined, the WSMATRIX is held.

Return value

None

Remarks

See also:

GsSetAmbient

Set color and brightness of ambient lighting.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	5/22/97

Syntax

```
void GsSetAmbient(r, g, b)
unsigned long r, g, b;
```

Arguments

r, g, b Ambient color RGB values (0-4095)

Explanation

This function sets the color and brightness of the ambient lighting in the 3D world. Values for red, green, and blue are set independently. A value of 4096 corresponds to normal ambient brightness, 0 to minimum brightness. Values greater than 4096 strengthen that color.

1/1 is 4096 and 1/8 is 4096/8.

Return value

None

Remarks

See also:

GsSetClip

Sets a drawing clipping area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
void GsSetClip(*clip)  
RECT *clip;
```

Arguments

clip Beginning address of a RECT structure for setting a clipping area

Explanation

Sets clipping for drawing. This function is different from GsSetDrawBuffClip() in that its argument can be used to specify a clip area. Note that this clipping value is a relative one within the double buffer, and thus the clip position will not change even if the double buffer is swapped.

Return value

None

Remarks

Clipping is done by libgpu.

See also:

GsSetClip2

Sets a drawing clipping area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
DRAWENV *GsSetClip2(*clip)
RECT *clip;
```

Arguments

clip Beginning address of a RECT structure for setting a clipping area

Explanation

Sets the clipping rectangle for drawing to the rectangle specified by *clip*. This function is different from GsSetClip in that the DRAWENV and DISPENV structures are not updated. The return value of GsSetClip2 is a pointer to a DRAWENV structure that can be used if necessary to set the system DRAWENV structure using PutDrawEnv. Note that the global DRAWENV must have been previously specified in order for the information in this structure to be valid.

Note that this clipping rectangle is relative to whichever is the current buffer, even if double-buffering is used.

Return value

Returns a pointer to an updated DRAWENV structure (which can be used to update the system DRAWENV structure if desired).

Remarks

Clipping is done by libgpu.

See also:

GsSetClip2D

Sets two-dimensional clipping.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

GsSetClip2D(*rectp)

RECT *rectp;

Arguments

rectp Pointer to the area to be clipped

Explanation

This function sets the area given by RECT as the area to be clipped.

This setting is affected by the double buffer. This means that the function leads to the automatic clipping of the same area even though the double buffer has been swapped.

GsSetDrawBuffClip must be invoked in order to validate this setting immediately afterwards.

If GsSetDrawBuffClip is not specifically invoked, the setting is valid from the next frame.

Return value

None

Remarks

See also:

GsSetDrawBuffClip

Sets drawing clipping area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSetDrawBuffClip(void)
```

Arguments

None

Explanation

This function sets clipping for drawing. The clipping value set by GsClip2D() is set in libgs.

This clipping value is a relative one within the double buffers. The clipping position does not change when double buffers are swapped.

Return value

None

Remarks

This function does not execute correctly if GPU drawing is in progress. Use ResetGraph(1) to terminate any current drawing process or DrawSync() to wait until the process is completed.

See also:

GsSetDrawBuffOffset

Sets the drawing offset.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSetDrawBuffOffset(void)
```

Arguments

None

Explanation

GsSetDrawBuffOffset sets the drawing offset. The offset value set in the global variable "POSITION" is updated.

This offset is relative within the double buffer. The offset value is preserved even if double buffers are swapped.

GsSetDrawBuffOffset sets the libgte or libgpu offset.

Note:

Using the GsOFSGPU or GsOFSGTE macro for the third argument of GsInitGraph() determines whether the libgte or libgpu offset should be set.

Return value

None

Remarks

This function does not execute correctly if GPU drawing is in progress. Use ResetGraph(1) to terminate any current drawing process or DrawSync() to wait until the process is completed.

See also:

GsSetFlatLight

Sets parallel light source.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSetFlatLight(id, *lt)
unsigned int id;
GsF_LIGHT *lt;
```

Arguments

id Light source number (0, 1, 2)
lt Pointer to light source data

Explanation

GsSetFlatLight sets a parallel light source. Up to three light sources (ID = 0, 1, 2) may be set. Light source data is given GsF_LIGHT structure.

Return value

None

Remarks

Note that even when the contents of the GsF_LIGHT structure are written over, the setting will be not reflected in libgs unless this function is invoked.

See also:

GsSetFogParam

Sets the fog parameter.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSetFogParam(*fogparam)
GsFOGPARAM *fogparam;
```

Arguments

fogparam Pointer to a fog parameter structure

Explanation

GsSetFogParam sets the fog parameter. Fog is valid only in lighting mode 1 and 3. (Light mode 3 is not supported.)

Return value

None

Remarks

See also:

GsSetLightMatrix

Sets a light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSetLightMatrix(*mp)
MATRIX *mp;
```

Arguments

mp Pointer to matrix

Explanation

This function multiplies the local screen light matrix *mp* by the matrices for the three light vectors, and places the results in *libgte*.

When using *libgte* during application execution of light source calculations, *GsSetLightMatrix()* must be set in advance.

Depending on the type of model data, some *GsSortObject...()* will calculate the light source during execution. In this case, also, you must use *GsSetLightMatrix()* to set a light matrix in advance.

Matrices to be set as *GsSetLightMatrix()* arguments are usually local screen matrices.

Return value

None

Remarks

See also:

GsSetLightMatrix2

Sets a light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	5/22/97

Syntax

```
void GsSetLightMatrix2(*mp)
MATRIX *mp;
```

Arguments

mp Pointer to matrix

Explanation

The three light source vector matrices and the local screen light matrix *mp*, passed as a parameter, are multiplied and placed in libgte.

This matrix must be set in advance when performing light-source calculations using libgte. GsSortObject... may perform light-source calculations during execution, depending on the type of modeling data handled. You must use GsSetLightMatrix2() to set the light matrix in these cases as well.

Generally, the matrix set as a parameter in GsSetLightMatrix2() will be a local world matrix.

Return value

None

Remarks

The difference between GsSetLightMatrix() and this function is whether the GTE rotation matrix and the parameter *mp* are destroyed or not. GsSetLightMatrix2() destroys these values, however, GsSetLightMatrix2() is faster than GsSetLightMatrix().

You must call GsSetLightMatrix() before GsSetLsMatrix().

See also:

GsSetLightMode

Sets light source mode.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSetLightMode(mode)
int mode;
```

Arguments

mode Light source mode value (0-3)
 0: Normal lighting
 1: Normal lighting with fog ON
 2: Material lighting (not currently supported)
 3: Material lighting with fog ON (not currently supported)

Explanation

This function sets the default light source mode. The method of light source calculation can be also set using status bits for each object. The setting of the status bit overrides the default setting.

Return value

None

Remarks

See also:

GsSetLsMatrix

Sets a local screen matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSetLsMatrix(*mp)
MATRIX *mp;
```

Arguments

mp Pointer to local screen matrix to be set

Explanation

This function sets a GTE local screen matrix. When you use GsSetLsMatrix for LIBGTE perspective transform processing, you must first set a local screen matrix in LIBGTE.

For GsSortObject---() calls to perform perspective transformations and use them in LIBGTE, you must first execute GsSetLsMatrix.

Return value

None

Remarks

See also:

GsSetOffset

Sets an offset.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	5/22/97

Syntax

```
void GsSetOffset(offx, offy)
long offx;
long offy;
```

Arguments

offx Drawing offset X
offy Drawing offset Y

Explanation

Specifies a drawing offset. This function is different from GsSetDrawBuffOffset() in that it sets an offset provided as an argument while GsSetDrawBuffOffset() sets a value for the global variable, POSITION. The offset to be provided as an argument is a relative offset inside the double buffer. In other words, the double buffer base offset is added to the offset provided by the argument.

Using the GsOFSGPU or GsOFSGTE macro for the third argument of GsInitGraph() determines whether the libgte or libgpu offset should be set.

Return value

None

Remarks

This function does not execute correctly if GPU drawing is in progress. Use ResetGraph(1) to terminate any current drawing process of DrawSync() to wait until the process is completed.

See also:

GsSetOrign

The offset is valid if the screen is switched.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	5/22/97

Syntax

```
void GsSetOrign (
long x ,
long y
)
```

Arguments

x Drawing offset X
y Drawing offset Y

Explanation

Specifies a drawing offset. The function is different to GsSetOffset() in that the offset value set in GsSetOffset() is temporary and becomes invalid when GsSwapDispBuff() and GsSetDrawBuffOffset() are called, while the offset value set in GsSetOrign() is valid until the GsSetOrign() is called again.

The offset to be provided as an argument is a relative offset inside the double buffer. In other words, the double buffer base offset is added to the offset provided by the argument.

The location is the same as the GsSetClip2D() in clipping.

Note: Using the GsOFSGPU or GsOFSGTE macro for the third argument of GsInitGraph() determines whether the libgte or libgpu offset should be set.

In fact they are set by the external variable POSITIONs offx, offy.

Return value

None

Remarks

This function does not execute correctly when GPU drawing is in progress, so it is necessary to call this function after terminating drawing using ResetGraph(1).

See also:

GsSetProjection

Sets the projection plane position.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSetProjection(h)
unsigned long h;
```

Arguments

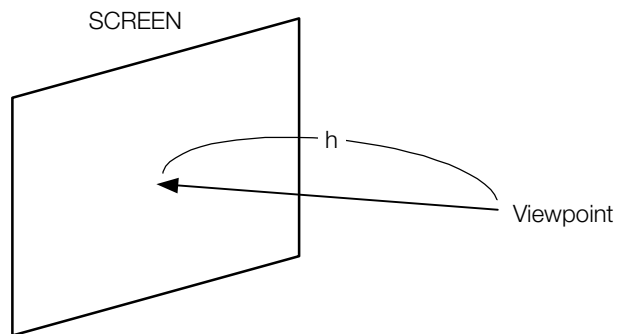
h Distance (projection) between the viewpoint and projection plane
Default: 1000

Explanation

This function adjusts the drawing angle.

A projection is the distance from the viewpoint to the projection plane.

Figure 9-1: Projection



The size of the projection plane is specified by (xres, yres) in the GsInitGraph() function. The size of the projection plane is constant with respect to the resolution, so the drawing angle is reduced as projection is increased, and the drawing angle is increased as projection is decreased.

Depending on the resolution, the aspect ratio may not be 1 to 1. In this case, set the X coordinate scale to 1/2 and adjust the aspect ratio.

Table 9-7: Resolution and Aspect Ration

Resolution	Aspect ration
640x480	1:1
640x240	2:1
320x240	1:1

Return value

None

Remarks

See also:

GsSetRefView2

Sets viewpoint position.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
int GsSetRefView2(*pv)
GsRVIEW2 *pv;
```

Arguments

pv Pointer to viewpoint position information

Explanation

Calculates WSMATRIX using viewpoint information. *pv* is a pointer to a GsRVIEW2 structure.

Since WSMATRIX will not change unless the viewpoint is moved, it need not be called for each frame. However, if the viewpoint is moved, WSMATRIX must be called for each frame in order for changes to be updated.

Call GsSetRefView2() for each frame if the GsRVIEW2 member *super* is set to anything other than WORLD; even if the other parameters are not changed, if the parameters of the superior coordinate system are changed, the viewpoint will have moved.

Return value

Upon success, the function returns 0. Upon failure, it returns 1.

Remarks

See also:

GsSetRefView2L

Sets viewpoint (High Precision Version).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.5	7/31/96

Syntax

```
int GsSetRefView2L(*pv)
GsRVIEW2 *pv;
```

Arguments

pv Pointer to viewpoint location information (view/reference point type)

Explanation

This function calculates WSMATRIX using the viewpoint information. The parameter is structure GsRVIEW2. It is not necessary to call this function for every frame if the viewpoint is not changed since WSMATRIX is not changed. However, if the viewpoint changes, this must be called for every frame to update.

When setting a GsRVIEW2 member, "super" to values other than WORLD, GsSetRefView2L() must be called for every frame since the viewpoint moves as a result of parent coordinate parameter change even when other parameters are not changed. The difference between GsSetRefView2L() and GsSetRefView2() is precision level. In GsSetRefView2L(), viewpoint wobbling caused by insufficient precision is improved compared with GsSetRefView2().

The execution time of GsSetRefView2L(), however, is doubled.

Return value

0 for successful viewpoint set

1 for error.

Remarks

See also:

GsSetView2

Sets viewpoint.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	3/25/98

Syntax

```
int GsSetView2(*pv)
GsVIEW2 *pv;
```

Arguments

pv Pointer to viewpoint position data (matrix form)

Explanation

Sets the WS matrix directly.

If you use GsSetRefView2() to determine the WS matrix from the viewpoint and the focal point, insufficient precision may cause errors when you move the viewpoint; it is more effective to use GsSetView2().

When the GsVIEW2 "super" member is set to anything besides WORLD, even if the other parameters remain unchanged, the viewpoint will move if the parent coordinate system parameters are changed. In such cases, you must call GsSetView2() for each frame.

If GsIDMATRIX2 is used as the base matrix, then the aspect ratio of the screen will be adjusted automatically.

Return value

Settings successful: 0; 1 if unsuccessful.

Remarks

See also:

GsSetWorkBase

Sets address for storing drawing commands.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSetWorkBase(*base_addr)
PACKET *base_addr;
```

Arguments

base_addr Pointer to an address storing drawing commands

Explanation

This function sets the memory address for storing drawing primitives generated by functions like GsSortObject...(), GsSortSprite(), and GsSortBg().

Primitives must be stored at the starting address of a packet area reserved by the user at the beginning of processing for each frame.

Return value

None

Remarks

See also:

GsSortBg, GsSortFastBg

Registers BG in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```

void GsSortBg(*bg, *otp, pri)
GsBG *bg;
GsOT *otp;
unsigned short pri;

void GsSortFastBg(*bg, *otp, pri)
GsBG *bg;
GsOT *otp;
unsigned short pri;

```

Arguments

```

bg      Pointer to BG
otp     Pointer to OT
pri     Position in OT

```

Explanation

This function assigns BG indicated by *bg* to the ordering table indicated by *otp*. *pri* refers to the priority of the Sprite in the ordering table. The highest priority is zero, with the lowest priority depending on the size of the ordering table. Values beyond the ordering table size are clipped to the available maximum value.

Turning off extension and rotation functions in the bg attributes gives higher-speed processing.

In GsSortFastBg(), not using enlargement, rotation, and reduction functions results in higher-speed processing. The Sprite structure members values *mx*, *my*, *scalex*, *scaley*, and *rotate* are ignored.

Return value

None

Remarks

See also:

GsSortBoxFill

Registers rectangle in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSortBoxFill(*bp, *ot, pri)
GsBOXF *bp;
GsOT *ot;
unsigned short pri;
```

Arguments

bp Pointer to GsBOXF
ot Pointer to OT
pri Position in OT

Explanation

This function assigns a rectangle indicated by *bp* to the ordering table indicated by *ot*.

Return value

None

Remarks

See also:

GsSortClear

Registers a screen clear command in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSortObject(r, g, b, *otp)
unsigned char r, g, b;
GsOT *otp;
```

Arguments

r, g, b Background color RGB values
otp Pointer to OT

Explanation

Sets a screen clear command at the start of the OT indicated by *otp*.

Return value

None

Remarks

This function only registers a screen clear command in the OT; actual clearing will not be executed until the GsDrawOt() function is used to start drawing.

This function is called after GsSwapDispBuff().

See also:

GsSortFastSpriteB

Registers a sprite to the OT (for arcade).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
void GsSortFastSpriteB (
GsSPRITE *sp,
GsOT *otp,
unsigned short pri,
unsigned short flip
)
```

Arguments

sp Pointer to the sprite
otp Pointer to the OT
pri Position in the OT
flip Flag indicating whether FLIP will take place
0:FLIP is not performed
1:x direction inversion
2:y direction inversion
3:x and y direction inversion

Explanation

This function allocates the sprite displayed by *sp* to the ordering table displayed by *otp*. The sprite display position parameters are all provided by *sp* members. *pri* is the priority level of the sprite in the ordering table. The highest priority level is 0 and the lowest depends on the size of the ordering table. A numerical value greater than the size of the ordering table is clipped to the maximum value available.

flip is the flag which determines whether both a horizontal direction inversion and a vertical direction inversion will be performed. In `GsSortFastSpriteB()`, not using enlargement, rotation and reduction functions results in higher-speed processing. The sprite structure members values *mx*, *my*, *scalex*, *scaley* and *rotate* are ignored.

Return value

None

Remarks

This function is for arcade use.

See also:

GsSortFixBg16

Registers high-speed BG in the OT

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```

void GsSortFixBg16(*bg, *work, *otp, pri)
GsBG *bg;
unsigned long *work;
GsOT *otp;
unsigned short pri;

```

Arguments

<i>bg</i>	Pointer to GsBG
<i>work</i>	Pointer to work area (primitive area)
<i>otp</i>	Pointer to OT
<i>pri</i>	Position in OT

Explanation

This function performs high-speed BG registration processing. It is less CPU-intensive than GsSortFastBg(), with the following restrictions.

- BG rotation/enlargement/reduction is not possible
- Fixed cell size: 16 for GsSortFixBg16, 32 for GsSortFixBg32
- Texture patter color mode is only 4-bit/8-bit
- Map size is optional
- Scroll is possible (in 1-pixel units)
- Only full-screen

This function uses the work area to store drawing primitives. The work area uses an unsigned long array; this must be initialized beforehand by GsInitFixBg16() or GsInitFixBg32(). This function does not use the packet area (an area set by GsSetWorkBase()).

Return value

None

Remarks

See also:

GsSortFixBg32

Registers high-speed BG in the OT

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
void GsSortFixBg32(*bg, *work, *otp, pri)
GsBG *bg;
unsigned long *work;
GsOT *otp;
unsigned short pri;
```

Arguments

bg Pointer to GsBG
work Pointer to work area (primitive area)
otp Pointer to OT
pri Position in OT

Explanation

This function performs high-speed BG registration processing. It is less CPU-intensive than GsSortFastBg(), with the following restrictions.

- BG rotation/enlargement/reduction is not possible
- Fixed cell size: 16 for GsSortFixBg16, 32 for GsSortFixBg32
- Texture patten color mode is only 4-bit/8-bit
- Map size is optional
- Scroll is possible (in 1-pixel units)
- Only full-screen

This function uses the work area to store drawing primitives. The work area uses an unsigned long array; this must be initialized beforehand by GsInitFixBg16() or GsInitFixBg32(). This function does not use the packet area (an area set by GsSetWorkBase()).

Return value

Remarks

See also:

GsSortGLine

Registers straight line in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
void GsSortGLine(*lp, *ot, pri)
GsLINE *lp;
GsOT *ot;
unsigned short pri;
```

Arguments

lp Pointer to GsLINE/GsGLINE
ot Pointer to OT
pri Position in OT

Explanation

This function assigns the straight line indicated by *lp* to the ordering table indicated by *ot*.

The GsSortLine() function registers single-color straight lines in OT, and the GsSortGLine() function graded straight lines in OT.

Return value

None

Remarks

See also:

GsSortLine

Registers straight line in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	7/31/96

Syntax

```
void GsSortLine(*lp, *ot, pri)
GsLINE *lp;
GsOT *ot;
unsigned short pri;
```

Arguments

lp Pointer to GsLINE/GsGLINE
ot Pointer to OT
pri Position in OT

Explanation

This function assigns the straight line indicated by *lp* to the ordering table indicated by *ot*.

The GsSortLine() function registers single-color straight lines in OT, and the GsSortGLine() function graded straight lines in OT.

Return value

None

Remarks

See also:

GsSortObject3

Assigns an object to the ordering table (for use with GsDOBJ3).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	5/22/97

Syntax

```
void GsSortObject3(*objp, *otp, shift)
GsDOBJ3 *objp;
GsOT *otp;
int shift;
```

Arguments

objp Pointer to an object
otp Pointer to OT
shift Specifies how many bits the Z value must be shifted to the right when assigning an object to the OT

Explanation

Performs perspective transformation and light source calculation for a three-dimensional object handled by GsDOBJ3, and creates a drawing command within the PMD format packet memory. Performs Z-sort of the drawing commands generated immediately afterwards and assigns them to the OT indicated by *otp*.

The accuracy of Z may be adjusted with the value of *shift*. The maximum size of the ordering table (resolution) is 14 bits, but if this value is set to 12 bits, for example, the shift value must be set at 2 (=14-12), so that it will not be larger than the ordering table area.

Return value

None

Remarks

See also:

GsSortObject4

Assigns an object to the ordering table (for use with GsDOBJ2).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	5/22/97

Syntax

```
void GsSortObject4(*objp, *otp, shift, *scratch)
GsDOBJ2 *objp;
GsOT *otp;
int shift;
unsigned long *scratch;
```

Arguments

objp Pointer to an object
otp Pointer to OT
shift Specifies how many bits the Z value must be shifted to the right when assigning an object to the OT
scratch Pointer to the address of the scratch pad

Explanation

Performs perspective transformation and light source calculation for a three-dimensional object to be handled by GsDOBJ2, and creates a drawing command within the packet area specified by GsSetWorkBase(). Performs Z-sort of the drawing commands generated immediately afterwards and assigns them to the OT indicated by *otp*.

The accuracy of Z may be adjusted with the value of *shift*. The maximum size of the ordering table (resolution) is 14 bits. If this value is set to 12 bits, for example, the shift value must be set at 2 (=14-12), so that it will not be larger than the ordering table area.

scratch is the specified scratchpad address used as work when automatic division is being performed. The scratchpad runs for 256 words from 0x1f800000 in cache memory.

To use the GsOBJ2 member attribute to enable division, perform an OR operation on the macros GsDIV1 through GsDIV5 (defined in libgs.h). For GsDIV1, a single polygon will be divided into four segments of 2 x 2. For GsDIV5, a single polygon will be divided into 1,024 segments of 32 x 32.

The insignificant addresses from the addresses specified by *scratch* are used as work. The usage volume is as follows:

Type	Scrach area usage volume (Unit: Byte)
Triangle polygon	96+88*N
Quadrilateral polygon	120+140*N

N is the division parameters GsDIV1 N=1, GsDIV2 N=2, GsDIV3 N=3 ...GsDIV5 N=5.

Return value

None

Remarks

See also:

GsSortObject4J

Allocation of object to ordering table (Function TABLE version).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	5/22/97

Syntax

```
void GsSortObject4J(*objp, *otp, shift, *scratch)
GsDOBJ2 *objp;
GsOT *otp;
int shift;
unsigned long *scratch;
```

Arguments

objp Pointer to an object
otp Pointer to OT
shift Specifies how many bits the Z value must be shifted to the right when assigning an object to the OT
scratch Pointer to the address of the scratch pad

Explanation

When all the insignificant functions have been registered, this function's features are equal to those of GsSortObject4 (). In addition, for the programmer to be able to control the functions registered to the table, he can increase the code efficiency by taking care not to call the unnecessary insignificant functions.

GsSortObject4 () is used for prototyping, but ultimately memory can be saved if you switch to GsSortObject4J ().

A maximum of 40kbytes can be saved.

If 'dmy' is written first of all at the head of the function name for slots which do not register GsFCALL4, even if by chance that insignificant function is called, it will not cause a BUS ERROR and since the function name used when the function was first called is printed out, delete the 'dmy' and register.

GsFCALL4 GsSortObject4J () reference function table.

Return value

None

Remarks

See also:

GsSortObject5

Assigns an object to the ordering table (for use with GsDOBJ5).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	5/22/97

Syntax

```
void GsSortObject5(*objp, *otp, shift, *scratch)
GsDOBJ2 *objp;
GsOT *otp;
int shift;
unsigned long *scratch;
```

Arguments

objp Pointer to an object
otp Pointer to OT
shift Specifies how many bits the Z value must be shifted to the right when assigning an object to the OT
scratch Pointer to the address of the scratch pad

Explanation

Performs transparency transformation and light source calculation for a three-dimensional object to be handled by GsDOBJ5, and creates in the preset packet area drawing commands that do not divide, and in the packet area specified by GsSetWorkBase() those drawing commands that do divide. Performs Z-sort of the drawing commands generated immediately afterwards and assigns them to the OT indicated by *otp*.

The accuracy of Z may be adjusted using the *shift* value. The maximum size of the ordering table (resolution) is 14 bits. If this value is set to 12 bits, for example, the shift value must be set at 2 (=14-12), so that it will not be larger than the ordering table area.

scratch is used as work when automatic division is being performed. To use attribute to enable division, perform an OR operation on the macros GsDIV1-GsDIV5 of libgs.h. For GsDIV1, a single polygon will be divided into four segments of 2 x 2. For GsDIV5, a single polygon will be divided into 1,024 segments of 32 x 32.

scratch is the specified scratchpad address used as work when automatic division is being performed. The scratchpad runs for 256 words from 0x1f800000 in cache memory.

To use the GsOBJ2 member attribute to enable division, perform an OR operation on the macros GsDIV1 through GsDIV5 (defined in libgs.h). For GsDIV1, a single polygon will be divided into four segments of 2 x 2. For GsDIV5, a single polygon will be divided into 1,024 segments of 32 x 32.

The insignificant addresses from the addresses specified by *scratch* are used as work. The usage volume is as follows:

Type	Scrach area usage volume (Unit: Byte)
Triangle polygon	96+88*N
Quadrilateral polygon	120+140*N

N is the division parameters GsDIV1 N=1, GsDIV2 N=2, GsDIV3 N=3 ...GsDIV5 N=5.

Return value

None

Remarks

See also:

GsSortObject5J

Assigns an object to the ordering table.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	5/22/97

Syntax

```
void GsSortObject5J(*objp, *otp, shift, *scratch)
GsDOBJ2 *objp;
GsOT *otp;
int shift;
unsigned long *scratch;
```

Arguments

objp Pointer to an object
otp Pointer to OT
shift Specifies how many bits the Z value must be shifted to the right when assigning an object to the OT
scratch Pointer to the address of the scratch pad

Explanation

When all the insignificant functions have been registered, this function's features are equal to those of GsSortObject5 (). In addition, for the programmer to be able to control the functions registered to the table, he can increase the code efficiency by taking care not to call the unnecessary insignificant functions.

GsSortObject5() is used for prototyping, but ultimately memory can be saved if you switch to GsSortObject5J().

A maximum of 40kbytes can be saved.

If 'dmy' is written first of all at the head of the function name for slots which do not register GsFCALL5, even if by chance that insignificant function is called, it will not cause a BUS ERROR and since the function name used when the function was first called is printed out, delete the 'dmy' and register.

GsFCALL5 GsSortObject5J () reference function table

Return value

None

Remarks

See also:

GsSortOt

Assigns an OT to another OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.3	6/22/98

Syntax

```
GsOT *GsSortOt(*ot_src, *ot_dest)
```

```
GsOT *ot_src;
```

```
GsOT *ot_dest;
```

Arguments

ot_src Pointer to source OT

ot_dest Pointer to destination OT

Explanation

This function assigns the OT given by *ot_src* to the OTZ location within *ot_dest*.

The OTZ value used at this time is calculated as follows:

$$\text{OTZ} = \text{ot_src} \rightarrow \text{point} - \text{ot_dest} \rightarrow \text{offset}$$

The integrated OT is inserted into *ot_dest*.

Return value

Pointer to the integrated OT.

Remarks

See also:

GsSortPoly

Registers polygon drawing primitive in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSortPoly(*prim, *ot, pri)
void *prim;
GsOt *ot;
unsigned short pri;
```

Arguments

prim Pointer to drawing primitive
ot Pointer to OT
pri Location in OT

Explanation

This function assigns the drawing primitive given by *prim* to the ordering table given by *ot*.

Out of the primitives defined by libgpu, the drawing primitive refers only to (POLY_....).

libgs requires no double buffering, since the contents of the primitive structure are copied in the packet generation area. Drawing coordinate values match the drawing coordinate system handled by libgs.

Return value

None

Remarks

See also:

GsSortSprite, GsSortFastSprite, GsSortFlipSprite

Registers a Sprite in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

```
void GsSortSprite(*sp, *otp, pri)
GsSPRITE *sp;
GsOT *otp;
unsigned short pri;

void GsSortFastSprite(*sp, *otp, pri)
GsSPRITE *sp;
GsOT *otp;
unsigned short pri;

void GsSortFlipSprite(*sp, *otp, pri)
GsSPRITE *sp;
GsOT *otp;
unsigned short pri;
```

Arguments

sp Pointer to a Sprite
otp Pointer to OT
pri Position in OT

Explanation

This function assigns the Sprite given by *sp* to the ordering table provided by *otp*.

All the parameters including Sprite indication locations are given by the *sp* members.

pri refers to the priority of the Sprite in the ordering table. The highest priority value is zero, with the lowest value depending on the size of the ordering table. Values beyond the size of the ordering table are clipped to the maximum ordering table value.

The GsSortFastSprite function provides high-speed processing, though enlargement, rotation, and reduction cannot be used. The Sprite structure members *nx*, *my*, *scalex*, *scaley*, and *rotate* are ignored.

GsSortFlipSprite() does not use the enlargement / rotation / reduction functions, and only supports flipping.

Return value

None

Remarks

See also:

GsSortSpriteB

Registers a sprite to the OT (for arcade).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
void GsSortSpriteB (
GsSPRITE *sp,
GsOT *otp,
unsigned short pri,
unsigned short flip
)
```

Arguments

sp Pointer to a sprite
otp Pointer to the OT
pri Position in the OT
flip Flag which determines whether FLIP is performed
 0:FLIP is not performed
 1:x direction inversion
 2:y direction inversion
 3:x and y direction inversion

Explanation

This function assigns the sprite given by *sp* to the ordering table provided by *otp*. All the parameters including sprite indication locations are given by the *sp* members. *pri* refers to the priority of the sprite in the ordering table. The highest priority value is zero and the lowest depends on the size of the ordering table. Values greater than the size of the ordering table are clipped to the maximum ordering table value. *flip* is the flag which determines whether both a horizontal inversion and a vertical inversion will be performed.

Return value

None

Remarks

This function is for arcade use.

See also:

GsSwapDispBuffer

Swaps double buffers.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	7/31/96

Syntax

void GsSwapDispBuffer(void)

Arguments

None

Explanation

This function exchanges the display buffer with the drawing buffer according to double buffer data set by GsSetDefDispBuffer(). Normally, swapping is done immediately after beginning vertical blanking.

This function performs the following:

- Sets display starting address
- Cancels blanking
- Sets double buffer index
- Switches two-dimensional clipping
- Sets libgte or libgpu offset
- Sets libgs offset

Note: The double buffer is implemented by offset. Using the GsOFSGPU or GsOFSGTE macro for the third argument of GsInitGraph() determines whether the libgte or libgpu offset should be set.

Return value

None

Remarks

This function does not execute correctly when GPU drawing is in progress, so it is necessary to call this function after terminating drawing using ResetGraph (1).

See also:

GsTMDdiv...

Low-level functions for GsSortObject4J() (performs fixed division).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	6/22/98

Syntax

```
PACKET *GsTMDdiv... (  
TMD_P_... *op,  
VERT      *vp,  
VERT      *np,  
PACKET     *pk,  
int        n,  
int        shift,  
GsOT       *ot,  
DIVPOLYGON... *divp  
)  
  
PACKET *GsTMDdivN... (  
TMD_P_... *op  
VERT      *vp,  
PACKET     *pk,  
int        n,  
int        shift,  
GsOT       *ot,  
DIVPOLYGON... *divp  
)
```

Arguments

<i>op</i>	Pointer to starting address of TMD data primitives
<i>vp</i>	Pointer to starting address of TMD data vertices TMD
<i>np</i>	Pointer to starting address of TMD data normals
<i>pk</i>	Pointer to top address of GPU packet buffer
<i>n</i>	Number of primitives
<i>shift</i>	OT shift bit
<i>ot</i>	Pointer to GsOT
<i>divp</i>	Pointer to DIVPOLYGON3 or DIVPOLYGON4

Explanation

These are low-level functions for GsSortObject4J().

These functions perform fixed division based on the number of divisions (ndiv) that is provided.

To use these functions, they must be registered in GsFCALL4 as low-level functions.

These functions perform polygon division based on the value of ndiv, coordinate & perspective transformation, backface clipping, and light source calculation for *n* primitives, complete the GPU packet in the buffer, and link it into the OT.

ndiv=1: 2x2 division

ndiv=2: 4x4 division

ndiv=3: 8x8 division

ndiv=4: 16x16 division

ndiv=5: 32x32 division

For function types which do not operate on normals within the data (e.g. GsTMDdivN...), light source calculations are not performed so fewer parameters are passed compared to those function types which operate on normals (e.g. GsTMDdiv...),

Low-level functions in libgs that support fixed division are shown below.

GsTMDdiv...() [have normals]

Low-level function name	First arg (op) type	Description
GsTMDdivF3L	TMD_P_F3	Flat triangle (light source calculation)
GsTMDdivF3LB	TMD_P_F3	Flat triangle (light source calculation + 2face)
GsTMDdivF3LFG	TMD_P_F3	Flat triangle (light source calculation +FOG)
GsTMDdivF3LFGB	TMD_P_F3	Flat triangle (light source calculation +FOG + 2face)
GsTMDdivF3NL	TMD_P_F3	Flat triangle
GsTMDdivF3NLB	TMD_P_F3	Flat triangle (2face)
GsTMDdivF4L	TMD_P_F4	Flat quadrilateral (light source calculation)
GsTMDdivF4LB	TMD_P_F4	Flat quadrilateral (light source calculation + 2face)
GsTMDdivF4LFG	TMD_P_F4	Flat quadrilateral (light source calculation +FOG)
GsTMDdivF4LFGB	TMD_P_F4	Flat quadrilateral (light source calculation +FOG + 2face)
GsTMDdivF4NL	TMD_P_F4	Flag quadrilateral
GsTMDdivF4NLB	TMD_P_F4	Flag quadrilateral (2face)
GsTMDdivG3L	TMD_P_G3	Gouraud triangle (light source calculation)
GsTMDdivG3LB	TMD_P_G3	Gouraud triangle (light source calculation + 2face)
GsTMDdivG3LFG	TMD_P_G3	Gouraud triangle (light source calculation +FOG)
GsTMDdivG3LFGB	TMD_P_G3	Gouraud triangle (light source calculation +FOG + 2face)
GsTMDdivG3NL	TMD_P_G3	Gouraud triangle
GsTMDdivG3NLB	TMD_P_G3	Gouraud triangle (2face)
GsTMDdivG4L	TMD_P_G4	Gouraud quadrilateral (light source calculation)
GsTMDdivG4LB	TMD_P_G4	Gouraud quadrilateral (light source calculation + 2face)
GsTMDdivG4LFG	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG)
GsTMDdivG4LFGB	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG + 2face)
GsTMDdivG4NL	TMD_P_G4	Gouraud quadrilateral
GsTMDdivG4NLB	TMD_P_G4	Gouraud quadrilateral (2face)
GsTMDdivTF3L	TMD_P_TF3	Textured flat triangle (light source calculation)
GsTMDdivTF3LB	TMD_P_TF3	Textured flat triangle (light source calculation + 2face)

Low-level function name	First arg (op) type	Description
GsTMDdivTF3LFG	TMD_P_TF3	Textured flat triangle (light source calculation +FOG)
GsTMDdivTF3LFGB	TMD_P_TF3	Textured flat triangle (light source calculation +FOG + 2face)
GsTMDdivTF3NL	TMD_P_TF3	Textured flat triangle
GsTMDdivTF3NLB	TMD_P_TF3	Textured flat triangle (2face)
GsTMDdivTF4L	TMD_P_TF4	Textured flat quadrilateral (light source calculation)
GsTMDdivTF4LB	TMD_P_TF4	Textured flat quadrilateral (light source calculation + 2face)
GsTMDdivTF4LM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +mip-map)
GsTMDdivTF4LFG	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG)
GsTMDdivTF4LFGB	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG + 2face)
GsTMDdivTF4LFGM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG+mip-map)
GsTMDdivTF4NL	TMD_P_TF4	Textured flat quadrilateral
GsTMDdivTF4NLB	TMD_P_TF4	Textured flat quadrilateral (2face)
GsTMDdivTF4NLM	TMD_P_TF4	Textured flat quadrilateral (mip-map)
GsTMDdivTG3L	TMD_P_TG3	Textured Gouraud triangle (light source calculation)
GsTMDdivTG3LB	TMD_P_TG3	Textured Gouraud triangle (light source calculation + 2face)
GsTMDdivTG3LFG	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG)
GsTMDdivTG3LFGB	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG + 2face)
GsTMDdivTG3NL	TMD_P_TG3	Textured Gouraud triangle
GsTMDdivTG3NLB	TMD_P_TG3	Textured Gouraud triangle (2face)
GsTMDdivTG4L	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation)
GsTMDdivTG4LB	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation + 2face)
GsTMDdivTG4LM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +mip-map)
GsTMDdivTG4LFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG)
GsTMDdivTG4LFGB	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG + 2face)
GsTMDdivTG4LFGM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG + mip-map)
GsTMDdivTG4NL	TMD_P_TG4	Textured Gouraud quadrilateral

Low-level function name	First arg (op) type	Description
GsTMDdivTG4NLB	TMD_P_TG4	Textured Gouraud quadrilateral (2face)
GsTMDdivTG4NLM	TMD_P_TG4	Textured Gouraud quadrilateral (mip-map)
GsTMDdivN...() [no normals]		
Low-level function name	First arg (op) type	Description
GsTMDdivNF3	TMD_P_NF3	Flat triangle
GsTMDdivNF3B	TMD_P_NF3	Flat triangle (2face)
GsTMDdivNF4	TMD_P_NF4	Flat quadrilateral
GsTMDdivNF4B	TMD_P_NF4	Flat quadrilateral (2face)
GsTMDdivNG3	TMD_P_NG3	Gouraud triangle
GsTMDdivNG3B	TMD_P_NG3	Gouraud triangle (2face)
GsTMDdivNG4	TMD_P_NG4	Gouraud quadrilateral
GsTMDdivNG4B	TMD_P_NG4	Gouraud quadrilateral (2face)
GsTMDdivTNF3	TMD_P_TNF3	Textured flat triangle
GsTMDdivTNF3B	TMD_P_TNF3	Textured flat triangle (2face)
GsTMDdivTNF4	TMD_P_TNF4	Textured flat quadrilateral
GsTMDdivTNF4B	TMD_P_TNF4	Textured flat quadrilateral (2face)
GsTMDdivTNF4M	TMD_P_TNF4	Textured flat quadrilateral (mip-map)
GsTMDdivTNG3	TMD_P_TNG3	Textured Gouraud triangle
GsTMDdivTNG3B	TMD_P_TNG3	Textured Gouraud triangle (2face)
GsTMDdivTNG4	TMD_P_TNG4	Textured Gouraud quadrilateral
GsTMDdivTNG4B	TMD_P_TNG4	Textured Gouraud quadrilateral (2face)
GsTMDdivTNG4M	TMD_P_TNG4	Textured Gouraud quadrilateral (mip-map)

Return value

Starting address of unused packet area.

Remarks

ndiv (number of divisions) is specified in the attribute.

pih, piv: display screen resolution (using clip) is the resolution already set in libgs.

For high speed operation, libgte contains tuned assembly-level low-level functions.

See also: GsSortObject4J().

GsTMDfast...

Low-level functions for GsSortObject4J().

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	6/22/98

Syntax

```
PACKET *GsTMDfast... (  
TMD_P_... *op,  
VERT *vp,  
VERT *np,  
PACKET *pk,  
int n,  
int shift,  
GsOT *ot,  
unsigned long *scratch  
)
```

```
PACKET *GsTMDfastN... (  
TMD_P_... *op;  
VERT *vp;  
PACKET *pk;  
int n;  
int shift;  
GsOT *ot;  
unsigned long *scratch;  
)
```

Arguments

<i>op</i>	Pointer to starting address of TMD data primitives
<i>vp</i>	Pointer to starting address of TMD data vertices TMD
<i>np</i>	Pointer to starting address of TMD data normals
<i>pk</i>	Pointer to top address of GPU packet buffer
<i>n</i>	Number of primitives
<i>shift</i>	OT shift bit
<i>ot</i>	Pointer to GsOT
<i>scratch</i>	Pointer to starting address of unused scratch pad

Explanation

These are low-level functions for GsSortObject4J().

To use these functions, they must be registered in GsFCALL4 as low-level functions.

These functions perform coordinate & perspective transformation, backface clipping, and light source calculation for *n* primitives, complete the GPU packet in the buffer, and link it into the OT.

For low-level functions that provide material attenuation, the lighting mode must be “material ON.” This attenuates the brightness during light source calculation based on the parameter which is provided in the attribute of GsDOBJ2.

For low-level functions that FLIP, the direction of the normal clip is reversed.

For function types which do not operate on normals within the data (e.g. GsTMDfastN...), light source calculations are not performed so fewer parameters are passed compared to those function types which operate on normals (e.g. GsTMDfast...),

Low-level functions supported in libgs are shown below.

GsTMDfast...() [have normals]

Low-level function name	First arg (op) type	Description
GsTMDfastF3L	TMD_P_F3	Flat triangle (light source calculation)
GsTMDfastF3LB	TMD_P_F3	Flat triangle (light source calculation + 2face)
GsTMDfastF3LFG	TMD_P_F3	Flat triangle (light source calculation +FOG)
GsTMDfastF3LFGB	TMD_P_F3	Flat triangle (light source calculation +FOG + 2face)
GsTMDfastF3NL	TMD_P_F3	Flat triangle
GsTMDfastF3NLB	TMD_P_F3	Flat triangle (2face)
GsTMDfastF4L	TMD_P_F4	Flat quadrilateral (light source calculation)
GsTMDfastF4LB	TMD_P_F4	Flat quadrilateral (light source calculation + 2face)
GsTMDfastF4LFG	TMD_P_F4	Flat quadrilateral (light source calculation +FOG)
GsTMDfastF4LFGB	TMD_P_F4	Flat quadrilateral (light source calculation +FOG + 2face)
GsTMDfastF4NL	TMD_P_F4	Flag quadrilateral
GsTMDfastF4NLB	TMD_P_F4	Flag quadrilateral (2face)
GsTMDfastG3L	TMD_P_G3	Gouraud triangle (light source calculation)
GsTMDfastG3LB	TMD_P_G3	Gouraud triangle (light source calculation + 2face)
GsTMDfastG3LFG	TMD_P_G3	Gouraud triangle (light source calculation +FOG)
GsTMDfastG3LFGB	TMD_P_G3	Gouraud triangle (light source calculation +FOG + 2face)
GsTMDfastG3NL	TMD_P_G3	Gouraud triangle
GsTMDfastG3NLB	TMD_P_G3	Gouraud triangle (2face)
GsTMDfastG4L	TMD_P_G4	Gouraud quadrilateral (light source calculation)
GsTMDfastG4LB	TMD_P_G4	Gouraud quadrilateral (light source calculation + 2face)
GsTMDfastG4LFG	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG)
GsTMDfastG4LFGB	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG + 2face)
GsTMDfastG4NL	TMD_P_G4	Gouraud quadrilateral
GsTMDfastG4NLB	TMD_P_G4	Gouraud quadrilateral (2face)
GsTMDfastTF3L	TMD_P_TF3	Textured flat triangle (light source calculation)
GsTMDfastTF3LB	TMD_P_TF3	Textured flat triangle (light source calculation + 2face)
GsTMDfastTF3LFG	TMD_P_TF3	Textured flat triangle (light source calculation +FOG)

Low-level function name	First arg (op) type	Description
GsTMDfastTF3LFGB	TMD_P_TF3	Textured flat triangle (light source calculation +FOG + 2face)
GsTMDfastTF3NL	TMD_P_TF3	Textured flat triangle
GsTMDfastTF3NLB	TMD_P_TF3	Textured flat triangle (2face)
GsTMDfastTF4L	TMD_P_TF4	Textured flat quadrilateral (light source calculation)
GsTMDfastTF4LB	TMD_P_TF4	Textured flat quadrilateral (light source calculation + 2face)
GsTMDfastTF4LM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +mip-map)
GsTMDfastTF4LFG	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG)
GsTMDfastTF4LFGB	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG + 2face)
GsTMDfastTF4LFGM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG+mip-map)
GsTMDfastTF4NL	TMD_P_TF4	Textured flat quadrilateral
GsTMDfastTF4NLB	TMD_P_TF4	Textured flat quadrilateral (2face)
GsTMDfastTF4NLM	TMD_P_TF4	Textured flat quadrilateral (mip-map)
GsTMDfastTG3L	TMD_P_TG3	Textured Gouraud triangle (light source calculation)
GsTMDfastTG3L_FLIP	TMD_P_TG3	Textured Gouraud triangle (light source calculation + FLIP)
GsTMDfastTG3LB	TMD_P_TG3	Textured Gouraud triangle (light source calculation + 2face)
GsTMDfastTG3LFG	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG)
GsTMDfastTG3LFG_FLIP	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG + FLIP)
GsTMDfastTG3LFGB	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG + 2face)
GsTMDfastTG3NL	TMD_P_TG3	Textured Gouraud triangle
GsTMDfastTG3NL_FLIP	TMD_P_TG3	Textured Gouraud triangle (FLIP)
GsTMDfastTG3NLB	TMD_P_TG3	Textured Gouraud triangle (2face)
GsTMDfastTG4L	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation)
GsTMDfastTG4LB	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation + 2face)
GsTMDfastTG4LM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +mip-map)

Low-level function name	First arg (op) type	Description
GsTMDfastTG4LFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation + FOG)
GsTMDfastTG4LFGB	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation + FOG + 2face)
GsTMDfastTG4LFGM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation + FOG + mip-map)
GsTMDfastTG4NL	TMD_P_TG4	Textured Gouraud quadrilateral
GsTMDfastTG4NLB	TMD_P_TG4	Textured Gouraud quadrilateral (2face)
GsTMDfastTG4NLM	TMD_P_TG4	Textured Gouraud quadrilateral (mip-map)
GsTMDfastF3GL	TMD_P_F3G	Flat gradation triangle (light source calculation)
GsTMDfastF3GLFG	TMD_P_F3G	Flat gradation triangle (light source calculation + FOG)
GsTMDfastF3GNL	TMD_P_F3G	Flat gradation triangle
GsTMDfastG3GL	TMD_P_G3G	Gouraud gradation triangle (light source calculation)
GsTMDfastG3GLFG	TMD_P_G3G	Gouraud gradation triangle (light source calculation + FOG)
GsTMDfastG3GNL	TMD_P_G3G	Gouraud gradation triangle
GsTMDfastF4GL	TMD_P_F4G	Flat gradation quadrilateral (light source calculation)
GsTMDfastF4GLFG	TMD_P_F4G	Flat gradation quadrilateral (light source calculation + FOG)
GsTMDfastF4GNL	TMD_P_F4G	Flat gradation quadrilateral
GsTMDfastG4GL	TMD_P_G4G	Gouraud gradation quadrilateral (light source calculation)
GsTMDfastG4GLFG	TMD_P_G4G	Gouraud gradation quadrilateral (light source calculation + FOG)
GsTMDfastG4GNL	TMD_P_G4G	Gouraud gradation quadrilateral
GsTMDfastF3M	TMD_P_F3	Flat triangle (light source + material attenuation)
GsTMDfastF3MFG	TMD_P_F3	Flat triangle (light source + FOG + material attenuation)
GsTMDfastF4M	TMD_P_F4	Flat quadrilateral (light source + material attenuation)
GsTMDfastF4MFG	TMD_P_F4	Flat quadrilateral (light source + FOG + material attenuation)
GsTMDfastG3M	TMD_P_G3	Gouraud triangle (light source + material attenuation)
GsTMDfastG3MFG	TMD_P_G3	Gouraud triangle (light source + FOG + material attenuation)
GsTMDfastG4M	TMD_P_G4	Gouraud quadrilateral (light source + material attenuation)
GsTMDfastG4MFG	TMD_P_G4	Gouraud quadrilateral (light source + FOG + material attenuation)

Low-level function name	First arg (op) type	Description
GsTMDfastTF3M	TMD_P_TF3	Textured flat triangle (light source + material attenuation)
GsTMDfastTF3MFG	TMD_P_TF3	Textured flat triangle (light source + FOG + material attenuation)
GsTMDfastTF4M	TMD_P_TF4	Textured flat quadrilateral (light source + material attenuation)
GsTMDfastTF4MFG	TMD_P_TF4	Textured flat quadrilateral (light source + FOG + material attenuation)
GsTMDfastTG3M	TMD_P_TG3	Textured Gouraud triangle (light source + material attenuation)
GsTMDfastTG3MFG	TMD_P_TG3	Textured Gouraud triangle (light source + FOG + material attenuation)
GsTMDfastTG4M	TMD_P_TG4	Textured Gouraud quadrilateral (light source + material attenuation)
GsTMDfastTG4MFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source + FOG + material attenuation)

GsTMDfastN...() [no normals]

Low-level function name	First arg (op) type	Description
GsTMDfastNF3	TMD_P_NF3	Flat triangle
GsTMDfastNF3B	TMD_P_NF3	Flat triangle (2face)
GsTMDfastNF4	TMD_P_NF4	Flat quadrilateral
GsTMDfastNF4B	TMD_P_NF4	Flat quadrilateral (2face)
GsTMDfastNG3	TMD_P_NG3	Gouraud triangle
GsTMDfastNG3B	TMD_P_NG3	Gouraud triangle (2face)
GsTMDfastNG4	TMD_P_NG4	Gouraud quadrilateral
GsTMDfastNG4B	TMD_P_NG4	Gouraud quadrilateral (2face)
GsTMDfastTNF3	TMD_P_TNF3	Textured flat triangle
GsTMDfastTNF3B	TMD_P_TNF3	Textured flat triangle (2face)
GsTMDfastTNF4	TMD_P_TNF4	Textured flat quadrilateral
GsTMDfastTNF4B	TMD_P_TNF4	Textured flat quadrilateral (2face)
GsTMDfastTNF4M	TMD_P_TNF4	Textured flat quadrilateral (mip-map)
GsTMDfastTNG3	TMD_P_TNG3	Textured Gouraud triangle
GsTMDfastTNG3_FLIP	TMD_P_TNG3	Textured Gouraud triangle (FLIP)
GsTMDfastTNG3B	TMD_P_TNG3	Textured Gouraud triangle (2face)
GsTMDfastTNG4	TMD_P_TNG4	Textured Gouraud quadrilateral
GsTMDfastTNG4B	TMD_P_TNG4	Textured Gouraud quadrilateral (2face)
GsTMDfastTNG4M	TMD_P_TNG4	Textured Gouraud quadrilateral (mip-map)

Return value

Starting address of unused packet area.

Remarks

With gradation, each vertex of the TMD polygon has a different RGB value.

For high speed operation, libgte contains tuned assembly-level low-level functions.

See also: GsSortObject4J().

GsIncFrame

Updates the frame ID.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	7/31/96

Syntax

GsIncFrame() (macro)

Arguments

None

Explanation

GsIncFrame is a macro called from within GsSwapDispBuff(). It increments the global variable PSDCNT by 1. PSDCNT is 32 bits in length, and restarts at 1 rather than 0 when it overflows.

PSDCNT is used by GsGetLw(),GsGetLs(),GsGetLws() when determining the validity of the matrix cache.

If you are not using GsSwapDispBuff() to swap double buf you must call GsIncFrame to swap the buffers when you use GsGetLw(), GsGetLs(), and GsGetLws().

Return value

None

Remarks

Use GsDefDispBuff() to establish settings the first time.

See also: GsGetLw (p. 9-47), GsGetLs (p. 9-47), GsGetLws (p. 9-49), GsSwapDispBuff (p. 9-109).

GsSetAzwh

Sets conditions for active subdivision.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.3	7/31/96

Syntax

```
void GsSetAzwh (
int z,
short w,
short h
)
```

Arguments

z Critical near z value for activate subdivision
w, h Size of polygon within subdivision routine at which no more subdivision will be done

Explanation

Sets the conditions for active subdivision.

Z is the near z value for the start of the subdivision fragmentation and *w, h* is the polygon size for halting the subdivision.

Return value

None

Remarks

See also:

List of External Variables

Name of external variable	Type	Description
CLIP2	RECT	Two-dimensional clipping area. Can also be set with GsClip2D
PSDBASEX[2]	short	Double buffer origin (X coordinate) Set with GsDefDispbuff()
PSDBASEY[2]	short	Double buffer origin (Ycoordinate) Set with GsDefDispbuff()
PSDIDX	short	Double buffer index
PSDCNT	u_long	A number that is incremented with each frame switch
POSITION	_GsPOSITION	Two-dimensional offset
GsDRAWENV	DRAWENV	Gs draw environment
GsDISPENV	DISPENV	Gs display environment
GsLSMATRIX	MATRIX	Gs local screen matrix Set with GsSetLs()
GsWSMATRIX	MATRIX	Gs world screen matrix Set with GsSetRefView(), etc.
GsLIGHT_MODE	int	Default light mode
HWD0	long	Horizontal resolution
VWD0	long	Vertical resolution
GsLIGHTWSMATRIX	MATRIX	Gs light matrix. Set with GsSetFlatLight()
GsIDMATRIX	MATRIX	Unit matrix
GsIDMATRIX2	MATRIX	Unit matrix (includes aspect conversion)
GsLIGHT_FUNC	Function pointer	Pointer to default light-source calculation routines used by GsDOBJ1, GsDOBJ2
GsOUT_PACKET_P	u_long	Pointer for saving start of packet area. Set with GsSetWorkBase()
GsMATE_C	u_long	Result of decoding attribute (attenuation coefficient)

GsLMODE	u_long	Result of decoding attribute (Light mode)
GsLIGNR	u_long	Result of decoding attribute (Ignore light)
GsLIOFF	u_long	Result of decoding attribute (No shading)
GsZOVER	u_long	Result of decoding attribute (nearZ)
GsBACKC	u_long	Result of decoding attribute (Two-sided polygons)
GsNDIV	u_long	Result of decoding attribute (Number of divisions)
GsTRATE	u_long	Result of decoding attribute (Semi-transparency rate)
GsTON	u_long	Result of decoding attribute (Semi-transparency)
GsDISPON	u_long	Result of decoding attribute (Display ON/OFF)
GsADIVH	short	Condition for active automatic divisions: Vertical size Set with GsSetAzwh(z,w,h)
GsADIVW	short	Condition for active automatic divisions: Horizontal size Set with GsSetAzwh(z,w,h)
GsADIVZ	long	Condition for active automatic divisions: Z value Set with GsSetAzwh(z,w,h)
GsFCALL5	Structure	Function table for GsSortObject5J()
GsFCALL4	Structure	Function table for GsSortObject5J()

Chapter 10: CD/Streaming Library

Table of Contents

Structures	
CdIATV	10-3
CdIFILE	10-4
CdIFILTER	10-5
CdILOC	10-6
StHEADER	10-7
Functions	
CdComstr	10-8
CdControl	10-9
CdControlB	10-11
CdControlF	10-12
CdDataCallback	10-13
CdDataSync	10-14
CdDiskReady	10-15
CdFlush	10-17
CdGetDiskType	10-18
CdGetSector	10-19
CdGetSector2	10-20
CdGetToc	10-21
CdInit	10-22
CdIntstr	10-23
CdIntToPos	10-24
CdLastCom	10-25
CdLastPos	10-26
CdMix	10-27
CdMode	10-28
CdPlay	10-29
CdPosToInt	10-30
CdRead	10-31
CdRead2	10-32
CdReadBreak	10-33
CdReadCallback	10-34
CdReadExec	10-35
CdReadFile	10-36
CdReadSync	10-37
CdReady	10-38
CdReadyCallback	10-39
CdReset	10-40
CdSearchFile	10-41
CdSetDebug	10-42
CdStatus	10-43
CdSync	10-44
CdSyncCallback	10-45
StCdInterrupt	10-46
StClearRing	10-47
StFreeRing	10-48
StGetBackloc	10-49
StGetNext	10-50
StGetNextS	10-51
StNextStatus	10-52
StRingStatus	10-53
StSetChannel	10-54
StSetEmulate	10-55
StSetMask	10-56

StSetRing	10-57
StSetStream	10-58
StUnSetRing	10-60

CdIATV

Audio attenuation structure.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	10/01/97

Structure

```
typedef struct {
    unsigned char val0;
    unsigned char val1;
    unsigned char val2;
    unsigned char val3;
} CdIATV;
```

Members

val0 CD (L) --> SPU (L) reduction
val1 CD (L) --> SPU (R) reduction
val2 CD (R) --> SPU (R) reduction
val3 CD (R) --> SPU (L) reduction

Explanation

Sets CD audio volume for the structure.

CD audio consists of CD-DA audio and CD-XA audio.

Val0 - val3 can range from 0 (minimum volume) to 255 (maximum volume).

For adjusting normal stereo volume, set the L channel volume in val0 and the R channel volume in val1. Val0 and val3 should be set to 0.

Remarks

See also:

CdIFILE

ISO-9660 file system file descriptor.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Structure

```
typedef struct {
    CdILOC pos;
    unsigned long size;
    char name[16];
} CdIFILE;
```

Members

pos File position
size File size
name File name

Explanation

Get position and size of ISO-9660 CD-ROM file.

Remarks

See also:

CdIFILTER

ADPCM channel.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    u_char file;
    u_char chan;
    u_short pad;
} CdIFILTER;
```

Members

file File ID
chan Channel ID
pad System reserved

Explanation

Sets the multi-channel ADPCM play channel.

Remarks

See also: CdlSetfilter (p.).

CdILOC

Time-code based CD-ROM disc position.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Structure

```
typedef struct {
    unsigned char minute;
    unsigned char second;
    unsigned char sector;
    unsigned char track;
} CdILOC;
```

Members

<i>minute</i>	Minute
<i>second</i>	Second
<i>sector</i>	Sector
<i>track</i>	Track number

Explanation

Structure defining a time-code position on a CD-ROM. The time code is based on the time needed to reach that position when playing the disc from the beginning at normal speed.

Remarks

The track member is not used at present.

See also:

StHEADER

Sector header.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Structure

```
typedef struct {
    unsigned short id;
    unsigned short type;
    unsigned short secCount;
    unsigned short nSectors;
    unsigned long frameCount;
    unsigned long frameSize;
    unsigned short width;
    unsigned short height;
    unsigned long dummy1;
    unsigned long dummy2;
    CdlLoc loc;
} StHEADER;
```

Members

<i>id</i>	Reserved by system
<i>type</i>	Data type (always 0x0160)
<i>secCount</i>	Sector offset within 1 frame
<i>nSectors</i>	Number of sectors comprising one frame
<i>frameCount</i>	Movie absolute frame number
<i>frameSize</i>	Movie data size (in long words)
<i>width</i>	Movie horizontal size
<i>height</i>	Movie vertical size
<i>dummy1</i>	Reserved by system
<i>dummy2</i>	Reserved by system
<i>loc</i>	File location

Explanation

Movie sector header.

If a header obtained with `StGetNext()` is written to this structure, the various items of information can be accessed through the structure members.

For details of information structure, refer to “Data Format” in the Run-time Library Overview manual.

Remarks

See also:

CdComstr

Get character string corresponding to command code (for debugging).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
char *CdComstr(com)
unsigned char com;
```

Arguments

com Command completion code

Explanation

For debugging. Get corresponding character string from processing status code. For example, get the following character strings for these codes.

Table 10–1

Command Code	Character String
CdINop	"CdINop"
CdISetloc	"CdISetloc"
CdIPlay	"CdIPlay"
CdIForward	"CdIForward"
CdIBackword	"CdIBackword"

Return value

Pointer to start of character string

Remarks

See also:

CdControl

Issue primitive command to CD-ROM system.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	5/22/97

Syntax

```
int CdControl(com, *param, *result)
unsigned char com, *param, *result;
```

Arguments

com Command code
param Pointer to command arguments
result Pointer to return value storage buffer (requires 8 bytes)

Explanation

Issues the primitive command *com* to the CD-ROM system. If the command takes an argument, CdControl() sets these arguments in *param*. Uses *result* to store the return value of the command in the specified buffer.

The stored contents of command (*com*), the arguments (*param*), and the return value (*result*) are listed below.

This function is a non-blocking function, so it is necessary to use CdSync to detect actual transfer termination.

Table 10–2: Primitive command overview

Symbol	Code	Type	Contents
CdINop	0x01	B	NOP (No Operation)
CdISetloc	0x02	B	Sets the seek target position
CdIPlay	0x03	B	Commence CD-DA play
CdIForward	0x04	B	Forward
CdIBackword	0x05	B	Rewind
CdIReadN	0x06	B	Start data read (with retry)
CdIStandby	0x07	N	Stand by with disk rotating
CdIStop	0x08	N	Disk stopped
CdIPause	0x09	N	Pause at current position
CdIMute	0x0b	B	CD-DA mute
CdIDemute	0x0c	B	Cancel mute
CdISetfilter	0x0d	B	Choose ADCPM play sector
CdISetmode	0x0e	B	Set basic mode
CdIGetparam	0x0f	B	Gets current CD subsystem mode
CdIGetlocL	0x10	B	Get logical location (data sector)
CdIGetlocP	0x11	B	Get physical location (audio sector)
CdISseekL	0x15	N	Logical seek (data sector seek)
CdISseekP	0x16	N	Physical seek (audio sector seek)
CdIReadS	0x1b	B	Commence data read (no retry)

B: Blocking, N: Non-Blocking operation

Table 10–3: Primitive commands that take arguments and their arguments

Symbol	Parameter	Type	Contents
CdlSetloc	ICdlLOC	*	Start sector location
CdlReadN	ICdlLOC	*	Start sector location
CdlReadS	ICdlLOC	*	Start sector location
CdlPlay	ICdlLOC	*	Start sector location
CdlSetfilter	ICdlFILTER	*	Set ADCPM sector play
CdlSetmode	lu_char	*	Set basic mode

Table 10–4: Return values of primitive commands

Symbol	Return Value and Stored Byte Position							
	0	1	2	3	4	5	6	7
CdlNop	Status							
CdlSetloc	Status							
CdlPlay	Status							
CdlForward	Status							
CdlBackword	Status							
CdlReadN	Status							
CdlStanby	Status							
CdlStop	Status							
CdlPause	Status							
CdlMute	Status							
CdlDemute	Status							
CdlSetfilter	Status							
CdlSetmode	Status							
CdlGetparam	Satus	Mode						
CdlGetlocL	Min	Sec	Sector	Mode	File	Chan		
CdlGetlocP	Track	Index	Min	Sec	Frame	Amin	Asec	Aframe
CdlSeekL	Status	Btrack	Etrack					
CdlSeekP	Status	Min	Sec					
CdlReadS	Status							

Return value

1 if the command is issued successfully; 0 if failed.

Remarks

Set *param* to 0 for commands that do not require arguments. If result is set to 0, the return value is not stored. With non-blocking functions, the actual transmission completion must be detected by CdSync()

See also:

CdControlB

Issue primitive command to CD-ROM system (Blocking-type function).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdControlB(com, *param, *result)
unsigned char com, *param, *result;
```

Arguments

com Command code
param Pointer to command arguments
result Pointer to return value storage buffer (requires 8 bytes)

Explanation

Issues the primitive command *com* to the CD-ROM system. If the command takes an argument, CdControlB() sets these arguments in *param*. Uses *result* to store the return value of the command in the specified buffer.

CdControlB() is identical to CdControl() except for the block function that waits to return until processing terminates.

For details, see the commands and arguments of CdControl(), and the Run-time Library 3.0 Overview manual.

Return value

1 if issued successfully; 0 if failed.

Remarks

Set *param* to 0 for commands that do not require arguments. If *result* is set to 0, the return value is not stored.

See also:

CdControlF

Issue primitive command to CD-ROM system (highspeed type).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdControlF(com, *param)
unsigned char com, *param;
```

Arguments

com Command code (see separate item)
param Pointer to an argument for command

Explanation

Issues the primitive command *com* to the CD-ROM system. If the command takes an argument, CdControlF() sets these arguments in *param*. Uses result to store the return value of the command in the specified buffer.

CdControlF() is fast because it does no handshaking with the subsystem (it does not even wait for command acknowledgement (ACK)).

For details, see the commands and arguments of CdControl(), and the Run-time Library 3.0 Overview.

Return value

1 if issued successfully; 0 if failed.

Remarks

Set *param* to 0 for commands that do not require arguments. At present 1 is always returned, so "return value" has no significance.

See also:

CdDataCallback

Defines a routine that will be executed when a data transfer initiated by CdGetSector() or CdGetSector2() is completed.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.0	6/22/98

Syntax

```
void *CdDataCallback (
void (*func) ()
)
```

Arguments

func Pointer to address of callback function

Explanation

Defines a routine that will be executed when the data transfer of data initiated by the CdGetSector() or CdGetSector2() function has been completed. The *func* parameter is the address of the desired routine. If *func* is 0, then any previous callback routine is disabled.

Return Value

Address of previously set callback

Remarks

While *func* is executing, subsequent data transfer complete interrupts are masked. Therefore, *func* should return as soon as the necessary processing is completed.

See also: CdGetSector(), CdGetSector2(), CdDataSync()

CdDataSync

Waits for a data transfer initiated by CdGetSector2() to be completed.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.2	3/26/98

Syntax

```
int CdDataSync(mode)
int mode;
```

Arguments

mode Polling mode:
 0: Blocking
 1: Non-blocking

Explanation

Waits for the data transfer of data initiated by the CdGetSector2() function to be completed. The *mode* parameter determines the method of polling. If *mode* is 0, then the function will wait for the data transfer to be completed. If *mode* is 1, then the function will poll the current status and return.

Return Value

Returns 0 if transfer is completed. Returns 1 if transfer is still being performed. Returns -1 if an error occurred.

Remarks

See also:

CdDiskReady

Determine CD-ROM status after disc change.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	7/31/96

Syntax

```
int CdDiskReady (
  int mode
)
```

Arguments

mode Check mode
 0: Blocking type
 1: Non-blocking type

Explanation

This function checks the CD-ROM status after a disc change to determine whether a command can be issued. Immediately after a disc is changed, there is a delay of a few seconds during which commands may not be issued. This function checks the status so that your program knows when issuing a command is safe.

When the *mode* parameter is 0, this function waits until the CD-ROM status has stabilized and commands may be issued before returning. When the *mode* parameter is 1, this function simply returns the current status.

Return value

CdIComplete The state where a command can be issued

CdIDiskError Blocking type:
 No discs or defected disc
 Non-blocking type:
 Not stable, no discs, or defected disc

CdIStatShellOpen Disc cover is open

Remarks

It is recommended that your program use this function immediately after initiating a disc change to wait for the disc cover to be closed and the CD-ROM status to stabilize. After this is done, check the disc format using the CDGetDiskType() function and proceed accordingly.

Note:

Following is the maximum wait time required for returning from a blocking type function call:

DebuggingStation:

CD-R Maximum of 12 seconds
 CD-DA Maximum of 12 seconds
 No disc Approximately 5 seconds

PlayStation:

Black CD Maximum of 3 seconds
 CD-DA Maximum of 5 seconds
 No disc Approximately 5 seconds

10-16 CD/Streaming Library Functions

Although non-blocking type function returns immediately after checking the disc status, it cannot differentiate two error cases, the non-stable status and no disc case. Thus it is recommended to manage the time out according to the wait time shown above.

Note:

This function does not operate correctly on the DTL-H2000 development system.

See also:

CdFlush

CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.2	7/31/96

Syntax

void CdFlush(*void*)

Arguments

None

Explanation

This function obtains the disc format. Currently only CD-ROM format can be recognized.

Return value

CdICdromFormat CD-ROM format
 CdIOtherFormat Other format
 CdIStatShellOpen Disc cover is open
 CdIStatNoDisk No discs

Remarks

On DebuggingStation, although PlayStation disc (black disc), CD-R, and other CD-ROM (ISO9600 format) can be recognized as a CD-ROM, on PlayStation (consumer model), only the PlayStation disc can be recognized as CD-ROM. CD-DA is always recognized as "Other Format."

Note:

Immediately after changing discs, it is recommended that your program call the CdDiskReady() function, followed by the CdDiskType() function.

Note:

This function does not operate correctly on the DTL-H2000 development system.

See also:

CdGetDiskType

Obtains disc format.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	7/31/96

Syntax

```
int CdGetDiskType(void)
```

Arguments

None

Explanation

This function obtains the disc format. Currently only CD-ROM format can be recognized.

Return value

CdICdromFormat CD-ROM format
 CdIOtherFormat Other format
 CdIStatShellOpen Disc cover is open
 CdIStatNoDisk No discs

Remarks

On DebuggingStation, although PlayStation disc (black disc), CD-R, and other CD-ROM (ISO9600 format) can be recognized as a CD-ROM, on PlayStation (consumer model), only the PlayStation disc can be recognized as CD-ROM. CD-DA is always recognized as "Other Format".

Note:

Immediately after changing discs, it is recommended that your program call the CdDiskReady() function, followed by the CdDiskType() function.

Note:

This function does not operate correctly on the DTL-H2000 development system.

See also:

CdGetSector

Transfers sector buffer data to main memory.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	6/22/98

Syntax

```
int CdGetSector (
void *madr,
int size
)
```

Arguments

madr Main memory address to which the data is transferred
size Size of data to be transferred (long word)

Explanation

Transfers sector data on the sector buffer to the main memory. Transferring is processed in background.

Return Value

Always returns 1.

Remarks

Sector size differs from mode to mode.

Although the data transfer can be divided several times to different memory addresses, the sector data must be read after the buffer becomes ready and before the buffer is overwritten by the next sector data. All data transfer will be completed upon function return.

Since the current CdGetSector() resets the buffer read pointer by Ready callback, there is no need to issue a dummy read to update the pointer.

See also: CdDataSync().

CdGetSector2

Transfers sector buffer data to main memory (Non-blocking type).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	4.0	6/22/98

Syntax

```
int CdGetSector2 (
void *madr,
int size
)
```

Arguments

madr Main memory address to which data is transferred
size Size of data to be transferred (long words)

Explanation

This function transfers sector buffer data to the main memory. Since the transfer is carried out in cycle steal mode, an interrupt can be inserted within the transfer. Because this function is of the non-blocking type which starts the transfer and returns, transfer completion must be monitored by CdDataSync() or CdDataCallback().

Furthermore, although an interrupt can be inserted during transfer in cycle steal mode, since a normal program cannot use a data bus, if there is a command to access the memory through a bus, processing will be blocked at that point.

Return value

Always returns 1.

Remarks

Data transfer in cycle steal mode takes longer than in block mode (the mode used by CdGetSector()).

In the current CdGetSector2(), since the buffer read pointer is reset by Ready Callback, there is no need to transfer dummy data to update the pointer

See also: CdDataCallback(), CdDataSync(), CdGetSector().

CdGetToc

Read CD-ROM table of contents information.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdGetToc(*loc)
CdILOC *loc;
```

Arguments

loc Pointer to location table

Explanation

Get starting position of each track on the CD-ROM disc.

Return value

Positive integer returns a track number; anything else returns Error.

Remarks

The maximum number of tracks is 100.

See also:

CdInit

Initializes CD-ROM system.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	3/26/98

Syntax

```
void CdInit(mode)
CdInit mode;
```

Arguments

mode Reset mode

Explanation

Resets the CD-ROM subsystem. The *mode* parameter is not used by current versions of the library and should be set to 0.

Return value

If initialization is successful, 1 is returned; if it fails, 0 is returned.

Remarks

When initialization fails, up to four retries will be performed. Since CdInit() and CdReset() reset the SPU sound volume and CD input volume to the SPU, etc., they must be called before libspu/libsnd initialization and setting functions. If initialization and setting functions are performed after CdInit() or CdReset(), they must be reset.

See also: CdReset()

CdIntstr

Get character string corresponding to command processing status (for debugging).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
char *CdIntstr(intr)
unsigned char intr;
```

Arguments

intr Processing status code

Explanation

For debugging. Get character string corresponding to processing status code *intr*. For debugging.

Table 10–5

Processing Status	Character String
CdINoIntr	"CdINoIntr"
CdIComplete	"CdIComplete"
CdIDiskError	"CdIDiskError"

Return value

Pointer to start of character string

Remarks

See also:

CdIntToPos

Translate CD position information from an absolute sector number to a minute/seconds/sector time code.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
CdILOC *CdIntToPos(i, *p)
```

```
int i;
```

```
CdILOC *p;
```

Arguments

i Absolute sector number

p Pointer to a CdILOC structure that will be set to the position time code

Explanation

Calculate value for minute/second/sector from absolute sector number.

Return value

Pointer to *p*

Remarks

See also:

CdLastCom

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.2	7/31/96

Syntax

int CdLastCom(void)

Arguments

None

Explanation

Returns the last command issued by CDControl/CDControlB.

Table 10–6

Symbol	Code	Type	Contents
CdINop	0x01	B	NOP (No Operation)
CdISetloc	0x02	B	Sets the seek target position
CdIPlay	0x03	B	Commence CD-DA play
CdIForward	0x04	B	Forward
CdIBackword	0x05	B	Rewind
CdIReadN	0x06	B	Start data read (with retry)
CdIStandby	0x07	N	Stand by with disk rotating
CdIStop	0x08	N	Disk stopped
CdIPause	0x09	N	Pause at current position
CdIMute	0x0b	B	CD-DA mute
CdIDemute	0x0c	B	Cancel mute
CdISetfilter	0x0d	B	Choose ADCPM play sector
CdISetmode	0x0e	B	Set basic mode
CdIGetparam	0x0f	B	Gets current CD subsystem mode
CdIGetlocL	0x10	B	Get logical location (data sector)
CdIGetlocP	0x11	B	Get physical location (audio sector)
CdISseekL	0x15	N	Logical seek (data sector seek)
CdISseekP	0x16	N	Physical seek (audio sector seek)
CdIReadS	0x1b	B	Commence data read (no retry)

B: Blocking, N: Non-Blocking operation

Return Value

Last command

Remarks

See also:

CdLastPos

Obtains the CD-ROM location most recently specified.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	7/31/96

Syntax

CdILOC *CdLastPos(*void*)

Arguments

None

Explanation

This function returns the latest location that was specified by the sub command, CdSetloc/CdIPlay/CdISeekL/CdISeekP/CdIRead/CdIReadS.

Return value

Pointer to the structure, CdILOC containing the CD-ROM location

Remarks

See also:

CdMix

Set attenuation for CD audio.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdMix (
CdIATV *vol
)
```

Arguments

vol Pointer to attenuator volume

Explanation

Set audio volume value for CD audio (CD-DA, ADPCM).

Return value

Always returns 1

Remarks

See also:

CdMode

Obtains the latest CD-ROM mode.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	7/31/96

Syntax

```
int CdMode(void)
```

Arguments

None

Explanation

This function returns the latest CD-ROM mode set.

Return value

CD-ROM mode

Remarks

High speed since this function only refers to the status in the main memory. The mode buffer is updated when a CdSetMode command is issued and also when the mode is specified in the arguments as in CdRead(), etc.

See also:

CdPlay

Plays CD-DA tracks.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	7/31/96

Syntax

```
int CdPlay (
int mode,
int *tracks,
int offset
)
```

Arguments

mode 0: Stops playing
 1: Plays track numbers specified in the *tracks* array in the specified order. Stop at end.
 2: Plays track numbers specified in the *tracks* array in the specified order. Repeat at end.
 3: Returns an index of the array corresponding to the track currently being played.

tracks Pointer to array specifying the track to be played. Must ends with 0.

offset Index of the "tracks" to be played.

Explanation

This function plays multiple tracks specified by the array "tracks" in order. After playing the last track of the array, it repeats or stops playing according to the mode specified.

Return value

Index of the "tracks" currently being played. Not the track number. -1 when it has already stopped playing.

Remarks

All playing is done in the unit of track. The playing or stopping in the middle of the track is not allowed.

See also:

CdPosToInt

Translate CD position information from a minutes/seconds/sector time code to an absolute sector number.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdPosToInt(*p)
CdILOC *p;
```

Arguments

p Pointer to a CdILOC structure that contains the position time code

Explanation

Translate a minutes/seconds/sectors time code contained in a cldLOC structure pointed to by the *p* parameter into an absolute sector value.

Return value

Absolute sector number

Remarks

See also:

CdRead

Read multiple sectors from the CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	5/22/97

Syntax

```
int CdRead(sectors, *buf, mode)
int sectors;
unsigned long *buf;
int mode;
```

Arguments

sectors Read sector count
buf Pointer to read buffer
mode CD-ROM subsystem mode, as defined for CdISetMode command (see the description of the CdControl() function).

Explanation

Reads one or more sectors of data from the CD-ROM to the specified buffer in memory. The starting position for the read is the position last specified for CdISetloc. Each CdRead will require a separate CdISetloc command.

The CdRead() call is non-blocking. Check for completion using the CdReadSync() or CdReadCallback() functions. The CdRead() function uses the CdReadyCallback() function internally, so that function cannot be used with CdRead().

Return value

1 if command issued successfully, otherwise 0.

Remarks

Note that the return code from CdRead only indicates if the command was issued successfully or not. For information about CD-ROM errors which occur during reading, check the result array of the CdReadSync function.

See also:

CdRead2

Starts reading data from the CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdRead2(mode)
int mode;
```

Arguments

mode CD-ROM subsystem mode, as defined for CdISetMode command (see the description of the CdControl() function).

Explanation

Seeks to the position specified by CdISetloc and commences reading data into the internal sector buffer. Commences streaming when the CdIModeStream flag is set in the *mode* parameter. Commences ADPCM audio play when the CdIModeRT flag is set in the *mode* parameter.

This function must be used in conjunction with the CdGetSector() function to transfer data from the internal sector buffer to the program's desired destination buffer. The CdGetSector() function should be called to transfer data as soon as either the CdReady() or CdReadyCallback() functions return the CdIDataReady flag.

Return value

1 if command issued successfully, otherwise 0.

Remarks

See also:

CdReadBreak

Interrupts CdRead.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	4.0	5/22/97

Syntax

```
void CdReadBreak(void)
```

Arguments

None

Explanation

This function is used to interrupt CdRead(). Data which was read up until the interrupt is not secured.

Return value

None

Remarks

Executing CdReadBreak() immediately after executing CdRead() (when 1 sector has not been read), can cause an error.

See also: CdRead().

CdReadCallback

Defines a callback function to be executed on completion of CdRead.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
unsigned long CdReadCallback(*func)  
void(*func)(int status, unsigned char *result);
```

Arguments

func Pointer to callback function address
status Return code of the CdReadSync()
result Pointer to an 8-byte array containing status and result information

Explanation

func defines the callback called when CdRead() completes. *func* is passed two arguments. The *status* argument will be either CdlComplete or CdlDiskError, corresponding to the return code of the CdReadSync() function. The *result* argument is a pointer to an 8-byte array containing status and result information, corresponding to the *result* argument of the CdReadSync() function.

If *func* is set as 0, callback does not occur.

Return value

Address of previously set callback

Remarks

While *func* is executing, subsequent data transfer complete interrupts are masked. Therefore, *func* should return as soon as the necessary processing is completed.

To restore the previous callback, preserve the return value and when processing finishes, use it to restore the previous callback address.

See also:

CdReadExec

Loads PlayStation-format executable program file from CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	6/22/98

Syntax

```
struct EXEC *CdReadExec (
char *file
)
```

Arguments

file Pointer to executable file name

Explanation

This function loads the executable program specified by *file* into main memory at the address specified by the program file header. The file must be an executable program in the PlayStation EXE format. To determine when the load is complete, use the CdReadSync() or CdReadCallback() functions. After loading, the program can be executed as a child process using the Exec() function.

Return value

Pointer to an EXEC structure that describes the loaded program

Remarks

Load address of the executable file should not overlap with the region of its parent process.

See also:

CdReadFile

Reads a file on CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	2/13/97

Syntax

```
int CdReadFile (
char *file,
u_long *addr,
int nbyte
)
```

Arguments

file Pointer to file name
addr Pointer to main memory address to be read-in
nbyte Data size to be read-in

Explanation

Reads *nbyte* of the file on CD-ROM. *nbyte* must be multiple of 2048. Reads entire file when 0 is specified for *nbyte*. When NULL is specified for *file*, it starts reading from the next byte after the byte read by the last CdReadFile.

Return value

0 Read error
Other Number of bytes read

Remarks

The filename must contain a full path specification. All lowercase letters will be converted to uppercase. Reading is performed in the background. Use CdReadSync() or CdReadCallback() to determine when reading is completed.

See also:

CdReadSync

Wait for completion of CdRead and related CD-ROM functions.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdReadSync(mode, *result)
int mode;
unsigned char *result;
```

Arguments

mode Await read completion
result Pointer to status storage buffer of command most recently completed

Explanation

Checks the current status of a data read operation initiated by CdRead, CdReadFile, and other related functions. Depending on the value of the *mode* parameter, either returns the current status immediately or waits for the operation to complete.

Table 10–7

Value	Contents
0	Waits for completing of read and returns
1	Determines current status and promptly returns

Return value

Returns the values below:

Table 10–8

Return value	Contents
Positive integer	Number of sectors remaining
0	Completion
-1	Read error

Remarks

See also:

CdReady

Wait for CD-ROM data to be ready.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdReady(mode, *result)
int mode;
unsigned char *result;
```

Arguments

mode Wait until data is prepared
result Pointer to status storage buffer of command most recently completed.

Explanation

This function is used after a CD-ROM read is initiated using CdRead2(), CdControl (CdIReadS), or CdControl (CdIReadN) to determine if there is data available in the sector buffer which is ready to be transferred using the CdGetSector() function. Depending on the value of the *mode* parameter, either returns the current status immediately or waits for the operation to complete.

Table 10–9

Value	Contents
0	Data waits until it can be prepared and returns
1	Determines current status and promptly returns

Return value

Data-available status is indicated by the following values:

Table 10–10

Return value	Meaning
CdIDataReady	There is data available for transfer
CdIDiskError	Error detected
CdINolntr	No preparation-completed data

Remarks

See also:

CdReadyCallback

Define CdReady callback function.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
unsigned long CdReadyCallback(*func)  
void(*func)(int status, unsigned char *result);
```

Arguments

func Pointer to callback function address
status Return code of the CdReadySync()
result Pointer to an 8-byte array containing status and result information

Explanation

Defines a callback routine to be executed when there is data available in the sector buffer following a CD-ROM read initiated using CdRead2(), CdControl (CdIReadS) or CdControl (CdIReadN). The *func* parameter specifies the address of the desired callback routine. If *func* is NULL, any previous callback routine is disabled.

func is passed two arguments. The *status* argument will be either CdIComplete or CdIDiskError, corresponding to the return code of the CdReadySync() function. The *result* argument is a pointer to an 8-byte array containing status and result information, corresponding to the *result* argument of the CdReadySync() function.

Return value

Address of previously set callback

Remarks

While *func* is executing, subsequent data available interrupts are masked. Therefore *func* should return as soon as the necessary processing is completed.

To restore the previous callback, preserve the return value and when processing finishes, use it to restore the previous callback address.

See also:

CdReset

Initialization of CD-ROM subsystem.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	3/26/98

Syntax

```
int CdReset (
int mode
)
```

Arguments

mode Reset mode

Explanation

Initializes the CD-ROM subsystem. CdInit () low-level function.

Unlike CdInit, this function does not initialize the event environment related to CD-ROM.

In reset mode the following values can be specified

Table 10-11

Mode	Contents
0	Initialization of CD subsystem only
1	Initialization of CD subsystem and CD audio volume (CD-DA, ADPCM)

When mode has been specified as 0, and initialization of CD audio volume is not performed, the volume settings specified in previous sound libraries will be saved.

Return value

If initialization is successful, returns 1. If fails, returns 0.

Remarks

No retry is carried out. Since CdInit() and CdReset() reset the SPU sound volume and CD input volume to the SPU, etc., they must be called before libspu/libsnd initialization and setting functions. If initialization and setting functions are performed after CdInit() or CdReset(), they must be reset.

See also: CdInit (p. 10-22).

CdSearchFile

Get location and size from CD-ROM file name.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	6/22/98

Syntax

```
CdIFILE *CdSearchFile(*fp, *name)
CdIFILE *fp;
char *name;
```

Arguments

fp Pointer to CD-ROM file structure pointer
name Pointer to a file name

Explanation

Determine the position time code (minutes, seconds, sectors) and total length of the specified file on the CD-ROM. The result is stored in the CdIFILE structure pointed to by the *fp* parameter.

Return value

Return Value	Meaning
0	File not found
Other than 0	Pointer to the CD-ROM file structure obtained

Remarks

The *file* specification must be a complete path to the file.

The CdSearchFile() function caches directory information, so subsequent consecutive calls for files in the same directory do not require additional CD-ROM reads. Note that only one directory is cached at a time and reading information for a file in another directory will invalidate the entire cache.

For the best possible performance, include file location and size information in your program at compile time instead of using CdSearchFile.

See also:

CdSetDebug

Set debug level.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdSetDebug(level)
```

```
int level;
```

Arguments

level Debug level

Explanation

Set debug level for CD-ROM subsystem. The possible values of *level* are shown below.

Table 10-12

Value	Contents
0	No checks performed
1	Check primitive commands
2	Print execution status of primitive commands

Return value

Previously set debug mode

Remarks

See also:

CdStatus

Obtains the latest CD-ROM status.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.0	7/31/96

Syntax

```
int CdStatus(void)
```

Arguments

None

Explanation

This function obtains the latest reported CD-ROM status.

Return value

CD-ROM Status

Remarks

This function operates at high speed because it simply returns the status code maintained by the CD-ROM system. The status buffer is updated whenever a CD-ROM command is issued. To explicitly obtain the absolute most current status, issue a CdControl(CdINop) command immediately before your CdStatus() call.

See also:

CdSync

Wait for completion of CD-ROM command.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int CdSync(mode, *result)
```

```
int mode;
```

```
unsigned char *result;
```

Arguments

mode Waits for command termination

result Pointer to status storage buffer of command most recently completed.

Explanation

Waits for actual termination of a command issued by CdControl(). The *mode* parameter specifies whether to wait and return command termination.

Table 10-13

Value	Contents
0	Waits for command termination and returns
1	Determines current status and promptly returns

Return value

Command execution status is indicated by the following values:

Table 10-14

Return value	Meaning
CdIComplete	Command complete
CdIDiskError	Error detected
CdINoIntr	Command is being executed

Remarks

See also:

CdSyncCallback

Define CdSync callback function.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
unsigned long CdSyncCallback(*func)
void(*func)(int status, unsigned char *result);
```

Arguments

func Callback function address
status Return code of CdSync() function
result Pointer to an 8-byte array containing status and result information

Explanation

Defines a callback routine to be executed when a CdControl() command is completed. The *func* parameter specifies the address of the desired callback routine. If *func* is NULL, any previous callback routine is disabled.

func is passed two arguments. The *status* argument will be either CdIComplete or CdIDiskError, corresponding to the return code of the CdSync() function. The *result* argument is a pointer to an 8-byte array containing status and result information, corresponding to the *result* argument of the CdSync() function.

Return value

Address of previously set callback.

Remarks

While *func* is executing, subsequent CD-ROM command complete interrupts are masked. Therefore, *func* should return as soon as the necessary processing is completed.

To restore the previous callback, preserve the return value and when processing finishes, use it to restore the previous callback address.

See also:

StCdInterrupt

Handler for interrupts from CD-ROM (internal function).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	02/15/98

Syntax

void StCdInterrupt(*void*)

Arguments

None

Explanation

This function is used as the CdReadyCallback routine by the StStartStream() and StStartEmulation() functions. It transfers sectors from the CD controller to the streaming ring buffer as they become available. This function does not need to be called directly by the user when playing movies in 16-bit mode.

When playing a movie in 24-bit mode, there is a potential hardware conflict between the CD subsystem and the MDEC image decompression system which can result in corrupted data. To avoid this, the StCdInterrupt() function may defer transferring a sector and instead set a flag variable called StCdInterFlag to indicate that a CD sector is ready to be transferred. Once the MDEC is finished transferring data, your application should check StCdInterFlag and call StCdInterrupt() directly if it is set. Please see the Sony sample code for movie playback for examples of the proper workaround.

Return value

None

Remarks

See also: CdGetSector(), DsGetSector().

StClearRing

Flush ring buffer.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
void StClearRing(void)
```

Arguments

None

Explanation

Flush ring buffer. Flushing the ring buffer when jumping tracks and so forth is effective in preventing excess frames from showing up.

Return value

None

Remarks

See also:

StFreeRing

Release ring buffer.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
unsigned long StFreeRing(*base)
unsigned long *base;
```

Arguments

base Pointer to starting address of user data area of released 1 frame

Explanation

The area obtained by StGetNext() is locked. StFreeRing() releases this locked region. The released region is the region for one frame's worth of data which is used as the base for the starting address of the user region. Linked sector header regions are also released.

If a region locked by StGetNext() is not released when its use ends, the ring buffer will soon overflow and streaming will come to a halt.

Return value

A return value of 0 indicates successful release. 1 denotes a failed release (for example, trying to release something that wasn't locked).

Remarks

See also:

StGetBackloc

Returns the location and ID of the first frame in the ring buffer in order to access frame data without any frame skip. The frame skip due to ring buffer overflow can be avoided by re-accessing the frame location obtained by this function. This function is not appropriate for data with XA AUIO since it requires data access.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	7/31/96

Syntax

```
int StGetBackloc(*loc)
CdILOC *loc;
```

Arguments

loc Pointer to latest location of the first frame.

Explanation

This function returns the latest location information and ID of the frame on the current ring buffer.

The location information obtained here is used as the access target value in order to avoid frame skip due to ring buffer overflow.

Please refer to `\psx\sample\cd\movie\tuto3.c` for usage example.

This function is valid only for StModeStream2 mode.

Return value

Frame ID that should be used upon the streaming restart. -1 for error indicating non StModeStream2 mode.

Remarks

See also:

StGetNext

Get one frame of ring buffer data.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
unsigned long StGetNext(*addr, *header)
unsigned long *addr;
unsigned long *header;
```

Arguments

addr Pointer to user data region starting address for 1 frame of retrieved data
header Pointer to sector header region starting address for 1 frame of retrieved data

Explanation

This function gets one frame of ring buffer data. If the next frame of data is ready in the ring buffer, the starting address of the user data and the sector header are stored in *addr* and *header* respectively. 0 is returned.

The region the data is taken from is locked until StFreeRing() is called, so it cannot be destroyed by new data.

The data region has a continuous address and the ring buffer does not loop in mid-data.

Return value

If 1 FRAME of data is taken from the ring buffer, 0 is returned. If it is not ready, 1 is returned.

Remarks

See also:

StGetNextS

Get one frame of ring buffer data from memory.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	7/31/96

Syntax

```
unsigned long StGetNext(*addr, *header)
unsigned long *addr;
unsigned long *header;
```

Arguments

addr Pointer to user data region starting address for 1 frame of retrieved data
header Pointer to sector header region starting address for 1 frame of retrieved data

Explanation

This function gets one frame of ring buffer data. The starting addresses and the sector header are stored in *addr* and *header* respectively. 0 is returned.

Return value

When one frame of data is taken from the ring buffer, 0 is returned.

Remarks

See also:

StNextStatus

Returns the status of the next frame. This function checks whether the next frame data is available on the ring buffer without affecting the internal state.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	7/31/96

Syntax

```
unsigned long StNextStatus(*addr, *header)
```

```
unsigned long *addr;
```

```
unsigned long *header;
```

Arguments

addr Pointer to starting address of the user data region for 1 frame of retrieved data

header Pointer to starting address of sector header region for 1 frame of retrieved data

Explanation

This function obtains the status of the next frame of ring buffer data.

The internal state is not affected by calling this function.

Following is the possible status:

StFREE Next frame is not on the ring buffer.

StCOMPLETE Next frame is completely read into the ring buffer.

StBUSY Next frame is being read into the ring buffer.

StLOCK Next frame is being processed (one frame is obtained by calling StGetNext but StFreeRing has not been called).

Return value

Next frame status as shown above.

Remarks

See also:

StRingStatus

Returns the status of the ring buffer. Frame skip caused by insufficient free space in the ring buffer can be prevented by calling this function.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	7/31/96

Syntax

```
void StRingStatus (*free_sectors, *over_sectors)
short *free_sectors, *over_sectors;
```

Arguments

free_sectors Pointer to the number of free sectors on the ring buffer.
over_sectors Pointer to the difference between the sector positions of CD-ROM data read in and the sector positions currently being processed.

Explanation

This function reports the ring buffer status with two variables specified as arguments.

The first argument, "free_sectors," is the number of sectors with no data in the unused area of the ring buffer. The larger the "free_sectors" is, the more free space on the ring buffer.

The second argument, "over_sectors," is the difference between the sector positions for CD-ROM data read in and the sector positions currently being processed. The larger the "over_sectors" is, the more unprocessed data on the ring buffer.

The sum of "free_sectors" and "over_sectors" and the total ring buffer size is nearly equal. The reason for not having an exact match in size is that when one frame cannot fit in completely close to the end, rewind occurs.

Return value

None

Remarks

See also:

StSetChannel

Set streaming channel.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
int StSetChannel(ch)  
unsigned long ch;
```

Arguments

ch Playback channel

Explanation

Sets streaming playback channel. *ch* sets the channel (0-31). The channel stores the STR data at the authoring level.

Return value

If the channel is set, return 0; otherwise, return 1.

Remarks

See also:

StSetEmulate

Set parameters for streaming emulation.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	6/22/98

Syntax

```
void StStartEmulate(*addr, loc, start_frame, end_frame, f1, f2)
unsigned long *addr, loc, start_frame, end_frame;
void (*func1)(), (*func2)();
```

Arguments

addr Pointer to emulation data starting address
loc Set color mode
start_frame Streaming start frame
end_frame Streaming end frame
func1: Address of function called back for each 1 FRAME of data. If specified as 0, it will not be called back.
func2: Address of function called back when streaming ends. If specified as 0, it will not be called back.

Explanation

Sets parameters for streaming emulation. Emulation means that CD-ROM data is put into memory in advance and data streaming is performed from memory, not from the CD-ROM, which provides only data-ready timing. In streaming emulation, play time is limited to a few seconds because of limits in memory capacity. Still, emulation is easier than using a CD-ROM emulator.

STR-format data needs to be loaded to *addr* in advance. See StSetStream() for details on other arguments. (*loc* is the same as *mode*.)

Return value

None

Remarks

See also:

StSetMask

Controls the playing of streaming.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
void StSetMask(mask, start, end)
unsigned long mask, start, end;
```

Arguments

mask Streaming play on/off
start StSetStream() start_frame
end StSetStream() end_frame

Explanation

Turns streaming play ON/OFF. There is no mechanical timing lag compared to CD-ROM drive pause and playback, and instant ON/OFF is possible.

Values that can be specified in *mask* are as follows.

Table 10–15

Value	Contents
0	Play
1	Pause

Resets start and end of SetStream() trigger frame values.

Return value

None

Remarks

See also:

StSetRing

Set ring buffer.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

```
void StSetRing(*ring_addr, ring_size)
unsigned long *ring_addr;
unsigned long ring_size;
```

Arguments

ring_addr Pointer to ring buffer starting address
ring_size Ring buffer size (in sectors)

Explanation

Secure a ring buffer of a size specified by *ring_size* from an address specified by *ring_addr*.

To use the Streaming Library, you must first call it.

Because only form-1 CD-ROM sectors are supported at present, one sector of data area is 2048 bytes.

It is necessary to secure this area in the main program.

Return value

None

Remarks

See also:

StSetStream

Set streaming parameters.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	6/22/98

Syntax

```
void StSetStream(mode, start_frame, end_frame, f1, f2)
unsigned long mode, start_frame, end_frame;
void (*func1)(), (*func2)();
```

Arguments

mode Set color mode
start_frame Frame to start streaming
end_frame Frame to end streaming
func1: Address of function called back for each 1 FRAME of data. If specified as 0, it will not be called back.
func2: Address of function called back when streaming ends. If specified as 0, it will not be called back.

Explanation

Sets streaming parameters.

The specified values and contents of each argument are as follows:

a) *mode*

Sets color mode. The values you may specify are as follows:

Table 10-16

Value	Contents
0	16-bit mode
1	24-bit mode

b) *start_frame*

Specifies the frame number (stored in STR data) that starts streaming.

Streaming will not begin until this Streaming Library frame is reached. If you want to play the data starting in the middle, you must specify an appropriate frame number. When you specify 0, streaming commences no matter what the frame number is.

c) *end_frame*

Specifies the frame number (stored in STR data) that ends streaming. Streaming ends when this Streaming Library frame is reached. If you specify a number large enough, it plays the CD-ROM data to the end and terminates. When you specify 0, all the data is stored in the ring buffer and the function automatically terminates. This takes a large ring buffer, and the function is successful when streaming is from memory.

d) *func1*

Generates one frame's worth of data and specifies the address of the callback function called.

e) *func2*

Sets the address of the callback function called at the time streaming is completed.

Return value

None

Remarks

To correctly exit from a streaming application, the end of streaming should not be set by `end_frame`. Set `end_frame` to `0xffffffff`, and code an appropriate endpoint from within the loop.

See also:

StUnSetRing

Release interrupt used by streaming library.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	7/31/96

Syntax

void StUnSetRing(*void*)

Arguments

None

Explanation

Release two interrupt functions CdDataCallback() and CdReadyCallback() hooked by CDRead2(CdIModeStream) and return to initial state.

If the streaming library is not used when streaming ends and control transfers to another program, the interrupt hooks which call this function need to be returned to the initial state.

Return value

None

Remarks

See also:

Chapter 11: Extended CD-ROM Library

Table of Contents

Structures	
DslATV	11-3
DslFILE	11-4
DslFILTER	11-5
DslLOC	11-6
Functions	
DsClose	11-7
DsCommand	11-8
DsControl	11-10
DsControlB	11-11
DsControlF	11-12
DsDataCallback	11-13
DsDataSync	11-14
DsEndReadySystem	11-15
DsFlush	11-16
DsGetDiskType	11-17
DsGetSector	11-18
DsGetSector2	11-19
DsGetToc	11-20
DsInit	11-21
DsIntToPos	11-23
DsLastPos	11-24
DsMix	11-26
DsPacket	11-27
DsPlay	11-29
DsPosToInt	11-30
DsQueueLen	11-31
DsRead	11-32
DsRead2	11-33
DsReadBreak	11-34
DsReadCallback	11-35
DsReadExec	11-36
DsReadFile	11-37
DsReadSync	11-38
DsReady	11-39
DsReadyCallback	11-40
DsReadySystemMode	11-41
DsReset	11-42
DsSearchFile	11-43
DsSetDebug	11-44
DsShellOpen	11-45
DsStartReadySystem	11-46
DsStatus	11-47
DsSync	11-48
DsSyncCallback	11-49
DsSystemStatus	11-50

DslATV

Audio attenuator.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    u_char val0;
    u_char val1;
    u_char val2;
    u_char val3;
} DslATV;
```

Members

val0 CD (L) -> SPU (L) attenuation
val1 CD (L) -> SPU (R) attenuation
val2 CD (R) -> SPU (R) attenuation
val3 CD (R) -> SPU (L) attenuation

Explanation

Structure for setting the CD volume (CD-DA and CD-XA).

The values for *val0* - *val3* can range from 0 to 128. For standard stereo volume adjustments,

- val0* is set to the L channel volume
- val1* is set to 0
- val2* is set to the R channel volume
- val3* is set to 0

Remarks

See also:

DsIFILE

9660 file descriptor.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    DsILOC pos;
    u_long size;
    char name[16];
} DsIFILE;
```

Members

pos File position
size File size (in bytes)
name Filename

Explanation

Structure which stores the position and size of a type 9660 CD-ROM.

Remarks

See also:

DsIFILTER

ADPCM channel.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    u_char file;
    u_char chan;
    u_short pad;
} DsIFILTER;
```

Members

file File number
chan Channel number
pad Reserved for system use

Explanation

Structure which specifies the ADPCM sector channel to be played back.

Remarks

See also:

DsILOC

CD-ROM location.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    u_char minute;
    u_char second;
    u_char sector;
    u_char track;
} DsILOC;
```

Members

minute Minutes
second Seconds
sector Sectors
track Track number

Explanation

Structure which specifies the CD-ROM position. Each element is specified using BCD

Remarks

Track is currently unused.

See also:

DsClose

Close the libds system.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	10/01/97

Syntax

```
void DsClose(void)
```

Arguments

None

Explanation

The libds system is closed. When this function is called, the libds kernel state machine is reset. The callback function which controls the libds kernel that has been forked in the system is detached.

Return value

None

Remarks

This function must be called to close libds whenever control is passed to a child process, a LoadExec is performed, or when CD control functions outside of libds are used. Call DsInit() to reopen libds.

See also: DsInit().

DsCommand

Add primitive command to the command queue.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsCommand (
  u_char com,
  u_char *param,
  DsICB cbsync,
  int count
)
```

Arguments

com Command code
param Pointer to argument for command (u_char[4])
cbsync Pointer to callback function
count Number of retries (0: no retries, -1: unlimited retries)

Explanation

Commands are performed in the background. If execution of the command fails, it will be retried count times. An error is returned if the command failed to complete after it was retried.

If count has a value of 0 it indicates no retry processing should be performed. If count has a value of -1 it indicates the command should be retried indefinitely.

Separate callback functions can be set for each command. The callback triggers when the command completes (or returns an error). The execution status of a command can be obtained with DsSync().

Return value

The command ID (>0) is returned if the command issued successfully, otherwise 0 is returned.

Remarks

See also: DsSync().

DsComstr

Get the character string corresponding to each command code (for debugging)

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.3	6/22/98

Syntax

```
char* DsComstr (
u-char com
)
```

Arguments

com Command completion code

Explanation

Used for debugging.

Gets the corresponding character string from the process status code. For example, character strings such as those below are obtained for the corresponding codes:

Command code	Character string
DslNop	DslNop
DslSetloc	DslSetLoc
DslPlay	DslPlay
DslForward	DslForward
DslBackward	DslBackward

Return value

Pointer to start of character string.

See also: Dslnit().

DsControl

CdControl compatibility function.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsControl (
  u_char com,
  u_char *param,
  u_char *result
)
```

Arguments

com Command code
param Pointer to arguments for command (u_char[4])
result Pointer to storage for the return value (u_char[8])

Explanation

This function provides the same interface as CdControl. Unlike CdControl, however, the command is handled such that the function blocks until the end of the operation, even if the command itself is non-blocking.

Return value

0: Execution of command failed
 1: Execution of command was successful

Remarks

See also:

DsControlB

CdControlB compatibility function.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsControlB (
  u_char com,
  u_char *param,
  u_char *result
)
```

Arguments

com Command code
param Arguments for command (u_char[4])
result Return value for command (u_char[8])

Explanation

This function provides the same interface as CdControlB. The actual timing will differ somewhat since the command queue is used.

Return value

0: Execution of command failed
 1: Execution of command was successful

Remarks

See also:

DsControlF

CdControlF compatibility function.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsControlF (
u_char com,
u_char *param
)
```

Arguments

com Command code
param Pointer to arguments for command (u_char[4])

Explanation

This function provides roughly the same interface as CdControlF except for a few differences such as timing.

Internally, the specified command is simply added to the command queue. Note that the command ID is provided in the return value.

Return value

The command ID (>0) is returned if the command was successfully added to the queue, otherwise 0 is returned.

Remarks

CdControlF waits for the previous command to complete execution before issuing the new command, but DsControlF adds the new command to the queue if the previous command has not completed execution.

See also: DsSync(), DsSyncCallback().

DsDataCallback

Sets the exit callback for DsGetSector() and DsGetSector2().

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	02/15/98

Syntax

```
void (*DsDataCallback(void(*func())))
```

Arguments

func Pointer to callback function

Explanation

The callback function specified by *func* is set as the callback when DsGetSector() or DsGetSector2() exits. *func* is called when reading is complete. No callback is generated when *func* is set to 0. This callback is not particularly meaningful since the transfer of data is finished when DsGetSector() exits.

Return value

Pointer to previous callback.

Remarks

See also: DsGetSector(), DsGetSector2().

DsDataSync

Wait for completion of DsGetSector2.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	02/15/98

Syntax

```
int DsDataSync (
  int mode
)
```

Arguments

mode 0: Wait for end of transfer
 1: Check current status and return immediately

Explanation

Waits for the transfer performed by DsGetSector2() to complete.

Return value

1: transfer is in progress
 0: transfer is complete

Remarks

See also: DsGetSector(), DsGetSector2().

DsEndReadySystem

End simple callback system.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
void DsEndReadySystem(void)
```

Arguments

None

Explanation

Ends simple callback system.

Return value

None

Remarks

This function is executed within a callback function.

See also: DsStartReadySystem().

DsFlush

Flush the command queue.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

void DsFlush(*void*)

Arguments

None

Explanation

All commands that have been entered in the command queue are flushed.

Any commands that are currently executing are allowed to complete but the results are not saved and callbacks are not invoked.

Return value

None

Remarks

If a command is executing when this function is called it is allowed to complete but subsequent commands are held (the commands are added to the queue).

See also:

DsGetDiskType

Get CD type.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsGetDiskType(void)
```

Arguments

None

Explanation

The type of CD currently installed is obtained and the value is returned by the function.

The resulting type can be either a PlayStation (black) or non-PlayStation disk.

This function blocks until the system status (the status obtained from DsSystemStatus()) changes to DslReady.

Return value

DslCdromFormat PlayStation disk

DslOtherFormat Any other type of CD

DslStatNoDisk CD is not installed

DslStatShellOpen CD cover is open

Remarks

The debugging station recognizes ISO9660 CDs (including CD-Rs) as type DslCdromFormat.

This function does not operate properly on the DTL-H2000 or the DTL-H5000.

See also:

DsGetSector

Transfer data from the sector buffer to main memory.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	6/22/98

Syntax

```
int DsGetSector (
void *madr,
int size
)
```

Arguments

madr Pointer to destination area in main memory
size Transfer size (long word)

Explanation

Data is transferred from the sector buffer to the storage area in main memory pointed to by *madr*.

This function blocks until the end of the transfer operation.

Return value

Always returns 1.

Remarks

The sector size varies according to the mode.

The data from the sector buffer can be transferred to memory over a number of iterations. The sector data in the buffer must be transferred to memory before it is overwritten by data from the next sector.

The transfer is complete when the function returns.

See also: DsDataCallback(), DsDataSync(), DsGetSector2().

DsGetSector2

Transfer data from the sector buffer to main memory (non-blocking).

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	06/22/98

Syntax

```
int DsGetSector2 (
void *madr,
int size
)
```

Arguments

madr Pointer to destination area in main memory
size Transfer size (long word)

Explanation

Data is transferred from the sector buffer to the storage area in main memory pointed to by *madr*.

The transfer is performed in cycle-stealing mode so interrupts may be received during the transfer.

Since `DsGetSector2()` is a non-blocking function that can return after the transfer starts, the completion of transfer must be determined using `DsDataSync()` or `DsDataCallback()`.

Receiving interrupts and accessing memory from the CPU are possible even during transfers in cycle-stealing mode. However, other DMA's will be blocked until the transfer is completed.

Return value

Always returns 1.

Remarks

Data transfers in cycle-stealing mode are more time-consuming compared to those in blocking mode (the mode used by `DsGetSector()`).

See also: `DsDataCallback()`, `DsDataSync()`, `DsGetSector()`.

DsGetToc

TOC read.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.1	10/01/97

Syntax

```
int DsGetToc (  
  DsILOC *loc  
)
```

Arguments

loc Location table

Explanation

The starting position of each track on the CD-ROM is obtained.

Return value

Positive integer: track number

Other values: error

Remarks

The largest track number is 100.

See also: DsILOC.

DsInit

Perform system initialization.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	02/15/98

Syntax

```
int DsInit(void)
```

Arguments

None

Explanation

Initializes the libds system.

DsInit() needs to be called just once at the beginning of a program or when restarting a system that was stopped with DsClose().

Return value

1 is returned if successful.

0 is returned if the operation failed.

Remarks

Calling DsInit() in the middle of a program may cause improper operation. DsReset() should be used if initialization needs to be performed in the middle of a program.

Because DsInit() resets the SPU sound volume and the CD input volume to SPU, etc., it should either be called before libspu and libsnd initialization/setting functions or they should be reset after DsInit() is called.

See also: DsClose(), DsReset().

DsInstr

Gets the corresponding character string for the command process status (for debugging).

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.3	6/22/98

Syntax

```
char* DsIntStr (
u_char intr
)
```

Arguments

intr Execution status code

Explanation

For debugging. Gets the corresponding character string from the process status code.

Process status	Character string
DsINolntr	NoIntr
DsIComplete	Complete
DsIDiskError	Disk Error

Return value

Pointer to start of character string.

See also:

DsIntToPos

Get minutes, seconds, sectors from absolute sector number.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
DsLOC* DsIntToPos (
  int i,
  DsLOC *p
)
```

Arguments

i Absolute sector number
p Pointer to buffer for storing result

Explanation

The absolute sector number specified by *i* is converted to minutes, seconds, and sectors and the result is stored in the DsLOC structure pointed to by *p*.

Return value

Pointer to result buffer

Remarks

See also: DsPosToInt(), DsLOC().

DsLastCom

Get the command issued last.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.3	6/22/98

Syntax

u_char DsLastCom (void)

Arguments

None

Explanation

Returns the primitive command code issued last.

Return value

Primitive command code

Remarks

See also: DsControl

DsLastPos

Get the last setloc position.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
DsILOC *DsLastPos (
DsiLOC *p
)
```

Arguments

p Pointer to buffer in which position is stored

Explanation

The last setloc position is obtained and the result is stored in the DsiLOC structure pointed to by *p*.

Return value

Pointer to result buffer

Remarks

See also: DsiLOC().

DsMix

Set attenuator.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsMix (
DslATV *vol
)
```

Arguments

vol Attenuator volume

Explanation

The CD audio volume (CD-DA/ADPCM) is set to the value in the DslATV structure pointed to by *vol*.

Return value

Always returns 1.

Remarks

See also: DslATV().

DsPacket

Adds a sequence of commands to the queue.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsPacket (
  u_char mode,
  DsLOC *pos,
  u_char com,
  DsICB *cbsync,
  int count
)
```

Arguments

mode Operating mode
pos Pointer to DsLOC structure specifying target position
com Last command to be executed
cbsync Pointer to callback function to be triggered when all the commands have been executed
count Retry count (0: no retries, -1: unlimited retries)

Explanation

A sequence of commands which perform a data read (playback) is added to the queue.

The commands added to the queue are as follows.

DsIPause

DsSetmode mode

DsSetloc pos

The command specified by com.

The following commands can be specified for com.

- * DsIPlay
- * DsIReadN
- * DsIReadS
- * DsISeekP
- * DsISeekL

If either DsIPlay, DsIReadN, or DsIReadS is specified, execution is performed up to and including the data read (playback).

If either DsISeekP or DsISeekL is specified, the seek is performed and the system enters pause state when the operation completes.

If any of the commands in the sequence generates an error, a retry is performed starting with the first command. The number of retries to be performed is specified by the count parameter. Retries will not be performed if count = 0, and unlimited retries will be performed if count=-1.

An error is generated if the operation is not successful after count retries.

If all the commands in the sequence are successful or if an error is generated, the callback function specified by cbsync is triggered.

DsSync() can be used to obtain the execution status.

Return value

The command ID (>0) is returned if the command was added to the queue. A 0 is returned if the command failed.

Remarks

An error is generated if the queue does not have enough space for the command sequence.

See also: DsCommand().

DsPlay

Play back CD-DA tracks.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsPlay (
int mode,
int *tracks,
int offset
)
```

Arguments

mode Mode

tracks Pointer to array that specifies the tracks to be played back; the last element of the array must be 0.

offset Index for track to begin playback

Explanation

The tracks specified by the tracks array are played in the background in sequence.

When the final track in the series is done, playback is repeated or is stopped, depending on the mode. The values available for mode are shown below.

Mode	Description
0	Stop playback
1	Play back the tracks specified by tracks in sequence Stop playback once all the tracks have been played
2	Play back the tracks specified by tracks in sequence Repeat from the beginning once all the tracks have been played
3	Return the index of the track currently being played

Return value

Returns the track currently being played.

The value returned is the index in the tracks array rather than the track number.

A return value of -1 indicates that all tracks have finished playing.

Remarks

Playback is performed in increments of tracks. Playback cannot start or stop in the middle of a track.

See also:

DsPosToInt

Get absolute sector number from minutes, seconds, sectors.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsPosToInt (
DslLOC *p
)
```

Arguments

p Pointer to DslLoc structure containing minutes, seconds, sectors

Explanation

The absolute sector number is calculated from the minutes, seconds, and sectors in the DslLOC structure pointed to by *p* and its value is returned.

Return value

The absolute sector number is returned.

Remarks

See also: DsIntToPos(), DslLOC().

DsQueueLen

Get the number of commands stored in the command queue.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsQueueLen(void)
```

Arguments

None

Explanation

The number of primitive commands stored in the command queue is obtained and returned by the function.

The commands issued by DsPacket() are not removed from the queue until all of the commands in the packet successfully complete. Therefore the number of commands returned by DsQueueLen() will remain unchanged during execution of the packet.

Return value

None

Remarks

The command currently executing is considered as being in the queue. The maximum number of commands that the queue can hold is defined by the DsIMaxCOMMANDS macro constant.

See also:

DsRead

Read data.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	10/01/97

Syntax

```
int DsRead (
DslLOC *pos,
int sectors,
u_long *buf,
int mod
)
```

Arguments

pos Pointer to DslLOC structure specifying starting position of CD
sectors Number of sectors to read
buf Pointer to buffer where read data will be stored
mode Operating mode to be used when data is being read

Explanation

CD data starting at the location specified by the DslLOC structure pointed to by *p* is read. The data is stored in the buffer pointed to by *buf*.

The operation is performed in units of sectors so the size of *buf* must be a multiple of 1 sector=2048 bytes (512 words).

The operation is performed in the background after the function has executed and exited. Note that the successful execution of the function does not indicate that the data has been successfully read.

Return value

Positive integer: function execution was successful (the id of the packet that was issued within the function is returned).

1: function execution failed.

Remarks

Note that the arguments are different from CdRead(). With DsRead(), the starting position of the data must be specified.

See also: DsReadBreak(), DsReadCallback(), DsReadSync().

DsRead2

Begin playback of movie.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsRead2 (
DslLOC *pos,
int mod
)
```

Arguments

pos Pointer to DslLOC structure specifying starting position of CD
mode Operating mode during playback

Explanation

The movie starting at the location specified by the DslLOC structure pointed to by *pos* is played back. A streaming library callback (provided by the library) is set and the reading of data is begun with DslReadS.

Return value

The command ID (>0) is returned if the function succeeded, otherwise 0 is returned if the command failed.

Remarks

Note that the arguments are different from CdRead2(). With DsRead2(), the starting position of the data must be specified.

See also:

DsReadBreak

Interrupt DsRead() operation.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

void DsReadBreak(*void*)

Arguments

None

Explanation

Interrupts a DsRead() operation.

Return value

None

Remarks

See also: DsRead().

DsReadCallback

Set a callback function to be called when DsRead() is finished.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
DsICB DsReadCallback (
DsICB func
)
```

Arguments

func Pointer to callback function

Explanation

The callback to be triggered when DsRead() completes is set to the function pointed to by *func*.

Return value

Pointer to previous callback function

Remarks

See also: DsRead().

DsReadExec

Read an executable file.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	6/22/98

Syntax

```
struct EXEC* DsReadExec (
char *file
)
```

Arguments

file Filename

Explanation

The executable file specified by *file* is loaded from the CD-ROM and placed at an appropriate location in main memory.

The read is performed in the background, so `DsReadSync()` should be used to see if the read has completed. The loaded file is executed as a child process using `Exec()`.

Return value

Pointer to EXEC structure of the loaded executable file.

Remarks

The load address of the executable file must not overlap with the area used by the parent process.

See also:

DsReadFile

Read a file from CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsReadFile (
char *file,
u_long *addr,
int nbyte
)
```

Arguments

file Filename
addr Pointer to buffer in memory for storing read data
nbyte Number of bytes to read

Explanation

nbyte bytes are read from the CD-ROM file specified by *file* and stored at the buffer pointed to by *addr*.

nbyte must be a multiple of 2048. If *nbyte* is set to 0 the entire file is read. If *file* is set to NULL the read operation begins from the point where the previous *DsReadFile()* left off.

Return value

0 is returned if an error occurred, otherwise the number of bytes read is returned.

Remarks

The filenames must all be represented by absolute paths. Lowercase characters are automatically converted to uppercase.

The read is performed in the background. *DsReadSync()* is used to check for completion of reading.

See also:

DsReadSync

Wait for completion of DsRead().

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsReadSync (
  u_char *result
)
```

Arguments

result Pointer to buffer holding execution results (u_char[8])

Explanation

Waits for completion of DsRead(). Returns the execution status of DsRead() at the point when DsReadSync() was called.

Return value

0 is returned if DsRead() has completed.
 -1 is returned when an error is detected (DsRead() was interrupted).
 Otherwise, the number of remaining sectors is returned.

Remarks

See also:

DsReady

Check for arrival of data.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsReady (
  u_char *result
)
```

Arguments

result Pointer to buffer for storing results (u_char[8])

Explanation

The status of a data read operation (DslReadS/DslReadN) is determined and the result is stored in the buffer pointed to by *result*.

In report mode, DsReady() checks for arrival of the report from DA playback.

Return value

DslDataReady New data has arrived in the sector buffer.
 DslNoIntr New data has not arrived.
 DslDataEnd Final sector has been confirmed (only for DA playback).

Remarks

The sector buffer value is meaningful only for data reads

See also:

DsReadyCallback

Set up Ready callback function.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
DsICB DsReadyCallback (  
DsICB func  
)
```

Arguments

func Pointer to callback function

Explanation

The Ready callback is set to the function pointed to by *func*.

The Ready callback function is called for data ready interrupts, data end interrupts (generated only for DA playback), and all error interrupts.

Return value

Pointer to previous callback function

Remarks

See also:

DsReadySystemMode

Sets the action of cover open/close for the simple callback.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.1	10/01/97

Syntax

```
int DsReadySystemMode (
int mode
)
```

Arguments

mode 0: When the cover is open, end the simple callback
 1: After the cover opens or closes, perform an automatic retry

Explanation

Sets the action of cover open/close for the simple callback.

When *mode* = 0, if the cover is opened during execution, stop processing and end the simple callback. Then call the user-specified callback function with *intr* = *DsIdiskError*.

When *mode* = 1, if the cover is opened or closed, reissue the command with an error and continue processing. The user-specified callback function is not called.

Furthermore, when *mode* = 1 and the cover is closed, if the disk is not set, the simple callback is completed and the user-specified callback function is called with *intr* = *DsIdiskError*.

Return value

Previously updated *mode*.

Remarks

Initial value of *mode* is 0.

The *mode* is valid until the next time it is set.

See also: *DsStartReadySystem()*, *DsEndReadySystem()*

DsReset

Reset system.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsReset(void)
```

Arguments

None

Explanation

The libds system is reset.

Return value

1 is returned if the reset was successful, otherwise 0 is returned.

Remarks

Always use DsReset() when initializing the system in the middle of a program. DsInit() cannot be used in the middle of a program.

See also: DsInit().

DsSearchFile

Get position and size of CD-ROM file.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	6/22/98

Syntax

```
DsIFILE *DsSearchFile (  
DsIFILE *fp,  
char *name  
)
```

Arguments

fp Pointer to CD-ROM file structure
name Filename

Explanation

The absolute position (minutes, seconds, sectors) and size of the CD-ROM file specified by filename is obtained and the results is stored in the DsIFILE structure pointed to by *fp*.

Return value

0: file not found.
-1: search failed.
Other: pointer to the retrieved file structure.

Remarks

Filenames must be represented by their absolute paths.

The position data for all the files in the same directory as the file specified by *fp* is stored in memory in a cache. Thus, when DsSearchFile() is performed consecutively for files from a single directory, access is faster from the second file on.

A return value of -1 indicates that the read operation on the directory failed for some reason.

See also:

DsSetDebug

Set the debug level.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsSetDebug (
  int level
)
```

Arguments

level Debug level

Explanation

Sets the debug level for the CD-ROM subsystem to the value specified by *level*.

Level can have the values shown below.

Value	Description
0	Do not perform any checks
1	Check primitive commands

Return value

Previous debug level

Remarks

See also:

DsShellOpen

Get the number of times the cover was opened.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsShellOpen(void)
```

Arguments

None

Explanation

The number of times the cover was opened since the program began is obtained and the value is returned by the function.

Note that the count is initialized to 1 when the program starts.

Return value

Number of times the cover was opened.

Remarks

This function returns the correct value only when DsSystemStatus()=DsReady.

See also: DsSystemStatus().

DsStartReadySystem

Start the simple callback.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	10/01/97

Syntax

```
int DsStartReadySystem (
    DslRCB func,
    int count
)
```

Arguments

func Pointer to callback function
count Retry count (-1: unlimited retries)

Explanation

Starts the simple callback.

When the simple callback is started, a DslDiskError will result in a retry of the last command. The number of retries to be performed is specified by *count*. If *count* has a value of -1, unlimited retries are performed. The retry count is the total number of retries from the point when the system is started.

The callback function specified by *func* normally triggers when a data read successfully completes. The only time an error will make the function trigger is if the cover is opened or an error is generated after the maximum number of retries.

When a retry is performed, the position from which to re-read is determined by the library, but the callback function will trigger from the sector following the previous call. Thus, internally, the callback function does not need to be aware of the retry.

Return value

1 is returned if the function was successful.
 0 is returned if the function failed (system has already been started).

Remarks

This function is always executed from a callback from a corresponding data read (playback) command. Executing the function at other times may corrupt the error handling system.

DsReadyCallback() should be used internally for simple callback. Simultaneous use from the application is not allowed.

See also: DsEndReadySystem().

DsStatus

Get the status of the CD subsystem.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
u_char DsStatus(void)
```

Arguments

None

Explanation

The last reported status of the CD subsystem is obtained and is returned by the function.

The retrieved status is handled by the library so in certain rare cases the value may be different from the current CD subsystem status.

Use DsINop to get the most recent status. ().

Return value

Status of the CD subsystem.

Remarks

See also:

DsSync

Check for completion of primitive command.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	02/15/98

Syntax

```
int DsSync (
  int id,
  u_char *result
)
```

Arguments

id Command ID
result Pointer to buffer for storing result (u_char[8])

Explanation

The execution status of the primitive command specified by *id* is obtained and stored in the memory area pointed to by *result*.

The execution status refers to the command corresponding to the command ID that was active when the function was called. The result value is valid only for DslComplete or DslDiskError.

If *id* is set to 0, the most current result regardless of the type of command can be obtained.

Return value

DslComplete Command has terminated normally.
 DslDiskError Command failed.
 DslNoIntr Command has not yet been executed.
 DslNoResult Execution has terminated but the results have already been destroyed.

Remarks

A certain number of execution results from commands are saved. The maximum number of saved results is defined by the DslMaxRESULTS macro constant.

See also:

DsSyncCallback

Set sync Callback function

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
DsICB DsSyncCallback (
DsICB func
)
```

Arguments

func Pointer to callback function

Explanation

The Sync callback is set to the function pointed to by *func*.

The Sync callback function is triggered for all command termination and error interrupts.

Note that if the queue performs retries for commands that generate errors, the Sync callback function is triggered, rather than the individual callback function set for the command itself.

Return value

Pointer to previous callback function.

Remarks

See also:

DsSystemStatus

Get status of command queue.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	5/22/97

Syntax

```
int DsSystemStatus(void)
```

Arguments

None

Explanation

The status of the command queue is obtained and returned by the function.

If no command is being executed, a DslReady is returned.

If a command is being executed or commands cannot be executed because the cover is open, etc., a DslBusy is returned.

If no CD is installed, a DslNoCD is returned.

Commands issued when the status is not DslReady are all added to the queue.

Return value

DslReady No command is being executed.

DslBusy Command is being executed, or command cannot be executed (e.g., because cover is open).

DslNoCD No CD is installed.

Remarks

See also:

Chapter 12: Controller/Peripherals Library

Table of Contents

Functions	
CheckCallback	12-3
DisableTAP	12-4
EnableTAP	12-5
GetVideoMode	12-6
InitGUN	12-7
InitTAP	12-9
PadChkVsync	12-11
PadEnableCom	12-12
PadEnableGun	12-13
PadGetState	12-14
PadInfoAct	12-15
PadInfoComb	12-17
PadInfoMode	12-19
PadInit	12-21
PadInitDirect	12-22
PadInitGun	12-25
PadInitMtap	12-27
PadRead	12-29
PadRemoveGun	12-30
PadSetAct	12-31
PadSetActAlign	12-33
PadSetMainMode	12-34
PadStartCom	12-35
PadStop	12-36
PadStopCom	12-37
RemoveGUN	12-38
ResetCallback	12-39
ResetGUN	12-40
RestartCallback	12-41
SelectGUN	12-42
SendTAP	12-43
SetVideoMode	12-45
StartGUN	12-46
StartTAP	12-47
StopCallback	12-48
StopGUN	12-49
StopTAP	12-50

CheckCallback

Determines whether the program is executing a callback.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	2.x	7/31/96

Syntax

```
int CheckCallback(void)
```

Arguments

None

Explanation

The CheckCallback() function determines whether the program is currently executing within a callback context or normal context.

Return value

Normal context returns 0. Callback context returns 1.

Remarks

See also:

DisableTAP

Disables communication with the controller

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.6	5/22/97

Syntax

void DisableTAP(*void*)

Arguments

None

Explanation

Temporarily disables communication with the controller.

Although StopTAP() deletes the controller handler activated by Vsync interrupts, this function simply skips controller communication with a flag operation.

Return value

None

Remarks

Although a normal controller communicates via Vsync interrupts, this function is used only with timing longer than 1/60 sec when the controller status is not needed.

See also: EnableTAP()

EnableTAP

Enables occurrence of an event.

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.6	5/22/97

Syntax

```
void EnableTAP(void)
```

Arguments

None

Explanation

Enables communication with a controller which was disabled with DisableTAP().

Return value

None

Remarks

Although a normal controller communicates via Vsync interrupts, this function is used only with timing longer than 1/60 sec when the controller status is not needed.

See also: DisableTAP()

GetVideoMode

Obtains present video signaling system.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.1	7/31/96

Syntax

long GetVideoMode(*void*)

Arguments

None

Explanation

Returns the present video signaling system declared in SetVideoMode.

Return value

Video signaling system mode

Table 12-1

Return Value	Contents
MODE_NTSC system	NTSC system video signaling
MODE_PAL	PAL system video signaling system

Remarks

When SetVideoMode () is not called, no matter what the machine, it will return MODE_NTSC.

See also:

InitGUN

Initializes gun.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.5	6/22/98

Syntax

```
void InitGUN (
char *bufA, long lenA
char *bufB,, long lenB,
char *buf0, char *buf1,
long len
)
```

Arguments

bufA, *bufB* Controller receive data buffer for port 0 and port 1
lenA, *lenB* Length in bytes of *bufA* and *bufB*
buf0, *buf1* Pointer to horizontal/vertical position receive buffer (necessary buffer size is $len*4+2$ bytes)
len Number of gun interrupts allowed between vertical blank periods (20 maximum)

Explanation

This function defines the buffers used to receive data from the light gun and other controllers. Standard controller information for buttons and analog controllers is returned in *bufA* and *bufB*. InitGUN() cannot be used at the same time as InitPAD() or InitTAP().

As of library v4.0, DMA operations and interrupts are blocked within the gun interrupt handler in order to improve the accuracy of the gun.

The more gun interrupts you specify between VBLANK periods, the more processing is required. Set the *len* value as low as possible to reduce overhead.

Since the horizontal direction counter value returns the system clock value, multiply the following coefficients according to the horizontal direction resolution in order to obtain pixel values:

Table 12-2: System Clock/Pixel Clock Variable Table

Mode	Horizontal Direction Resolution	Coefficient
NTSC:	256	0.158532
	320	0.198166
	384	0.226475
	512	0.317065
	640	0.396332
PAL:	256	0.157086
	320	0.196358
	384	0.224409
	512	0.314173
	640	0.392717

[Pixel value] = [Coefficient] x [System Block value] + [Offset]

Return value

Remarks

See also: InitPAD(), SelectGUN (p. 12-41), StartGUN(), StopGUN(), RemoveGUN().

InitTAP

Controller initialization.

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.4	7/31/96

Syntax

```
void InitTAP (
char *bufA,
char *bufB,
long lenA,
long lenB
)
```

Arguments

bufA, bufB Pointer to receive data buffer
lenA, lenB Receive data buffer length (unit: byte)

Explanation

This function registers a receive data buffer for the controller.

Return value

None

Remarks

The data format stored in the received buffer is as follows:

Table 12–3: Buffer Data Format

Byte	Contents
0	Received result 0x00: success, 0xff: failure
1	Significant 4 bits: terminal classification 0x1: Mouse 0x2: 16 button analog 0x3: Gun controller 0x4: 16 button 0x5: Analog joystick 0x8: Multi-tap Insignificant 4 bits: Received data byte number/2
3	Button condition; 1: release, 0: push No. 2 bit: right, No. 3 bit: left
4	Transfer volume x direction (-128-127)
5	Transfer volume y direction (-128-127)
6	Transfer volume z direction (-128-127)
7,8	Transfer volume ? direction (-128-127)

Table 12–4: 16 button analog, analog joystick

Byte	Contents
2,3	Button condition 1: release, 0: push
4,5,6,7 ,-	Analog channel value

Table 12-5: Gun controller, 16 buttons

Byte	Contents
2,3	Button condition 1: release, 0: push

Table 12-6: Multi-tap received data configuration

Byte	Contents
0	Received result 0x00: success, 0xff: failure
1	0x80 fixed
2	Connector #1 received result; 0: success, 0xff: failure
3	Same as above (terminal classification<<4) / (received data byte number/2)
4-9	Same as above (received data)
10	Connector #2: received result
11	Same as above (terminal classification<<4) / (Received data byte number/2)
12-17	Same as above; received data
18	Connector #3: received result; 0:success, 0xff: failure
19	Same as above (terminal classification<<4) / (Received data byte number/2)
20-25	Same as above received data
26	Connector #4: received result; 0:success, 0xff: failure
27	Same as above (terminal classification<<4) / (Received data byte number/2)
28-33	Same as above received data

The significant 4 bits from the No. 1 byte of the buffer are terminal classifications, and the insignificant four bits are half the value of the number of bytes of received data from the terminal (the buffer's no. 3 byte onwards are stored). Please refer to each terminal's documentation for the physical positioning of the buttons and channels, etc. and compatibility.

*Existing terminal type

The standard controller is a 16 button model terminal.

The mouse produced by this company supports only the two x and y directions. Also, a mouse such as that from Namco Co., Ltd. [neGcon] is a four channel, in other words, the insignificant 4 bits from the No. 1 byte of the buffer are 3 ((Button portion is 2 bytes + channel portion is 4 bytes) by 2, 16 button analog model terminal.

See also: StartTAP (p. 12-45), StopTAP (p. 12-48).

PadChkVsync

Checks communication with controller.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
int PadChkVsync(void)
```

Arguments

None

Explanation

This function is called once during Vsync to confirm that communication with the controller has occurred in a frame.

A 1 is returned if communication with the controller took place. A 0 is returned if there was no communication or if the function is called twice or more in a frame. The function should be called once per frame to confirm controller communication.

Return value

Called once per frame (1/60 sec).

1: Communication with controller took place (regardless of success/failure)

0: Communication with controller did not take place

Remarks

See also:

PadEnableCom

Enables/disables communication with the controller.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
unsigned PadEnableCom (
    unsigned mode
)
```

Arguments

mode Flag to enable/disable communication with the controller
 Bit 0 is used to enable/disable port 0, and bit 1 is used to enable/disable port 1.
 The bits have the following meaning:
 1 = enabled
 0 = disabled

Explanation

In general, communication with the controller takes place once per frame (1/60th of a second). However, when a lower update rate is desired (e.g. when polling for a button press or when only one of the ports is used), communication with the controller can be temporarily disabled with this function to provide the application with greater processing time.

The vertical retrace interrupt itself is not enabled or disabled, so PadEnableCom() only works between PadStartCom() and PadStopCom().

Ports 0 and 1 have a default value of "enabled." Calling PadInitDirect(), PadInitMtap(), PadStartCom(), or PadStopCom() will not affect the enable/disable state set by PadEnableCom().

If communication is suspended for three seconds or more, the controller will be reset. If communication is subsequently restarted, the return value from PadState() will temporarily be PadStateDiscon and a retry will be generated that will refetch controller information. For this reason, the return value from PadGetState() will need to be monitored so that refetched actuator information can be properly processed.

Return value

The previous enable/disable state of communication before the function was called (the previous argument) is returned.

Remarks

See also: PadStartCom(), PadStopCom()

PadEnableGun

Enables/disables gun interrupts.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
void PadEnableGun (
  unsigned char mask
)
```

Arguments

mask Set bits in the mask to 1 to enable gun interrupts for the corresponding ports as shown below.

<i>mask</i> bits	D7	D6	D5	D4	D3	D2	D1	D0
Port number	0x13	0x12	0x11	0x10	0x03	0x02	0x01	0x00

Explanation

This function enables gun interrupts when the corresponding mask bit is set to 1.

Specific gun interrupts can be masked off if horizontal and vertical position information is not needed for those guns.

The default setting is mask disabled for all ports. Retrieval of horizontal and vertical position information begins when a gun is connected.

Return value

None

Remarks

See also: PadInitGun(), PadRemoveGun()

PadGetState

Obtains controller connection state.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
int PadGetState (  
int port  
)
```

Arguments

port The port number to be checked

	Port 1	Port 2
Direct connection	0x00	0x10
Multi-tap A	0x00	0x10
Multi-tap B	0x01	0x11
Multi-tap C	0x02	0x12
Multi-tap D	0x03	0x13

Explanation

Checks that the controller is connected, determines when button-press information is valid, and determines when information from the actuators is valid.

Return value

Value	Macro (libpad.h)	Controller connection state
0	PadStateDiscon	Controller disconnected
1	PadStateFindPad	Find controller connection (checking)
2	PadStateFindCTP1	Check for controller connection with controllers other than DUAL SHOCK (Complete the acquisition of controller information)
4	PadStateReqInfo	Actuator information being retrieved (data being retrieved)
5	PadStateExecCmd	Library is communicating with controller (e.g. PadSetActAlign())
6	PadStateStable	Retrieval of actuator information completed, or library-controller communication completed (PadSetActAlign(), etc. can be called)

Remarks

See also:

PadInfoAct

Obtains actuator information.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
int PadInfoAct (
int port,
int actno,
int term
)
```

Arguments

port Port number of the controller to be checked

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

actno Actuator number to be checked
(ranging from 0 to total number of mounted actuators -1)
Set *actno* to -1 to get the total number of actuators
(in this case, the third argument *term* is ignored)

term Information to be checked about actuator

Explanation

Obtains the actuator function number, sub-function number, actuator parameter data size, and actuator current drain.

Return value

The return value corresponds to the third argument *term* as follows.

Third argument	Macro	Return value
1	InfoActFunc	Function number (1: continuous-rotation vibration)
2	InfoActSub	Sub-function number (When the function number is 1, 1: low-speed rotation, 2: high-speed rotation)
3	InfoActSize	Parameter data length (0: 1 bit (ON/OFF only), 1 or greater: number of bytes)
4	InfoActCurr	Maximum current drain

If the parameter data length for an actuator is more than one byte, each of the parameter write offsets for that actuator can be set by writing the actuator number in each of the corresponding offsets using `PadSetActAlign()`. The controller will interpret the position of the lowest numbered offset as the high-order byte of the parameter. If the actuator number is written such that the data length for the parameter is exceeded, the settings beyond the allowed parameter data length will be ignored, beginning with the lowest numbered offset.

Note:

Up to 60 units of current can be supplied by the main PlayStation unit. Therefore, the current drain of the actuators should be checked to make sure that it does not exceed 60 units. If actuator parameters are set so that the 60-unit limit is exceeded, the actuators connected to the larger port numbers will be ignored (they will be forcibly stopped). This is particularly important for applications that use multi-taps.

The `PadInfoAct()` function will always return a 0 for a controller other than a DUAL SHOCK controller. Even for a DUAL SHOCK controller, the return value will be 0 as long as the state returned from `PadGetState()` is anything other than `PadStateStable` (during `PadStateReqInfo`). When obtaining actuator information, the application should wait for the return value from `PadGetState()` to become `PadStateStable`.

Remarks

See also: `PadInfoComb()`, `PadInfoMode()`

PadInfoComb

Obtains information on actuator combinations that can be used simultaneously.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	6/22/98

Syntax

```
int PadInfoComb (
int port,
int listno,
int offs
)
```

Arguments

port Port number of the controller to be checked

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

listno List number of the combination list to be checked
(ranging from 0 to total combination list -1)
Specify -1 for *listno* to obtain the total number of combination lists.
In this case, the *offs* argument will be ignored

offs Offset within the combination list
(ranging from 0 to total number of actuators contained in the list -1)
Specify -1 for *offs* to obtain the total number of actuators contained in the combination list.

Explanation

Checks combinations of actuators that can be used simultaneously based on restrictions imposed by the physical arrangement of the actuators, etc.

Return value

<i>listno</i>	<i>offs</i>	Return value
-1	---	Total number of combination lists
0 to (Total number-1)	-1	Total number of actuators contained in list number <i>listno</i>
0 to (Total number-1)	0 to (Total number-1)	Actuator number stored at offset <i>offs</i> within list number <i>listno</i> .

Note:

The PadInfoComb() function will always return 0 for a controller other than a DUAL SHOCK controller. Even for a DUAL SHOCK controller, the return value will be 0 as long as the state returned from PadGetState() is anything other than PadStateStable (during PadStateReqInfo). When obtaining information about combinations of actuators that can be used simultaneously, the application should wait for the return value from PadGetState() to become PadStateStable.

Remarks

12-18 Controller/Peripherals Library Functions

See also: PadInfoAct(), PadInfoMode()

PadInfoMode

Obtains information about the controller mode.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
int PadInfoMode (
int port,
int term,
int offs
)
```

Arguments

port Port number of the controller to be checked

	Port 1	Port 2
Direct connection	0x00	0x10
Multi-tap A	0x00	0x10
Multi-tap B	0x01	0x11
Multi-tap C	0x02	0x12
Multi-tap D	0x03	0x13

term Item to be checked

offs The offset in the controller mode ID table containing the desired controller mode ID

Explanation

Checks the currently active controller mode ID, distinguishes DUAL SHOCK controllers from other controllers, and checks the controller mode ID supported by the DUAL SHOCK controller.

Return value

The return value corresponds to the second and third arguments(*) in the following manner.

2 nd Argument	Macro	Return value
1	InfoModeCurID	Currently active controller mode ID. Valid range: 4 bits. (Same as value of terminal type for offset 1 in receive buffer)
2	InfoModeCurExID	Currently active controller mode ID. Valid range: 16 bits. (0 for controllers other than DUAL SHOCK)
3	InfoModeCurExOffs	Offset within the controller mode ID table that stores the currently active controller mode ID.
4	InfoModeldTable	Controller mode ID stored at the offset specified by the third argument <i>offs</i> within the controller mode ID table.

(*): If the second argument has a value other than 4(InfoModeldTable) the third argument *offs* is ignored.

When the second argument is 1 (InfoModeCurID) or 2 (InfoModeCurExID), the function may be called at any time, regardless of the value returned by PadGetState().

When the second argument is 4(InfoModeldTable), the return value will be 0 if PadGetState() does not return PadStateStable (for PadStateReqInfo). This is true even if the controller is a DUAL SHOCK controller.

After calling `PadLoadInfo()`, the application should wait for the return value from `PadGetState()` to become `PadStateStable`.

Remarks

When DUAL SHOCK controller SCPH-1200 is connected and `PadLoadInfo()` is called, initialization will complete and the return value will be as shown below.

Controller information and contents immediately after initialization:

Currently active controller mode ID	4
Currently active controller mode ID (DUAL SHOCK)	4
Controller mode ID table offset for currently active controller mode ID	0
Contents of controller mode ID table	See table below

Contents of controller mode table ID:

Controller mode ID table offset	Controller mode ID
0	4
1	7

See also: `PadInfoAct()`, `PadInfoComb()`

PadInit

Initializes a controller.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.0	02/15/98

Syntax

```
void PadInit(mode)
```

Arguments

mode Controller type

Explanation

This function initializes all connected controllers of the type specified by the *mode* parameter

Return value

None

Remarks

At present, only type 0 controllers are supported.

This function is for prototyping purposes only.

See also:

PadInitDirect

Initializes the controller environment (for direct connection to the main PlayStation unit).

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
void PadInitDirect (
  unsigned char *pad1,
  unsigned char *pad2
)
```

Arguments

pad1 Port 1 receive results (34 bytes)
pad2 Port 2 receive results (34 bytes)

Explanation

Initializes the control environment for a controller.

Note that the controller connected to port A of the multi-tap will not be recognized as a multi-tap even if a multi-tap is connected. Instead, it will be considered as being directly connected to the main PlayStation unit (controllers connected to ports B - D will be ignored). If a multi-tap is not used, using this function for initialization will reduce program size by about 1.6KB.

Return value

None

Remarks

PadInitMtap() of libpad.lib and InitPAD(), InitGUN(), InitTAP(), and PadInit() of libapi.lib cannot be used together.

In libpad, controller connection state is maintained by the library. If the connection state is invalid, the controller will not be recognized. Therefore, when a controller is used by both parent and child processes, each process must call PadInitDirect().

The standard data format stored in the receive buffer is as follows.

Receive buffer data format:

Offset	Contents
0	Receive results 0x00: successful, other values: failed
1	Upper four bits: terminal type Lower four bits: number of data bytes received / 2
2	Receive data: button state (high-order byte)
3	Receive data: button state (low-order byte)
4	Receive data: analog channel value 1
5	Receive data: analog channel value 2
6	Receive data: analog channel value 3
7	Receive data: analog channel value 4
...	Receive data: ...

Terminal types:

Terminal type	Controller name	Main controller part number
1	Mouse	SCPH-1030
2	16-button analog	SLPH-00001 (Namco Ltd.)
3	Gun controller	SLPH-00014 (Konami Ltd.)
4	16-button	SCPH-1080,1150,1200
5	Analog joystick	SCPH-1110
6	Gun controller	SLPH-00034 (Namco Ltd.)
7	Analog controller	SCPH-1150,1200

Button state bit assignments:

Bit	D15	D14	D13	D12	D11	D10	D9	D8
16-button	←	↓	→	↑	ST			SEL
Analog controller	←	↓	→	↑	ST	R3	L3	SEL
Analog joystick	←	↓	→	↑	ST			SEL
16-button analog	←	↓	→	↑	ST			
Mouse								
Gun controller (Konami)					ST			
Gun controller (Namco)					A			

Bit	D7	D6	D5	D4	D3	D2	D1	D0
16-button	□	X	O	△	R1	L1	R2	L2
Analog controller	□	X	O	△	R1	L1	R2	L2
Analog joystick	□	X	O	△	R1	L1	R2	L2
16-button analog			A	B	R			
Mouse					Left	Right		
Gun controller (Konami)	TRG	O						
Gun controller (Namco)		B	TRG					

Meaning of analog values:

	Analog value 1	Analog value 2	Analog value 3	Analog value 4
Analog controller	Position along the X axis (right) (0 ~ 80 ~ FF)	Position along the Y axis (right) (0 ~ 80 ~ FF)	Position along the X axis (left) (0 ~ 80 ~ FF)	Position along the Y axis (left) (0 ~ 80 ~ FF)
Analog joystick	Position along the X axis (right) (0 ~ 80 ~ FF)	Position along the Y axis (right) (0 ~ 80 ~ FF)	Position along the X axis (left) (0 ~ 80 ~ FF)	Position along the Y axis (left) (0 ~ 80 ~ FF)
Gun controller (Namco)	Position along the X axis Low-order byte	Position along the X axis High-order byte	Position along the Y axis Low-order byte	Position along the Y axis High-order byte
Mouse	Displacement along the X axis (80 ~ 0 ~ 7F)	Displacement along the Y axis (80 ~ 0 ~ 7F)	None	None

See also: PadInitMtap(), PadInitGun(), PadStartCom(), PadStopCom()

PadInitGun

Initializes the controller environment (for guns that use interrupts).

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
void PadInitGun (
  unsigned char *buff,
  int size
)
```

Arguments

buff Horizontal/vertical position receive buffer (required buffer size = size*4+2 bytes)
size Maximum number of gun interrupts for 1Vsync (maximum 20)

Explanation

Sets up the horizontal/vertical position receive buffer. Retrieval of the horizontal and vertical positions is triggered by an interrupt from the gun.

In order to improve the accuracy of the gun, interrupts and DMAs are blocked within the interrupt handler. Setting a large number of interrupts per 1Vsync consumes 1Hsync of time for each interrupt, so this value should be set low.

Structure of horizontal/vertical position receive buffer:

Offset	Contents
0	Port number for retrieved horizontal/vertical positions
1	Number of valid horizontal and vertical counters
2,3	Vertical counter value 0
4,5	Horizontal counter value 0
6,7	Vertical counter value 1
8,9	Horizontal counter value 1
.	.
.	.
.	.
78,79	Vertical counter value 19
80,81	Horizontal counter value 19

(Counter values are half-words (LSB first))

The horizontal counter value returns the system clock value. The pixel value can be obtained by multiplying by the coefficient corresponding to the horizontal resolution shown in the table below.

System clock - pixel clock conversion table:

Mode	Horizontal resolution	Coefficient
NTSC:	256	0.158532
	320	0.198166
	384	0.226475
	512	0.317065
	640	0.396332
PAL:	256	0.157086
	320	0.196358
	384	0.224409
	512	0.314173
	640	0.392717

[Pixel value] = [Coefficient] x [System clock value] + [Offset]

Horizontal/vertical gun positions are fetched from ports to which a terminal type=3 controller is connected and for guns whose interrupts have been enabled by PadEnableGun().

Gun horizontal/vertical positions can be fetched during each frame, in sequence, beginning with the smallest port number for those ports with guns which are interrupt-enabled.

Check offset 0 in the horizontal/vertical position receive buffer ("Port number for retrieved horizontal/vertical positions") to determine the port number associated with the retrieved horizontal/vertical position.

PadInitGun() is provided only to initialize the gun interrupt environment. In order to communicate with a gun controller, PadInitDirect() or PadInitMtap() must be called first.

Return value

None

Remarks

In libpad, gun connection state is maintained by the library. If the connection state is invalid, gun position information cannot be obtained. Therefore, as an example, when both parent and child processes use an ID=3 gun, each process must call PadInitGun().

See also: PadRemoveGun(), PadEnableGun(), PadInitDirect(), PadInitMtap()

PadInitMtap

Initializes controller environment (for multi-taps).

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
void PadInitMtap (
  unsigned char *pad1,
  unsigned char *pad2
)
```

Arguments

pad1 Port 1 receive result (34 bytes)
pad2 Port 2 receive result (34 bytes)

Explanation

Initializes the control environment for a controller. If a multi-tap is connected, it is treated as a multi-tap. If a controller is connected directly to the main PlayStation unit, the structure of the receive buffer will be the same as when it is initialized with PadInitDirect().

Return value

None

Remarks

PadInitDirect() of libpad.lib and InitPAD(), InitGUN(), InitTAP(), and PadInit() of libapi.lib cannot be used together.

In libpad, controller connection state is maintained by the library. If the connection state is invalid, the controller cannot be recognized. Therefore, when a controller is used by both parent and child processes, each process must call PadInitMtap().

The format of the data stored in the receive buffer is shown below.

Receive buffer structure (when the controller is connected directly to the main PlayStation unit):

Offset	Contents
0	Result from receive operation 0x00: successful, other values: failed
1	High-order 4 bits: terminal type Low-order 4 bits: Byte count of received data / 2
2	Receive data: button state (high-order byte)
3	Receive data: button state (low-order byte)
4	Receive data: analog channel value 1
5	Receive data: analog channel value 2
6	Receive data: analog channel value 3
7	Receive data: analog channel value 4
...	Receive data:...

Receive buffer structure (when the controller is connected via a multi-tap):

Offset		Contents
0		Results from receive operation 0x00: successful, other values: failed
1		0x80
2	Port A	Results from receive operation 0x00: successful, other values: failed
3		High-order 4 bits: terminal type Low-order 4 bits: Byte count of received data / 2
4 ~ 9		Receive data
10	Port B	Results from receive operation 0x00: successful, other values: failed
11		High-order 4 bits: terminal type Low-order 4 bits: Byte count of received data / 2
12 ~ 17		Receive data
18	Port C	Results from receive operation 0x00: successful, other values: failed
19		High-order 4 bits: terminal type Low-order 4 bits: Byte count of received data / 2
20 ~ 25		Receive data
26	Port D	Results from receive operation 0x00: successful, other values: failed
27		High-order 4 bits: terminal type Low-order 4 bits: Byte count of received data / 2
28 ~ 33		Receive data

Notes:

A multi-tap may not be recognized if a controller is not connected to port A of the multi-tap. Therefore, a controller should always be connected to port A of the multi-tap. This should also be mentioned in the instruction manual.

See also: PadInitDirect(), PadInitGun(), PadStartCom(), PadStopCom()

PadRead

Read data from the controller.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.0	02/15/98

Syntax

```
unsigned long PadRead(id)
unsigned short id;
```

Argument

id Controller ID

Explanation

This function reads data from the controller specified by the *id* parameter.

Return value

The return value is controller button status.

Remarks

Currently, *id* has no meaning.

This function is for prototyping purposes only.

See also:

PadRemoveGun

Stops retrieval of horizontal/vertical gun position.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
void PadRemoveGun(void)
```

Arguments

None

Explanation

Stops the retrieval of horizontal/vertical gun position information.

Return value

None

Remarks

See also: PadInitGun(), PadEnableGun()

PadSetAct

Sets transmit buffer.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	6/22/98

Syntax

```
void PadSetAct (
int port,
unsigned char *data,
int len
)
```

Arguments

port Target port number

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

data Transmit data buffer

len Length of transmit data buffer (in bytes)

Explanation

The function PadSetAct() registers the transmit data buffer in the library, so it is not necessary to call this function again if the transmit buffer doesn't change. When the operation of the actuator changes and the contents of the buffer change, the library reads out the buffer every Vsync and automatically transmits the contents to the controller.

When controlling the DUAL SHOCK actuator, not only must the transmit buffer be specified using PadSetAct(), but PadSetActAlign() must also be used to inform the controller of the offset in the transmit buffer where the actuator parameters are located. (The calling sequence of PadSetActAlign() and PadSetAct() is not specified.)

The data length that can be handled by the actuator can be determined with PadInfoAct(). For the SCPH-1200, the data lengths are 1 bit and 1 byte for actuator numbers 0 and 1, respectively. Thus, the actuator can be controlled by using PadSetActAlign() to send the parameter for actuator number 0 at transmit buffer offset 0, and the parameter for actuator number 1 at offset 1. In this case, offset 0 would contain a value of 0 or 1, and offset 1 would contain a value between 0 and 255.

The actuator will be stopped when the parameter value is 0, and will rotate faster for larger values.

Once the actuator parameters have been sent to the controller during a vertical retrace interrupt, the actuator will continue operating even if communication with the controller is suspended by PadEnableCom() or PadStopCom(). However, if communication is suspended for three seconds or more, the controller will be reset, at which point the actuator will stop operating. Even if the interval during which communication is suspended is less than three seconds, the actuator will temporarily stop operating when PadStartCom() reinitiates communication (if communication was suspended with PadStopCom(), it can only be restarted by calling PadStartCom()).

If communication is suspended for three seconds, or if the actuator is halted due to a PadStartCom() read, the value returned from PadState() will temporarily be PadStateDiscon and a retry will be generated for

refetching controller information. For this reason, the return value from `PadGetState()` must be monitored so that refetched actuator information can be properly processed.

If the parameter data length for an actuator is more than one byte, each of the parameter write offsets for that actuator can be set by writing the actuator number in each of the corresponding offsets using `PadSetActAlign()`. The controller will interpret the position of the lowest numbered offset as the high-order byte of the parameter. If the actuator number is written such that the data length for the parameter is exceeded, the settings beyond the allowed parameter data length will be ignored, beginning with the lowest numbered offset.

Return value

None

Remarks

For a DUAL SHOCK controller, the offsets in the transmit buffer where actuator parameters are written can be specified with `PadSetActAlign()`. However, with analog controller SCPH-1150, there is a pre-determined method for setting up the transmit buffer. The method for setting up the transmit buffer and relevant points to be observed are described below.

Offset	Contents
0: target device ID	High-order 2 bits: 0x01 Low-order 6 bits: undefined (vibration device)
1: transmit data	bit 0: 1 = vibration ON, 0 = vibration OFF remaining bits (bit 7 ~ 1): undefined
2... : transmit data	Always 0x00. Other values: undefined

Note:

The actuator can be turned on only during the 1 Vsync interval before communication with the next controller takes place. During the interval in which the actuator is to be operated, the target device ID should be entered in the transmit data buffer and vibrations should be set to ON at each vertical sync interrupt. The target device ID is valid when the two high-order bits are set to 01. The remaining bits are reserved for the system and should be set to 0. Vibrations are set to ON when the low-order bit of the first transmit data byte is set to 1. The remaining bits should be set to 0 as with the target device ID.

See also: `PadSetActAlign()`

PadSetActAlign

Sets actuator parameter details to be sent to the controller.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
int PadSetActAlign (
int port,
char *data
)
```

Arguments

port Port number of the controller to which the actuator parameter details are to be sent

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

data Actuator parameter transmission details (6 bytes)

Explanation

The position in the transmit buffer where the actuator parameters are located is indicated to the controller by writing the actuator numbers in the appropriate positions in the 6-byte array.

In the table shown below, offset 0 of the transmit buffer is used for actuator number 0, and offset 1 is used for actuator number 1. The remaining offsets are not used. (The actuator number is entered at positions where transmission is desired, and FF is entered at unused positions.)

Offset	0	1	2	3	4	5
Contents	00	01	FF	FF	FF	FF

Return value

A 1 is returned if the actuator parameter details request is accepted. 0 is returned if the request is not accepted.

Remarks

This function will not accept requests if the library is communicating with the controller. The return value should be checked to see if the request was accepted. The request will be accepted if PadGetState() returns a PadStateStable, so the value from PadGetState() can be checked to confirm that the request was accepted. However, when PadSetActAlign() and PadSetMainMode() are called, the result from PadGetState() changes immediately from PadStateStable to PadStateExecCmd, and PadSetActAlign() and PadSetMainMode() calls will not be accepted until three vertical sync interrupts (six for multi-taps) have elapsed. Thus, these two functions cannot be called one after the other. If PadState() is called instead of checking the return value, PadGetState() should be called right before calling the functions to confirm that the return value is PadStateStable.

See also: PadGetState(), PadSetAct()

PadSetMainMode

Switches / locks the controller mode selector.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	6/22/98

Syntax

```
int PadSetMainMode (  
int port,  
int offs,  
int lock  
)
```

Arguments

port Port number for which the controller mode is to be switched

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

offs The controller mode ID table offset which contains the controller mode to be switched
lock If bit 1 is set to 0, the locked/unlocked state of the selector button is kept in its current state.
If bit 1 is set to 1 and bit 0 is set to 0, the selector button is unlocked.
If bit 1 is 1 and bit 0 is 1, the selector button is locked.

Explanation

Selects the controller mode and switches between locked and unlocked settings for the controller mode selection button on the main controller unit. When this function is called and the controller mode is changed, controller information is retrieved. Therefore, the value returned from PadGetState() needs to be monitored so that actuator information can be properly refetched.

Return value

1 when a controller mode setting request was accepted. 0 if the request was not accepted.

Remarks

This function will not accept requests if the library is communicating with the controller. The return value should be checked to see if the request was accepted. The request will be accepted if PadGetState() returns a PadStateStable, so the value from PadGetState() can be checked to confirm that the request was accepted. However, when PadSetMainMode() or PadSetActAlign() is called, the result from PadGetState() changes immediately from PadStateStable to PadStateExecCmd, and PadSetActAlign() and PadSetMainMode() will not accept requests. Therefore, these two functions cannot be called one after the other. When PadState() is checked instead of the return value, PadGetState() must be checked right before calling the functions.

See also: PadGetState()

PadStartCom

Starts reading from controller.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
void PadStartCom(void)
```

Arguments

None

Explanation

Initiates a controller read operation triggered by a vertical retrace interval interrupt.

Return value

None

Remarks

See also: PadInitDirect(), PadInitMtap(), PadStopCom(), PadEnableCom()

PadStop

Halts controller.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	2.x	02/15/98

Syntax

void PadStop(*void*)

Argument

None

Explanation

Halts all currently connected controllers.

Return value

None

Remarks

When processing is complete, it is necessary to call this function without fail and halt the controller driver. This function is for prototyping purposes only.

See also:

PadStopCom

Stops controller read.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	02/15/98

Syntax

```
void PadStopCom(void)
```

Arguments

None

Explanation

Stops a controller read operation.

(Stops handling all vertical interval interrupts related to controller services.)

Return value

None

Remarks

See also: PadInitDirect(), PadInitMtap(), PadStartCom(), PadEnableCom()

RemoveGUN

Removes gun.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.6	5/22/97

Syntax

void RemoveGUN(*void*)

Arguments

None

Explanation

This function removes the gun driver registered in InitGUN().

Return value

Remarks

See also: InitGUN(), StartGUN(), StopGUN(), SelectGUN(), RemoveGUN().

ResetCallback

Initializes all callbacks.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.0	7/31/96

Syntax

```
void ResetCallback(void)
```

Arguments

None

Explanation

Initializes all system callbacks. Sets all callback functions to 0 (unregistered), and after securing the interrupt context stack, sets up the environment for accepting interrupts.

Return value

None

Remarks

ResetCallback() must be called after program boot, before any other processing is performed.

The environment initialized by ResetCallback() will remain valid until StopCallback() is called.

It is acceptable to continuously call ResetCallback() without StopCallback(). However, the second and subsequent calls will be ignored.

See also:

ResetGUN

Resets the light gun handler.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.5	7/31/96

Syntax

void RESETGUN(*void*)

Arguments

None

Explanation

Resets the light gun handler and disables related interrupts.

Return value

Remarks

See also:

RestartCallback

Restarts halted call-back.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.2	7/31/96

Syntax

```
int RestartCallback(void)
```

Arguments

None

Explanation

Restores the halted call-back to the status immediately prior to when it was halted.

Differs from ResetCallback () in that the call-back functions and call-back stack are not initialized.

Return value

None

Remarks

ResetCallback () must be executed before executing RestartCallBack ().

The environment initialized by RestartCallback () is valid until StopCallback () is called.

There is no problem even if RestartCallback () is successively called without inserting StopCallback (), but calls from the second one onwards will be ignored.

See also:

SelectGUN

Selects gun.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.5	5/22/97

Syntax

```
void SelectGUN (
  int ch,
  unsigned char mask
)
```

Arguments

ch Gun channel (0 or 1)
mask Interruption mask setting
 0: interruption prohibited
 1: interruption permitted

Explanation

Reports the on/off of the interruption mask for the gun.

It is not possible to cancel more than two masks at the same time.

Return value

Remarks

See also: InitGUN(), StartGUN(), StopGUN(), RemoveGUN().

SendTAP

Initializes controller.

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.7	5/22/97

Syntax

```
void SendTAP (
char *bufA,
long lenA,
char *bufB,
long lenB
)
```

Arguments

bufA, *bufB* Transmission data buffer
lenA, *lenB* Transmission data buffer length (Unit= bytes)

Explanation

This is an initialization function which registers the controller transmission data buffer.

The target device ID and data set in the buffer are transmitted when communicating with each vertical synchronization interrupt controller.

Byte	Contents
0	Flag which specifies whether buffer data is valid or invalid 0x00 is invalid, 0x01 is valid. Operation with other values is undefined.
1	Target device ID (Uses upper level 2 bits, other bits are system reserved)
2~6	Transmission data (data length is the same as received data length)
7	Controller B Target Device ID
8~12	Transmission data to Controller B
13	Controller C Target Device ID
14~18	Transmission data to Controller C
19	Controller D Target Device ID
20~24	Transmission data to Controller D

Return value

None

Remarks

The buffer settings and Notes regarding the new version of the analog joystick equipped with a vibration device are as follows:

Byte	Contents
Target device ID	Upper-level 2 bit: 0x01 Lower-level 6 bit: Operation undefined (vibration device)
Transmission data (first byte)	bit0:1 = vibration ON, 0 = vibration OFF Others (bit 7~1): Operation undefined
Transmission data (second byte onwards)	Always 0x00. Others: Operation undefined

Notes:

Communication with the controller is carried out when the vibration device is set to On and is effective only until the 1 vertical synchronization period when communication with the controller is carried out next. When the device is in operation set the target device ID and vibration on the transmission data buffer of each vertical synchronization interrupt to “on” and continue to place the buffer in an effective state. Although the target device ID is effective when the upper two bits are 01, other bits are system reserved so should all be 0. Transmission data is effective when both the first byte and the upper 1 bit are 1, the other bits should be 0.

SetVideoMode

Declares current video signaling system.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.1	7/31/96

Syntax

```
long SetVideoMode (
long mode
)
```

Arguments

mode Video signaling system mode

Explanation

Declares the video signaling system indicated by mode to the libraries.

Table 12-7

Mode	Contents
MODE_NTSC	NTSC system video signaling system
MODE_PAL	PAL system video signaling system

Related libraries will be able to conform to the actions of the declared video signaling system environment.

Return value

Previously-set video signaling system mode.

Remarks

Gets called in advance of all library functions.

See also:

StartGUN

Starts controller reading.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.6	5/22/97

Syntax

long StartGUN(*void*)

Arguments

None

Explanation

Starts controller reading with vertical return section interruption carried out as trigger.

Return value

When successful, returns 1. When fails, returns 0

Remarks

Returns in interruption-approved state

See also: InitGUN(), StopGUN(), SelectGUN(), RemoveGUN()

StartTAP

Starts controller reading.

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.4	6/22/98

Syntax

void StartTAP(*void*)

Arguments

None

Explanation

Starts controller reading with vertical return section interruption carried out as trigger.

Return value

None

Remarks

Returns in interruption-approved state

See also: InitTAP (p. 12-9).

StopCallback

Stops all callbacks.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.0	7/31/96

Syntax

void StopCallback(*void*)

Arguments

None

Explanation

Stops all system callbacks.

Return value

None

Remarks

Before terminating programs, StopCallback() must be called to disable all interrupts.

See also:

StopGUN

Halts controller reading.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.6	5/22/97

Syntax

```
void StopGUN(void)
```

Arguments

None

Explanation

Halts the controller reading. Does not prohibit interruption.

Return value

None

Remarks

See also: InitTAP(), StartGUN(), SelectGUN(), RemoveGUN()

StopTAP

Halts controller reading.

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.4	7/31/96

Syntax

void StopTAP(*void*)

Arguments

None

Explanation

Halts the controller reading. Does not prohibit interruption.

Return value

None

Remarks

See also: InitTAP (p. 12-9).

Chapter 13: Link Cable Library

Table of Contents

Functions	
_comb_control	13-3
AddCOMB	13-4
ChangeClearSIO	13-5
DelCOMB	13-6
Macros	
CombAsyncRequest	13-7
CombBytesRemaining	13-8
CombBytesToRead	13-9
CombBytesToWrite	13-10
CombCancelRead	13-11
CombCancelWrite	13-12
CombControlStatus	13-13
CombCTS	13-14
CombGetBPS	13-15
CombGetMode	13-16
CombGetPacketSize	13-17
CombReset	13-18
CombResetError	13-19
CombResetVBLANK	13-20
CombSetBPS	13-21
CombSetControl	13-22
CombSetMode	13-23
CombSetPacketSize	13-24
CombSetRTS	13-25
CombSioStatus	13-26
CombWaitCallback	13-27

`_comb_control`

Link cable driver control.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	3.0	7/31/96

Syntax

```
long _comb_control (
unsigned long cmd
unsigned long arg
unsigned long param
)
```

Arguments

cmd Command
arg Subcommand
param Argument

Explanation

Offers the same functionality as `ioctl()` to an SIO device.

Return value

The return value depends on the control command used in *cmd*.

Remarks

See also:

AddCOMB

Initializes link cable driver.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	3.0	7/31/96

Syntax

void AddCOMB(*void*)

Arguments

None

Explanation

Initializes link cable driver.

Return value

None

Remarks

See also:

ChangeClearSIO

Clears interrupt from expanded SIO in the driver.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	3.0	7/31/96

Syntax

```
void ChangeClearSIO (
long val
)
```

Arguments

val Interrupt cause clear flag

Explanation

If *val* is set as non-0, an interrupt from an expansion SIO in the driver is cleared. This is used only when other expansion SIO drivers are also present.

Return value

None

Remarks

See also:

DelCOMB

Removes link cable driver from kernel.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	3.0	7/31/96

Syntax

void DelCOMB(*void*)

Arguments

None

Explanation

Removes link cable driver from kernel.

Return value

None

Remarks

See also:

CombAsyncRequest

Gets asynchronous communication request status.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

```
long CombAsyncRequest (
long param
)
```

Arguments

param 0: asynchronous write, 1: asynchronous read

Explanation

Determines whether an asynchronous input/output request has been made.

If *param* is 0, gets asynchronous write status.

If *param* is 1, gets asynchronous read status.

Return value

If request has been made, returns 1. Otherwise, returns 0.

Remarks

Equivalent to `_comb_control (0, 6, param)`.

A combat cable command macro.

See also:

CombBytesRemaining

Gets remaining transmit or receive data.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

```
long CombBytesRemaing (
long param
)
```

Arguments

param 0: asynchronous write, 1: asynchronous read

Explanation

Gets the remaining data count from the asynchronous read or the asynchronous write being processed.

If *param* is 0, get bytes remaining for asynchronous write.

If *param* is 1, get bytes remaining for asynchronous read.

Return value

Returns the number of bytes remaining.

Remarks

Equivalent to `_comb_control (0, 5, param)`.

A combat cable command macro.

See also:

CombBytesToRead

Gets number of bytes left to receive.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombBytesToRead(*void*)

Arguments

None

Explanation

Obtains the number of bytes left in the current asynchronous read operation.

Return value

Returns the number of bytes remaining.

Remarks

Equivalent to `_comb_control (0, 5, 1)`.

A combat cable command macro.

See also:

CombBytesToWrite

Gets number of bytes left to send.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombBytesToWrite(*void*)

Arguments

None

Explanation

Obtains the number of bytes remaining in the current asynchronous write operation.

Return value

Returns the number of bytes remaining.

Remarks

Equivalent to `_comb_control (0, 5, 0)`.

A combat cable command macro.

See also:

CombCancelRead

Cancels asynchronous read.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombCancelRead(*void*)

Arguments

None

Explanation

Cancels current asynchronous read operation.

Return value

Always returns 0.

Remarks

Equivalent to `_comb_control (2, 3, 0)`.

A combat cable command macro.

See also:

CombCancelWrite

Cancels asynchronous write.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombCancelWrite(*void*)

Arguments

None

Explanation

Cancels current asynchronous write operation.

Return value

Always returns 0.

Remarks

Equivalent to `_comb_control (2, 2, 0)`.

A combat cable command macro.

See also:

CombControlStatus

Gets control line status.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombControlStatus(*void*)

Arguments

None

Explanation

Obtains the control line status.

Return value

Returns the control line status.

Bit fields are as follows.

Table 13-1: Control Line Status

bit	Contents
31-2	undefined
1	1: RTS on
0	1: DTR on

Remarks

Equivalent to `_comb_control (0, 1, 0)`.

A combat cable command macro.

See also:

CombCTS

Gets status of CTS signal.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombCTS(*void*)

Arguments

None

Explanation

Obtains the state of the serial controller CTS bit.

Return value

Returns a 1 if CTS is 1. Otherwise, returns 0.

Remarks

Equivalent to `_comb_control (3, 1, 0)`.

A combat cable command macro.

See also:

CombGetBPS

Gets communication speed.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombGetBPS(*void*)

Arguments

None

Explanation

Obtains the communication speed (in bps).

Return value

Returns the communication speed (in bps).

Remarks

Equivalent to `_comb_control (0, 3, 0)`.

A combat cable command macro.

See also:

CombGetMode

Gets communication mode.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombGetMode(*void*)

Arguments

None

Explanation

Obtains the communication mode.

Return value

Returns the communication mode.

Table 13-2: Communication Mode

bit	Contents
31-8	undefined
7,6	stop bit length 01: 1 10: 1.5 11: 2
5	parity2 1:odd 0:even
4	parity1 1:enabled
3,2	character length 00: 5 bits 01: 6 10: 7 11: 8
1	always 1
0	always 0

Remarks

Equivalent to `_comb_control (0, 2, 0)`.

A combat cable command macro.

See also:

CombGetPacketSize

Gets receive packet size.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombGetPacketSize(*void*)

Arguments

None

Explanation

Obtains the receive packet size.

Return value

Returns the receive packet size.

Remarks

Equivalent to `_comb_control (0, 4, 0)`.

A combat cable command macro.

See also:

CombReset

Initializes the serial controller.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombReset(*void*)

Arguments

None

Explanation

Initializes the serial controller.

Controller status, communication mode and communication speed remain unchanged.

Return value

Always returns 0.

Remarks

Equivalent to `_comb_control (2, 0, 0)`.

A combat cable command macro.

See also:

CombResetError

Initializes error flags.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombResetError(*void*)

Arguments

None

Explanation

Clears error-related bits from driver status.

Return value

Always returns 0.

Remarks

Equivalent to `_comb_control (2, 1, 0)`.

A combat cable command macro.

See also:

CombResetVBLANK

Resets vertical blanking signal.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombResetVBLANK(*void*)

Arguments

None

Explanation

Resets the vertical blanking signal.

Return value

Always returns 0.

Remarks

Equivalent to `_comb_control (5, 0, 0)`.

A combat cable command macro.

See also:

CombSetBPS

Sets communication speed.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

```
long CombSetBPS (
long bps
)
```

Arguments

bps Communication speed (in bps)

Explanation

Sets the communication speed.

bps must be in the range 9600 - 2073600 and evenly divisible into 2073600.

If asynchronous write is used, the maximum communication speed is 57600 bps.

Return value

Always returns 0.

Remarks

Equivalent to `_comb_control (1, 3, bps)`.

A combat cable command macro.

See also:

CombSetControl

Sets control line status.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

```
long CombSetControl (
long val
)
```

Arguments

val Control line status

Explanation

Sets the control line status.

Table 13-3: Control Line Status

bit	Contents
31-2	unused
1	1: RTS on
0	1: DTR on

Return value

Always returns 0.

Remarks

Equivalent to `_comb_control (1, 1, val)`.

A combat cable command macro.

See also:

CombSetMode

Sets communication mode.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

```
long CombSetMode (
long mode
)
```

Arguments

mode Communication mode

Explanation

Sets the communication mode.

Return value

Always returns 0.

Table 13-4: Communication Mode

bit	Contents
31-8	Unused
7,6	stop bit length 01: 1 10: 1.5 11: 2
5	Parity2 1: odd 0: even
4	Parity1 1: enabled
3,2	Character length 00: 5 bits 01: 6 10: 7 11: 8
1	Always 1
0	Always 0

Remarks

Equivalent to `_comb_control (1, 2, mode)`.

A combat cable command macro.

See also:

CombSetPacketSize

Sets receive packet size.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

```
long CombSetPacketSize (
long size
)
```

Arguments

size Packet size (1, 2, 4, or 8)

Explanation

Sets receive packet size.

The receive packet size sets the byte count used for generating interrupts in asynchronous communication.

For example, if the receive packet size were set to 4, the serial controller would generate an interrupt after every four bytes of data received. A large packet size will lower the frequency of interrupts, thus improving overall system performance. However, if asynchronous transmission is being performed, the receive packet size must be set to 1.

Return value

Always returns 0.

Remarks

Equivalent to `_comb_control (1, 4, size)`.

A combat cable command macro.

See also:

CombSetRTS

Sets RTS signal.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombSetRTS(*void*)

Arguments

None

Explanation

Sets the RTS bit in control line status to 1.

Return value

Always returns 0.

Remarks

Equivalent to `_comb_control (3, 0, 1)`.

A combat cable command macro.

See also:

CombSioStatus

Gets serial controller status.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

long CombSioStatus(*void*)

Arguments

None

Explanation

Obtains serial controller status.

Return value

Returns the serial controller status.

Bit fields are as shown below.

Table 13-5: Serial Controller Status

Bit	Contents
31-10	undefined
9	1: interrupts on
8	1: CTS is on
7	1: DSR is on
6	undefined
5	1:frame error generated
4	1:overflow error generated
3	1:parity error generated
2	1:no data to transmit
1	1:receive data available
0	1:transmit data available

Remarks

Equivalent to `_comb_control (0, 0, 0)`.

A combat cable command macro.

See also:

CombWaitCallback

Sets wait callback function.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	02/15/98

Syntax

```
long CombWaitCallback (
long func
)
```

Arguments

func Pointer to the wait callback function

Explanation

The value of *func* is entered as a pointer to the wait callback function.

Return value

Returns the value of the previous callback function.

Remarks

Equivalent to the `_comb_control (4, 0, func)`.

A combat cable command macro.

See also:

Chapter 14: Extended Sound Library

Table of Contents

Structures	
ProgAtr	14-5
SndRegisterAttr	14-6
SndVoiceStats	14-7
SndVolume	14-8
SndVolume2	14-9
VabHdr	14-10
VagAtr	14-11
_SsFCALL	14-13
Functions	
dmy_Ss...	14-15
SsAllocateVoices	14-16
SsBlockVoiceAllocation	14-17
SsChannelMute	14-18
SsEnd	14-19
SsGetActualProgFromProg	14-20
SsGetChannelMute	14-21
SsGetCurrentPoint	14-22
SsGetMute	14-23
SsGetMVol	14-24
SsGetNck	14-25
SsGetRVol	14-26
SsGetSerialAttr	14-27
SsGetSerialVol	14-28
SsGetVoiceMask	14-29
SsInit	14-30
SsInitHot	14-31
SsIsEos	14-32
SsPitchFromNote	14-33
SsPlayBack	14-34
SsQueueKeyOn	14-35
SsQueueRegisters	14-36
SsQueueReverb	14-38
SsQuit	14-39
SsSepClose	14-40
SsSepOpen	14-41
SsSepOpenJ	14-42
SsSepPause	14-43
SsSepPlay	14-44
SsSepReplay	14-45
SsSepSetAccelerando	14-46
SsSepSetCrescendo	14-47
SsSepSetDecrescendo	14-48
SsSepSetRitardando	14-49
SsSepSetVol	14-50
SsSepStop	14-51
SsSeqCalledTbyT	14-52
SsSeqClose	14-53
SsSeqGetVol	14-54
SsSeqOpen	14-55
SsSeqOpenJ	14-56
SsSeqPause	14-57
SsSeqPlay	14-58
SsSeqPlayPtoP	14-59

SsSeqReplay	14-60
SsSeqSetAccelerando	14-61
SsSeqSetCrescendo	14-62
SsSeqSetDecrescendo	14-63
SsSeqSetNext	14-64
SsSeqSetRitardando	14-65
SsSeqSetVol	14-66
SsSeqSkip	14-67
SsSeqStop	14-68
SsSetAutoKeyOffMode	14-69
SsSetCurrentPoint	14-70
SsSetLoop	14-71
SsSetMarkCallback	14-72
SsSetMono	14-73
SsSetMute	14-74
SsSetMVol	14-75
SsSetNck	14-76
SsSetNext	14-77
SsSetNoiseOff	14-78
SsSetNoiseOn	14-79
SsSetReservedVoice	14-80
SsSetRVol	14-81
SsSetSerialAttr	14-82
SsSetSerialVol	14-83
SsSetStereo	14-84
SsSetTableSize	14-85
SsSetTempo	14-86
SsSetTickCallBack	14-87
SsSetTickMode	14-88
SsSetVoiceMask	14-90
SsSetVoiceSettings	14-91
SsStart	14-93
SsStart2	14-94
SsUnBlockVoiceAllocation	14-95
SsUtAllKeyOff	14-96
SsUtAutoPan	14-97
SsUtAutoVol	14-98
SsUtChangeADSR	14-99
SsUtChangePitch	14-100
SsUtFlush	14-101
SsUtGetDetVVol	14-102
SsUtGetProgAtr	14-103
SsUtGetReverbType	14-104
SsUtGetVabHdr	14-105
SsUtGetVagAddr	14-106
SsUtGetVagAddrFromTone	14-107
SsUtGetVagAtr	14-108
SsUtGetVBaddrInSB	14-109
SsUtGetVVol	14-110
SsUtKeyOff	14-111
SsUtKeyOffV	14-112
SsUtKeyOn	14-113
SsUtKeyOnV	14-114
SsUtPitchBend	14-115
SsUtReverbOff	14-116
SsUtReverbOn	14-117
SsUtSetDetVVol	14-118

SsUtSetProgAtr	14-119
SsUtSetReverbDelay	14-120
SsUtSetReverbDepth	14-121
SsUtSetReverbFeedback	14-122
SsUtSetReverbType	14-123
SsUtSetVabHdr	14-124
SsUtSetVagAtr	14-125
SsUtSetVVol	14-126
SsVabClose	14-127
SsVabFakeBody	14-128
SsVabFakeHead	14-129
SsVabOpen	14-130
SsVabOpenHead	14-131
SsVabOpenHeadSticky	14-132
SsVabTransBody	14-134
SsVabTransBodyPartly	14-135
SsVabTransCompleted	14-136
SsVabTransfer	14-137
SsVoiceCheck	14-138
SsVoKeyOff	14-139
SsVoKeyOn	14-140

ProgAtr

Program header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	10/01/97

Structure

```
typedef struct ProgAtr {
    unsigned char tones;
    unsigned char mvol;
    unsigned char prior;
    unsigned char mode;
    unsigned char mpan;
    char reserved0;
    short attr;
    unsigned long reserved1;
    unsigned long reserved2;
};
```

Members

<i>tones</i>	Number of VAG attribute sets contained in the program
<i>mvol</i>	Master volume for the program
<i>prior</i>	Program priority (0-15)
<i>mode</i>	Sound source mode
<i>mpan</i>	Program pan
<i>reserved0</i>	Reserved by the system
<i>attr</i>	Program attribute
<i>reserved 1</i>	Reserved by the system
<i>reserved2</i>	Reserved by the system

Explanation

Remarks

See also:

SndRegisterAttr

SPU register attributes.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Structure

```
struct SndRegisterAttr {
    SndVolume2 volume;
    short pitch;
    short mask;
    short addr;
    short adsr1;
    short adsr2;
} SndRegisterAttr;
```

Members

<i>volume</i>	Volume data for left and right channels
<i>pitch</i>	Pitch rate at which to play back waveform data
<i>mask</i>	Bitfield designating which attributes to set
<i>addr</i>	Waveform data start address
<i>adsrc1</i>	Bitfield for setting adsrc information

ar_m

15	14	ar	9	8	dr	5	4	sl	0
----	----	----	---	---	----	---	---	----	---

ar_m: attack rate mode 0 = linear, 1 = exponential
ar: attack rate
dr: decay rate
sl: sustain level

adsr2 Bitfield for setting adsr information

sr_m sr_s

rr_m

15	14	13	12	sr	6	5	4	rr	0
----	----	----	----	----	---	---	---	----	---

sr_m:	sustain rate mode	0 = linear, 1 = exponential
sr_s:	sustain rate sign	0 = positive, 1 = negative
sr:	sustain rate	
rr_m:	release rate mode	0 = linear, 1 = exponential
rr:	release rate	
note: bit 13 is unused		

Explanation

This structure is used in the function `SsQueueRegisters()` to set Spu voice information.

Remarks

See also: SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsAllocateVoices (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsGetActualProgFromProg (p. n-nn), SsPitchFromNote (p. n-nn), SsQueueKeyOn (p. n-nn), SsQueueRegisters (p. n-nn), SsQueueReverb (p. n-nn), SsSetVoiceSettings (p. n-nn), SsUnBlockVoiceAllocation (p. n-nn), SsVoiceCheck (p. n-nn).

SndVoiceStats

Internal libsnd voice variables.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Structure

```
struct SndVoiceStats {
    short vagld;
    short vabld;
    unsigned short pitch;
    short vol;
    char pan;
    short note;
    short tone;
    short prog_num;
    short prog_actual;
} SndVoiceStats;
```

Members

<i>vagld</i>	VAG number pointed to by tone information (1-254)
<i>vabld</i>	VAB number containing tone information (0-15)
<i>pitch</i>	Playback rate of voice
<i>vol</i>	Volume of voice (0-127)
	Not valid for 3D sound input
<i>pan</i>	Voice pan (0-127; 0 = left, 64 = center, 127 = right)
	Not valid for 3D sound input
<i>note</i>	Note at which tone information keyed on
<i>tone</i>	Tone number (0-15)
<i>prog_num</i>	Program number containing tone information (0-127)
<i>prog_actual</i>	The "real" program number within which the tone information resides. The "real" number is only incremented by valid programs (programs containing one or more tones) and so may differ from the program number. Example: In a VAB with valid programs 0-10 and 100-127, the <i>prog_num</i> of program 127 = 127, while the <i>prog_actual</i> of program 127 = 38. Used to calculate offset in VAB header of tone information.

Explanation

This structure is used to fill the internal libsnd voice structures in the function `SsSetVoiceStats()`.

Remarks

See also: `SndRegisterAttr` (p. n-nn), `SndVolume2` (p. n-nn), `SsAllocateVoices` (p. n-nn), `SsBlockVoiceAllocation` (p. n-nn), `SsGetActualProgFromProg` (p. n-nn), `SsPitchFromNote` (p. n-nn), `SsQueueKeyOn` (p. n-nn), `SsQueueRegisters` (p. n-nn), `SsQueueReverb` (p. n-nn), `SsSetVoiceSettings` (p. n-nn), `SsUnBlockVoiceAllocation` (p. n-nn), `SsVoiceCheck` (p. n-nn).

SndVolume

Volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Structure

```
typedef struct SndVolume {
    unsigned short left;
    unsigned short right;
};
```

Members

left L channel volume value
right R channel volume value

Explanation

Remarks

See also:

SndVolume2

Volume-greater range.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Structure

```
struct SndVolume2 {
    short left;
    short right;
} SndVolume2;
```

Members

left Left volume value, -0x4000 ~ 0x3fff
right Right volume value, -0x4000 ~ 0x3fff

Explanation

This structure allows for a greater range of volume inputs, including negative volumes, when used with the libsnd keyon emulation series: SsBlockVoiceAllocation() -> SsAllocateVoices() -> SsSetVoiceStats() -> SsQueueRegisters() -> SsQueueKeyOn() -> SsQueueReverb() -> SsUnBlockVoiceAllocation().

Return value

Remarks

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SsAllocateVoices (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsGetActualProgFromProg (p. n-nn), SsPitchFromNote (p. n-nn), SsQueueKeyOn (p. n-nn), SsQueueRegisters (p. n-nn), SsQueueReverb (p. n-nn), SsSetVoiceSettings (p. n-nn), SsUnBlockVoiceAllocation (p. n-nn), SsVoiceCheck (p. n-nn).

VabHdr

Bank header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	10/01/97

Structure

```
typedef struct VabHdr {
    long form;
    long ver;
    long id;
    unsigned long fsize;
    unsigned short reserved0;
    unsigned short ps;
    unsigned short ts;
    unsigned short vs;
    unsigned char mvol;
    unsigned char pan;
    unsigned char attr1;
    unsigned char attr2;
    unsigned long reserved1;
};
```

Members

<i>form</i>	Format name (always 'VABp')
<i>ver</i>	Format version number
<i>id</i>	Bank (VAB) number
<i>fsize</i>	Bank file size
<i>reserved0</i>	Reserved by the system
<i>ps</i>	Total number of programs contained in the bank
<i>ts</i>	Total number of tones contained in the bank
<i>vs</i>	Number of VAGs contained in the bank
<i>mvol</i>	Master volume
<i>pan</i>	Master pan level
<i>attr1</i>	Bank attribute 1 that can be defined by the user
<i>attr2</i>	Bank attribute 2 that can be defined by the user
<i>reserved1</i>	Reserved by the system

Explanation

The VAB bank header contains information, such as sound source data set size and sound source numerals, that is used at the time of execution.

When `SsVabOpenHead()` is called, it is read by the system and wave form data is generated in the SPU's local memory. Also, volume setting and panning setting are referred at the time of voice allocation.

Information about VAB, the program and each VAG header can change at the time of execution by the user, and the attribute value is reflected in the voice application after the next KEY ON.

Remarks

See also:

VagAtr

Waveform header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	10/01/97

Structure

```
typedef struct VagAtr {
    unsigned char prior;
    unsigned char mode;
    unsigned char vol;
    unsigned char pan;
    unsigned char center;
    unsigned char shift;
    unsigned char min;
    unsigned char max;
    unsigned char vibW;
    unsigned char vibT;
    unsigned char porW;
    unsigned char porT;
    unsigned char pbmin;
    unsigned char pbmax;
    unsigned char reserved1;
    unsigned char reserved2;
    unsigned short adsr1;
    unsigned short adsr2;
    short prog;
    short vag;
    short reserved[4];
};
```

Members

<i>prior</i>	Priority (0-15)
<i>mode</i>	Sound source mode (Bit values 0: normal, 1: reverb)
<i>vol</i>	Volume (0-127, 0:min, 127:max)
<i>pan</i>	Pan pot (0-127, 0:left, 63:center, 127:right)
<i>center</i>	Center note (0-127)
<i>shift</i>	Pitch correction (0-127, in cents) (center note fine tune)
<i>min</i>	Minimum note limit
<i>max</i>	Maximum note limit
<i>vibW</i>	Vibrato width (0-127 over one octave)
<i>vibT</i>	Period of vibrato cycle (in ticks)
<i>porW</i>	Portamento width
<i>porT</i>	Period of portamento duration (in ticks)
<i>pbmin</i>	Minimum pitch bend limit
<i>pbmax</i>	Maximum pitch bend limit
<i>reserved1</i>	Reserved by the system
<i>reserved2</i>	Reserved by the system
<i>adsr1</i>	Set ADSR value 1
<i>adsr2</i>	Set ADSR value 2
<i>prog</i>	Master program containing the VAG attribute
<i>vag</i>	VAG's ID number utilized by the VAG attribute
<i>reserved</i> [0...3]	Reserved by the system

Explanation

Remarks

See also:

_SsFCALL

Function table type referenced in SsSeqOpenJ() and SsSepOpenJ().

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	6/22/98

Structure

```
typedef struct {
    void (*noteon) ();
    void (*programchange) ();
    void (*pitchbend) ();
    void (*metaevent) ();
    void (*control[13]) ();
    void (*ccentry[20]) ();
} _SsFCALL;
```

Members

All members hold pointers to low level MIDI functions.

<u>Member</u>	<u>Pointer to</u>	
<i>noteon</i>	<i>_SsNoteOn</i>	<i>MIDI status data</i>
<i>programchange</i>	<i>_SsSetProgramChange</i>	
<i>pitchbend</i>	<i>_SsSetPitchBend</i>	
<i>metaevent</i>	<i>_SsGetMetaEvent</i>	
<i>control [CC_NUMBER]</i>	<i>_SsSetControlChange</i>	<i>Specified when using Control Change</i>
<i>control [CC_BANKCHANGE]</i>	<i>_SsContBankChange</i>	<i>Control Change#1 (Bank Change)</i>
<i>control [CC_DATAENTRY]</i>	<i>_SsContDataEntry</i>	<i>Control Change #6 (Data Entry)</i>
<i>control [CC_MAINVOL]</i>	<i>_SsContMainVol</i>	<i>Control Change #7 (Main Volume)</i>
<i>control [CC_PANPOT]</i>	<i>_SsContPanpot</i>	<i>Control Change #10 (Pan Pot)</i>
<i>control [CC_EXPRESSION]</i>	<i>_SsContExpression</i>	<i>Control Change #11 (Expression)</i>
<i>control [CC_DAMPER]</i>	<i>_SsContDamper</i>	<i>Control Change #64 (Damper pedal)</i>
<i>control [CC_NRPN1]</i>	<i>_SsContNrpn1</i>	<i>Control Change #98 (NRPN)</i>
<i>control [CC_NRPN2]</i>	<i>_SsContNrpn2</i>	<i>Control Change #99 (NRPN)</i>
<i>control [CC_RPN1]</i>	<i>_SsContRpn1</i>	<i>Control Change #100 (RPN)</i>
<i>control [CC_RPN2]</i>	<i>_SsContRpn2</i>	<i>Control Change #101 (RPN)</i>
<i>control [CC_EXTERNAL]</i>	<i>_SsContExternal</i>	<i>Control Change #91 (External Effect Depth)</i>
<i>control [CC_RESETALL]</i>	<i>_SsContResetAll</i>	<i>Control Change #121 (Reset All)</i>
<i>ccentry [DE_PRIORITY]</i>	<i>_SsSetNrpnVabAttr0</i>	<i>priority</i>
<i>ccentry [DE_MODE]</i>	<i>_SsSetNrpnVabAttr1</i>	<i>mode</i>
<i>ccentry [DE_LIMITL]</i>	<i>_SsSetNrpnVabAttr2</i>	<i>limit low</i>
<i>ccentry [DE_LIMITH]</i>	<i>_SsSetNrpnVabAttr3</i>	<i>limit high</i>
<i>ccentry [DE_ADSR_AR_L]</i>	<i>_SsSetNrpnVabAttr4</i>	<i>ADSR (AR-L)</i>
<i>ccentry [DE_ADSR_AR_E]</i>	<i>_SsSetNrpnVabAttr5</i>	<i>ADSR (AR-E)</i>
<i>ccentry [DE_ADSR_DR]</i>	<i>_SsSetNrpnVabAttr6</i>	<i>ADSR (DR)</i>
<i>ccentry [DE_ADSR_SL]</i>	<i>_SsSetNrpnVabAttr7</i>	<i>ADSR (SL)</i>
<i>ccentry [DE_ADSR_SR_L]</i>	<i>_SsSetNrpnVabAttr8</i>	<i>ADSR (SR-L)</i>
<i>ccentry [DE_ADSR_SR_E]</i>	<i>_SsSetNrpnVabAttr9</i>	<i>ADSR (SR-E)</i>
<i>ccentry [DE_ADSR_RR_L]</i>	<i>_SsSetNrpnVabAttr10</i>	<i>ADSR (RR-L)</i>
<i>ccentry [DE_ADSR_RR_E]</i>	<i>_SsSetNrpnVabAttr11</i>	<i>ADSR (RR-E)</i>
<i>ccentry [DE_ADSR_SR]</i>	<i>_SsSetNrpnVabAttr12</i>	<i>ADSR (SR)</i>
<i>ccentry [DE_VIB_TIME]</i>	<i>_SsSetNrpnVabAttr13</i>	<i>vibrate time (no support)</i>
<i>ccentry [DE_PORTA_DEPTH]</i>	<i>_SsSetNrpnVabAttr14</i>	<i>portamento depth (no support)</i>

<i>ccentry [DE_REV_TYPE]</i>	<i>_SsSetNrpnVabAttr15</i>	<i>reverb type</i>
<i>ccentry [DE_REV_DEPTH]</i>	<i>_SsSetNrpnVabAttr16</i>	<i>reverb depth</i>
<i>ccentry [DE_ECHO_FB]</i>	<i>_SsSetNrpnVabAttr17</i>	<i>echo feed back</i>
<i>ccentry [DE_ECHO_DELAY]</i>	<i>_SsSetNrpnVabAttr18</i>	<i>echo feed delay</i>
<i>ccentry [DE_DELAY]</i>	<i>_SsSetNrpnVabAttr19</i>	<i>delay time</i>

Explanation

Functions SsSeqPlay() and SsSepPlay() analyze the MIDI status data and call low level functions. Although there are many low level functions, an application would not necessarily use all these functions. These low level function groups will all be linked in by calling either SsSeqOpen() or SsSepOpen().

In order to reduce code size by not linking unnecessary low level functions, two new functions SsSeqOpenJ() and SsSepOpenJ() have been added; these functions can replace SsSeqOpen() and SsSepOpen() respectively. In these new functions, all low level functions are in a jump table so that the user can set only the desired function groups.

_SsFCALL is a structure that defines this function table. Necessary functions can be linked by assigning the pointer to the applicable low level function. Alternately, low level functions can be eliminated by not setting the member.

Note that if a MIDI sequence calls a low level function which has not been linked in, a BUS ERROR will result. To determine which functions will be called by a MIDI sequence, use a test program. Set all pointers for low level functions to the correspondingly named dmy_Ss... function. Each low level function called by the MIDI sequence will output a message via the printf() function.

Return value

Remarks

SsFCALL is the name of the actual libsnd variable that must be used by the programmers to link the low level MIDI functions.

See also: dmy_Ss...() SsSeqOpenJ() SsSepOpenJ()

dmy_Ss...

Test (dummy) function for low-level MIDI jump table.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	5/22/97

Syntax

```
void dmy_Ss...()
```

Arguments

None

Explanation

Hook these functions into the libsnd variable SsFCALL (structure type _SsFCALL).

Ex: SsFCALL.noteon = (void (*)()) dmy_Ss_NoteOn();

When these functions are called for the first time, the name of the low level MIDI function will be output by the printf() function. After all of the low level MIDI functions that need to be called by your program have been determined, replace the registered dmy_Ss... calls with the appropriate _Ss... calls. Unused dmy_Ss... should be deleted.

Return value

None

Remarks

This function is provided for debugging.

See also:

SsAllocateVoices

Compares the priorities of the desired number of voices and allocates those voices where possible.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

```
long SsAllocateVoices (  
  unsigned char voices,  
  unsigned char priority  
)
```

Arguments

voices The desired number of voices required to be keyed on simultaneously
priority Priority of the desired voices
 Range: 0-127, with 0 being lowest priority and 127 being highest priority

Explanation

Emulates the libsnd voice allocation system, but allows more than one voice to be allocated simultaneously. Searches through all 24 SPU voices for SPU voices in the state SPU_OFF, ENV_OFF (that is, SPU voices which are not currently sounding). If there are fewer SPU voices in a state of SPU_OFF, ENV_OFF than the total number of desired *voices*, the levels of voice priority are compared, and the lowest priority SPU voice number will be allocated for the desired *voices* (if the lowest priority is less than the value set in *priority*). In cases where the priorities are equivalent, the SPU voice number with the lowest envelope will be allocated to the desired *voices*. In cases where the priorities and envelope size are the same, the oldest SPU voices will be allocated to the desired *voices*.

Return value

Returns a bifold containing information about which voices were allocated for key on.

AND the return value and SPU_xxCH (xx = 0-23)

Table 14-1

Result of AND	Description
0	Voice not allocated
1	Voice allocated

Remarks

This function should only be used as part of the series:

- SsBlockVoiceAllocation()
- SsAllocateVoices()
- SsSetVoiceSettings()
- SsQueueRegisters()
- SsQueueKeyOn()
- SsQueueReverb()
- SsUnBlockVoiceAllocation()

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsGetActualProgFromProg (p. n-nn), SsPitchFromNote (p. n-nn), SsQueueKeyOn (p. n-nn), SsQueueRegisters (p. n-nn), SsQueueReverb (p. n-nn), SsSetVoiceSettings (p. n-nn), SsUnBlockVoiceAllocation (p. n-nn), SsVoiceCheck (p. n-nn).

SsBlockVoiceAllocation

Blocks voice allocation system used by SsUtKeyOn(), SsUtKeyOnV(), and MIDI key on commands.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

char SsBlockVoiceAllocation(void)

Arguments

None

Explanation

This function blocks the voice allocation system for libsnd functions SsUtKeyOn() and SsUtKeyOnV(). Once this function is called, those functions will return (-1). MIDI key on commands also will be blocked until SsUnBlockVoiceAllocation() is called.

Return value

Returns 1 if successful.

Returns -1 if voice allocation system already blocked, by either a previous call to this function with no corresponding call to SsUnBlockVoiceAllocation() or a call to SsUtKeyOn(), SsUtKeyOnV() or a MIDI key on command.

Remarks

The time spent until SsUnBlockVoiceAllocation() should be short, in order to reduce missed key on commands.

This function should only be used as part of the series:

```
SsBlockVoiceAllocation()
SsAllocateVoices()
SsSetVoiceSettings()
SsQueueRegisters()
SsQueueKeyOn()
SsQueueReverb()
SsUnBlockVoiceAllocation()
```

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsAllocateVoices (p. n-nn) SsSetVoiceSettings (p. n-nn) SsQueueRegisters (p. n-nn) SsQueueKeyOn (p. n-nn) SsQueueReverb (p. n-nn) SsUnBlockVoiceAllocation (p. n-nn) SsFindPitch (p. n-nn) SsGetActualProgFromProg (p. n-nn) SsVoiceCheck (p. n-nn).

SsChannelMute

Select SEQ channel and play.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	10/23/96

Syntax

```
void SsChannelMute (
short acn,
short tn,
long channels
)
```

Arguments

acn SEP access number
tn SEQ number within SEP
 (0 when the music score data is SEQ)
channels MIDI channel

Explanation

This function specifies MIDI channel in SEQ with 16bit upon playing SEQ. The parts specified with the channel bits can be muted. It is possible to switch to Mute when playing is in progress.

Return value

None

Remarks

See also: SsSeqPlay(), SsSepPlay().

SsEnd

Stops the sound system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsEnd(void)
```

Arguments

None

Explanation

If `SsSetTickMode()` is used to set the mode that automatically calls `SsSeqCalledTbyT()`, this function, after it is called, stops `SsSeqCalledTbyT()` from being called for every Tick.

Return value

None

Remarks

See also: `SsStart`, `SsSetTickMode` (p. 14-86), `SsSeqCalledTbyT` (p. 14-50), `SsQuit` (p. 14-39).

SsGetActualProgFromProg

Converts a program number into a “real” program or offset number.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

```
short SsGetActualProgFromProg (
short vabId,
short ProgNum
)
```

Arguments

vabId VAB number containing desired tone information
ProgNum Program number containing tone information

Explanation

Used to determine the “real” program number of tone information. The “real” number is only incremented by valid programs (programs containing one or more tones) and so may differ from the program number. Example: In a VAB with valid programs 0-10 and 100-127, the *prog_num* of program 127 = 127, while the *prog_actual* of program 127 = 38. This number is used to calculate the offset of tone information in the VAB header.

Return value

The “real” program number is returned upon success.

-1 is returned if *vabId* or *ProgNum* are out of range.

Remarks

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsAllocateVoices (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsPitchFromNote (p. n-nn), SsQueueKeyOn (p. n-nn), SsQueueRegisters (p. n-nn), SsQueueReverb (p. n-nn), SsSetVoiceSettings (p. n-nn), SsUnBlockVoiceAllocation (p. n-nn), SsVoiceCheck (p. n-nn).

SsGetChannelMute

Gets muted channel number

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.0	5/22/97

Syntax

```
long SsGetChannelMute (
short sep_num,
short seq_num
)
```

Arguments

sep_num SEQ/SEP access number
seq_num SEQ number within SEP data

Explanation

Gets number of muted MIDI channel.

Return value

Number (16bit form) of muted MIDI channel.

Remarks

See also: SsChannelMute().

SsGetCurrentPoint

Obtain SEQ/SEP address currently read-in.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	10/23/96

Syntax

```
unsigned char *SsGetCurrentPoint (
short acn,
short trn
)
```

Arguments

acn SEP access number
trn SEQ number within SEP
 (0 when the music score data is SEQ)

Explanation

This function obtains the current read-in address for the SEQ/SEP data that is being played.

Return value

SEP/SEQ data address.

Remarks

See also: SsSeqPlay(), SsSepPlay().

SsGetMute

Obtains mute attribute.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	6/22/98

Syntax

char SsGetMute (*void*)

Arguments

None

Explanation

This function obtains the mute attribute.

Return value

SS_MUTE_ON ... Mute on

SS_MUTE_OFF ... Mute off

Remarks

See also: SsSetMute (p. 14-74).

SsGetMVol

Main volume value acquisition.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsGetMVol(m_vol)  
SndVolume *m_vol;
```

Arguments

m_vol Pointer to main volume value

Explanation

Returns the main volume value to *m_vol*.

Return value

None

Remarks

See also: SsSetMVol (p. 14-69).

SsGetNck

Not supported as of Library Version 4.0.

Gets noise clock value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	5/22/97

Syntax

short SsGetNck(*void*)

Arguments

None

Explanation

Returns the noise clock value.

Return value

Noise clock value.

Remarks

Use libspu noise functions or create a noise VAG from recorded AIFF.

See also: SsSetNck (p. 14-76).

SsGetRVol

Gets reverb volume value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsGetRVol(r_vol)
SndVolume *r_vol;
```

Arguments

r_vol Pointer to reverb volume value

Explanation

Returns the reverb volume value to *r_vol*.

Return value

None

Remarks

See also: SsSetRVol (p. 14-81).

SsGetSerialAttr

Gets a serial attribute value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
char SsGetSerialAttr(s_num, attr)
```

```
char s_num;
```

```
char attr;
```

Arguments

s_num Serial Number

attr Attribute

Explanation

Returns the specified serial attribute value.

(a) *s_num*

Table 14-2

Macro	Contents
SS_SERIAL_A	Serial A (CD input)
SS_SERIAL_B	Serial B (external digital input)

(b) *attr*

Table 14-3

Macro	Contents
SS_MIX	Mixing
SS_REV	Reverb

Return value

Attribute: Returns 1 if on. Returns 0 if off.

Remarks

See also: SsSetSerialAttr (p. 14-80).

SsGetSerialVol

Gets a serial volume value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	7/31/96

Syntax

```
void SsGetSerialVol(s_num, s_vol)
char s_num;
SndVolume *s_vol;
```

Arguments

s_num Serial number
s_vol Pointer to volume value

Explanation

Returns the specified serial number volume value.

Table 14-4

Macro	Contents
SS_SERIAL_A	Serial A (CD input)
SS_SERIAL_B	Serial B (external digital input)

Return value

None

Remarks

See also: SsSetSerialVol (p. 14-83).

SsGetVoiceMask

Returns a bit mask containing the voices which are blocked from access by the libsnd voice allocation system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

unsigned long SsGetVoiceMask(void)

Arguments

None

Explanation

Returns a bit mask containing the voices which are blocked from access by the libsnd voice allocation system.

Return value

AND the return value and SPU_xxCH(xx=0~23).

Table 14-5

Result of AND	Description
0	Voice not blocked from allocation system
1	Voice blocked from allocation system

Remarks

See also: SsSetVoiceMask(), SsSetReservedVoice().

SsInit

Initializes sound system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

void SsInit(*void*)

Arguments

None

Explanation

This function initializes the sound system, clearing the sound local memory.

Return value

None

Remarks

See also: SsInitHot (p. 14-31), SsEnd (p. 14-11), Spulnit (see libspu).

SsInitHot

Initializes sound system (hot reset).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	7/31/96

Syntax

```
void SsInitHot(void)
```

Arguments

None

Explanation

This function performs initialization of the sound system without destroying the data that has been transferred to the sound buffer. Using Exec-related functions, when you want to initialize the sound system by a child process, with the sound buffer in its current state, the child process should call SsInitHot instead of calling SsInit as it normally would.

Return value

None

Remarks

See also: SsInit (p. 14-29), Exec-related functions, SpulnitHot (p. 14-31).

SsIsEos

Determines whether or not a song is being played.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

short SsIsEos(*access_num*, *seq_num*)

short *access_num*;

short *seq_num*;

Arguments

access_num SEQ/SEP access number

seq_num SEQ number inside SEP data

Explanation

Determines whether or not a specified song is being played.

When using this function for SEQ data, set 0 in *seq_num*; when using this function for SEP data, set the number that contains the SEQ to be played.

Return value

Returns 1 if the song is being played.

Returns 0 if the song is not being played.

Remarks

See also:

SsPitchFromNote

Converts MIDI note information into a pitch playback rate.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

```
unsigned short SsPitchFromNote (
short note,
short fine,
unsigned char center,
unsigned char shift
)
```

Arguments

note The MIDI note number (0-127) at which the tone will be keyed on.

fine The fine-tuning adjustment in cents, of *note*. Values range from 0-127, but are scaled to cents.

center The MIDI note number (0-127) at which the tone was created; the center member of the VagAtr structure.

shift The fine-tuning adjustment in cents, of *center*; the shift member of the VagAtr structure. Values range from 0-127, but are scaled to cents.

Explanation

Calculates a pitch value to be applied to a voice in the function SsQueueRegisters().

Return value

Returns the calculated pitch value.

Remarks

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsAllocateVoices (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsGetActualProgFromProg (p. n-nn), SsQueueKeyOn (p. n-nn), SsQueueRegisters (p. n-nn), SsQueueReverb (p. n-nn), SsSetVoiceSettings (p. n-nn), SsUnBlockVoiceAllocation (p. n-nn), SsVoiceCheck (p. n-nn).

SsPlayBack

Reads SEQ/SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsPlayBack(access_num, seq_num, l_count)
short access_num;
short seq_num;
short l_count;
```

Arguments

<i>access_num</i>	SEQ/SEP access number
<i>seq_num</i>	SEQ number inside SEP data
<i>l_count</i>	Song repetition count

Explanation

In the current play mode, no event occurs when a function is called again during execution. However, this function, if called again during execution, stops the song being played, returns to the start of the song, and begins playing it again.

When using this function for SEQ data, set 0 in *seq_num*; when using this function for SEP data, set the number that contains the SEQ to be played. Specify a song repetition count in *l_count*.

For infinite play repetition, specify SSPLAY_INFINITY.

Return value

None

Remarks

See also: SsSeqPlay (p. 14-56), SsSepPlay (p. 14-42).

SsQueueKeyOn

Sets the desired voices in the key on queue to be processed at the next tick callback.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

```
void SsQueueKeyOn (
long voices
)
```

Arguments

voices Bitfield containing voices to be set in the key on queue

Explanation

Hooks into the key on system of libsnd.

Set the arguments as follows:

voices OR SPU_xxCH(0-23) for the desired voices to be set in the key on queue.

Return value

None

Remarks

This function should only be used as part of the series:

- SsBlockVoiceAllocation()
- SsAllocateVoices()
- SsSetVoiceSettings()
- SsQueueRegisters()
- SsQueueKeyOn()
- SsQueueReverb()
- SsUnBlockVoiceAllocation()

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsAllocateVoices (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsGetActualProgFromProg (p. n-nn), SsPitchFromNote (p. n-nn), SsQueueRegisters (p. n-nn), SsQueueReverb (p. n-nn), SsSetVoiceSettings (p. n-nn), SsUnBlockVoiceAllocation (p. n-nn), SsVoiceCheck (p. n-nn).

SsQueueRegisters

Places the specified values in the register queue to be set at the next tick callback.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

```
void SsQueueRegisters (  
long voice,  
SndRegisterAttr *SRA  
)
```

Arguments

voice The voice number for which the SPU registers need to be set (0-23)
**SRA* Values to be sent to the SPU registers

Explanation

Queues the SndRegisterAttr to be processed at the next tick callback.

The setting of *SRA* members is as follows:

volume.left and *volume.right*

Any value between -0x4000 ~ 0x3fff. This is equivalent to volume mode SPU_VOICE_DIRECT (See SpsSetVoiceAttr() for details). Normally, libsnd takes only values from 0-127 and converts them to short int via the algorithm listed in the overview of libsnd. The volume calculations may be emulated or an algorithm to use fewer CPU cycles or do 3-D sound volume calculations may be substituted.

pitch

When using this function in the libsnd emulation functions series, the value set in pitch must match the value of the pitch member of the SndVoiceStats structure set in SsSetVoiceSettings(). When using this function to change pitch when the voice is currently sounding, SsPitchFromNote() may be used to calculate the pitch, or a user-defined pitch lookup table may be used.

mask

The mask may be set by Oring the desired following list of attributes:

Table 14-6

SND_VOLL	left volume
SND_VOLR	right volume
SND_PITCH	pitch
SND_ADDR	waveform data start address
SND_ADSR1	adsr1 information
SND_ADSR2	adsr2 information

If mask is set to 0, all attributes will be set

addr

Contains the waveform data start address. May be obtained using SsUtGetVagAddrFromTone().

adsr1

Contains adsr information for the voice. Should initially be set to the adsr1 member of VagAtr for the tone assigned to the voice.

adsr2

Contains adsr information for the voice. Should initially be set to the adsr2 member of VagAtr for the tone assigned to the voice.

Return value

None

Remarks

This function must be used in the series: SsBlockVoiceAllocation() -> SsAllocateVoices() -> SsSetVoiceSettings() -> SsQueueRegisters()-> SsQueueKeyOn() -> SsQueueReverb() -> SsUnBlockVoiceAllocation() when keying sounds on via the libsnd emulation method.

This function may also be called at any time after the sound has been keyed on to change any of the members of SndRegisterAttr, provided that the function SsVoiceCheck() is called first to ensure voice integrity.

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsAllocateVoices (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsGetActualProgFromProg (p. n-nn), SsPitchFromNote (p. n-nn), SsQueueKeyOn (p. n-nn), SsQueueReverb (p. n-nn), SsSetVoiceSettings (p. n-nn), SsUnBlockVoiceAllocation (p. n-nn), SsVoiceCheck (p. n-nn).

SsQueueReverb

Sets the desired voices in the reverb queue to be processed at the next tick callback.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

```
void SsQueueReverb (
long voices,
long reverb
)
```

Arguments

voices Bitfield containing voices for which reverb will be changed
reverb Bitfield containing reverb on/off data to be set in the reverb queue

Explanation

Hooks into the reverb queueing systems of libsnd, for applying reverb to sounds about to key on, altering the reverb of currently playing sounds, or applying reverb for libspu voices when both sound libraries are used together.

Set the arguments as follows:

voices OR SPU_xxCH(0-23) for the desired voices to be set in the reverb queue.
reverb Reverb queue bitfield.
 Reverb will be affected for all high bits of *voices*.
 If bit = 0 Reverb will be turned off for that voice
 If bit = 1 Reverb will be turned on for that voice

Return value

None

Remarks

If this function is being used as part of the libsnd key-on emulation series:

SsBlockVoiceAllocation() -> SsAllocateVoices() -> SsSetVoiceSettings() -> SsQueueRegisters() -> SsQueueKeyOn() -> SsQueueReverb() -> SsUnBlockVoiceAllocation(), the *mode* member of the VagAtr for the voice may be checked to determine whether reverb should affect that voice.

This function may also be used as a workaround for the reverb conflict between libspu and libsnd. Voices being used by libspu may have reverb changed using this function.

Also, this function is an ideal solution to changing reverb mode during playback of sound. In libsnd it is currently difficult to change reverb mode during playback of sound effects or MIDI music.

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsAllocateVoices (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsGetActualProgFromProg (p. n-nn), SsPitchFromNote (p. n-nn), SsQueueKeyOn (p. n-nn), SsQueueRegisters (p. n-nn), SsSetVoiceSettings (p. n-nn), SsUnBlockVoiceAllocation (p. n-nn), SsVoiceCheck (p. n-nn).

SsQuit

Terminate the sound system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

void SsQuit(*void*)

Arguments

None

Explanation

Terminates the sound system. After this function is called, transfer to the sound buffer will be disabled. To enable transfer to the sound buffer again, SsInit () must be called.

SsEnd () must be called before SsQuit ().

Return value

None

Remarks

See also: SsEnd (p. 14-1), SsStart (p. 14-88), SsSetTickMode (p. 14-86), SsSeqCalledTbyT (p. 14-50).

SsSepClose

Close SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	5/22/97

Syntax

```
void SsSepClose(sep_access_num)
short sep_access_num;
```

Arguments

sep_access_num SEP access number

Explanation

Closes SEP data possessing *sep_access_num* that is no longer needed.

Because closing is performed on a SEP unit basis, all SEQ data stored in the closed SEP will become inaccessible. Before executing this function, use SsSepStop() with the applicable SEP.

Return value

Remarks

See also: SsSepOpen (p. 14-35).

SsSepOpen

OpenOpens SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	5/22/97

Syntax

```
short SsSepOpen(addr, vab_id, seq_num)
unsigned long *addr;
short vab_id;
short seq_num;
```

Arguments

addr Pointer to starting address of SEP data within the main memory
vab_id VAB id
seq_num Number of SEQs contained in SEP

Explanation

Analyzes the SEP data located in the main memory, and returns a SEP access number. Maximum of 32 pieces of SEP data can be opened simultaneously when combined with the number of open SEQ data.

The VB data from the VAB data ID specified by this function is carried out after transmission completion has been confirmed by SsVabTransCompleted().

Return value

SEP access number.

An internal SEP data management table number that possesses the same characteristics as the SEQ access number.

Remarks

For Library Versions 4.0 and earlier:

- 1) Do not call this from inside a callback;
- 2) If your s_max from SsSetTableSize is less than 32, you must keep track of your open SEQs/SEPs so as not to exceed the limit set by s_max.

See also: SsSepClose (p. 14-40).

SsSepOpenJ

Opens SEP data (function table version).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	5/22/97

Syntax

```
short SsSepOpenJ (
unsigned long *addr,
short vab_id,
short num2
)
```

Arguments

addr Pointer to starting address of SEP data within the main memory
vab_id VAB id
num2 Number of SEQs contained in SEP

Explanation

This function is equivalent to SsSepOpen() if all the low level functions were registered. In addition to the SsSepOpen() capability, this function enables a programmer to control functions to be registered to the table and thus improve code efficiency by not calling unnecessary low level functions.

For those slots that SsFCALL will not register, use dummy functions, standard function names with the prefix "dmy" so that even when a lower function was called, no BUS ERROR would occur and the function names would be printed out. After checking the called function names, register the function names without "dmy".

The VB data from the VAB data ID specified by this function is carried out after transmission completion has been confirmed by SsVabTransCompleted().

Return value

SEQ Access Number: Used in the SEQ data access function, being the inner SEQ data control table number.

Remarks

For Library Versions 4.0 and earlier:

- 1) Do not call this from inside a callback;
- 2) If your s_max from SsSetTableSize is less than 32, you must keep track of your open SEQs/SEPs so as not to exceed the limit set by s_max.

See also: SsSepOpen().

SsSepPause

Pause the reading of SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSepPause(sep_access_num, seq_num)
short sep_access_num;
short seq_num;
```

Arguments

sep_access_num SEP access number
seq_num SEQ number inside SEP data

Explanation

Pauses the reading (playing) of the *seq_num* SEQ data of SEP data possessing *sep_access_num*.

Return value

None

Remarks

See also: SsSepReplay (p. 14-42).

SsSepPlay

Reads (plays) SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSepPlay(sep_access_num, seq_num, play_mode, l_count)
```

```
short sep_access_num;
```

```
short seq_num;
```

```
char play_mode;
```

```
short l_count;
```

Arguments

<i>sep_access_num</i>	SEP access number
<i>seq_num</i>	SEP data SEQ number
<i>play_mode</i>	Play mode
<i>l_count</i>	Song repetition count

Explanation

Begins to read (play) SEQ data specified by the SEP data *seq_num* specified by *sep_access_num*, or, depending on the *play_mode* value, you may choose a pause state. For infinite play repetition, specify SSPLAY_INFINITY.

Table 14-7

Play_mode	Actions
SSPLAY_PAUSE	Makes a pause state
SSPLAY_PLAY	Plays immediately

Examples:

- (1) Opens SEP data containing four pieces of SEQ data:
sep1 = SsSepOpen (addr, vab_id, 4);
- (2) Immediately plays the third piece of data of the opened SEP data twice.
SsSepPlay (sep1, 2, SSPLAY_PLAY, 2);

Return value

None

Remarks

See also: SsSepStop (p. 14-51).

SsSepReplay

Resume (replay) the reading of SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSepReplay(sep_access_num, seq_num)
short sep_access_num;
short seq_num;
```

Arguments

sep_access_num SEP access number
seq_num SEQ number inside SEP data

Explanation

Resumes the reading of the *seq_num* SEQ data of SEP data possessing *sep_access_num*, that was paused by SsSepPause.

Return value

None

Remarks

See also: SsSepPause (p. 14-42).

SsSepSetAccelerando

Accelerate the tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSepSetAccelerando(sep_access_num, seq_num, tempo, v_time)
```

```
short sep_access_num;
```

```
short seq_num;
```

```
long tempo;
```

```
long v_time;
```

Arguments

<i>sep_access_num</i>	SEQ access number
<i>seq_num</i>	SEQ number inside SEP data
<i>tempo</i>	Song tempo
<i>v_time</i>	Time (in ticks)

Explanation

Increases the tempo of the *seq_num*-th SEQ data of SEP data possessing *sep_access_num* down to tempo within *v_time*.

However, if the specified tempo is smaller (slower) than the current tempo, this function acts the same as SsSepSetRitardando.

Return value

None

Remarks

See also: SsSepSetRitardando (p. 14-49).

SsSepSetCrescendo

Crescendo (valid for individual SEQ in SEP).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	10/01/97

Syntax

```
void SsSepSetCrescendo(sep_access_num, seq_num, vol, v_time)
short sep_access_num;
short seq_num;
short vol;
long v_time;
```

Arguments

<i>sep_access_num</i>	SEP access number
<i>seq_num</i>	SEQ number inside SEP data
<i>vol</i>	Volume value (0-127)
<i>v_time</i>	Time (in tick units)

Explanation

Raises the main volume of the *seq_num* SEQ data of SEP data possessing *sep_access_num* by *vol* within *v_time*.

Note that this function will have no effect if the volume of each voice is at the maximum or if *vol* is a negative number. It is recommended that *v_time* be set to an integer multiple of *vol*.

Return value

None

Remarks

See also: SsSepSetDecrescendo (p. 14-48).

SsSepSetDecrescendo

Decrescendo (valid for individual SEQ in SEP).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSepSetDecrescendo(sep_access_num, seq_num, hort vol, v_time)
short sep_access_num;
short seq_num;
short vol;
long v_time;
```

Arguments

<i>sep_access_num</i>	SEP access number
<i>seq_num</i> SEQ	Number inside SEP data
<i>vol</i>	Volume value (0-127)
<i>v_time</i>	Time (in tick units)

Explanation

Lowens the main volume of the *seq_num* SEQ data of SEP data possessing *sep_access_num* by *vol* within *v_time*.

Note that this function will have no effect if the volume of each voice is at the minimum or if *vol* is a negative number. It is recommended that *v_time* be set to an integer multiple of *vol*.

Return value

None

Remarks

See also: SsSepSetCrescendo (p. 14-47).

SsSepSetRitardando

Slows the tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSepSetRitardando(sep_access_num, seq_num, tempo, v_time)
short sep_access_num;
short seq_num;
long tempo;
long v_time;
```

Arguments

<i>sep_access_num</i>	SEQ access number
<i>seq_num</i>	SEQ number inside SEP data
<i>tempo</i>	Song tempo
<i>v_time</i>	Time (in tick units)

Explanation

Slows the tempo of the *seq_num* SEQ data of SEP data possessing *sep_access_num* down to tempo within *v_time*.

However, if the specified tempo is larger (faster) than the current tempo, this function acts the same as SsSepSetAccelerando.

Return value

None

Remarks

See also: SsSepSetAccelerando (p. 14-45).

SsSepSetVol

SEP volume setting (valid for individual SEQ in SEP).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSepSetVol(sep_access_num, seq_num, voll, volr)
short sep_access_num;
short seq_num;
short voll;
short volr;
```

Arguments

<i>sep_access_num</i>	SEP access number
<i>seq_num</i>	SEQ number inside SEP data
<i>voll</i>	L channel main volume value
<i>volr</i>	R channel main volume value

Explanation

Sets the L and R channels for the main volume of the *seq_num* SEQ data of SEP data possessing *sep_access_num* to specified values.

A value between 0 and 127 can be set.

Return value

None

Remarks

See also:

SsSepStop

Stops the reading of SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSepStop(sep_access_num, seq_num)
short sep_access_num;
short seq_num;
```

Arguments

sep_access_num SEP access number
seq_num SEQ number inside SEP data

Explanation

Terminates the reading (playing) of the *seq_num* SEQ data of SEP data possessing *sep_access_num*.

Return value

None

Remarks

See also: SsSepPlay (p. 14-42).

SsSeqCalledTbyT

It is called at each 1 Tick, interprets SEQ/SEP data and carries out playback.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

void SsSeqCalledTbyT(*void*)

Arguments

None

Explanation

At each Tick this function is called; it interprets SEQ/SEP data and carries out playback. Tick is set by SsSetTickMode(), but this Tick merely regulates the internal sound system, without depending either on the speed or resolution determined by SEQ/SEP data.

When SsSetTickMode is specified, the sound system calls this function with the given resolution if the tick_mode is macro SS_TICK60 or SS_TICK240. However, if SS_NOTICK is specified, this function must be called by the program at each 1/60 second interval (usually with vertical sync (VSync()) timing).

Return value

None

Remarks

See also: SsSetTickMode (p. 14-86).

SsSeqClose

Closes SEQ data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	5/22/97

Syntax

```
void SsSeqClose(seq_access_num)
short seq_access_num;
```

Arguments

seq_access_num SEQ access number

Explanation

This function closes SEQ data with an un-needed *seq_access_num*.

Before executing this function, use SsSeqStop() with the applicable SEQ.

Return value

None

Remarks

See also: SsSeqOpen (p. 14-55).

SsSeqGetVol

Obtaining SEQ volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSeqGetVol (
short access_num,
short seq_num,
short *voll,
short *volr
)
```

Argument

<i>access_num</i>	SEQ/SEP access number
<i>seq_num</i>	SEQ number of SEP data
<i>voll</i>	L volume of SEQ data
<i>volr</i>	R volume of SEQ data

Explanation

This function returns current left and right SEQ volume to *voll* and *volr*. Set *seq_num* at 0 for SEQ data, and set it at appropriate SEQ number for SEP data.

The volume value set by *SsSepSetVol()* can be obtained by this function.

Return value

None

Remarks

See also: *SsSeqSetVol* (p. 14-66), *SsSepSetVol* (p. 14-50).

SsSeqOpen

Opens SEQ data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	5/22/97

Syntax

```
short SsSeqOpen(addr, vab_id)
unsigned long *addr;
short vab_id;
```

Arguments

addr Pointer to start address of SEQ data in the main storage
vab_id VAB id

Explanation

This function analyzes SEQ data in the main memory to return the SEQ access number. When you try to open more than 32 SEP data (combined with SEQ data) at the same time, the return value will be -1.

The VB data from the VAB data ID specified by this function is carried out after transmission completion has been confirmed by SsVabTransCompleted ().

Return value

SEQ access number: because this is used in the SEQ data access function, this is the SEQ data control table number.

Remarks

For Library Versions 4.0 and earlier:

- 1) Do not call this from inside a callback;
- 2) If your s_max from SsSetTableSize is less than 32, you must keep track of your open SEQs/SEPs so as not to exceed the limit set by s_max.

See also: SsSeqClose (p. 14-53).

SsSeqOpenJ

Opens SEQ data (function table version).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	5/22/97

Syntax

```
short SsSeqOpenJ (
unsigned long *addr,
short vab_id
)
```

Arguments

addr Pointer to start address of SEQ data in the main storage
vab_id VAB id

Explanation

This function is equivalent to SsSeqOpen() if all the low level functions were registered. In addition to the SsSeqOpen() capability, this function enables a programmer to control functions to be registered to the table and thus improve code efficiency by not calling unnecessary low level functions.

For those slots that SsFCALL will not register, use dummy functions, standard function names with the prefix "dmy" so that even when a lower function was called, no BUS ERROR would occur and the function names would be printed out. After checking the called function names, register the function names without "dmy".

The VB data from the VAB data ID specified by this function is carried out after transmission completion has been confirmed by SsVabTransCompleted ().

Return value

SEQ Access Number: Used in the SEQ data access function, being the inner SEQ data control table number.

Remarks

For Library Versions 4.0 and earlier:

- 1) Do not call this from inside a callback;
- 2) If your s_max from SsSetTableSize is less than 32, you must keep track of your open SEQs/SEPs so as not to exceed the limit set by s_max.

See also: SsSeqOpen () .

SsSeqPause

Pauses SEQ data reading.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSeqPause(seq_access_num)
short seq_access_num;
```

Arguments

seq_access_num SEQ access number

Explanation

This function stops reading (playing) SEQ data with *seq_access_num*.

Return value

None

Remarks

See also: SsSeqReplay (p. 14-60).

SsSeqPlay

Reads (plays) SEQ data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSeqPlay(seq_access_num, play_mode, l_count)
short seq_access_num;
char play_mode;
short l_count;
```

Arguments

<i>seq_access_num</i>	SEQ access number
<i>play_mode</i>	Performance mode
<i>l_count</i>	Number of repeats of the music

Explanation

This function selects either immediate SEQ data reading or sets a pause state at the start of SEQ data. Designate repeat play of the music by *l_count*, using SSPLAY_INFINITY if play is unlimited. For play mode, the parameters below may be specified.

Table 14-8

Macro	State
SSPLAY_PAUSE	Pause at start of piece
SSPLAY_PLAY	Immediate performance

Return value

None

Remarks

See also: SsSeqPause (p. 14-57).

SsSeqPlayPtoP

Reads SEQ data (play interval).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.2	02/15/98

Syntax

```
void SsSeqPlayPtoP (
short access_num,
short seq_num,
unsigned char *start_point,
unsigned char *end_point,
char play_mode,
short l_count
)
```

Arguments

<i>access_num</i>	SEQ/SEP access number
<i>seq_num</i>	SEQ number within SEP (0 if music data is SEQ)
<i>start_point</i>	Load (play) start point
<i>end_point</i>	Load (play) end point
<i>play_mode</i>	Performance mode
<i>l_count</i>	Loop count

Explanation

The data interval specified by *start_point* and *end_point* is loaded (played). Values obtained from *SsGetCurrentPoint()* are specified for *start_point* and *end_point*.

The value of *play_mode* can be used to select whether SEQ data should be read (played) immediately or whether a pause should be entered at the start of the SEQ data (the start of the song).

l_count specifies the number of times the song is to be looped. Use *SSPLAY_INFINITY* for an infinite loop.

<i>play_mode</i>	Operation
<i>SSPLAY_PAUSE</i>	Enter pause state
<i>SSPLAY_PLAY</i>	Begin playing immediately

Return value

None

Remarks

See also: *SsSeqPlay()*, *SsSepPlay()*, *SsGetCurrentPoint()*

SsSeqReplay

Resumes SEQ data reading (Replay) .

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSeqReplay(seq_access_num)  
short seq_access_num;
```

Arguments

seq_access_num SEQ access number

Explanation

This function resumes reading SEQ data with *seq_access_num* stopped by SsPause

Return value

None

Remarks

See also: SsSeqPause (p. 14-56).

SsSeqSetAccelerando

Quickens tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSeqSetAccelerando(seq_access_num, tempo, v_time)
short seq_access_num;
long tempo;
long v_time;
```

Arguments

<i>seq_access_num</i>	SEQ access number
<i>tempo</i>	Music tempo
<i>v_time</i>	Time (in ticks)

Explanation

This function quickens the SEQ data with *seq_access_num* to the tempo resolution in *v_time*. With the specified resolution smaller (slower) than the current resolution, the function provides the same effect as SsSeqSetRitardando.

Return value

None

Remarks

See also: SsSeqSetRitardando (p. 14-65).

SsSeqSetCrescendo

Crescendo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	10/01/97

Syntax

```
void SsSeqSetCrescendo(seq_access_num, vol, v_time)
short seq_access_num;
short vol;
long v_time;
```

Arguments

<i>seq_access_num</i>	SEQ access number
<i>vol</i>	Volume value (0-127)
<i>v_time</i>	Time (in ticks)

Explanation

This function increases the main volume of SEQ data with *seq_access_num* by the *vol* value in *v_time*. With the maximum voice volume, or if *vol* is a negative number, the function provides no effect. It is recommended that *v_time* be set to an integer multiple of *vol*.

Return value

None

Remarks

See also: SsSeqSetDecrescendo (p. 14-62).

SsSeqSetDecrescendo

Decrescendo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	10/01/97

Syntax

```
void SsSeqSetDecrescendo(seq_access_num, vol, v_time)
short seq_access_num;
short vol;
long v_time;
```

Arguments

<i>seq_access_num</i>	SEQ access number
<i>vol</i>	Volume value (0-127)
<i>v_time</i>	Time (in ticks)

Explanation

Lowers main volume of SEQ data with *seq_access_num* by the *vol* value in *v_time*. If each voice volume is the maximum value, or if *vol* is a negative number, there is no effect. It is recommended that *v_time* be set to an integer multiple of *vol*.

Return value

None

Remarks

See also: SsSeqSetCrescendo (p. 14-62).

SsSeqSetNext

Specifies subsequent SEQ data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSeqSetNext(seq_access_num1, seq_access_num2)
short seq_access_num1;
short seq_access_num2;
```

Arguments

seq_access_num1 SEQ access number
seq_access_num2 SEQ access number

Explanation

This function specifies the SEQ access number (*seq_access_num2*) of SEQ data to be performed after the SEQ data with *seq_access_num1*.

Return value

None

Remarks

See also: SsSetNext (p. 14-77)

SsSeqSetRitardando

Slows tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSeqSetRitardando(seq_access_num, tempo, v_time)
```

```
short seq_access_num;
```

```
long tempo;
```

```
long v_time;
```

Arguments

seq_access_num SEQ access number

tempo Music tempo

v_time Time (in ticks)

Explanation

This function slows the SEQ data with *seq_access_num* to the tempo resolution *in v_time*. With the specified resolution larger (faster) than the current resolution, however, the function provides the same effect as SsSeqSetAccelerando.

Return value

None

Remarks

See also: SsSeqSetAccelerando (p. 14-60).

SsSeqSetVol

Sets SEQ volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSeqSetVol(seq_access_num, voll, volr)
short seq_access_num;
short voll;
short volr;
```

Arguments

<i>seq_access_num</i>	SEQ access number
<i>voll</i> <i>L</i>	Channel's main volume value
<i>volr</i> <i>R</i>	Channel's main volume value

Explanation

This function sets the main volume of music with *seq_access_num* at values specified for the L and R channels. *voll* and *volr* range from 0 to 127.

Return value

None

Remarks

See also:

SsSeqSkip

Skips SEQ data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.2	02/15/98

Syntax

```
int SsSeqSkip (
short access_num,
short seq_num,
char unit,
short count
)
```

Arguments

<i>access_num</i>	SEQ/SEP access number
<i>seq_num</i>	SEQ number within SEP (0 if music data is SEQ)
<i>unit</i>	Skip unit
<i>count</i>	Skip amount

Explanation

The playback pointer is moved forward the number of times specified by *count*, and in units specified by *unit*. The pointer is moved beginning with the sequence specified by *access_num*.

unit	Skip unit
SSSKIP_TICK	TICK unit
SSSKIP_NOTE4	Quarter note unit
SSSKIP_NOTE8	One-eighth note unit
SSSKIP_BAR	Measure unit

Return value

0 if the function was successful
-1 if the operation failed

Remarks

See also:

SsSeqStop

Terminates SEQ data reading.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsSeqStop(seq_access_num)
short seq_access_num;
```

Arguments

seq_access_num SEQ access number

Explanation

This function terminates the reading of SEQ data with *seq_access_num* (performance).

Return value

None

Remarks

See also: SsSeqPlay (p. 14-56).

SsSetAutoKeyOffMode

Sets the automatic KeyOff mode.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSetAutoKeyOffMode(mode)
short mode;
```

Arguments

mode 0 Automatically keys off
mode 1 Does not key off until a KeyOff request comes in

Explanation

Sets the automatic KeyOff mode. The default is the automatic KeyOff mode. If the envelopes for the past 16 interrupts contain all 0's, the automatic KeyOff mode assumes that waveform playback has been automatically terminated, and uses the voice for other waveform playback.

Return value

None

Remarks

See also:

SsSetCurrentPoint

Sets data address in SEQ/SEP.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.2	02/15/98

Syntax

```
int SsSetCurrentPoint (
short acn,
short tn,
unsigned char *point
)
```

Arguments

acn SEP access number
tn SEQ number within SEP (0 if music data is SEQ)
point SEQ/SEP data address

Explanation

Sets the data address obtained from SsGetCurrentPoint() in SEQ/SEP.

Return value

0 if the function was successful
-1 if the operation failed

Remarks

See also: SsGetCurrentPoint()

SsSetLoop

Sets a song repetition count.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSetLoop(access_num, seq_num, l_count)
short access_num;
short seq_num;
short l_count;
```

Arguments

<i>access_num</i>	SEQ/SEP access number
<i>seq_num</i>	SEQ number inside SEP data
<i>l_count</i>	Song repetition count

Explanation

Sets a song repetition count. This function is useful for changing the song repetition count set in SsSeqPlay. After this function is called, the current song repetition count will be reset, and the song will be played for the number of times set by the new count.

When using this function for SEQ data, set 0 in *seq_num*; when using this function for SEP data, set the number that contains the SEQ to be played.

Return value

None

Remarks

See also:

SsSetMarkCallback

Register a function to be called when a mark is detected.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
typedef void (*SsSeqMarkCallbackProc) (short, short, short);
void SsSetMarkCallback (
short access_num,
short seq_num,
SsMarkCallbackProc proc
)
```

Arguments

access_num SEQ/SEP access number
seq_num SEQ number inside SEP data
proc Callback function to be called when Mark is detected

Explanation

When a mark is detected inside a song possessing *access_num*, a Callback function will be called. During this operation, SEQ/SEP number will be handed over to the first argument; SEQ number inside SEP data will be handed over to the second argument; and the data2 value set in Mark will be handed over to the third argument. Set the second argument to 0 when using SEQ. The function clears the Callback function when NULL is given to *proc*.

Only one Callback function can be registered at a time.

Sample

```
/* Callback function-definition*/
SsMarkCallbackProc proc (short ac_no, short tr_no, short
data);
:
/* Opens SEQ */
short seq_a_num = SsSeqOpen (addr, vab_id);
/* Sets Callback function */
SsSetMarkCallback (seq_a_num, 0, (SsMarkCallbackProc) proc);
```

Return value

None

Remarks

See also:

SsSetMono

Set monaural mode.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsSetMono(void)
```

Arguments

None

Explanation

Sets the output to monaural mode.

Return value

None

Remarks

See also: SsSetStereo (p. 14-84)

SsSetMute

Set a Mute.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsSetMute(mode)
char mode;
```

Arguments

mode Setting mode

Explanation

This function sets a mute. The values below may be specified for *mode*.

Table 14-9

Macro	Contents
SS_MUTE_ON	Mute on
SS_MUTE_OFF	Mute off

Return value

None

Remarks

See also:

SsSetMVol

Set main volume value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsSetMVol(voll, volr)
```

```
short voll;
```

```
short volr;
```

Arguments

voll L channel volume value

volr R channel volume value

Explanation

This function sets the main volume values for *voll* and *volr*. The value ranges from 0 to 127.

You must set this before playing sequence (SEQ, SEP) data.

Return value

None

Remarks

See also: SsGetMVol (p. 14-24).

SsSetNck

Not supported as of Library Version 4.0.

Sets noise clock value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	5/22/97

Syntax

```
void SsSetNck(n_clock)
```

```
short n_clock;
```

Arguments

n_clock Noise clock value

Explanation

Sets the noise clock value. The possible values are from 0 - 0x3f. Noise is lower if the value is smaller, and louder if the value is larger.

Return value

None

Remarks

Use libspu noise functions or create a noise VAG from recorded AIFF.

See also: SsGetNck (p. 14-15).

SsSetNext

This function sets the next SEQ/SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.2	7/31/96

Syntax

```
void SsSetNext (
short ac_no1,
short tr_no1,
short ac_no2,
short tr_no2
)
```

Arguments

<i>ac_no1</i>	SEP/SEQ access number
<i>tr_no1</i>	SEQ number in SEP (If the score data is SEQ, <i>tr_no1</i> is 0.)
<i>ac_no2</i>	SEP/SEQ access number
<i>tr_no2</i>	SEQ number in SEP (If the score data is SEQ, <i>tr_no2</i> is 0.)

Explanation

This function sets the score data with SEP/SEQ access numbers (*ac_no2*, *tr_no2*) to be played after SEP/SEQ data (*ac_no1*, *tr_no1*).

The next score data is played automatically after the previous score finishes playing.

Return value

None

Remarks

See also: SsSeqSetNext (p. 14-64).

SsSetNoiseOff

Not supported as of Library Version 4.0.

Sets Noise off.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	5/22/97

Syntax

void SsSetNoiseOff(void)

Arguments

None

Explanation

Makes noise off

Return value

None

Remarks

Use libspu noise functions or create a noise VAG from recorded AIFF.

See also: SsSetNoiseOn (p. 14-79).

SsSetNoiseOn

Not supported as of Library Version 4.0.

Sets Noise on.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	522/97

Syntax

```
void SsSetNoiseOn(voll, volr)
```

```
short voll;
```

```
short volr;
```

Arguments

voll L channel volume value

volr R channel volume value

Explanation

Makes Noise On with the given volume value. Volume values may be between 0-127. It sets the Noise Clock value with SsSetNck before making Noise On.

Return value

None

Remarks

Use libspu noise functions or create a noise VAG from recorded AIFF.

See also: SsSetNoiseOff (p. 14-78).

SsSetReservedVoice

Declares the number of voices to be allocated by libsnd library.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	5/22/97

Syntax

```
char SsSetReservedVoice(voices)
char voices;
```

Arguments

voices Voice count

Explanation

Declares the number of voices libsnd library will use for allocation. Other voices can be used in libspu by the user. (They must always be called in "all key off.")

For example, if *char* = 20, then:

- Voices 0-19 will be used for allocation by libsnd.
- Voices 20-23 will be available for libspu.

Return value

Returns the set voice count if successful. Returns -1 if unsuccessful.

Remarks

Should only be called once, before start of sound processing (SsStart/SsStart2()).

See also:

SsSetRVol

Sets reverberant volume values.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsSetRVol(voll, volr)
```

```
short voll;
```

```
short volr;
```

Arguments

voll L channel's volume value

volr R channel's volume value

Explanation

This function sets the reverberant volume values for *voll* and *volr*. The value ranges from 0 to 127.

Return value

None

Remarks

See also: SsGetRVol (p. 14-26).

SsSetSerialAttr

Sets a serial attribute.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsSetSerialAttr(s_num, attr, mode)
char s_num;
char attr;
char mode;
```

Arguments

- s_num* Serial number
- attr* Attribute value
- mode* Setting mode

Explanation

Sets a serial attribute.

(a) *s_num*

Table 14-10

Macro	Contents
SS_SERIAL_A	Serial A (CD input)
SS_SERIAL_B	Serial input line B (external digital input)

(b) *attr*

Table 14-11

Macro	Contents
SS_MIX	Mixing
SS_REV	Reverb

(c) *mode*

Table 14-12

Macro	Contents
SS_SON	attr on
SS_SOFF	attr off

Return value

None

Remarks

See also: SsGetSerialAttr (p. 14-27).

SsSetSerialVol

Sets a serial volume value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	5/22/97

Syntax

```
void SsSetSerialVol(s_num, voll, volr)
char s_num;
short voll;
short volr;
```

Arguments

s_num Serial number
voll L channel volume value
volr R channel volume value

Explanation

Sets the value of the serial volume in *voll*, *volr*. The volume values may be set between 0-127.

Table 14-13

Macro	Contents
SS_SERIAL_A (SS_CD)	Serial A (CD input)
SS_SERIAL_B (SS_EXT)	Serial B (external digital input)

Return value

None

Remarks

See also: SsGetSerialVol (p. 14-28).

SsSetStereo

Sets stereo mode.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
void SsSetStereo(void)
```

Arguments

None

Explanation

Sets the output to stereo mode. The sound system default output is stereo.

Return value

None

Remarks

See also: SsSetMono (p. 14-72).

SsSetTableSize

Specifies the area of a SEQ/SEP data attribute table.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	5/22/97

Syntax

```
void SsSetTableSize(*table, s_max, t_max)
char *table;
short s_max;
short t_max;
```

Arguments

table Pointer to SEQ/SEP data attribute table area variable
s_max Maximum frequency of opening SEQ/SEP data
t_max Number of SEQ included in SEP

Explanation

The area of a SEQ/SEP data attribute table is set in the library. The library uses this area to analyze SEQ/SEP data, then saves it and plays it back.

s_max specifies the maximum number of times SEQ/SEP data may be opened. The upper limit is 32. Once the upper limit is reached, unused SEQ/SEP data must be closed with SsSeqClose/SsSepClose before more data can be opened. *t_max* specifies the number of SEQ included in the SEP data. Set *t_max* to 1 to handle only SEQ data and not use SEP data. The upper limit of *t_max* is 16.

In table, you must preserve the area by using global variables or functions like malloc() (auto variables cannot be used in a function).

Use the following to find the size from the library:

(SS_SEQ_TABSIZ x *s_max* x *t_max*)

where the constant SS_SEQ_TABSIZ is declared in libsnd.h. Note that the value of this constant varies from version to version, so use the constant when saving the table area.

SsSetTableSize() is called immediately after SsInit(). Both functions are set to be called only once; what happens when multiple calls are made is unclear.

Return value

None

Remarks

See also: SsInit(), SsInitHot(), SsSeqOpen (p. 14-55), SsSepOpen (p. 14-41).

SsSetTempo

Set a tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsSetTempo(access_num, seq_num, tempo)
short access_num;
short seq_num;
short tempo;
```

Arguments

access_num SEQ/SEP access number
seq_num SEQ number inside SEP data
tempo Song tempo

Explanation

Sets a tempo. This function is useful for changing the tempo set in SsSeqPlay.

After this function is called, the current tempo will be changed to the new tempo specified for playing songs.

When using this function for SEQ data, set 0 in *seq_num*; when using this function for SEP data, set the number that contains the SEQ to be played.

Return value

None

Remarks

See also:

SsSetTickCallback

Sets the TickCallback function called with every TICK.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
int SsSetTickCallback (
void (*cb)()
)
```

Arguments

cb Pointer to TickCallback function called with every Tick

Explanation

Sets the TickCallback function called with every Tick. Only when SS_NOTICK has not been set on SsSetTickMode, after SsStart () or SsStart2 () have been called, function *cb* will be called with each Tick.

When tick Callback function has not been set using SsSetTickCallback, the default will be set as SsSeqCalledTbyT ().

Return value

Previously-set TickCallback function.

Remarks

See also: SsSetTickMode (p. 14-88), SsStart (p. 14-93), SsStart2 (p. 14-94), SsSeqCalledTbyT (p. 14-52).

SsSetTickMode

Sets Tick.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	5/22/97

Syntax

```
void SsSetTickMode(tick_mode)
long tick_mode;
```

Arguments

tick_mode Tick mode

Explanation

Sets the resolution of Tick. Call this function only once before calling SsSeqOpen(), SsSepOpen() or SsStart() for the first time. When it is called multiple times, correct operation cannot be guaranteed.

Tick Mode does not depend on the speed or resolution specified by SEQ/SEP data, and merely specifies the resolution inside the sound system.

In Tick Mode, the effects of SS_TICK50, SS_TICK60, and SS_TICKVSYNC differ according to the specification of SetVideoMode() (see the individual Tick Mode entries below).

tick_mode may be specified with the following values.

Table 14-14

<i>tick_mode</i>	Setting
SS_TICK50	Tick = 1/50 second SsSeqCalledTbyT() Automatic call
SS_TICK60	Tick = 1/60 second SsSeqCalledTbyT() Automatic call
SS_TICKVSYNC	Tick = resolution of vertical sync SsSeqCalledTbyT() Automatic call
SS_TICK120	Tick = 1/120 second SsSeqCalledTbyT() Automatic call
SS_TICK240	Tick = 1/240 second SsSeqCalledTbyT() Automatic call
SS_NOTICK	Tick = 1/60 second SsSeqCallTbyT() No automatic call
Any resolution	Tick = 1/ <i>tick_mode</i> seconds SsSeqCalledTbyT() Automatic call
(Any resolution SS_NOTICK)	Tick = 1/ <i>tick_mode</i> seconds SsSeqCallTbyT() No automatic call

1. *tick_mode* = SS_TICK50
1/50 second is 1 Tick; the SEQ file will be played at this resolution.
When the mode specified by SetVideoMode() is MODE_PAL, use the OS Root Counter Management Service RCntCNT3 with this resolution. PAL vertical sync timing (1/50 sec) is 1 Tick; the SEQ file will be played at this resolution. For MODE_NTSC, generate this resolution with the OS Root Counter Management Service RCntCNT2, and interpret and play back the SEQ file. You cannot use RCntCNT2 in programs at any other resolution.
2. *tick_mode* = SS_TICK60
1/60 seconds is 1 Tick; the SEQ file will be played at this resolution.

When the mode specified by `SetVideoMode()` is `MODE_NTSC`, use the OS Root Counter Management Service `RCntCNT3` with this resolution. NTSC vertical sync timing (1/60 sec) is 1 Tick; the SEQ file will be played at this resolution. For `MODE_PAL`, generate this resolution with the OS Root Counter Management Service `RCntCNT2`, and interpret play back the SEQ file. You cannot use `RCntCNT2` in programs at any other resolution.

3. `tick_mode = SS_TICKVSYNC`

Vertical sync timing is 1 Tick; the SEQ file will be played at this resolution.

When the mode specified by `SetVideoMode()` is `MODE_NTSC`, NTSC vertical sync timing (1/60 sec) is the resolution; when the specified mode is `MODE_PAL`, PAL vertical sync timing (1/50 sec) is the resolution. The SEQ file is interpreted and played back.

Use the OS Root Counter Management Service `RCntCNT3` at both of these resolutions.

4. `tick_mode = SS_TICK120`

1/120 seconds are 1 Tick; the SEQ file will be played at this resolution. However, because the OS Root Counter Management Service `RCntCNT2` is used at this resolution, it cannot be used by programs at other than this resolution.

5. `tick_mode = SS_TICK240`

1/240 seconds is 1 Tick; the SEQ file will be played at this resolution. However, because the OS Root Counter Management Service `RCntCNT2` is used at this resolution, it cannot be used by programs at other resolutions.

6. `tick_mode = SS_NOTICK`

Vertical retrace timing (1/60 seconds) is 1 Tick; the SEQ file will be played at this resolution. However, because `SsSeqCalledTbyT()` will not be automatically called, it must be called inside the user program at the vertical retrace timing.

7. `tick_mode = Any resolution`

By setting a value between 60 and 240 in the argument, 1 Tick is set to (1/tick_mode), and the SEQ file is interpreted and played at this resolution. However, in this case, because the OS Root Counter Management Service `RCntCNT2` is used at this resolution, it cannot be used by programs at other than this resolution.

8. `tick_mode = (Any resolution | SS_NOTICK)`

By setting a value between 60 and 240 in the argument, 1 Tick is set at (1/tick_mode), and the SEQ file is interpreted and played at this resolution. The macros `SS_TICK50`, `SS_TICK60`, `SS_TICKVSYNC`, `SS_TICK120`, and `SS_TICK240` are not valid for the "any resolution." An actual value must be ORed with `SS_NOTICK`. Example: `65 | SS_NOTICK` will set up a resolution of 1/65th second. However, if `SS_NOTICK` is specified as "bit or" in an argument, `SsSeqCalledTbyT()` will not be called automatically, so the user must call `SsSeqCalledTbyT()` at the timing specified by the program.

Return value

None

Remarks

See also: `SsStart` (p. 14-88), `SsSeqCalledTbyT` (p. 14-50), `SetVideoMode` (see libetc), `SsSetTickCallback()`.

SsSetVoiceMask

Blocks the libsnd voice allocation system from using the specified voices.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	6/22/98

Syntax

```
void SsSetVoiceMask (
    unsigned long s_voice
)
```

Arguments

s_voice voices that libsnd allocation system will not have access. Specify the voices set in *s_voice* by ORing together SPU_0CH-SPU_23CH.

Explanation

Blocks the voices specified in the bit mask *s_mask* from being accessed by the libsnd voice allocation system. These voices would then be available for use via libspu calls or SsUtKeyOnV() calls. This function differs from SsSetReservedVoice() in that a non-continuous block of voices may be available to the libsnd voice allocation system. Ex: SsSetVoiceMask(6) will give the libsnd voice allocations system access to 22 voices - voices 0, 2, and 4-23. SsSetReservedVoice(22) will give the libsnd voice allocation system access to voices 0-21. The two functions may be used together, so that individual voices within the limits set by SsSetReservedVoice() may be blocked, but an easier method would be to just make one call to SsSetVoiceMask().

Return value

None

Remarks

SsAllocateVoices() does not include code which will block these voices from the libsnd voice allocation system.

See also: SsGetVoiceMask(), SsSetReservedVoice().

SsSetVoiceSettings

Sets the internal libsnd variables for the given voice.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

```
void SsSetVoiceSettings (
long voice,
SndVoiceStats *Snd_v_attr
)
```

Arguments

voice Voice number (0-23)
**Snd_v_attr* Voice attributes to be set

Explanation

Hooks into the internal libsnd variables before key on of sound so that libsnd functions such as SsUtPitchBend() can be used to alter the *voice* after key on.

The voice attributes *Snd_v_attr* must be set as follows:

vagld

VAG number which tone to be keyed on points to. Can be obtained by using SsUtGetVagAtr().

vabld

VAB in which information for tone to be keyed on resides.

pitch

Original playback rate of tone. May be obtained by using SsFindPitch() or by creating a user-defined pitch lookup table.

vol

Maximum volume tone will be played back at. Choose the greater value between left and right channel. Negative values are not valid (*vol* is invalid when used with 3d sound), and values range from 0-127.

pan

Relative pan value at which tone will be keyed on at. Not valid with 3d sound. Use the formula $pan = volr * 64 / voll$ if $volr > voll$ and $pan = \max \text{ volume (either } 0x3ff \text{ or } 127) - voll * 64 / volr$ if $voll > volr$. If $voll = volr$, $pan = 64$.

note

Note at which tone will be keyed on.

tone

Tone number to be keyed on. Determined by comparing the desired note with the min and max members of VagAtr for all tones within the specified program and *vabld*.

prog_num

The program number within which the tone information resides.

prog_actual

The “real” program number within which the tone information resides. The “real” number is only incremented by valid programs (programs containing one or more tones) and so may differ from the program number. Example: In a VAB with valid programs 0-10 and 100-127, the *prog_num* of program

127 = 127, while the *prog_actual* of program 127 = 38. This number is used to calculate the offset of tone information in the VAB header and may be obtained using the function `SsGetActualProgFromProg()`.

Return value

None

Remarks

This function should only be used as part of the series:

- `SsBlockVoiceAllocation()`
- `SsAllocateVoices()`
- `SsSetVoiceSettings()`
- `SsQueueRegisters()`
- `SsQueueKeyOn()`
- `SsQueueReverb()`
- `SsUnBlockVoiceAllocation()`

See also: `SndRegisterAttr` (p. n-nn), `SndVoiceStats` (p. n-nn), `SndVolume2` (p. n-nn), `SsAllocateVoices` (p. n-nn), `SsBlockVoiceAllocation` (p. n-nn), `SsGetActualProgFromProg` (p. n-nn), `SsPitchFromNote` (p. n-nn), `SsQueueKeyOn` (p. n-nn), `SsQueueRegisters` (p. n-nn), `SsQueueReverb` (p. n-nn), `SsUnBlockVoiceAllocation` (p. n-nn), `SsVoiceCheck` (p. n-nn).

SsStart

Starts the sound system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

void SsStart(*void*)

Arguments

None

Explanation

Carries out the sound system start process.

When the mode is set to call SsSeqCalledTbyT () automatically by SsSetTickMode (), SsSeqCalledTbyT () is called in each Tick after calling this function.

Return value

None

Remarks

See also: SsEnd (p. 14-1), SsSetTickMode (p. 14-86), SsSeqCalledTbyT (p. 14-50).

SsStart2

Starts the sound system (VSyncCallback version).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	7/31/96

Syntax

void SsStart(void)

Arguments

None

Explanation

Carries out the sound system start process.

When the mode is set to call SsSeqCalledTbyT() automatically by SsSetTickMode(), SsSeqCalledTbyT() is called in each Tick after calling this function.

Set SsSeqCalledTbyT() in VSyncCallback() only when SS_TICK60 on NTSC or SS_TICK50 on PAL is specified in SsSetTickMode(). The setting of SsSeqCalledTbyT() in other Tick modes is the same as SsStart()

Return value

None

Remarks

See also: SsStart (p. 14-88), SsEnd (p. 14-19), SsSetTickMode (p. 14-86), SsSeqCalledTbyT (p. 14-50).

SsUnBlockVoiceAllocation

Releases voice allocation system used by SsUtKeyOn(), SsUtKeyOnV(), and MIDI key on commands.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

```
char SsUnBlockVoiceAllocation(void)
```

Arguments

None

Explanation

Releases the voice allocation system used by SsUtKeyOn(), SsUtKeyOnV(), SsAllocateVoices(), and MIDI key on commands. Must follow every call to SsBlockVoiceAllocation().

Return value

Returns 1 if successful.

Returns -1 if voice allocation system was not currently blocked.

Remarks

This function should only be used as part of the series:

- SsBlockVoiceAllocation()
- SsAllocateVoices()
- SsSetVoiceSettings()
- SsQueueRegisters()
- SsQueueKeyOn()
- SsQueueReverb()
- SsUnBlockVoiceAllocation()

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsAllocateVoices (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsGetActualProgFromProg (p. n-nn), SsPitchFromNote (p. n-nn), SsQueueKeyOn (p. n-nn), SsQueueRegisters (p. n-nn), SsQueueReverb (p. n-nn), SsSetVoiceSettings (p. n-nn), SsVoiceCheck (p. n-nn).

SsUtAllKeyOff

Keys off all voices.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsUtAllKeyOff (
short mode
)
```

Arguments

mode Always 0

Explanation

Forcibly keys off all voices used by libsnd.

Return value

None

Remarks

See also:

SsUtAutoPan

Automatically changes panning.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
short SsUtAutoPan(vc, start_pan, end_pan, delta_time)
short vc;
short start_pan;
short end_pan;
short delta_time;
```

Arguments

<i>vc</i>	Voice number (0-23)
<i>start_pan</i>	Panning change starting value (0-127)
<i>end_pan</i>	Panning change starting value (0-127)
<i>delta_time</i>	Change starting time (in units of 1/60 sec, to a maximum of 180 Seconds) (0-10800)

Explanation

Linearly changes the panning from *start_pan* to *end_pan* at *delta_time* (1/60 sec increments) for voice *vc*.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtKeyOn (p. 14-113), SsUtKeyOnV (p. 14-114), SsVoKeyOn (p. 14-140).

SsUtAutoVol

Automatically changes voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

short SsUtAutoVol(*vc*, *start_vol*, *end_vol*, *delta_time*)

short *vc*;

short *start_vol*;

short *end_vol*;

short *delta_time*; ;

Arguments

<i>vc</i>	Voice number (0-23)
<i>start_vol</i>	Volume change starting value (0-127)
<i>end_vol</i>	Volume change starting value (0-127)
<i>delta_time</i>	Change starting time (in units of 1/60 sec, to a maximum of 180 Seconds) (0-10800)

Explanation

Linearly changes the volume from *start_vol* to *end_vol* at *delta_time* (1/60 sec increments) for voice *vc*.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtKeyOn (p. 14-113), SsUtKeyOnV (p. 14-114), SsVoKeyOn (p. 14-140).

SsUtChangeADSR

Changes ADSR.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	5/22/97

Syntax

```
short SsUtChangeADSR(vc, vabld, prog, old_note, adsr1, adsr2)
```

```
short vc;
```

```
short vabld;
```

```
short prog;
```

```
short old_note;
```

```
unsigned short adsr1;
```

```
unsigned short adsr2;
```

Arguments

<i>vc</i>	Voice number (0-23)
<i>vabld</i>	VAB number (0-31) from the return value of the function SsVabOpenHead
<i>prog</i>	Program number (0-127)
<i>old_note</i>	Previous pitch specification in half-tone units (note number)(0-127)
<i>adsr1</i>	ADSR1
<i>adsr2</i>	ADSR2

Explanation

Changes the ADSR of the key on voice using SsUtKeyOn().

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtKeyOn (p. 14-113), SsUtKeyOnV (p. 14-114), SsVoKeyOn (p. 14-140), SsVabOpenHead().

SsUtChangePitch

Changes the pitch.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

short SsUtChangePitch(*voice, vabld, prog, old_note, old_fine, new_note, new_fine*)

short *voice*;

short *vabld*;

short *prog*;

short *old_note*;

short *old_fine*;

short *new_note*;

short *new_fine*;

Arguments

<i>voice</i>	Voice number (0-23)
<i>vabld</i>	VAB number (0-31) from the return value of the function SsVabOpenHead
<i>prog</i>	Program number (0-127)
<i>old_note</i>	Previous pitch specification in semitones (note number) (0-127)
<i>old_fine</i>	Previous fine pitch specification (100/127 cents) (0-127)
<i>new_note</i>	New pitch specification in semitones (note number) (0-127)
<i>new_fine</i>	New fine pitch specification (100/127 cents) (0-127)

Explanation

Changes the pitch of the voice.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtPitchBend (p. 14-115), SsUtKeyOn (p. 14-113), SsUtKeyOnV (p. 14-114), SsVoKeyOn (p. 14-140).

SsUtFlush

Executes KeyOn/KeyOff requests that have been queued. (Flushing)

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

void SsUtFlush(*void*)

Arguments

None

Explanation

Executes KeyOn/KeyOff requests that have been queued.

Normally, flushing is performed by an automatic interrupt of Sound Library (when the mode is set by SsSetTickMode to mode other than SS_NOTICK) or by a clear call of SsSeqCalledTbyT (when the mode is set by SsSetTickMode to SS_NOTICK).

However, if neither of these is used, use this function for flushing.

An interval of at least 1/44100 sec must be inserted before calling this function.

Return value

None

Remarks

See also:

SsUtGetDetVVol

Obtains a detailed value of voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
short SsUtGetDetVVol(vc, *detvoll, *detvolr)
short vc;
short *detvoll;
short *detvolr;
```

Arguments

vc Voice number (0-23)
detvoll Pointer to detailed volume, left (0-16383)
detvolr Pointer to detailed volume, right (0-16383)

Explanation

Returns the detailed value of the voice volume.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtSetDetVVol (p. 14-118), SsUtKeyOn (p. 14-113), SsUtKeyOnV (p. 14-114), SsVoKeyOn (p. 14-140).

SsUtGetProgAtr

Gets a program attribute table.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
short SsUtGetProgAtr(vabld, progNum, *progrptr)
```

```
short vabld;
```

```
short progNum;
```

```
ProgAtr *progrptr;
```

Arguments

vabld VAB number (0-31) from the return value of the function SsVabOpenHead
progNum Program number (0-127)
progrptr Pointer to program attribute table

Explanation

Specifies a VAB number and a program number, and returns the VAB attribute table to *progrptr*.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtSetProgAtr (p. 14-115), ProgAtr (p. 14-4).

SsUtGetReverbType

Obtains a reverb type.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

short SsUtGetReverbType(void)

Arguments

None

Explanation

Obtains the current reverb type value.

Return value

Current reverb type value.

Remarks

See also: SsUtSetReverbType (p. 14-104).

SsUtGetVabHdr

Returns VAB attribute header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
short SsUtGetVabHdr(vabId, *vabhdrptr)
```

```
short vabId;
```

```
VabHdr *vabhdrptr;
```

Arguments

vabId VAB number (0-31) from the return value of the function SsVabOpenHead
vabhdrptr Pointer to VAB attribute header

Explanation

Specifies the VAB number (the return value of SsVabOpenHead()) and returns the VAB attribute header to *vabhdrptr*.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: VabHdr (p. 14-10).

SsUtGetVagAddr

Returns an SPU buffer address stored by VAG.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	7/31/96

Syntax

long SsUtGetVagAddr(*vabld*, *vagld*)

short *vabld*;

short *vagld*;

Arguments

vabld VAB data id

vagld VAG data id

Explanation

Given VAB id (0-15) and VAG id (1-254), this function returns a 32-bit SPU buffer address (as bytes) stored by VAG.

Return value

Returns an SPU buffer address stored by VAG.

Remarks

See also: SsVabOpenHead (p. 14-131).

SsUtGetVagAddrFromTone

Returns the SPU buffer address where VAG data is stored.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.3	7/31/96

Syntax

```
unsigned long SsUtGetVagAddrFromTone (
short vabid,
short progid,
short toneid
)
```

Arguments

vabid VAB id
progid Program number
toneid Tone number

Explanation

This function returns the address in the sound buffer where the VAG wave form data with the specified VAB id, program number, and tone number are transferred.

Return value

Address in the sound buffer. If it fails, it returns -1.

Remarks

See also:

SsUtGetVagAtr

Returns a tone attribute table (VagAtr).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
short SsUtGetVagAtr(vabld, progNum, toneNum, *vagatrptr)
short vabld;
short progNum;
short toneNum;
VagAtr *vagatrptr;
```

Arguments

<i>vabld</i>	VAB number (0-31) from the return value of the function SsVabOpenHead
<i>progNum</i>	Program number (0-127)
<i>toneNum</i>	Tone number (0-15)
<i>vagatrptr</i>	Pointer to tone attribute table

Explanation

Specifies a VAB number, a program number, and a tone number, and returns a tone attribute table to *vagatrptr*.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtSetVagAtr (p. 14-125), VagAtr (p. 14-11).

SsUtGetVBaddrInSB

Returns the address inside the sound buffer to which VAB data specified by VAB id has been transferred.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

unsigned long SsUtGetVBaddrInSB(*vabid*)
 short *vabid*;

Arguments

vabid VAB id

Explanation

Returns the address inside the sound buffer to which VAB data specified by VAB id has been transferred.

Return value

Address inside the sound buffer. Returns -1 if unsuccessful.

Remarks

See also:

SsUtGetVVol

Obtains voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
short SsUtGetVVol(vc, *voll, *volr)
short vc;
short *voll;
short *volr;
```

Arguments

vc Voice number (0-23)
voll Pointer to volume, left (0-127)
volr Pointer to volume, right (0-127)

Explanation

Returns a volume value for a voice.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtSetVVol (p. 14-126), SsUtKeyOn (p. 14-113), SsUtKeyOnV (p. 14-114), SsVoKeyOn (p. 14-140).

SsUtKeyOff

Keys off voice.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

short SsUtKeyOff(*voice, vabld, prog, tone, note*)

short *voice*;

short *vabld*;

short *prog*;

short *tone*;

short *note*;

Arguments

voice Voice number (0-23) access number

vabld VAB number (0-31) from the return value of the function SsVabOpenHead

prog Program number (0-127)

tone Tone number (0-15)

note Pitch specification in half-tone units (note number) (0-127)

Explanation

Keys off the voice.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtKeyOn (p. 14-113), SsUtKeyOnV (p. 14-114), SsVoKeyOn (p. 14-140).

SsUtKeyOffV

Keys off the voice specified by the voice number.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

short SsUtKeyOffV(*voice*)

short *voice*;

Arguments

voice Voice number (0-23)

Explanation

Keys off the voice specified by the voice number (0-23).

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtKeyOnV (p. 14-114).

SsUtKeyOn

Keys on voice.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

short SsUtKeyOn(*vabld*, *prog*, *tone*, *note*, *fine*, *voll*, *volr*)

short *vabld*;

short *prog*;

short *tone*;

short *note*;

short *fine*;

short *voll*;

short *volr*;

Arguments

vabld VAB number (0-31) from the return value of the function SsVabOpenHead

prog Program number (0-127)

tone Tone number (0-15)

note Pitch specification in semitones (note number) (0-127)

fine Detailed pitch specification (100/127 cents) (0-127)

voll Volume, left (0-127)

volr Volume, right (0-127)

Explanation

Keys on the voice specified by the VAB number, the program number (0-127), and the tone number (0-15) at the specified pitch and volume, and returns the allocated voice number.

Return value

Returns the voice number (0-23) used for KeyOn.

Returns -1 if unsuccessful.

Remarks

See also: SsUtKeyOff (p. 14-111).

SsUtKeyOnV

Keys on the voice specified by voice number.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

short SsUtKeyOnV(*voice, vabld, prog, tone, note, fine, voll, volr*)

short *voice*;

short *vabld*;

short *prog*;

short *tone*;

short *note*;

short *fine*;

short *voll*;

short *volr*;

Arguments

voice Voice number (0-23)

vabld VAB number (0-31) from the return value of the function SsVabOpenHead

prog Program number (0-127)

tone Tone number (0-15)

note Pitch specification in semitones (note number) (0-127)

fine Detailed pitch specification (100/127 cents) (0-127)

voll Volume, left (0-127)

volr Volume, right (0-127)

Explanation

Keys on the voice specified by the voice number (0-23), the VAB number, the program number (0-127), and the tone number (0-15) at the specified pitch and volume, and returns the allocated voice number.

Return value

Returns the voice number (0-23) used for KeyOn.

Returns -1 if unsuccessful.

Remarks

See also: SsUtKeyOffV (p. 14-112).

SsUtPitchBend

Applies a pitch bend.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

short SsUtPitchBend(*voice, vabld, prog, note, pbend*)

short *voice*;

short *vabld*;

short *prog*;

short *note*;

short *pbend*;

Arguments

voice Voice number (0-23)

vabld VAB number (0-31) from the return value of the function SsVabOpenHead

prog Program number (0-127)

note Pitch specification in half-tone units (note number) (0-127)

pbend Pitch-bend value (0-127)

Explanation

Applies a pitch bend (0-127, 64:center) to the voice.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtChangePitch (p. 14-95), SsUtKeyOn (p. 14-113), SsUtKeyOnV (p. 14-114), SsVoKeyOn (p. 14-140).

SsUtReverbOff

Turns off Reverb.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

void SsUtReverbOff(void)

Arguments

None

Explanation

Turns off Reverb.

Return value

None

Remarks

See also: SsUtReverbOn (p. 14-117).

SsUtReverbOn

Turns on Reverb.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsUtReverbOn(void)
```

Arguments

None

Explanation

Turns on Reverb at the Type and Depth. Set by SsUtSetReverbType and SsUtSetReverbDepth.

Return value

None

Remarks

See also: SsUtReverbOff (p. 14-116), SsUtSetReverbType (p. 14-123), SsUtSetReverbDepth (p. 14-121).

SsUtSetDetVVol

Sets a detailed value of voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
short SsUtSetDetVVol(vc, detvoll, detvolr)
short vc;
short detvoll;
short detvolr;
```

Arguments

vc Voice number (0-23)
detvoll Detailed volume, left (0-16383)
detvolr Detailed volume, right (0-16383)

Explanation

Sets the detailed value of voice volume.

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtGetDetVVol (p. 14-101), SsUtKeyOn (p. 14-113), SsUtKeyOnV (p. 14-114), SsVoKeyOn (p. 14-140).

SsUtSetProgAtr

Sets a program attribute table.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
short SsUtSetProgAtr(vabld, progNum, *progrptr)
short vabld;
short progNum;
ProgAtr *progrptr;
```

Arguments

vabld VAB number (0-31) from the return value of the function SsVabOpen()
progNum Program number (0-127)
progrptr Pointer to program attribute table

Explanation

Specifies a VAB number and a program number, and changes the program attribute table, *progrptr*.

- Change allowed: mvol, mpan, prior, mode, attr
- Change not allowed: tones, reserved0, reserved 1, reserved 2

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtGetProgAtr (p. 14-101).

SsUtSetReverbDelay

Sets a Delay volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsUtSetReverbDelay(delay)  
short delay;
```

Arguments

delay 0-127

Explanation

Sets a delay volume for using Echo and Delay type reverb.

Return value

None

Remarks

See also:

SsUtSetReverbDepth

Sets a reverb depth.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsUtSetReverbDepth(ldepth, rdepth)
```

```
short ldepth;
```

```
short rdepth
```

Arguments

ldepth Left depth (0-127)

rdepth Right depth (0-127)

Explanation

ldepth 0-127

rdepth 0-127

Sets a reverb depth.

Return value

None

Remarks

See also: SsUtGetReverbDepth (p. 14-104).

SsUtSetReverbFeedback

Sets a feedback volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
void SsUtSetReverbFeedback(feedback)
short feedback;
```

Arguments

feedback Feedback (0-127)

Explanation

Sets a feedback volume for using Echo and Delay type reverb.

Return value

None

Remarks

See also:

SsUtSetReverbType

Sets reverb type.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
short SsUtSetReverbType(type)
```

```
short type;
```

Arguments

type Reverb type

Table 14-15: Reverb Type Overview (See Sound Delicatessen DSP)

Type	Mode	Delay time	Feedback
SPU_REV_TYPE_OFF	off	X	X
SPU_REV_TYPE_ROOM	room	X	X
SPU_REV_TYPE_STUDIO_A	studio (small)	X	X
SPU_REV_TYPE_STUDIO_B	studio (med)	X	X
SPU_REV_TYPE_STUDIO_C	studio (big)	X	X
SPU_REV_TYPE_HALL	hall	X	X
SPU_REV_TYPE_SPACE	space echo	X	X
SPU_REV_TYPE_ECHO	echo	O	O
SPU_REV_TYPE_DELAY	delay	O	O
SPU_REV_TYPE_PIPE	pipe echo	X	X

Explanation

Sets reverb type.

When a reverb type is set, reverb depth is automatically set to 0. Because noise will occur as soon as depth is set if data remains in the reverb work area, follow the procedure below.

```
SsUtSetReverbType (SS_REV...);
```

```
SsUtReverbOn();
```

Wait for several seconds.

```
SsUtSetReverbDepth (64, 64);
```

Number and type are as shown in the table above.

Return value

If setting was correctly performed, the Type number that was set is returned.

If setting was not correctly performed, -1 is returned.

Remarks

See also: SsUtGetReverbType (p. 14-104).

SsUtSetVabHdr

Sets a VAB attribute header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

short SsUtSetVabHdr(*vabId*, **vabhdrptr*)

short *vabId*;

VabHdr **vabhdrptr*;

Arguments

vabId VAB number (0-31) from the return value of the function SsVabOpenHead
vabhdrptr Pointer to VAB attribute header

Explanation

Specifies the VAB number (the return value of SsVabOpenHead()) and changes the VAB attribute header, *vabhdrptr*.

- Setting allowed: mvol, pan, attr1, attr2 only
- Setting not allowed: form, ver, id, fsize, reserved0, ps, ts, vs, reserved 1

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtGetVabHdr (p. 14-105).

SsUtSetVagAtr

Sets a tone attribute table (VagAtr).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	7/31/96

Syntax

```
short SsUtSetVagAtr(vabId, progNum, toneNum, *vagatrptr)
```

```
short vabId;
```

```
short progNum;
```

```
short toneNum;
```

```
VagAtr *vagatrptr;
```

Arguments

<i>vabId</i>	VAB number (0-31) from the return value of the function SsVabOpen()
<i>progNum</i>	Program number (0-127)
<i>toneNum</i>	Tone number (0-15)
<i>vagatrptr</i>	Pointer to tone attribute table

Explanation

Specifies a VAB number, a program number, and a tone number, and changes a tone attribute table, *vagatrptr*.

Change allowed: Items in VagAtr that are not listed below.

Change not allowed: prog, vag, reserved1, reserved2, reserved[0-3]

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: SsUtGetVagAtr (p. 14-104), VagAtr (p. 14-11).

SsUtSetVVol

Sets voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	5/22/97

Syntax

```
short SsUtSetVVol (
short vc,
short voll,
short volr
)
```

Arguments

vc Voice number (0-23)
voll Volume, left (0-127)
volr Volume, right (0-127)

Explanation

Sets the left and right volumes of the specified voice, *vc*. Since *libsnd* uses an exponential volume calculation for sounds being keyed on, the input volumes *voll* and *volr* will be modified as follows:

$lvol = voll * voll / 127$
 $rvol = volr * volr / 127$

Return value

Returns 0 if successful. Returns -1 if unsuccessful.

Remarks

See also: *SsUtGetVVol* (p. 14-109), *SsUtKeyOn* (p. 14-113), *SsUtKeyOnV* (p. 14-114), *SsVoKeyOn* (p. 14-140).

SsVabClose

Closes VAB data file.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	5/22/97

Syntax

```
void SsVabClos(vab_id)
short vab_id;
```

Arguments

vab_id VAB data id

Explanation

This function closes a VAB data file containing *vab_id*.

Execute it after closing the SEQ/SEP which use the specified VAB data ID.

Return value

None

Remarks

See also: SsVabOpen (p. 14-128), SsVabTransBody(), SsVabTransBodyPartly().

SsVabFakeBody

Recognizes sound source data in the sound buffer as the given VAB ID. This function does not perform any transfer.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	02/15/98

Syntax

```
short SsVabFakeBody(vabid)
short vabid;
```

Arguments

vabid VAB id

Explanation

This function recognizes sound source data in the sound buffer after SsVabFakeHead has recognized a header list on main RAM.

Although this function does perform VAB ID verification, it does not perform the actual transfer. Instead, it sets the internal state of the library to "Transferred to SPU."

It is not necessary to use SsVabTransCompleted after calling this function.

SsVabFakeHead() and SsVabFakeBody() perform sound buffer placement control in the program and they must be used together with SsVabOpenHeadSticky(). They cannot be used together with SsVabOpenHead().

Return value

VAB Identifying number. Returns -1 if unsuccessful.

Remarks

See also: SsVabFakeHead (p. 14-129), SsVabOpenHeadSticky (p. 14-132), SsVabTransBody (p. 14-134), SsVabTransBodyPartly (p. 14-135).

SsVabFakeHead

Rerecognizes a sound source header list.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	02/15/98

Syntax

```
short SsVabFakeHead(*addr, vabid, sbaddr)
unsigned char *addr;
short vabid;
unsigned long sbaddr;
```

Arguments

addr Pointer to VH leading address
vabid Desired VAB ID. If "-1", the library will make the allocation.
sbaddr Address inside the sound buffer, to which VB is being transferred.

Explanation

Rerecognizes the sound source header in the main memory, and sets the previously read VH data in the state that can be used by the library again.

Specify a VAB ID for opening. When VAB ID is -1, the function searches for an empty VAB ID (0 - 15) and allocates.

The user must specify the leading address in *sbaddr* for the area inside the sound buffer to which VB is being transferred.

SsVabFakeHead() and SsVabFakeBody() perform sound buffer placement control in the program and they must be used together with SsVabOpenHeadSticky(). They cannot be used together with SsVabOpenHead().

Return value

VAB Identifying number. Returns -1 if unsuccessful.

Remarks

See also: SsVabFakeBody (p. 14-128), SsUtGetVBAddrInSB (p. 14-107), SsVabOpenHead (p. 14-131), SsVabOpenHeadSticky (p. 14-132).

SsVabOpen

Opens VAB data.

NOTE: This function is no longer recommended for use. Instead, use SsVabOpenHead and SsVabTransBody or SsVabTransfer.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	7/31/96

Syntax

```
short SsVabOpen(*addr, *vab_header)
unsigned char *addr;
VabHdr *vab_header;
```

Arguments

addr Pointer to start address of VAB data in main storage
vab_header Pointer to address to VAB header structure corresponding to VAB id

Explanation

It analyses the VAB data header which is in the main memory, stores the header value in *vab_header*, and returns the VAB id that identifies the VAB given as the function's Return value. At the same time, it transmits to the SPU local memory the VAG data group (wave form) data contained in VAB.

Return value

It is the VAB id which identifies the given VAB. It is -1 in the event of failure.

Note:

This function is no longer recommended for use. Instead, use SsVabOpenHead and SsVabTransBody or SsVabTransfer.

Remarks

See also: SsVabClose (p. 14-125), SsVabOpenHead (p. 14-131), SsVabTransBody (p. 14-134).

SsVabOpenHead

Recognizes a sound source header list.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	5/22/97

Syntax

```
short SsVabOpenHead(*addr, vabid)
unsigned char *addr;
short vabid;
```

Arguments

addr Pointer to VAB data starting address
vabid VAB ID

Explanation

Recognizes a sound source header list in the main memory.

Sets the table in the main memory so that it can be used by the Sound Library. Specify a VAB ID for opening. When VAB ID is -1, the function searches for an empty VAB ID (0 - 15) and allocates it. SsVabTransCompleted () should be executed to confirm transmission completion.

Return value

VAB identification number. Returns -1 if unsuccessful.

Remarks

See also: SsVabTransBody (p. 14-134), SsVabTransBodyPartly (p. 14-135), SsVabOpenHeadSticky (p. 14-132), SsVabTransfer (p. 14-137).

SsVabOpenHeadSticky

Recognizes a sound source header list. (.VB transfer address specification).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	10/01/97

Syntax

```
short SsVabOpenHeadSticky(*addr, vabid, sbaddr)
unsigned char *addr;
short vabid;
unsigned long sbaddr;
```

Arguments

addr Pointer to leading address of VAB data in the main memory
vabid Desired VAB ID or -1
sbaddr Leading address inside the sound buffer to be used when transferring VabBody (.VB) to the sound buffer

Explanation

Recognizes a sound source header list in the main memory.

Sets the table in the main memory in the state that is usable by Sound Library. Specify a VAB ID for opening. When VAB ID is -1, the function searches for an empty VAB ID (0-15) and allocates.

Specify for *sbaddr* the leading address inside the sound buffer for transferring VabBody (.VB) to the sound buffer, within the range of 0x1010 to 0x7fff. When doing so, take .VB size into consideration and specify the address so that it will not be transferred into the reverb work area.

SsVabTransBody/SsVabTransBodyPartly that is called later transfers VabBody to *sbaddr*.

When using this function, because consistency cannot be maintained for the sound buffer memory management, SsVabOpenHead will not be able to be used when opening other VAB (.VH). Use this function to open all .VH.

When using this function, the consistency of sound buffer placement info which is managed by SpuMalloc() and related functions cannot be maintained. Consequently, when using SsVabOpenHead-related functions which use SpuMalloc(), as well as SsVabOpenHeadSticky (this function), the functions listed below must not be used within the program. All of these perform .VH open operations using SsVabOpenHeadSticky (this function) (Except for the case of setting placement info in SsVabOpenHeadSticky or managing the .VB region in SpuMalloc completely by the user.)

SsVabOpenHead, SsVabTransfer, SpuMalloc, SpuFree,
 SpuMallocWithStartAddr, SpuReserveReverbWorkArea

Return value

VAB identifying number. Returns -1 if unsuccessful

When -1 is returned, the cause is shown below.

1. The specified VabID was already open. In this case, an error is generated.
2. The specified VabID is not within the range 0-15. VabID must be within this range.
3. The number of VABs simultaneously open was exceeded. Up to 16 VABs can be open simultaneously.
4. Could not confirm the end of transfer using SsVabTransCompleted().

VB must be transferred to the sound buffer using a SsVabTransBody-type function.

It is necessary to confirm that the transfer has completed using `SsVabTransCompleted()`. If the completion of transfer cannot be confirmed, an error is reported when an attempt is made to open the VAB using a `SsVabOpenHead`-type function.

Remarks

See also: `SsVabOpenHead` (p. 14-131), `SsVabTransBody` (p. 14-134), `SsVabTransBodyPartly` (p. 14-135), `SsVabTransfer` (p. 14-137).

SsVabTransBody

Transfers sound source data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	02/15/98

Syntax

```
short SsVabTransBody(*addr, vabid)
```

```
unsigned char *addr;
```

```
short vabid;
```

Arguments

addr Pointer to VAB data leading address

vabid VAB ID

Explanation

After SsVabOpenHead is used for recognizing a header list, SsVabTransBody starts the transfer of the sound source data (VAB body) in the main memory to the SPU local memory.

The starting address within the sound buffer is specified within the 0x1010-0x7fff range when transferring VabBody(.VB) to the sound buffer in *addr*. At such times, the address must be specified so that it is not transferred into the area occupied by the reverb work area, giving consideration to the size of the .VB.

Return value

VAB identifying number. Returns -1 if unsuccessful.

Remarks

See also: SsVabOpenHead (14-131), SsVabTransBodyPartly (14-135), SsVabOpenHeadSticky (p. 14-132), SsVabTransfer (p. 14-137).

SsVabTransBodyPartly

Transfers sound source data in segments.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	02/15/98

Syntax

```
short SsVabTransBodyPartly(*addr, bufsize, vabid)
unsigned char *addr;
unsigned long bufsize;
short vabid;
```

Arguments

addr Pointer to starting address of the segment transfer buffer
bufsize Buffer size
vabid VAB ID

Explanation

Starts transfer to the SPU sound buffer of main memory sound source data (VAB body) whose data header list is recognized using SsVabOpenHead().

By continuously calling SsVabTransBodyPartly() while sequentially copying part of the sound source (VAB body) into the area possessing a *bufsize* indicated by *addr* (can be specified from 0x1010), transfers may be made to a contiguous area within the sound buffer using only a limited area in main memory.

In order to ensure continuity of transfer, you must use SsVabTransCompleted() to verify whether each transfer has been completed, after SsVabTransBodyPartly() has been called.

Return value

Transfer results return the following values.

Table 14-16

Return value	Status
-2	The size of the sound source data (VAB body) inherited from SsVabOpenHead() has not been completely transferred
-1	Transfer failed
vabid	Transfer successful

Remarks

See also: SsVabOpenHead (p. 14-131), SsVabTransBody (p. 14-134), SsVabOpenHeadSticky (p. 14-132), SsVabTransfer (p. 14-137).

SsVabTransCompleted

Gets VAB data transfer state.

Library	Header File	Introduced	Documentation Date
libsnd.lib	libsnd.h	3.0	7/31/96

Syntax

```
short SsVabTransCompleted(immediateFlag)  
short immediateFlag;
```

Arguments

immediateFlag Transfer status recognition flag

Explanation

Returns an indication of whether data transfer to SPU local memory has terminated.
immediateFlag may be specified with the following values:

Table 14-17

ImmediateFlag	Action
SS_IMMEDIATE	Immediately returns transfer state
SS_WAIT_COMPLETED	Loops until transfer is completed

Return value

Returns "1" if the transfer has been completed. Returns "0" if the transfer is ongoing.

Remarks

See also: SsVabOpenHead (p. 14-131), SsVabOpenHeadSticky (p. 14-132), SsVabTransfer (p. 14-137).
SsVabTransBody (p. 14-134), SsVabTransBodyPartly (p. 14-135).

SsVabTransfer

Recognizes and transfers sound source data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	02/15/98

Syntax

```
SsVabTransfer (  
  unsigned char *vh_addr,  
  unsigned char *vb_addr,  
  short vabid,  
  short i_flag  
)
```

Arguments

vh_addr Pointer to starting address of VH data
vb_addr Pointer to starting address of VB data
vabid VAB ID number
i_flag = SS_IMMEDIATE...Immediately returns return value (VAB ID number)
 = SS_WAIT_COMPLETED...Waits until transfer is completed

Explanation

This function recognizes a sound source header list(VH data) specified by *vh_addr* and transfers a sound source data(VB data) specified by *vb_addr*, to the SPU sound buffer. The VAB ID number is specified in the argument "vabid." When "vabid" is -1, the function searches for an empty VAB ID(0-15) and allocates. The "i_flag" determines whether the function should wait until transfer is completed or return immediately after the transfer starts (then checks with SsVabTransCompleted).

The starting address within the sound buffer is specified within the 0x1010-0x7fff range when transferring VabBody(.VB) to the sound buffer in *vb_addr*. At such times, the address must be specified so that it is not transferred into the area occupied by the reverb work area, giving consideration to the size of the .VB.

Return value

VAB ID number for successful return.

For error case the following value is returned.

- 1 VAB ID cannot be allocated or invalid VH
- 2 Invalid VB
- 3 and lower Other error

Remarks

See also: SsVabOpenHead (p. 14-131), SsVabOpenHeadSticky (p. 14-132), SsVabTransBody (p. 14-134), SsVabTransBodyPartly (p. 14-135), SsVabTransCompleted (p. 14-136).

SsVoiceCheck

Verifies that the tone information played by a voice that is to be modified is the intended tone information.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	10/01/97

Syntax

```
short SsVoiceCheck (
long voice,
long vabld,
short note
)
```

Arguments

voice Voice number containing tone information to be verified (0-23)
vabld The upper 8 bits of the lower 2 bytes of *vabld* contain the vab identification number as returned by SsVabOpenHead(), and the lower 8 bits of the lower 2 bytes of *vabld* contain the program number containing the tone information to be verified
note The note at which the tone to be verified was originally keyed on

Explanation

Verifies that the tone information played by a voice that is to be modified (key off, reverb change, pitch change etc.) is the intended tone information; that is, that the voice was not allocated to a different tone than the tone specified by *vabld* and *note*.

Return value

Returns 0 if successful.

Returns -1 if tone information is different than tone specified by *vabld* and *note* or *voice* is out of range.

Remarks

Should be called before each call to SsQueueRegisters() after key on of sound has occurred.

See also: SndRegisterAttr (p. n-nn), SndVoiceStats (p. n-nn), SndVolume2 (p. n-nn), SsAllocateVoices (p. n-nn), SsBlockVoiceAllocation (p. n-nn), SsGetActualProgFromProg (p. n-nn), SsPitchFromNote (p. n-nn), SsQueueKeyOn (p. n-nn), SsQueueRegisters (p. n-nn), SsQueueReverb (p. n-nn), SsSetVoiceSettings (p. n-nn), SsUnBlockVoiceAllocation (p. n-nn).

SsVoKeyOff

Key off.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	10/01/97

Syntax

```
long SsVoKeyOff(vab_pro, pitch)
long vab_pro;
long pitch;
```

Arguments

vab_pro VAB data id and program number
pitch Pitch

Explanation

Of the lower 16 bits of *vab_pro*, the upper 8 bits are used for VAB id, and the lower 8 bits specify a program number. Of the lower 16 bits of *pitch*, the upper 8 bits specify a key number in MIDI standard. To specify a finer pitch, specify a key number in the lower 8 bits of pitch in 1/128 semitones.

Return value

The keyed off voice number is returned.

Remarks

See also: SsVoKeyOn (p. 14-140).

SsVoKeyOn

Key on.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	10/01/97

Syntax

```
long SsVoKeyOn(vab_pro, pitch, voll, volr)
long vab_pro;
long pitch;
unsigned short voll;
unsigned short volr;
```

Arguments

vab_pro VAB data id and program number
pitch Pitch
voll L channel volume
volr R channel volume

Explanation

Of the lower 16 bits of *vab_pro*, the upper 8 bits are used for VAB id, and the lower 8 bits specify a program number. Of the lower 16 bits of *pitch*, the upper 8 bits specify a key number in MIDI standard. To specify a finer pitch, specify a key number in the lower 8 bits of *pitch* in 1/128 semitone units. The sound specified by *vab_pro* and *pitch* is keyed on at the specified *voll* and *volr*.

Return value

Returns which voices were keyed on.
KeyOn with volume set to 0 is the same as SsVoKeyOff.
AND the return value and SPU_xxCH (xx=0-23)

Table 14-18

Result of AND	Description
0	Voice not keyed on
1	Voice keyed on

Remarks

See also: SsVoKeyOff (p. 14-139).

Chapter 15: Basic Sound Library

Table of Contents

Structures	
SpuCommonAttr	15-5
SpuDecodeData	15-6
SpuEnv	15-7
SpuExtAttr	15-8
SpuReverbAttr	15-9
SpuStEnv	15-10
SpuStVoiceAttr	15-11
SpuVoiceAttr	15-12
SpuVolume	15-14
Functions	
SpuClearReverbWorkArea	15-15
SpuFlush	15-16
SpuFree	15-17
SpuGetAllKeysStatus	15-18
SpuGetCommonAttr	15-19
SpuGetCommonCDMix	15-20
SpuGetCommonCDReverb	15-21
SpuGetCommonCDVolume	15-22
SpuGetCommonMasterVolume	15-23
SpuGetCommonMasterVolumeAttr	15-24
SpuGetCommonMasterVolumeX	15-25
SpuGetIRQ	15-26
SpuGetIRQAddr	15-27
SpuGetKeyStatus	15-28
SpuGetMute	15-29
SpuGetNoiseClock	15-30
SpuGetNoiseVoice	15-31
SpuGetPitchLFOVoice	15-32
SpuGetReverb	15-33
SpuGetReverbModeDelayTime	15-34
SpuGetReverbModeDepth	15-35
SpuGetReverbModeFeedback	15-36
SpuGetReverbModeParam	15-37
SpuGetReverbModeType	15-38
SpuGetReverbVoice	15-39
SpuGetTransferMode	15-40
SpuGetTransferStartAddr	15-41
SpuGetVoiceADSR	15-42
SpuGetVoiceADSRAttr	15-43
SpuGetVoiceAR	15-44
SpuGetVoiceARAttr	15-45
SpuGetVoiceAttr	15-46
SpuGetVoiceDR	15-47
SpuGetVoiceEnvelope	15-48
SpuGetVoiceEnvelopeAttr	15-49
SpuGetVoiceLoopStartAddr	15-50
SpuGetVoiceNote	15-51
SpuGetVoicePitch	15-52
SpuGetVoiceRR	15-53
SpuGetVoiceRRAttr	15-54
SpuGetVoiceSampleNote	15-55
SpuGetVoiceSL	15-56
SpuGetVoiceSR	15-57

SpuGetVoiceSRAttr	15-58
SpuGetVoiceStartAddr	15-59
SpuGetVoiceVolume	15-60
SpuGetVoiceVolumeAttr	15-61
SpuGetVoiceVolumeX	15-62
Spulnit	15-63
SpulnitHot	15-64
SpulnitMalloc	15-65
SpulsReverbWorkAreaReserved	15-66
SpulsTransferCompleted	15-67
SpuMalloc	15-68
SpuMallocWithStartAddr	15-69
SpuNGetVoiceAttr	15-70
SpuNSetVoiceAttr	15-71
SpuQuit	15-72
SpuRead	15-73
SpuReadDecodedData	15-74
SpuReserveReverbWorkArea	15-76
SpuRGetAllKeysStatus	15-77
SpuRSetVoiceAttr	15-78
SpuSetCommonAttr	15-80
SpuSetCommonCDMix	15-82
SpuSetCommonCDReverb	15-83
SpuSetCommonCDVolume	15-84
SpuSetCommonMasterVolume	15-85
SpuSetCommonMasterVolumeAttr	15-86
SpuSetEnv	15-87
SpuSetIRQ	15-88
SpuSetIRQAddr	15-89
SpuSetIRQCallback	15-90
SpuSetKey	15-91
SpuSetKeyOnWithAttr	15-92
SpuSetMute	15-94
SpuSetNoiseClock	15-95
SpuSetNoiseVoice	15-96
SpuSetPitchLFOVoice	15-97
SpuSetReverb	15-98
SpuSetReverbDepth	15-99
SpuSetReverbModeDelayTime	15-100
SpuSetReverbModeDepth	15-101
SpuSetReverbModeFeedback	15-102
SpuSetReverbModeParam	15-103
SpuSetReverbModeType	15-105
SpuSetReverbVoice	15-106
SpuSetTransferCallback	15-107
SpuSetTransferMode	15-108
SpuSetTransferStartAddr	15-109
SpuSetVoiceADSR	15-110
SpuSetVoiceADSRAttr	15-111
SpuSetVoiceAR	15-112
SpuSetVoiceARAttr	15-113
SpuSetVoiceAttr	15-114
SpuSetVoiceDR	15-119
SpuSetVoiceLoopStartAddr	15-120
SpuSetVoiceNote	15-121
SpuSetVoicePitch	15-122
SpuSetVoiceRR	15-123

SpuSetVoiceRRAttr	15-124
SpuSetVoiceSampleNote	15-125
SpuSetVoiceSL	15-126
SpuSetVoiceSR	15-127
SpuSetVoiceSRAttr	15-128
SpuSetVoiceStartAddr	15-129
SpuSetVoiceVolume	15-130
SpuSetVoiceVolumeAttr	15-131
SpuStart	15-132
SpuStGetStatus	15-133
SpuStGetVoiceStatus	15-134
SpuStInit	15-135
SpuStQuit	15-136
SpuStSetPreparationFinishedCallback	15-137
SpuStSetStreamFinishedCallback	15-138
SpuStSetTransferFinishedCallback	15-139
SpuStTransfer	15-140
SpuWrite	15-142
SpuWrite0	15-143
SpuWritePartly	15-144

SpuCommonAttr

Common attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	7/31/96

Structure

```
typedef struct {
    unsigned long mask;
    SpuVolume mvol;
    SpuVolume mvolmode;
    SpuVolume mvolx;
    SpuExtAttr cd;
    SpuExtAttr ext;
} SpuCommonAttr;
```

Members

<i>mask</i>	Set mask
<i>mvol</i>	Master volume
<i>mvolmode</i>	Master volume mode
<i>mvolx</i>	Current master volume
<i>cd</i>	Cd input attributes
<i>ext</i>	External digital input attributes

Explanation

Used when setting/checking common attributes. The members needed for setting are set as bit values in *mask*.

Remarks

See also: SpuVolume (p. 15-14), SpuExtAttr (p. 15-7), SpuSetCommonAttr (p. 15-80), SpuGetCommonAttr (p. 15-19).

SpuDecodeData

Decode data.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	7/31/96

Structure

```
#define SPU_DECODEDATA_SIZE 0x200
```

```
typedef struct {
    short cd_left[SPU_DECODEDATA_SIZE];
    short cd_right[SPU_DECODEDATA_SIZE];
    short voice1[SPU_DECODEDATA_SIZE];
    short voice3[SPU_DECODEDATA_SIZE];
} SpuDecodeData;
```

Members

cd_left CD L channel data decoded by SPU
cd_right CD R channel data decoded by SPU
voice1 Voice 1 data decoded by SPU
voice3 Voice 3 data decoded by SPU

Explanation

Used when getting CD-ROM, voice 1 and voice 3 data decoded by the SPU.

The data which can actually be used is each member's first half 0x100 data or second half 0x100 data. This is determined by the return value of `SpuReadDecodeData()`.

Remarks

See also: `SpuReadDecodeData` (p. 15-74).

SpuEnv

SPU environment attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	5/22/97

Structure

```
typedef struct {
    unsigned long mask;
    unsigned long queueing;
} SpuEnv;
```

Members

mask Setting mask
queueing Event queueing

Explanation

Used to set the basic sound library environment. Currently, only queueing of the following events can be set
 - Key on, Key off, Pitch LFO voice setting, Noise Voice setting, and Reverb Voice setting.

The default value state is to perform the setting of these events immediately.

Remarks

See also: SpuSetEnv(), SpuFlush()

SpuExtAttr

External input attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	7/31/96

Structure

```
typedef struct {
    SpuVolume volume;
    long reverb;
    long mix;
} SpuExtAttr;
```

Members

volume Volume
reverb Reverb on/off
mix Mixing on/off

Explanation

Used when setting/checking CD and external digital input attributes.

Remarks

See also: SpuCommonAttr (p. 15-4), SpuVolume (p. 15-14).

SpuReverbAttr

Reverb attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    unsigned long mask;
    long mode;
    SpuVolume depth;
    long delay;
    long feedback;
} SpuReverbAttr;
```

Members

<i>mask</i>	Set mask
<i>mode</i>	Reverb mode
<i>depth</i>	Reverb depth
<i>delay</i>	DelayTime (ECHO, DELAY only)
<i>feedback</i>	Feedback (ECHO, DELAY only)

Explanation

Used when setting/checking reverb attributes. The members required at setting are set in the mask as bit values.

Remarks

See also: structure SpuVolume (p. 15-14), SpuSetReverbModeParam (p. 15-100), SpuGetReverbModeParam (p. 15-37), SpuSetReverbDepth (p. 15-99).

SpuStEnv

SPU streaming environment attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	5/22/97

Structure

```
typedef struct {
    long size;
    long low_priority;
    SpuStVoiceAttr voice[24];
} SpuStEnv
```

Members

size Stream buffer size

low_priority Determines priority of SPU Streaming compared to other CPU processes. Default is SPU_OFF. Setting *low_priority* to SPU_ON lowers SPU Streaming priority.

voice Each stream attribute set

Explanation

Used in SPU streaming library, streaming environment and each stream attribute setting.

Remarks

See also: SpuStVoiceAttr (p. 15-11), SpuStInit (p. 15-135).

SpuStVoiceAttr

SPU streaming voice attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    char status;
    char pad1;
    char pad2;
    char pad3;
    long last_size;
    unsigned long buf_addr;
    unsigned long data_addr;
} SpuStVoiceAttr;
```

Members

<i>status</i>	Stream status
<i>pad1</i>	Padding
<i>pad2</i>	Padding
<i>pad3</i>	Padding
<i>last_size</i>	The size of final data transfer ($last_size \leq (size / 2)$)
<i>buf_addr</i>	The start address of stream buffer
<i>data_addr</i>	The start address of stream in SPU RAM data in main RAM

Explanation

Holds each stream's attributes in the SPU streaming library.

Remarks

See also: SpuStEnv (p. 15-10), SpuStInit (p. 15-135).

SpuVoiceAttr

Voice attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Structure

```
typedef struct {
    unsigned long voice;
    unsigned long mask;
    SpuVolume volume;
    SpuVolume volmode;
    SpuVolume volumex;
    unsigned short pitch;
    unsigned short note;
    unsigned short sample_note;
    short envx;
    unsigned long addr;
    unsigned long loop_addr;
    long a_mode;
    long s_mode;
    long r_mode;
    unsigned short ar;
    unsigned short dr;
    unsigned short sr;
    unsigned short rr;
    unsigned short sl;
    unsigned short adsr1;
    unsigned short adsr2;
} SpuVoiceAttr;
```

Members

<i>voice</i>	Set voice (value is bit string)
<i>mask</i>	Set attribute bit (value is bit string)
<i>volume</i>	Volume
<i>volmode</i>	Volume mode
<i>volumex</i>	Current volume
<i>pitch</i>	Interval (set pitch)
<i>note</i>	Interval (set note)
<i>sample_note</i>	Interval (set note)
<i>envx</i>	Current envelope volume value
<i>addr</i>	Waveform data start address
<i>loop_addr</i>	Starting address of loop
<i>a_mode</i>	Attack rate mode
<i>s_mode</i>	Sustain rate mode
<i>r_mode</i>	Release rate mode
<i>ar</i>	Attack rate
<i>dr</i>	Decay rate
<i>sr</i>	Sustain rate
<i>rr</i>	Release rate
<i>sl</i>	Sustain level
<i>adsr1</i>	Same value as structure VagAtr adsr1
<i>adsr2</i>	Same value as structure VagAtr adsr2

Explanation

Used when setting/checking the attributes of each voice. The voice number is provided/obtained from the voice bit value, and the members needed for setting are set as bit values in the mask.

Note:

Constant macro names spelled SPU_ON, SPU_OFF have the same values as and are interchangeable with constant macros used in the program and spelled SpuOn, SpuOff.

Remarks

See also: structure SpuVolume (p. 15-14), SpuSetVoiceAttr (p. 15-114), SpuGetVoiceAttr (p. 15-46), SpuSetKeyOnWithAttr (p. 15-92).

SpuVolume

Volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	7/31/96

Structure

```
typedef struct {
    short left;
    short right;
} SpuVolume;
```

Members

left L channel value
right R channel value

Explanation

Used in attributes that require L channel/R channel values when setting/getting each voice.

Remarks

See also: SpuVoiceAttr (p. 15-12), SpuReverbAttr (p. 15-9), SpuExtAttr (p. 15-7), SpuCommonAttr (p. 15-4).

SpuClearReverbWorkArea

Clears reverb work area.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	5/22/97

Syntax

```
long SpuClearReverbWorkArea(rev_mode)
long rev_mode;
```

Arguments

rev_mode Reverb mode

Explanation

Clears the area occupied by the reverb work area corresponding to the reverb mode specified by *rev_mode*.

Regardless of whether or not it is reserved at this time, the function checks to see if the area is being used.

This operation uses synchronous DMA transfer, so depending on the reverb mode, some time may be needed.

This function should not be called while another Spu transfer is occurring, as the SpuTransferFinished callback is temporarily cleared inside this function and thus the end of the pending transfer may be missed.

Return value

If successful, 0 is returned.

SPU_ERROR is returned if the reverb work area corresponding to the reverb mode set by *rev_mode* is in use, or if the specified reverb mode value is wrong.

Remarks

See also: SpuSetReverbModeParam (p.15-103), SpuReserveReverbWorkArea (p. 15-76), SpuSetReverb (p. 15-98), SpuMalloc (p. 15-68), SpuMallocWithStartAddr (p. 15-69).

SpuFlush

Flushes queued events.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
unsigned long SpuFlush (
  unsigned long ev
)
```

Arguments

ev Event to be flushed

Explanation

This function flushes a queued event.

Set *ev* with bitwise inclusive ORed events to be flushed;

SPU_EVENT_KEY	Key ON/OFF
SPU_EVENT_PITCHLFO	Pitch LFO Voice Set
SPU_EVENT_NOISE	Noise Voice Set
SPU_EVENT_REVERB	Reverb Voice Set

When *ev* is set to SPU_EVENT_ALL, all events will be flushed.

Return value

Bitwise inclusive ORed value of the flushed event(s).

Remarks

See also: SpuSetEnv(), SpuSetKey(), SpuSetKeyOnWithAttr(), SpuSetPitchLFOVoice(), SpuSetNoiseVoice(), SpuSetReverbVoice().

SpuFree

Releases area allocated in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
void SpuFree(addr)
unsigned long addr;
```

Arguments

addr Start address of allocated area (in bytes)

Explanation

Releases area allocated in the sound buffer as indicated by the start address *addr*, and deletes that area's information from the management table.

Return value

None

Remarks

See also: SpuInitMalloc (p. 15-65), SpuMalloc (p. 15-68), SpuMallocWithStartAddr (p. 15-69).

SpuGetAllKeysStatus

Determines key on/off for voices in the designated range.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.0	7/31/96

Syntax

```
void SpuGetAllKeysStatus(*status)
char *status[24];
```

Arguments

status Pointer to the result of checking a voice

Explanation

Checks key on/key off and envelope status of all voices; checks actual key on/key off.
An error may result if multiple envelopes are set, and if volume goes to 0 in the course of changing envelope status.
The current key on/key off and envelope status of each voice is returned to status[24].

Table 15-1

Value	Status
SPU_ON	Key on status
	Not turned off by SpuSetKey
	Envelope not 0
SPU_ON_ENV_OFF	Key on status
	Not turned off by SpuSetKey
	Envelope 0
SPU_OFF_ENV_ON	Key off status
	Turned off by SpuSetKey
	Envelope not 0
SPU_OFF	Key off status
	Turned off by SpuSetKey
	Envelope 0

Return value

None

Remarks

See also: SpuSetKey (p. 15-91), SpuGetKeyStatus (p. 15-28), SpuRGetAllKeysStatus (p. 15-77).

SpuGetCommonAttr

Checks attributes common to all voices (infrequent change requests).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
void SpuGetCommonAttr(*attr)
SpuCommonAttr *attr;
```

Arguments

attr Pointer to attributes common to all voices

Explanation

Returns attributes common to all voices in *attr*. See `SpuSetCommonAttr()` for details.

Return value

None

Remarks

See also: `SpuSetCommonAttr` (p. 15-80), `SpuCommonAttr` (p. 15-4).

SpuGetCommonCDMix

Obtains CD input ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetCommonCDMix (
long *on_off
)
```

Arguments

on_off CD input on/off

Explanation

This function obtains the CD input on/off status.

It is equivalent to the process of obtaining the value of the `cd.mix` member of the `SpuCommonAttr` structure in `SpuGetCommonAttr()`.

Return value

None

Remarks

See also: `SpuGetCommonAttr()`, `SpuSetCommonCDMix()`.

SpuGetCommonCDReverb

Obtains CD input reverb ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetCommonCDReverb (
long *on_off
)
```

Arguments

on_off CD input reverb on/off

Explanation

This function obtains the CD input reverb on/off status.

It is equivalent to the process of obtaining the value of the `cd.reverb` member of the `SpuCommonAttr` structure in `SpuGetCommonAttr()`.

Return value

None

Remarks

See also: `SpuGetCommonAttr()`, `SpuSetCommonCDReverb()`.

SpuGetCommonCDVolume

Obtains CD input volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetCommonCDVolume (
short *cdvolL,
short *cdvolR
)
```

Arguments

cdvolL CD Input volume (left)
cdvolR CD Input volume (right)

Explanation

This function obtains the CD input volume.

It is equivalent to the process of obtaining the value of the cd.volume member of the SpuCommonAttr structure in SpuGetCommonAttr().

Return value

None

Remarks

See also: SpuGetCommonAttr(), SpuSetCommonCDVolume().

SpuGetCommonMasterVolume

Obtains master volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetCommonMasterVolume (
short *mvolL,
short *mvolR
)
```

Arguments

mvolL Master volume (left)
mvolR Master volume (right)

Explanation

This function obtains the master volume.

It is equivalent to the process of obtaining the value of the *mvol* member of the *SpuCommonAttr* structure in *SpuGetCommonAttr()*.

The only effective value is that obtained when the volume mode is set to 'direct mode'. Other volume modes are undefined.

When the volume mode is not 'Direct Mode' or when you would like to obtain both the volume and the volume mode at the same time, use *SpuGetCommonMasterVolumeAttr()*.

Return value

None

Remarks

See also: *SpuGetCommonAttr()*, *SpuGetCommonMasterVolumeAttr()*, *SpuSetCommonMasterVolume()*, *SpuSetCommonMasterVolumeAttr()*.

SpuGetCommonMasterVolumeAttr

Obtains master volume/master volume mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetCommonMasterVolumeAttr (
short *mvolL,
short *mvolR,
short *mvolModeL,
short *mvolModeR
)
```

Arguments

<i>mvolL</i>	Master volume (left)
<i>mvolR</i>	Master volume (right)
<i>mvolModeL</i>	Master Volume Mode (left)
<i>mvolModeR</i>	Master Volume Mode (right)

Explanation

This function obtains the master volume/master volume mode.

It is equivalent to the process of obtaining the value of the mvol and mvolmode members of the SpuCommonAttr structure in SpuGetCommonAttr().

Return value

None

Remarks

See also: SpuGetCommonAttr(), SpuGetCommonMasterVolume(), SpuSetCommonMasterVolume(), SpuSetCommonMasterVolumeAttr().

SpuGetCommonMasterVolumeX

Obtains current master volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetCommonMasterVolumeX (
short *mvolXL,
short *mvolXR
)
```

Arguments

mvolXL Current master volume (left)
mvolXR Current master volume (right)

Explanation

This function obtains the current master volume.

It is equivalent to the process of obtaining the value of the *mvolx* member of the *SpuCommonAttr* structure in *SpuGetCommonAttr()*.

Return value

None

Remarks

See also: *SpuGetCommonAttr()*, *SpuGetCommonMasterVolume()*, *SpuGetCommonMasterVolumeAttr()*, *SpuSetCommonMasterVolume()*, *SpuSetCommonMasterVolumeAttr()*.

SpuGetIRQ

Checks status of interrupt request on/off.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

long SpuGetIRQ(*void*)

Arguments

None

Explanation

Checks status of interrupt request on/off.

Return value

Currently set value.

SPU_ON Interrupt request is set

SPU_OFF Interrupt request is not set

Remarks

See also: SpuSetIRQ (p. 15-82).

SpuGetIRQAddr

Checks interrupt request address.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

unsigned long SpuGetIRQAddr(*void*)

Arguments

None

Explanation

Returns interrupt request address value.

Return value

Currently set address value

Remarks

See also: SpuSetIRQAddr (p. 15-88), SpuSetIRQ (p. 15-82).

SpuGetKeyStatus

Checks key on/key off status for specified voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpuGetKeyStatus(voice_bit)
unsigned long voice_bit;
```

Arguments

voice_bit Checked voice

Explanation

Checks key on/key off and envelope status of specified voices; checks actual key on/key off.

Explicitly specify the voices targeted in *voice_bit* by ORing together SPU_0CH-SPU_23CH. 1 function call gets the attributes of only 1 voice, so, in the case of multiple specifications, the smallest voice number specified is selected.

An error may result if multiple envelopes are set, and if volume goes to 0 in the course of changing envelope status.

Return value

If successful, the current key on/key off status and envelope status of the specified voice are returned. (See the table below.) If the specified voice is incorrect, `SpuGetKeyStatus()` returns -1.

Table 15-2

Value	Status
SPU_ON	Key on status Not turned off by SpuSetKey Envelope not 0
SPU_ON_ENV_OFF	Key on status Not turned off by SpuSetKey Envelope 0
SPU_OFF_ENV_ON	Key off status Turned off by SpuSetKey Envelope not 0
SPU_OFF	Key off status Turned off by SpuSetKey Envelope 0

Remarks

See also: `SpuSetKey` (p. 15-91), `SpuGetAllKeysStatus` (p. 15-17).

SpuGetMute

Checks status of sound muting.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

long SpuGetMute(*void*)

Arguments

None

Explanation

Checks current sound mute on/off status.

Return value

Currently set value (SPU_ON/SPU_OFF)

Table 15-3

Value	Description
SPU_ON	Mute off
SPU_OFF	Mute on

Remarks

See also: SpuSetMute (p. 15-94).

SpuGetNoiseClock

Checks noise source clock.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

long SpuGetNoiseClock(*void*)

Arguments

None

Explanation

Returns the value of noise source clock.

Return value

Currently set noise source clock value.

Remarks

See also: SpuSetNoiseClock (p. 15-95).

SpuGetNoiseVoice

Checks noise source ON/OFF for each voice

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

unsigned long SpuGetNoiseVoice(void)

Arguments

None

Explanation

Checks current status of noise source ON/OFF for each voice.

Return value

Returns the noise source ON/OFF value of the current voice. OR together SPU_0CH-SPU_23CH.

Distinguishes the noise source ON/OFF value by ANDing the return value and SPU_xxCH(xx=0~23).

Table 15-4

Result of AND	Description
0	Noise source off
Other than 0	Noise source on

Remarks

See also: SpuSetNoiseClock (p. 15-95), SpuSetNoiseVoice (p. 15-96).

SpuGetPitchLFOVoice

Checks pitch LFO ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

unsigned long SpuGetPitchLFOVoice(void)

Arguments

None

Explanation

Checks current status of pitch LFO ON/OFF for each voice.

Return value

Returns the pitch LFO ON/OFF value of the current voice. OR together SPU_0CH-SPU_23CH.

Distinguishes the pitch LFO ON/OFF value by ANDing the return value and SPU_xxCH(xx=0~23).

Table 15-5

Result of AND	Description
0	Pitch LFO off
Other than 0	Pitch LFO on

Remarks

See also: SpuSetPitchLFOVoice (p. 15-97).

SpuGetReverb

Checks reverb status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

long SpuGetReverb(*void*)

Arguments

None

Explanation

Checks current reverb ON/OFF status.

Return value

Set value (SPU_ON/SPU_OFF).

Table 15–6

Value	Description
SPU_ON	Reverb on
SPU_OFF	Reverb off

Remarks

See also: SpuSetReverb (p. 15-98).

SpuGetReverbModeDelayTime

Obtains reverb mode delay time.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetReverbModeDelayTime (
long *delay
)
```

Arguments

delay Reverb delay time

Explanation

This function obtains the relay delay time.

It is equivalent to the process of obtaining the value of the delay member of the SpuCommonAttr structure in SpuGetCommonAttr().

Return value

None

Remarks

See also: SpuSetReverbModeParam(), SpuGetReverbModeParam(), SpuSetReverbModeDelayTime().

SpuGetReverbModeDepth

Obtains reverb mode depth.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetReverbModeDepth (
short *depthL,
short *depthR
)
```

Arguments

depthL Reverb depth (left)
depthR Reverb depth (right)

Explanation

This function obtains the reverb depth.

It is equivalent to the process of obtaining the value of the depth member of the SpuCommonAttr structure in SpuGetCommonAttr().

Return value

None

Remarks

See also: SpuSetReverbModeParam(), SpuGetReverbModeParam(), SpuSetReverbModeDepth().

SpuGetReverbModeFeedback

Obtains reverb mode feedback.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetReverbModeFeedback (
long *feedback
)
```

Arguments

feedback Reverb feedback

Explanation

This function obtains the reverb feedback.

It is equivalent to the process of obtaining the value of the feedback member of the SpuCommonAttr structure in SpuGetCommonAttr().

Return value

None

Remarks

See also: SpuSetReverbModeParam(), SpuGetReverbModeParam(), SpuSetReverbModeFeedback().

SpuGetReverbModeParam

Checks reverb mode and parameters.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
void SpuGetReverbModeParam(*attr)
SpuReverbAttr *attr;
```

Arguments

attr Pointer to reverb attributes

Explanation

Gets currently set reverb mode and parameters.

For details see SpuSetReverbModeParam().

Return value

None

Remarks

See also: SpuSetReverbModeParam (p. 15-100), SpuReverbAttr (p. 15-9).

SpuGetReverbModeType

Obtains reverb mode type.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuGetReverbModeType (  
long *type  
)
```

Arguments

type Reverb mode type

Explanation

This function obtains the reverb mode type.

It is equivalent to the process of obtaining the value of the mode member of the SpuCommonAttr structure in SpuGetCommonAttr().

Return value

None

Remarks

See also: SpuSetReverbModeParam(), SpuGetReverbModeParam(), SpuSetReverbModeType().

SpuGetReverbVoice

Checks reverb ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

unsigned long SpuGetReverbVoice(*void*)

Arguments

None

Explanation

Checks current reverb ON/OFF status for each voice.

Return value

Returns the reverb ON/OFF value of the current voice. OR together SPU_0CH-SPU_23CH.

Distinguishes the noise source ON/OFF value by ANDing the return value and SPU_xxCH(xx=0~23).

Table 15-7

Result of AND	Description
0	Reverb off
Other than 0	Reverb on

Remarks

See also: SpuSetReverbVoice (p. 15-105).

SpuGetTransferMode

Checks sound buffer transfer mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

long SpuGetTransferMode(*void*)

Arguments

None

Explanation

Returns currently set value of the transfer mode when transferring from main memory to the sound buffer.

Return value

Current setting of transfer mode

SPU_TRANSFER_BY_DMA	DMA transfer setting
SPU_TRANSFER_BY_IO	I/O transfer setting

Remarks

See also: SpuSetTransferMode (p. 15-108), SpuWrite (p. 15-142), SpuWrite0 (), SpuWritePartly ().

SpuGetTransferStartAddr

Checks sound buffer transfer destination/transfer source start address.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

unsigned long SpuGetTransferStartAddr(*void*)

Arguments

None

Explanation

Returns currently set value for start address when transferring from main memory to the sound buffer, and from the sound buffer to main memory.

Return value

Currently set sound buffer starting address value.

Remarks

See also: SpuSetTransferStartAddr (p. 15-109), SpuWrite (p. 15-142), SpuWrite0 (), SpuWritePartly(), SpuRead (p. 15-73), SpuReadDecodedData ().

SpuGetVoiceADSR

Gets ADSR.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.6	10/23/96

Syntax

```
void SpuGetVoiceADSR (  
int voiceNum,  
unsigned short *AR,  
unsigned short *DR,  
unsigned short *SR,  
unsigned short *RR,  
unsigned short *SL  
)
```

Arguments

voiceNum	Voice number (0 - 23)
AR	ADSR attack rate
DR	ADSR decay rate
SR	ADSR sustain rate
RR	ADSR release rate
SL	ADSR sustain level

Explanation

This function obtains each ADSR attribute used in the voice, equivalent to the process to obtain the values for SpuVoiceAttr members, AR, DR, SR, RR, and SL using SpuGetVoiceAttr function.

The value obtained are valid only when the attack, sustain, and release rate are set to the mode as below:

-----+-----
Attack Rate Mode | SPU_VOICE_LINEARIncN (Linear Increase)
Sustain Rate Mode | SPU_VOICE_LINEARDecN (Linear Decrease)
Release_Rate_Mode | SPU_VOICE_LINEARDecN_(Linear Decrease)
-----+-----

For other mode, the obtained values are undefined. If you want to obtain multiple Rate Mode at the same time, use SpuSetVoiceADSRAttr.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceAR(), SpuGetVoiceDR(), SpuGetVoiceSR(), SpuGetVoiceRR(), SpuGetVoiceSL().

SpuGetVoiceADSRAttr

Gets ADSR and each mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceADSRAttr (
int voiceNum,
unsigned short *AR,
unsigned short *DR,
unsigned short *SR,
unsigned short *RR,
unsigned short *SL,
long *ARmode,
long *SRmode,
long *RRmode
)
```

Arguments

<i>voiceNum</i>	Voice number (0 - 23)
<i>AR</i>	ADSR attack rate
<i>DR</i>	ADSR decay rate
<i>SR</i>	ADSR sustain rate
<i>RR</i>	ADSR release rate
<i>SL</i>	ADSR sustain level
<i>ARmode</i>	ADSR attack rate mode
<i>SRmode</i>	ADSR sustain rate mode
<i>RRmode</i>	ADSR release rate mode

Explanaton

This function obtains each ADSR attribute used in the voice, equivalent to the process to obtain the values for SpuVoiceAttr members, AR, DR, SR, RR, SL, ARmode, SLmode, and RRmode using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceADSR(), SpuGetVoiceAR(), SpuGetVoiceDR(), SpuGetVoiceSR(), SpuGetVoiceRR(), SpuGetVoiceSL(), SpuGetVoiceARAttr(), SpuGetVoiceSRAttr(), SpuGetVoiceRRAttr().

SpuGetVoiceAR

Gets ADSR attack rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceAR (
int voiceNum,
unsigned short *AR
)
```

Arguments

voiceNum Voice number (0 - 23)
AR ADSR attack rate

Explanation

This function obtains ADSR attack rate used in voice. This function obtains voice volume, equivalent to the process to obtain the value for *SpuVoiceAttr* member, are using *SpuGetVoiceAttr* function.

The value obtained is valid only when ADSR attack rate mode is set to *SPU_VOICE_LINEARIncN* (Linear Increase). For other ADSR attack rate mode the value is undefined.

When both ADSR attack rate volume and ADSR attack rate mode need to be obtained at the same time, use *SpuGetVoiceARAttr*.

Return value

None

Remarks

See also: *SpuGetVoiceAttr()*, *SpuNGetVoiceAttr()*, *SpuGetVoiceARAttr()*.

SpuGetVoiceARAttr

Gets ADSR attack rate / attack rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceARAttr (
int voiceNum,
unsigned short *AR,
long *ARmode
)
```

Arguments

voiceNum Voice number (0 - 23)
AR ADSR attack rate
ARmode` ADSR attack rate mode

Explanation

This function obtains ADSR attack rate / ADSR attack rate mode used in voice, equivalent to the process to obtain the value for SpuVoiceAttr members, AR and ARmode using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNSGetVoiceAttr(), SpuGetVoiceAR().

SpuGetVoiceAttr

Checks voice attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
void SpuGetVoiceAttr(*attr)
SpuVoiceAttr *attr;
```

Arguments

attr Pointer to voice attributes

Explanation

Checks voice attributes.

Explicitly set the single voice (SPU_0CH, SPU_1CH, ... SPU_23CH) checked by *attr*.voice. All the attribute structure members are returned except mask. See SpuSetVoiceAttr() for the details of these attributes.

Return value

None (The argument *attr* is the return value.)

Remarks

See also: SpuSetVoiceAttr (p. 15-110), SpuRSetVoiceAttr, SpuSetKey (p. 15-91), SpuSetKeyOnWithAttr (p. 15-92), SpuVoiceAttr (p. 15-10).

SpuGetVoiceDR

Gets ADSR decay rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceDR (
int voiceNum,
unsigned short *DR
)
```

Arguments

voiceNum Voice number (0 - 23)
DR ADSR decay rate

Explanation

This function obtains ADSR decay rate used in voice, equivalent to the process to obtain the value for *SpuVoiceAttr* member, *DR* using *SpuGetVoiceAttr* function.

Return value

None

Remarks

See also: *SpuGetVoiceAttr()*, *SpuNGetVoiceAttr()*.

SpuGetVoiceEnvelope

Gets current envelope value.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceEnvelope (
  int voiceNum,
  short *envx
)
```

Arguments

<i>voiceNum</i>	Voice number (0 - 23)
<i>envx</i>	Current envelope value

Explanation

This function obtains the current voice envelope value, equivalent to the process to obtain the value for SpuVoiceAttr envx, using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr().

SpuGetVoiceEnvelopeAttr

Gets current voice envelope value and key ON/OFF status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceEnvelopeAttr (
int voiceNum,
long *keyStat,
short *envx
)
```

Arguments

<i>voiceNum</i>	Voice number (0 - 23)
<i>keyStat</i>	Status of voice envelope and key ON/OFF
<i>envx</i>	Current envelope value

Explanation

This function obtains the current voice envelope value and voice key ON/OFF and envelope status.

Refer to SpuGetVoiceAttr for values that can be specified in keystate, the key ON/OFF and envelope status.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceEnvelope(), SpuSetKey(), SpuGetAllKeysStatus(), SpuRGetAllKeysStatus().

SpuGetVoiceLoopStartAddr

Gets loop start address of waveform data in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceLoopStartAddr (
int voiceNum,
unsigned long *loopStartAddr
)
```

Arguments

voiceNum Voice number (0 - 23)
loopStartAddr Loop start address

Explanation

This function obtains loop start address of waveform data in the sound buffer, equivalent to the process to obtain the value for SpuVoiceAttr loop_addr, using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetTransferStartAddr().

SpuGetVoiceNote

Gets interval (note specification).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceNote (
int voiceNum,
unsigned short *note
)
```

Arguments

voiceNum Voice number (0 - 23)
note Interval (note specification)

Explanation

This function obtains Voice Interval (Note Specification), equivalent to the process to obtain the value for SpuVoiceAttr member, note using SpuGetVoiceAttr function.

Thus prior to call SpuSetVoiceNote, SpuSetVoiceAttr

SPU_VOICE_SAMPLE_NOTE

or the waveform data sample note feature for voice must be set.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceSampleNote(), SpuGetVoiceSampleNote().

SpuGetVoicePitch

Gets interval (pitch specification).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoicePitch (
int voiceNum,
unsigned short *pitch
)
```

Arguments

voiceNum Voice number (0 - 23)
pitch Interval (pitch specification)

Explanation

This function obtains voice interval (pitch specification), equivalent to the process to obtain the value for SpuVoiceAttr member, pitch using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr().

SpuGetVoiceRR

Gets ADSR release rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceRR (
int voiceNum,
unsigned short *RR
)
```

Arguments

voiceNum Voice number (0 - 23)
RR ADSR release rate

Explanation

This function obtains ADSR release rate in voice equivalent to, equivalent to the process to obtain the value for *SpuVoiceAttr* member, *rr* using *SpuGetVoiceAttr* function.

The value obtained is valid only when ADSR release rate mode is set to *SPU_VOICE_LINEARDecN* (Linear Decrease mode).

For other ADSR release rate mode, the value is undefined. If you want to obtain both ADSR release rate and ADSR release rate mode at the same time, use *SpuGetVoiceRRAttr*.

Return value

None

Remarks

See also: *SpuGetVoiceAttr()*, *SpuNGetVoiceAttr()*, *SpuGetVoiceRRAttr()*.

SpuGetVoiceRRAttr

Gets ADSR release rate / release rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceRRAttr (
int voiceNum,
unsigned short *RR,
long *RRmode
)
```

Arguments

voiceNum Voice number (0 - 23)
RR ADSR release rate
RRmode ADSR release rate mode

Explanation

This function obtains ADSR release rate / ADSR release rate mode used in voice, equivalent to the process to obtain the value for SpuVoiceAttr members, RR and RRmode using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceRR().

SpuGetVoiceSampleNote

Gets waveform data sample note.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceSampleNote (
int voiceNum,
unsigned short *sampleNote
)
```

Arguments

voiceNum Voice number (0 - 23)
sampleNote Sets waveform data sample note

Explanation

This function obtains waveform data sample note, equivalent to the process to obtain the value for *SpuVoiceAttr* member, *sample_note* using *SpuGetVoiceAttr* function..

Return value

None

Remarks

See also: *SpuGetVoiceAttr()*, *SpuNGetVoiceAttr()*, *SpuGetVoiceNote()*.

SpuGetVoiceSL

Gets ADSR sustain level.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceSL (
int voiceNum,
unsigned short *SL
)
```

Arguments

voiceNum Voice number (0 - 23)
SL ADSR sustain level

Explanation

This function obtains ADSR sustain level. equivalent to the process to obtain the value for SpuVoiceAttr member, SL using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceRRAttr().

SpuGetVoiceSR

Gets ADSR sustain rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceSR (
int voiceNum,
unsigned short *SR
)
```

Arguments

voiceNum Voice number (0 - 23)
SR ADSR sustain rate

Explanation

This function obtains ADSR sustain rate in voice equivalent to, equivalent to the process to obtain the value for *SpuVoiceAttr* member, *SR* using *SpuGetVoiceAttr* function.

The value obtained is valid only when ADSR sustain rate mode is set to *SPU_VOICE_LINEARDecN* (Linear Decrease mode).

For other ADSR sustain rate mode, the value is undefined. If you want to obtain both ADSR sustain rate and ADSR sustain rate mode at the same time, use *SpuGetVoiceSRAttr*.

Return value

None

Remarks

See also: *SpuGetVoiceAttr()*, *SpuNGetVoiceAttr()*, *SpuGetVoiceSRAttr()*.

SpuGetVoiceSRAttr

Gets ADSR sustain rate / sustain rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceSRAttr (
int voiceNum,
unsigned short *SR,
long *SRmode
)
```

Arguments

<i>voiceNum</i>	Voice number (0 - 23)
<i>SR</i>	ADSR sustain rate
<i>SRmode</i>	ADSR sustain rate mode

Explanation

This function obtains ADSR sustain rate / ADSR sustain rate mode used in voice, equivalent to the process to obtain the value for SpuVoiceAttr members, SR and SRmode using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceSR().

SpuGetVoiceStartAddr

Gets start address of waveform data in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceStartAddr (
int voiceNum,
unsigned long *startAddr
)
```

Arguments

voiceNum Voice number (0 - 23)
startAddr Waveform data start address

Explanation

This function obtains start address of waveform data in the sound buffer, equivalent to the process to obtain the value for SpuVoiceAttr member, addr using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetTransferStartAddr().

SpuGetVoiceVolume

Gets voice volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceVolume (
  int voiceNum,
  short *volumeL,
  short *volumeR
)
```

Arguments

<i>voiceNum</i>	Voice Number (0 - 23)
<i>volumeL</i>	Volume (Left)
<i>volumeR</i>	Volume (Right)

Explanation

This function obtains voice volume, equivalent to the process to obtain the value for `SpuVoiceAttr` member, `volume` using `SpuGetVoiceAttr` function.

The value obtained is valid only when the volume mode is set to "Direct Mode". For other volume mode, the value is undefined.

When the volume mode is not "Direct Mode" or both volume and volume mode need to be obtained at the same time, use `SpuGetVoiceVolumeAttr`.

Return value

None

Remarks

See also: `SpuGetVoiceAttr()`, `SpuNGetVoiceAttr()`, `SpuGetVoiceVolumeAttr()`.

SpuGetVoiceVolumeAttr

Gets voice volume/volume mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceVolumeAttr (
int voiceNum,
short *volumeL,
short *volumeR,
short *volModeL,
short *volModeR
)
```

Arguments

<i>voiceNum</i>	Voice Number (0 - 23)
<i>volumeL</i>	Volume (Left)
<i>volumeR</i>	Volume (Right)
<i>volModeL</i>	Volume mode (Left)
<i>volModeR</i>	Volume mode (Right)

Explanation

This function obtains voice volume / volume mode, equivalent to the process to obtain the value for SpuVoiceAttr members, volume and volumed using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceVolume().

SpuGetVoiceVolumeX

Gets current voice volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuGetVoiceVolumeX (
int voiceNum,
short *volumeL,
short *volumeR
)
```

Arguments

<i>voiceNum</i>	Voice Number (0 - 23)
<i>volumeXL</i>	Current volume (Left)
<i>volumeXR</i>	Current volume (Right)

Explanation

This functions obtains current voice volume. This function obtains voice volume, equivalent to the process to obtain the value for SpuVoiceAttr member, volumex using SpuGetVoiceAttr function.

Return value

None

Remarks

See also: SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceVolume(), SpuGetVoiceVolumeAttr().

Spulnit

SPU initialization.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.1	7/31/96

Syntax

void Spulnit(void)

Arguments

None

Explanation

Initializes SPU. Called only once within the program. After initialization, SPU may have the following states.

- Master volume is 0 for both L/R
- Reverb is off
- Reverb work area is not reserved
- Reverb depth is 0 for both L/R
- Reverb volume is 0 for both L/R
- Sound buffer transfer mode is DMA transfer
- For all voices:
 - Key off
 - Pitch LFO function not set
 - Noise function not set
 - Reverb function not set
- CD input volume is 0 for both L/R
- External digital input volume is 0 for both L/R
- DMA transfer initialization set

The status of the sound buffer is indeterminate after initialization.

Return value

None

Remarks

See also: SpulnitHot (p. 15-64), SpuStart (p. 15-119), SpuQuit (p. 15-29).

SpulnitHot

Spu initialization (hot reset), preserves sound buffer status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.1	7/31/96

Syntax

void SpulnitHot(*void*)

Arguments

None

Explanation

Initializes SPU. Call SpulnitHot() when you initialize the sound system and want to preserve the sound buffer status in a child process.

After initialization, status is as follows.

- L/R main volume are both 0
- Reverb is off
- Reverb work area is not reserved
- L/R reverb depth are both 0
- L/R reverb volume are both 0
- Transfer to sound buffer is DMA mode
- All voices:
 - Key off
 - Reverb functionality not yet set
- Sets DMA transfer initialization.

Sound buffer status is preserved after initialization, though not through a hardware reset.

Return value

None

Remarks

See also: Spulnit (p.15-47), SpuStart (p. 15-119), SpuQuit (p. 15-29).

SpulnitMalloc

Initializes the sound buffer memory management mechanism.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpulnitMalloc(num, *top)
long num;
char *top;
```

Arguments

num Maximum number of times memory is allocated
top Pointer to the start address of the area storing management table

Explanation

Performs initialization in order to divide the sound buffer into *num* number of areas and manage them. The individual *num* memory management blocks used by each request are allocated in the area provided by *top*. The size of the area must be as follows:

(SPU_MALLOC_RECSIZ • (num + 1)) bytes

Return value

Returns the number of memory management blocks allocated.

Remarks

When creating memory management blocks to be used by 10 SpuMalloc() calls, SpulnitMalloc() is called as follows:

```
char rec[SPU_MALLOC_RECSIZ * (10 + 1)];
SpuInitMalloc (10,          /*10 SpuMalloc calls can be used*/
               rec);        /*memory management block*/
```

See also: malloc (See libmath), SpuMalloc (p. 15-68), SpuMallocWithStartAddr (p. 15-69), SpuFree (p. 15-16).

SpulsReverbWorkAreaReserved

Checks to see if reverb work area is reserved/Checks to see if reverb work area can be reserved.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpulsReverbWorkAreaReserved(on_off)
long on_off;
```

Arguments

on_off Contents of the checking process

Explanation

Checks to see if the reverb work area corresponding to the current reverb mode is reserved, or checks to see if it can be reserved.

on_off specifies which action is performed. These settings are explained below.

Table 15–8

Value	Description
SPU_DIAG	Checks to see if reverb work area can be reserved
SPU_CHECK	Checks reverb work area reserve status

a) SPU_DIAG

Using sound buffer memory management mechanism information, SPU_DIAG checks to see whether or not the reverb work area is an area allocated by `SpuMalloc()`/`SpuMallocWithStartAddr()`. If it can be reserved, SPU_ON is returned. If it cannot be reserved, SPU_OFF is returned.

b) SPU_CHECK

Returns current reverb work area reserve status.

Return value

When *on_off* is SPU_DIAG, if the reverb work area can be reserved, SPU_ON is returned. If it cannot be reserved, SPU_OFF is returned.

When *on_off* is SPU_CHECK, if the reverb work area is reserved, SPU_ON is returned. If it is not reserved, SPU_OFF is returned.

Remarks

See also: `SpuReserveReverbWorkArea` (p. 15-76), `SpuSetReverbModeParam` (p. 15-100), `SpuSetReverb` (p. 15-98), `SpuMalloc` (p. 15-68), `SpuMallocWithStartAddr` (p. 15-69).

SpulsTransferCompleted

Checks completion of transfer to the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpulsTransferCompleted(flag)
long flag;
```

Arguments

flag Check flag

Explanation

Checks whether transfer is completed.

Flag values may be specified as follows.

Table 15-9

Value	Description
SPU_TRANSFER_WAIT	Wait until transfer ends
SPU_TRANSFER_PEEK result	Check whether transfer has ended return result
SPU_TRANSFER_GLANCE	Same as SPU_TRANSFER_PEEK

SpulsTransferCompleted is not functional when, using SpuSetTransferCallback, a callback function is set and started at the completion of DMA transfer.

Return value

Returns the status of transfer completion.

- 1 transfer completed
- 0 transfer not completed.

If flag = SPU_TRANSFER_WAIT, wait until transfer ends and always return 1.

If transfer mode is "I/O transfer", 1 is returned immediately.

SpulsTransferCompleted returns 1 when, using SpuSetTransferCallback, a callback function is set and started at the completion of DMA transfer.

Remarks

See also: SpuWrite (p. 15-142), SpuWritePartly (), ()SpuRead (p. 15-73), SpuReadDecodedData (), SpuSetTransferCallback ().

SpuMalloc

Allocates an area in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpuMalloc(size)
long size;
```

Arguments

size Size of area allocated (in bytes)

Explanation

Allocate an area of *size* bytes in the sound buffer.

The return value is the address of the start of the allocated area and must be greater than 0x100f. When this value is set by an argument of SpuSetTransferStartAddr(), and the transfer start address is set, SpuWrite() transfers waveform data.

The following states cause failure in allocation.

- The requested size cannot be continuously allocated.
- There is an area which satisfies the requested size, but that area is part or all of a reverb work area allocated by SpuReserveReverbWorkArea(), and essentially cannot be allocated.

Return value

If allocation is successful, the starting address of the allocated area is returned.

If unsuccessful, -1 is returned.

Remarks

See also: SpuInitMalloc (p. 15-65), SpuMallocWithStartAddr (p. 15-69), SpuFree (p. 15-16), SpuSetTransferStartAddr (p. 15-109), SpuWrite (p. 15-142), SpuReserveReverbWorkArea (p. 15-76), SpuSetReverb (p. 15-98), SpuSetReverbModeParam (p. 15-103).

SpuMallocWithStartAddr

Allocates an area from a specified start address in sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpuMallocWithStartAddr(addr, size)
unsigned long addr;
long size;
```

Arguments

addr Allocated area starting address (in bytes)
size Size of allocated area (in bytes)

Explanation

Allocates an area in the sound buffer of *size* bytes starting from the start address *addr*.

The allocatable area is 0x01010 - 0x7ffff.

If that address is in an area already allocated, an area of *size* bytes, starting from the nearest empty area after the *addr* area, is allocated.

The following states cause failure in allocation.

- The requested size cannot be continuously allocated.
- There is an area which satisfies the requested size, but that area is part or all of a reverb work area allocated by SpuReserveReverbWorkArea(), and essentially cannot be allocated.

Return value

If allocation is successful, the starting address of the allocated area is returned. If unsuccessful, -1 is returned.

Remarks

See also: SpuInitMalloc (p. 15-65), SpuMalloc (p.15-68), SpuFree (p. 15-16), SpuSetTransferStartAddr (p. 15-109), SpuWrite (p. 15-142), SpuReserveReverbWorkArea (p. 15-76), SpuSetReverb (p. 15-98), SpuSetReverbModeParam (p. 15-103).

SpuNGetVoiceAttr

Gets voice attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuNGetVoiceAttr (  
int voiceNum,  
SpuVoiceAttr *attr  
)
```

Arguments

voiceNum Voice number (0 - 23)
attr Voice attribute

Explanation

This function obtains voice attribute. Set voice number to be obtained explicitly into *voiceNum*.
All attributes except "mask" will be returned for "attr" structrue members.
Refer to *SpuSetVoiceAttr* for detail of each attribute.

Return value

None

Remarks

See also: *SpuGetVoiceAttr()*, *SpuSetVoiceAttr()*, *SpuNSetVoiceAttr()*, *SpuRSetVoiceAttr()*, *SpuSetKey()*, *SpuSetKeyOnWithAttr()*, *SpuGetVoiceVolume()*, *SpuGetVoiceVolumeAttr()*, *SpuGetVoiceVolumeX()*, *SpuGetVoicePitch()*, *SpuGetVoiceNote()*, *SpuGetVoiceSampleNote()*, *SpuGetVoiceEnvelope()*, *SpuGetVoiceStartAddr()*, *SpuGetVoiceLoopStartAddr()*, *SpuGetVoiceAR()*, *SpuGetVoiceDR()*, *SpuGetVoiceSR()*, *SpuGetVoiceRR()*, *SpuGetVoiceSL()*, *SpuGetVoiceARAttr()*, *SpuGetVoiceSRAttr()*, *SpuGetVoiceRRAttr()*, *SpuGetVoiceADSR()*, *SpuGetVoiceADSRAttr()*, *SpuGetVoiceEnvelopeAttr()*.

SpuNSetVoiceAttr

Sets voice attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	10/23/96

Syntax

```
void SpuNSetVoiceAttr (
int voiceNum,
SpuVoiceAttr *attr
)
```

Arguments

voiceNum Voice number (0 - 23)
attr Voice attribute

Explanation

This function sets the voice attribute. Set voice number to be obtained explicitly into *voiceNum*.

Set *attr.mask* with bitwise inclusive ORed attributes;

SPU_VOICE_VOLL	Volume (left)
SPU_VOICE_VOLR	Volume (right)
SPU_VOICE_VOLMODEL	Volume mode (left)
SPU_VOICE_VOLMODER	Volume mode (right)
SPU_VOICE_PITCH	Interval (pitch specification)
SPU_VOICE_NOTE	Interval (note specification)
SPU_VOICE_SAMPLE_NOTE	Waveform data sample note
SPU_VOICE_WDSA	Waveform data start address
SPU_VOICE_ADSR_AMODE	ADSR attack rate mode
SPU_VOICE_ADSR_SMODE	ADSR sustain rate mode
SPU_VOICE_ADSR_RMODE	ADSR release rate mode
SPU_VOICE_ADSR_AR	ADSR attack rate
SPU_VOICE_ADSR_DR	ADSR decay rate
SPU_VOICE_ADSR_SR	ADSR sustain rate
SPU_VOICE_ADSR_RR	ADSR release rate
SPU_VOICE_ADSR_SL	ADSR sustain level
SPU_VOICE_ADSR_ADSR1	ADSR adsr1 for 'VagAtr'
SPU_VOICE_ADSR_ADSR2	ADSR adsr2 for 'VagAtr'
SPU_VOICE_LSAX	Loop start address

Return value

None

Remarks

See also: *SpuSetVoiceAttr()*, *SpuNSetVoiceAttr()*, *SpuRSetVoiceAttr()*, *SpuGetVoiceAttr()*, *SpuNGetVoiceAttr()*, *SpuSetKey()*, *SpuSetKeyOnWithAttr()*, *SpuSetVoiceVolume()*, *SpuSetVoiceVolumeAttr()*, *SpuSetVoicePitch()*, *SpuSetVoiceNote()*, *SpuSetVoiceSampleNote()*, *SpuSetVoiceStartAddr()*, *SpuSetVoiceLoopStartAddr()*, *SpuSetVoiceAR()*, *SpuSetVoiceDR()*, *SpuSetVoiceSR()*, *SpuSetVoiceRR()*, *SpuSetVoiceSL()*, *SpuSetVoiceARAtr()*, *SpuSetVoiceSRAtr()*, *SpuSetVoiceRRAtr()*, *SpuSetVoiceADSR()*, *SpuSetVoiceADSRAttr()*.

SpuQuit

Terminates SPU processing.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

void SpuQuit(*void*)

Arguments

None

Explanation

Terminates SPU processing. Normally, during a game, all devices, including SPU, are usually reset with a hardware reset, so it is not necessary to call SpuQuit(), but because SpuQuit is called with the original debug environment, the item below is reset in the current specification.

After this setting is made, DMA transfer to the sound buffer cannot be used

Return value

None

Remarks

See also: Spulnit (p.15-47), SpulnitHot (p. 15-64).

SpuRead

Transfers data from the sound buffer to main memory.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	6/22/98

Syntax

```
unsigned long SpuRead(*addr, size)
unsigned char *addr;
unsigned long size;
```

Arguments

addr Pointer to transfer data start address in main memory
size Transferred data size (in bytes)

Explanation

Transfers *size* bytes of data from the sound buffer to main memory *addr*.

The transfer destination main memory address *addr* must fulfill the following conditions.

- It is an address of an allocated variable that is a global variable
- It is an address of an allocated variable that is in the heap and is allocated by a function such as malloc.
- It does not address a stack area (address of an auto variable) declared in a function.

After calling, either call SpulsTransferCompleted () to confirm transfer completion or set the DMA transfer completion Callback function in advance using SpuSetTransferCallback

Due to the limitations of the DMA transfer hardware, transfers are always performed in 64 byte units. When specifying values which are not multiples of 64 as secondary arguments, since the portion of the value which is a multiple of 64 will be transferred, there is the possibility of damaging the data in the SPU memory.

Return value

Transferred data size.

If the specified data size is larger than 512 KB, the actual transferred size is returned.

Remarks

See also: SpuWrite (p. 15-142), SpuWrite0 (), SpuWritePartly (), SpuSetTransferStartAddr (p. 15-109), SpuGetTransferStartAddr (p. 15-41), SpulsTransferCompleted (), SpuSetTransferCallback ().

SpuReadDecodedData

Transfers sound data decoded by the SPU from the sound buffer to main memory.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	6/22/98

Syntax

```
long SpuReadDecodedData(*d_data, flag)
SpuDecodedData *d_data;
long flag;
```

Arguments

d_data Pointer to start address of SpuDecodedData structure in main memory
flag SPU_CDONLY Set transfer of only CD input
 SPU_VOICEONLY Set transfer of only Voice 1, 3
 SPU_ALL Set transfer of all data

Explanation

Transfers waveform data decoded by the SPU from the sound buffer to main memory.

The SPU writes sound data after CD input volume processing and sound data after Voice 1 and Voice 3 envelope processing to the sound buffer's starting 0x1000 byte (0x800 short int) area 16 bits (1 short int) at a time at each clock (44.1 kHz). Each piece of sound data has 0x400 byte (0x200 short int) buffers.

Data is signed 16-bit data, so access is in units of 16 bits (1 short int). Data is arranged as shown below.

Table 15–10: Arrangement of Data

Map (short int)	Data Contents
0x000 - 0x1ff	CD Left channel
0x200 - 0x3ff	CD Right channel
0x400 - 0x5ff	Voice 1
0x600 - 0x7ff	Voice 3

These are divided into the first half (0x100 short int) and the second half (0x100 short int); which buffer area is currently being written to is decided by the return value.

```
SPU_DECODED_FIRSTHALF    . . . First half
SPU_DECODED_SECONDHALF  . . . Second half
```

The return value is the area currently being written to, so data that can actually be used is in the area not being reported.

The main memory address *addr* storing the transfer data must fulfill the following conditions.

- It is an address of an allocated variable that is a global variable
- It is an address of an allocated variable that is in the heap and is allocated by a function such as `malloc`.
- It does not address a stack area (address of an auto variable) declared in a function.

In order to confirm transfer completion, set the DMA transfer completion Callback function in advance using `SpuSetTransferCallback()`.

Return value

Returns the buffer area currently being written to, as shown below.

The return value is the area currently being written to, so data that can actually be used is in the area not being reported.

Table 15-11

Return value	Meaning
SPU_DECODE_FIRSTHALF	Writes the first half of data
SPU_DECODE_SECONDDHALF	Writes the second half of data

Remarks

See also: `SpuRead ()`, `SpuWrite` (p. 15-142), `SpuWrite0 ()`, `SpuWritePartly ()`, `SpuSetTransferStartAddr` (p. 15-109), `SpuGetTransferStartAddr` (p. 15-41), `SpulsTransferCompleted ()`, `SpuSetTransferCallback ()`.

SpuReserveReverbWorkArea

Reserve/release reverb work area.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpuReserveReverbWorkArea(on_off)
long on_off;
```

Arguments

on_off Reserve/release flag

Explanation

Reserves the current reverb work area corresponding to the current reverb mode in such a way that it is not allocated by SpuMalloc()/SpuMallocWithStartAddr(), or releases it so that it is allocated.

on_off specifies which action is performed. These settings are explained below.

Table 15-12

Value	Description
SPU_ON	Reserve reverb work area
SPU_OFF	Release reverb work area

a) SPU_ON

Reserves the reverb work area so that it is not an area allocated by SpuMalloc()/SpuMallocWithStartAddr(). Reserves the area without regard to reverb ON/OFF.

If the reverb work area has already been allocated by SpuMalloc() / SpuMallocWithStartAddr() as another area, it is not allocated and SPU_OFF is returned.

b) SPU_OFF

Releases the reverb work area so that it can be allocated by SpuMalloc() / SpuMallocWithStartAddr() as another area. Releases it regardless of reverb ON/OFF; reverb must have been turned off beforehand.

Return value

When *on_off* is set to SPU_ON, if the reverb work area has already been allocated by SpuMalloc()/SpuMallocWithStartAddr() as another area, it is not reserved and SPU_OFF is returned. If it is reserved, SPU_ON is returned.

When *on_off* is set to SPU_OFF, SPU_OFF is always returned.

Remarks

See also: SpuIsReverbWorkAreaReserved (p. 15-66), SpuSetReverbModeParam (p. 15-103), SpuSetReverb (p. 15-98), SpuMalloc (p. 15-68), SpuMallocWithStartAddr (p. 15-69).

SpuRGetAllKeysStatus

Checks key on/key off for the specified range of voices.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.1	6/22/98

Syntax

```
long SpuRGetAllKeysStatus(min, max, *status)
long min;
long max;
char *status[24];
```

Arguments

- min* Lower limit of the voice number to be checked
- max* Upper limit of the voice number to be checked
- status* Pointer to the result of checking a voice

Explanation

Checks key on/key off and envelope status of all voices whose range is specified by *min* and *max*; checks actual key on/key off.

An error may result if multiple envelopes are set, and if volume goes to 0 in the course of changing envelope status.

The current key on/key off and envelope status of each voice is returned to *status*[24].

Table 15-13

Value	Status
SPU_ON	Key on status Not turned off by SpuSetKey Envelope not 0
SPU_ON_ENV_OFF	Key on status Not turned off by SpuSetKey Envelope 0
SPU_OFF_ENV_ON	Key off status Turned off by SpuSetKey Envelope not 0
SPU_OFF	Key off status Turned off by SpuSetKey Envelope 0

Return value

- SPU_INVALID_ARGS Invalid voice range
- SPU_SUCCESS Keys status contained in *status*[24].

Remarks

See also: SpuSetKey (p. 15-91), SpuGetKeyStatus (p. 15-28).

SpuRSetVoiceAttr

Sets attributes of each voice in the designated range.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.1	6/22/98

Syntax

```
long SpuRSetVoiceAttr (min, max, *attr)
long min;
long max;
SpuVoiceAttr *attr;
```

Arguments

min Lower limit of the voice number to be checked
max Upper limit of the voice number to be checked
attr Pointer to voice attributes

Explanation

Sets attributes for each voice, the range of which is specified by min and max.

Explicitly set voices by ORing together SPU_0CH, SPU_1CH, ...SPU_23CH in attr.voice, where the range of voices is specified by min and max.

You can set each attribute in attr.voice by ORing together the terms shown below.

Table 15-14

Attribute	Description
SPU_VOICE_VOLL	Volume (left)
SPU_VOICE_VOLR	Volume (right)
SPU_VOICE_VOLMODEL	Volume mode (left)
SPU_VOICE_VOLMODER	Volume mode (right)
SPU_VOICE_PITCH	Interval (pitch specification)
SPU_VOICE_NOTE	Interval (note specification)
SPU_VOICE_SAMPLE_NOTE	Waveform data sample note
SPU_VOICE_WDSA	Waveform data start address
SPU_VOICE_ADSR_AMODE	ADSR Attack rate mode
SPU_VOICE_ADSR_SMODE	ADSR Sustain rate mode
SPU_VOICE_ADSR_RMODE	ADSR Release rate mode
SPU_VOICE_ADSR_AR	ADSR Attack rate
SPU_VOICE_ADSR_DR	ADSR Decay rate
SPU_VOICE_ADSR_SR	ADSR Sustain rate
SPU_VOICE_ADSR_RR	ADSR Release rate
SPU_VOICE_ADSR_SL	ADSR Sustain level
SPU_VOICE_ADSR_ADSR1	ADSR adsr1 for 'VagAtr'
SPU_VOICE_ADSR_ADSR2	ADSR adsr2 for 'VagAtr'
SPU_VOICE_LSAX	Loop start address

If attr.mask is 0, set all attributes.

The individual settings of each attribute are described in SpuSetVoiceAttr.

Return value

SPU_INVALID_ARGS Invalid voice range.
SPU_SUCCESS Voice attributes set for specified range.

Remarks

See also: `SpuGetVoiceAttr` (p. 15-42), `SpuSetKey` (p. 15-91), `SpuSetKeyOnWithAttr` (p. 15-92).

SpuSetCommonAttr

Sets attributes common to all voices (infrequent change requests).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
void SpuSetCommonAttr(*attr)
SpuCommonAttr *attr;
```

Arguments

attr Pointer to attributes common to all voices

Explanation

Sets attributes common to all voices.

You can set each attribute (members of *attr*) in *attr.mask* by ORing together the terms shown below. If *attr.mask* is 0, all attributes are set.

Table 15–15

Attribute	Description
SPU_COMMON_MVOLL	Master volume (left)
SPU_COMMON_MVOLR	Master volume (right)
SPU_COMMON_MVOLMODEL	Master volume mode (left)
SPU_COMMON_MVOLMODER	Master volume mode (right)
SPU_COMMON_CDVOLL	CD input volume (left)
SPU_COMMON_CDVOLR	CD input volume (right)
SPU_COMMON_CDREV	CD input reverb ON/OFF
SPU_COMMON_CDMIX	CD input ON/OFF
SPU_COMMON_EXTVOLL	External digital input volume (left)
SPU_COMMON_EXTVOLR	External digital input volume (right)
SPU_COMMON_EXTREV	External digital input reverb ON/OFF
SPU_COMMON_EXTMIX	External digital input ON/OFF

Individual setting parameters are explained below.

a) Master Volume and Master Volume Mode

Master volume is set in *attr.mvol*; master volume mode is set in *attr.mvolmode*. Left and right are set independently.

The volume range obtainable and the various modes are the same as the settings for each voice; see Table 13-35 under *SpuSetVoiceAttr()*.

b) CD Input Volume

CD input volume is set independently for left and right in *attr.cd.volume* in the range -0x8000 - 0x7fff. If the volume set is negative, the phase is inverted.

c) CD Input Reverb On/Off

Reverb is set in *attr.cd.reverb*. The values that may be specified are as follows.

Table 15-16

Value	Description
SPU_ON	Set reverb on
SPU_OFF	Set reverb off

d) CD Input Mixing On/Off

Sets CD input mixing in attr.cd.mix. The values that may be specified are as follows. CD input is not output unless this value is on.

Table 15-17

Value	Description
SPU_ON	Set mixing on
SPU_OFF	Set mixing off

e) External Digital Input Volume

External digital input volume is set independently for left and right in attr.ext.volume in the range - 0x8000 - 0x7fff. If the volume set is negative, the phase is inverted.

f) External Digital Input Reverb On/Off

Reverb is set in attr.ext.reverb. The values that may be specified are as follows.

Table 15-18

Value	Description
SPU_ON	Set reverb on
SPU_OFF	Set reverb off

g) External Digital Input Mixing On/Off

Reverb is set in attr.cd.mix. The values that may be specified are as follows. External digital input is not output unless this value is on.

Table 15-19

Value	Description
SPU_ON	Set mixing on
SPU_OFF	Set mixing off

Return value

None

Remarks

See also: SpuGetCommonAttr (p. 15-19), SpuSetVoiceAttr (p. 15-114).

SpuSetCommonCDMix

Sets CD input ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuSetCommonCDMix (
long on_off
)
```

Arguments

on_off CD Input on/off

Explanation

This function sets the CD input on/off.

It performs a process equivalent to the

SPU_COMMON_CDMIX

SpuSetCommonAttr() mask setting.

Refer to SpuSetCommandAttr() for values set to the CD Input on/off (*on_off*).

Return value

None

Remarks

See also: SpuSetCommonAttr(), SpuGetCommonCDMix().

SpuSetCommonCDReverb

Sets CD input reverb ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuSetCommonCDReverb (
long on_off
)
```

Arguments

on_off CD Input reverb on/off

Explanation

This function sets the CD input on/off.

It performs a process equivalent to the

SPU_COMMON_CDREV

mask setting of SpuSetCommonAttr().

Refer to SpuSetCommandAttr() for values set to the CD Input on/off (*on_off*).

Return value

None

Remarks

See also: SpuSetCommonAttr(), SpuGetCommonCDReverb().

SpuSetCommonCDVolume

Sets CD input volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuSetCommonCDVolume (
short cdvolL,
short cdvolR
)
```

Arguments

cdvolL CD input volume (left)
cdvolR CD input volume (right)

Explanation

This function sets the CD input volume.

It performs a process equivalent to the

SPU_COMMON_CDVOLL

SPU_COMMON_CDVOLR

mask settings of SpuSetCommonAttr().

Refer to SpuSetCommandAttr() for values set to the CD input volumes *cdvolL* and *cdvolR*.

Return value

None

Remarks

See also: SpuSetCommonAttr(), SpuGetCommonCDVolume().

SpuSetCommonMasterVolume

Sets master volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuSetCommonMasterVolume (
short mvoll,
short mvolR
)
```

Arguments

mvoll Master volume (left)
mvolR Master volume (left)

Explanation

This function sets the master volume.

It performs a process equivalent to the

SPU_COMMON_MVOLL

SPU_COMMON_MVOLR

mask settings of SpuSetCommonAttr().

The volume mode is 'direct mode' and the range of the values which can be set to the *mvoll* and *mvolR* volumes is equal to that of the 'direct mode' in SpuSetVoiceAttr().

If you would like to set both the volume and the volume mode simultaneously, use SpuSetCommonMasterVolumeAttr().

Refer to SpuSetCommonAttr() and SpuSetVoiceAttr() for values set to the *mvoll* and *mvolR* volumes.

Return value

None

Remarks

See also: SpuSetCommonAttr(), SpuSetVoiceAttr(), SpuSetCommonMasterVolumeAttr(), SpuGetCommonMasterVolume(), SpuGetCommonMasterVolumeAttr().

SpuSetCommonMasterVolumeAttr

Sets master volume/master volume mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuSetCommonMasterVolumeAttr (
short mvoll,
short mvolR,
short mvolModeL,
short mvolModeR
)
```

Arguments

<i>mvoll</i>	Master volume (left)
<i>mvolR</i>	Master volume (right)
<i>mvolModeL</i>	Master volume mode (left)
<i>mvolModeR</i>	Master volume mode (right)

Explanation

This function sets the master volume and master volume mode.

It performs a process equivalent to the

SPU_COMMON_MVOLL

SPU_COMMON_MVOLR

SPU_COMMON_MVOLMODEL

SPU_COMMON_MVOLMODER

mask settings of SpuSetCommonAttr().

Refer to SpuSetCommonAttr() and SpuSetVoiceAttr() for values set to the volume modes *mvolModeL* and *mvolModeR* and also the volumes *mvoll* and *mvolR*.

Return value

None

Remarks

See also: SpuSetCommonAttr(), SpuSetVoiceAttr(), SpuSetCommonMasterVolume(), SpuGetCommonMasterVolume(), SpuGetCommonMasterVolumeAttr().

SpuSetEnv

Sets basic sound library environment.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetEnv(*env)
SpuEnv *env;
```

Arguments

env Basic sound library environment attribute

Explanation

This function sets the basic sound library environment. In *env.mask* it is possible to set each attribute by ORing the desired attributes (only one attribute can currently be set).

Attribute Bit	Contents
SPU_ENV_EVENT_QUEUEING	Queues an event

When *env.mask* is set to 0, all the attributes will be set.

The individual setting conditions are explained below:

(1) Queueing an event

Specifying the following values in *env.queueing* establishes whether the following events are queued or not: Key On/Off, Pitch LFO Voice Setting, Noise Voice Setting, and Reverb Voice Setting.

Specified Value	Contents
SPU_ON	Queues an event
SPU_OFF	Performs event immediately. Does not queue event (Default value)

Return value

None

Remarks

See also: `SpuSetKey()`, `SpuSetKeyOnWithAttr()`, `SpuSetPitchLFOVoice()`, `SpuSetNoiseVoice()`, `SpuSetReverbVoice()`, `SpuFlush()`, `SpuEnv()`.

SpuSetIRQ

Sets interrupt request ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpuSetIRQ(on_off)
long on_off;
```

Arguments

on_off Sets interrupt request ON/OFF/RESET

Table 15–20

Value	Description
SPU_ON	Set interrupt request
SPU_OFF	Cancel interrupt request
SPU_RESET	Reset interrupt request (= set after cancel)

Explanation

Sets interrupt request ON/OFF.

Return value

Set value.

Table 15–21

Value	Description
SPU_ON	Set interrupt request
SPU_OFF	Cancel interrupt request
SPU_RESET	Reset interrupt request

Remarks

See also: SpuGetIRQ (p. 15-20).

SpuSetIRQAddr

Sets interrupt request address.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	6/22/98

Syntax

```
unsigned long SpuSetIRQAddr(addr)
unsigned long addr;
```

Arguments

addr Interrupt request address

Explanation

Sets interrupt request address value. The address value must be

- In bytes
- Divisible by 8
- Less than 512 KB

The interrupt request is carried out when the read/write to the set address has been performed.

Return value

Returns the value of the address that is set.

If the value of the set address *addr* is not divisible by 8, the set value is advanced to the next value divisible by 8, and that value is set and returned.

If the address exceeds 512 KB, 0 is returned.

Remarks

See also: SpuGetIRQAddr (p. 15-27), SpuSetIRQ (p. 15-82), SpuGetIRQ (p. 15-20).

SpuSetIRQCallback

Sets callback at the time of an interrupt request.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
SpuSetIRQCallback(func)
SpuIRQCallbackProc func;
```

Arguments

func The callback function activated at the time of an interrupt request

Explanation

Sets a callback function activated at the time of an interrupt request.

If the callback function value is set to NULL, the callback is cleared.

Return value

Pointer to the previously set function.

Remarks

See also: SpuSetIRQ (p. 15-82), SpuSetIRQAddr (p. 15-88).

SpuSetKey

Sets key on/key off for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
void SpuSetKey(on_off, voice_bit)
long on_off;
unsigned long voice_bit;
```

Arguments

on_off Sets key on/key off
voice_bit Set voice

Explanation

Sets each voice specified by *voice_bit* as key on/key off.

Values that may be set by *on_off* are as follows.

Table 15-22

Value	Description
SPU_ON	Set key on
SPU_OFF	Set key off

Sets *voice_bit* by ORing together SPU_0CH, SPU_1CH...SPU_23CH.

Return value

None

Remarks

When setting key on for voice 0 and voice 2, call SpuSetKey() as follows.

```
SpuSetKey (SPU_ON,   /* set key on */
           SPU_0CH | SPU_2CH);   /* 0 ch and 2 ch */
```

See also: SpuSetKeyOnWithAttr (p. 15-92), SpuSetVoiceAttr (p. 15-110).

SpuSetKeyOnWithAttr

Sets key on with attributes for voice using attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
void SpuSetKeyOnWithAttr(*attr)
SpuVoiceAttr *attr;
```

Arguments

attr Pointer to voice attributes

Explanation

Specifies attributes for each voice and sets key on.

Explicitly specify the voices to be produced by ORing together SPU_0CH, SPU_1CH...SPU_23CH in *attr.voice*.

You can each each attribute in *attr.voice* by ORing together the terms shown below.

Table 15–23

Attribute	Description
SPU_VOICE_VOLL	Volume (left)
SPU_VOICE_VOLR	Volume (right)
SPU_VOICE_VOLMODEL	Volume mode (left)
SPU_VOICE_VOLMODER	Volume mode (right)
SPU_VOICE_PITCH	Interval (pitch specification)
SPU_VOICE_NOTE	Interval (note specification)
SPU_VOICE_SAMPLE_NOTE	Waveform data sample note
SPU_VOICE_WDSA	Waveform data start address
SPU_VOICE_ADSR_AMODE	ADSR Attack rate mode
SPU_VOICE_ADSR_SMODE	ADSR Sustain rate mode
SPU_VOICE_ADSR_RMODE	ADSR Release rate mode
SPU_VOICE_ADSR_AR	ADSR Attack rate
SPU_VOICE_ADSR_DR	ADSR Decay rate
SPU_VOICE_ADSR_SR	ADSR Sustain rate
SPU_VOICE_ADSR_RR	ADSR Release rate
SPU_VOICE_ADSR_SL	ADSR Sustain level
SPU_VOICE_ADSR_ADSR1	ADSR adsr1 for VagAtr
SPU_VOICE_ADSR_ADSR2	ADSR adsr2 for VagAtr
SPU_VOICE_LSAX	Loop start address

If *attr.mask* is 0, all attributes will be set.

The individual settings of each attribute are described in *SpuSetVoiceAttr*.

Return value

None

Remarks

See also: `SpuSetKey` (p. 15-91), `SpuSetVoiceAttr` (p. 15-110), `SpuGetVoiceAttr` (p. 15-42), `SpuVoiceAttr` (p. 15-10).

SpuSetMute

Sets sound muting ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpuSetMute(on_off)
long on_off;
```

Arguments

on_off Mute ON/OFF

Explanation

Sets sound muting ON/OFF. *on_off* setting values are as follows.

Table 15-24

Value	Description
SPU_ON	Mute off
SPU_OFF	Mute on

However, CD input and external digital input are not muted by this mute ON/OFF.

Return value

Set value.

SPU_ON Mute off
SPU_OFF Mute on

Remarks

See also: SpuGetMute (p. 15-29).

SpuSetNoiseClock

Sets noise source clock.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	5/22/97

Syntax

```
long SpuSetNoiseClock(n_clock)
long n_clock;
```

Arguments

n_clock Noise source clock

Explanation

Set noise source clock to *n_clock*. The clock value *n_clock* must be 0-0x3f.

Noise source clock value will be applied to all voices that noise source is applied via SpuSetNoiseVoice().

Return value

Noise source clock value set.

Remarks

See also: SpuGetNoiseClock (p. 15-30), SpuSetNoiseVoice (p. 15-96), SpuGetNoiseVoice (p. 15-31).

SpuSetNoiseVoice

Sets noise source ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.0	10/01/97

Syntax

```
unsigned long SpuSetNoiseVoice(on_off, voice_bit)
long on_off;
unsigned long voice_bit;
```

Arguments

on_off Sets noise source ON (SPU_ON), OFF (SPU_OFF), or direct bit pattern (SPU_BIT)
voice_bit Set voice

Explanation

Sets each voice specified by *voice_bit* as noise on/noise off (i.e., use/do not use noise).

Values that may be set by *on_off* are as follows:

Table 15–25

Value	Description
SPU_ON	Sets noise source
SPU_OFF	Releases noise source
SPU_BIT	Set direct bit pattern

Specify the voices set in *voice_bit* by ORing together SPU_0CH-SPU_23CH.

If *on_off* is equal to SPU_BIT, bits that are 1 in the *voice_bit* bit pattern are set directly on, and those that are 0 are set off.

Return value

Returns the noise source ON/OFF value of the current voice. OR together SPU_0CH-SPU_23CH.

Distinguishes the noise source ON/OFF value by ANDing the return value and SPU_xxCH(xx=0~23).

Table 15–26

Result of AND	Description
0	Sets noise source off
Other than 0	Sets noise source on

Remarks

Set voice 0 and voice 2 noise source on as follows:

```
    spuSetNoiseVoice(SPU_ON,      /*set noise source on*/
    SPU_0CH | SPU_2CH);          /*0 ch and 2 ch*/
```

See also: SpuSetNoiseClock (p. 15-95), SpuGetNoiseClock (p. 15-30), SpuGetNoiseVoice (p. 15-31).

SpuSetPitchLFOVoice

Sets pitch LFO ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	10/01/97

Syntax

```
unsigned long SpuSetPitchLFOVoice(on_off, voice_bit)
```

```
long on_off;
```

```
unsigned long voice_bit;
```

Arguments

<i>on_off</i>	SPU_ON	Sets pitch LFO on
	SPU_OFF	Sets pitch LFO off
	SPU_BIT	Sets direct bit pattern
<i>voice_bit</i>	Set voice	

Explanation

Sets pitch LFO ON/OFF for each voice.

Voice *n*, having pitch LFO set on, is set so that LFO sets pitch when the volume of voice (*n*-1) undergoes a time change. To make this pitch LFO valid, voice *n* and voice (*n*-1) must produce sound, and the volume of voice (*n*-1) must be set to 0 in advance. Voice (*n* - 1) can produce sound at an optional timing after voice *n* produces sound; LFO is applied at the moment when voice (*n*-1) produces sound.

Specify the voices set in *voice_bit* by ORing together SPU_0CH, SPU_1CH...SPU_23CH.

If *on_off* is equal to SPU_BIT, bits that are 1 in the *voice_bit* bit pattern are set directly on, and those that are 0 are set off.

Return value

Returns the pitch LFO ON/OFF value of the current voice. OR together SPU_0CH, SPU_1CH...SPU_23CH.

Distinguishes the pitch LFO ON/OFF value by ANDing the return value and SPU_xxCH (xx=0~23).

Table 15-27

Result of AND	Description
0	Sets pitch LFO off
Other than 0	Sets pitch LFO on

Remarks

See also: SpuGetPitchLFOVoice (p. 15-32), SpuSetKey (p. 15-91), SpuSetKeyOnWithAttr (p. 15-92).

SpuSetReverb

Sets reverb ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpuSetReverb(on_off)
long on_off;
```

Arguments

<i>on_off</i>	SPU_ON	Set reverb on
	SPU_OFF	Set reverb off

Explanation

Sets reverb ON/OFF.

If a reverb work area is not reserved with `SpuReserveReverbWorkArea`, when `SPU_ON` is specified by `on_off`, `SpuSetReverb` checks whether the area used as a work area by `SpuMalloc/`
`SpuMallocWithStartAddr` is being used as another area, and if it is being used, reverb is set off and `SPU_OFF` is returned.

If it is not being used, reverb is set on and `SPU_ON` is returned. When a reverb work area is reserved, an `on_off` value of `SPU_ON` sets reverb and returns `SPU_ON`.

Return value

Set value

SPU_ON	Reverb on
SPU_OFF	Reverb off

Remarks

See also: `SpuGetReverb` (p. 15-33), `SpuSetReverbModeParam` (p. 15-100), `SpuReserveReverbWorkArea` (p. 15-76).

SpuSetReverbDepth

Sets the reverb depth parameter.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpuSetReverbDepth(*attr)
SpuReverbAttr *attr;
```

Arguments

attr Pointer to reverb attribute

Explanation

Sets the reverb depth parameter attribute.

You can set each attribute (members of *attr*) in *attr.mask* by ORing together the terms shown below.

Table 15–28

Value	Description
SPU_REV_DEPTHL	Reverb depth (left)
SPU_REV_DEPTHR	Reverb depth (right)

If *attr.mask* is 0, left and right attributes are set simultaneously.

a) Reverb Depth

Reverb depth is set independently for left and right. The range for this specification is -0x8000 -0x7fff.

If the value set is negative, the reverb sound (wet) phase is inverted.

Return value

Always returns 0.

Remarks

See also: *SpuSetReverbModeParam* (p. 15-100), *SpuGetReverbModeParam* (p. 15-37), *SpuReverbAttr* (p. 15-9).

SpuSetReverbModeDelayTime

Sets reverb delay time.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuSetReverbModeDelayTime (
long delay
)
```

Arguments

delay Reverb delay time

Explanation

This function sets the reverb delay time.

It performs a process equivalent to the

SPU_REV_DELAYTIME

mask setting of SpuSetReverbModeParam().

The reverb mode is effective only in SPU_REV_MODE_ECHO or SPU_REV_MODE_DELAY. There will be no effect if any other mode is set.

Refer to SpuSetReverbModeParam() for values set to the reverb delay time *delay*.

Return value

None

Remarks

See also: SpuSetReverbModeParam(), SpuGetReverbModeParam(), SpuGetReverbModeDelayTime().

SpuSetReverbModeDepth

Sets reverb mode depth.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuSetReverbModeDepth (
short depthL,
short depthR
)
```

Arguments

depthL reverb depth (left)
depthR reverb depth (right)

Explanation

This function sets the reverb depth.

It performs a process equivalent to the

SPU_REV_DEPTHL

SPU_REV_DEPTHR

mask settings of SpuSetReverbModeParam().

Refer to SpuSetReverbModeParam() for values set to the reverb depths *depthL* and *depthR*.

Return value

None

Remarks

See also: SpuSetReverbModeParam(), SpuGetReverbModeParam(), SpuGetReverbModeDepth(), SpuSetReverbDepth().

SpuSetReverbModeFeedback

Sets reverb feedback.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuSetReverbModeFeedback (
long feedback
)
```

Arguments

feedback Reverb feedback

Explanation

This function sets the reverb feedback.

It performs a process equivalent to the

SPU_REV_FEEDBACK

mask setting of SpuSetReverbModeParam().

The reverb mode is effective only in SPU_REV_MODE_ECHO or SPU_REV_MODE_DELAY. There will be no effect if any other mode is set.

Refer to SpuSetReverbModeParam() for values set to the reverb feedback *feedback*.

Return value

None

Remarks

See also: SpuSetReverbModeParam(), SpuGetReverbModeParam(), SpuGetReverbModeFeedback().

SpuSetReverbModeParam

Sets reverb mode and parameters.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
long SpuSetReverbModeParam(*attr)
SpuReverbAttr *attr;
```

Arguments

attr Pointer to reverb attributes

Explanation

Sets reverb mode and parameter attributes.

You can set each attribute (members of *attr*) in *attr.mask* by ORing together the terms shown below. If *attr.mask* is 0, all attributes are set.

Table 15–29

Attribute	Description
SPU_REV_MODE	Mode setting
SPU_REV_DEPTHL	Reverb depth (left)
SPU_REV_DEPTHR	Reverb depth (right)
SPU_REV_DELAYTIME	Delay time (ECHO, DELAY only)
SPU_REV_FEEDBACK	Feedback (ECHO, DELAY only)

a) **Reverb Mode** (Table 8-31)

Sets reverb mode. Setting attributes other than "Depth" (reverb depth) varies according to the reverb mode.

Table 15–30: Reverb Mode and Other Attributes

<i>attr.mode</i>	mode	Delay time	Feedback
SPU_REV_MODE_OFF	off		
SPU_REV_MODE_ROOM	room		
SPU_REV_MODE_STUDIO_A		studio (small)	
SPU_REV_MODE_STUDIO_B		studio (med)	
SPU_REV_MODE_STUDIO_C		studio (big)	
SPU_REV_MODE_HALL	hall		
SPU_REV_MODE_SPACE	space echo		
SPU_REV_MODE_ECHO	echo	can set	can set
SPU_REV_MODE_DELAY	delay	can set	can set
SPU_REV_MODE_PIPE	half echo		

When reverb mode is changed (this happens even at initial setting because the initial value is SPU_REV_MODE_OFF), the internal reverb Depth value is 0 even if Depth was previously set in SpuSetReverbModeParam(). This is because the work area size changes when this mode changes, so incorrect data in the work area produces noise. So after the reverb mode changes, Depth needs to be reset in SpuSetReverbModeParam() or SpuSetReverbDepth().

Based on reverb characteristics, the time to complete one scan of the work area is estimated and the mode/depth are set; or, after the mode is set, the work area data is erased then Depth is set (to be described later).

The sound buffer volume occupied by the work area depends on the reverb mode as shown in Table 8-31. However, this area is managed by a memory management mechanism such as `SpuMalloc()`. See `SpuMalloc()` for details.

Table 15-31: Volume Occupied by Reverb Mode In Sound Buffer

attr.mode	mode	hexadecimal	decimal
SPU_REV_MODE_OFF	off	0/80 (*)	0/128 (*)
SPU_REV_MODE_ROOM	room	26c0	9920
SPU_REV_MODE_STUDIO_A	studio (small)	1f40	8000
SPU_REV_MODE_STUDIO_B	studio (med)	4840	18496
SPU_REV_MODE_STUDIO_C	studio (big)	6fe0	28640
SPU_REV_MODE_HALL	hall	ade0	44512
SPU_REV_MODE_SPACE	space echo	f6c0	63168
SPU_REV_MODE_ECHO	echo	18040	98368
SPU_REV_MODE_DELAY	delay	18040	98368
SPU_REV_MODE_PIPE	half echo	3c00	15360

(*) If `SpuReserveReverbWorkArea (SPU_ON)` is used for address setting, it takes 128 bytes even if the mode is off. If `SpuReserveReverbWorkArea (SPU_OFF)` is used, it takes 0 bytes.

If `SPU_REV_MODE_CLEAR_WA` is ORed in `attr.mode`, it clears the area needed by reverb mode set when setting reverb mode. This is a measure against noise when changing modes. However, the sound buffer is cleared by synchronous DMA transfer, so other processing (drawing, sound generation) is not performed during this processing, and some wait time is needed, depending on the reverb type.

`SpuClearReverbWorkArea()` is used to forcibly clear the area used by the reverb mode specified when setting reverb mode with optional timing.

b) Reverb Depth

Set in `attr.depth`, independently for left and right. Values are set in the range -0x8000 - 0x7fff.

If the set value is negative, the reverb sound (wet) phase is inverted.

c) Delay Time

Set in `attr.delay`. Values are set in the range 0-127. Valid when mode is `SPU_REV_MODE_ECHO` or `SPU_REV_MODE_DELAY`.

d) Feedback

Valid when mode is `SPU_REV_MODE_ECHO` or `SPU_REV_MODE_DELAY`.

Delay time is set in `attr.feedback` with values from 0 to 127.

Return value

If the area used as a work area by the new mode is being used as another area by `SpuMalloc()`/
`SpuMallocWithStartAddr()`, none of the set reverb attributes are set and `SPU_ERROR` is returned. If it is not being used, the set reverb attributes are set and 0 is returned.

`SPU_ERROR` is also returned when an invalid `SPU_REV_MODE` is set.

Remarks

See also: `SpuGetReverbModeParam` (p. 15-37), `SpuMalloc` (p. 15-68), `SpuMallocWithStartAddr` (p. 15-69), `SpuReserveReverbWorkArea` (p. 15-76), `SpuClearReverbWorkArea` (p. 15-15).

SpuSetReverbModeType

Sets reverb mode type.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	2/13/97

Syntax

```
void SpuSetReverbModeType (
long type
)
```

Arguments

type Reverb mode type

Explanation

This function sets the reverb mode type.

It performs a process equivalent to the

SPU_REV_MODE

mask setting of SpuSetReverbModeParam().

Refer to SpuSetReverbModeParam() for values set to the reverb mode type *type*.

Return value

None

Remarks

See also: SpuSetReverbModeParam(), SpuGetReverbModeParam(), SpuMalloc(), SpuMallocWithStartAddr(), SpuReserveReverbWorkArea(), SpuClearReverbWorkArea(), SpuGetReverbModeType().

SpuSetReverbVoice

Sets reverb ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.0	10/01/97

Syntax

```
unsigned long SpuSetReverbVoice(on_off, voice_bit)
long on_off;
unsigned long voice_bit;
```

Arguments

on_off Sets reverb ON (SPU_ON), OFF (SPU_OFF), or direct bit pattern (SPU_BIT)
voice_bit Set voice

Explanation

Sets each voice specified by *voice_bit* as reverb on/reverb off.

Values that may be set by *on_off* are as follows:

Table 15-32

Value	Description
SPU_ON	Set reverb on
SPU_OFF	Set reverb off
SPU_BIT	Set direct bit pattern

Specify the voices set in *voice_bit* by ORing together SPU_0CH-SPU_23CH.

If *on_off* is equal to SPU_BIT, bits that are 1 in the *voice_bit* bit pattern are set directly on, and those that are 0 are set off.

Return value

Returns the reverb ON/OFF value of the current voice. OR together SPU_0CH-SPU_23CH.

Distinguishes the noise source ON/OFF value by ANDing the return value and SPU_xxCH(xx=0~23).

Table 15-33

Result of AND	Description
0	Sets reverb off
Other than 0	Sets reverb on

Remarks

Set voice 0 and voice 2 reverb on as follows:

```
    SpuSetReverbVoice(SPU_ON,    /*set reverb on*/  
                      SPU_0CH | SPU_2CH);    /*0 ch and 2 ch*/
```

See also: SpuGetReverbVoice (p. 15-39).

SpuSetTransferCallback

Sets callback function when DMA transfer is completed.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.1	7/31/96

Structure

```
SpuTransferCallbackProc SpuSetTransferCallback (
SpuTransferCallbackProc func
)
```

Arguments

func Starts callback function when DMA transfer is completed.

Explanation

Sets callback function started when DMA transfer is completed.

If the value of the callback function is set to NULL, the callback is cleared.

When a callback is set using SpuSetTransferCallback() and starting at DMA transfer completion, SpulsTransferCompleted does not function.

Return value

This functions returns the previously set callback function. If a callback function is not set, the function returns NULL.

Remarks

See also: SpuWrite (p. 15-142), SpuWrite0 (p. 15-143), SpuWritePartly (p. 15-144), SpuRead (p. 15-73), SpuReadDecodedData (), SpuSetTransferMode (), SpuGetTransferMode (), SpulsTransferCompleted (p. 15-67).

SpuSetTransferMode

Sets sound buffer transfer mode.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.0	7/31/96

Syntax

```
long SpuSetTransferMode(mode)
long mode;
```

Arguments

mode	SPU_TRANSFER_BY_DMA	DMA transfer setting
	SPU_TRANSFER_BY_IO	I/O transfer setting

Explanation

Sets mode when transferring data from main memory to the sound buffer.

Mode values are as shown below. DMA transfer is the default.

Table 15-34

Value	Description
SPU_TRANSFER_BY_DMA	DMA transfer setting Can do other processing during transfer
SPU_TRANSFER_BY_IO	I/O transfer setting Transfer uses the CPU; cannot do other processing during transfer

These specifications are valid only when transferring data from main memory to the sound buffer. DMA transfer is always used when transferring data from the sound buffer to main memory.

When transfer is done without first calling this function, transfer mode is set to a previously determined value.

Return value

Set transfer mode

SPU_TRANSFER_BY_DMA	DMA transfer setting
SPU_TRANSFER_BY_IO	I/O transfer setting

Remarks

See also: SpuGetTransferMode (p. 15-40), SpuWrite (p. 15-142), SpuWrite0 (), SpuWritePartly ().

SpuSetTransferStartAddr

Sets sound buffer transfer destination/transfer source start address.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
unsigned long SpuSetTransferStartAddr(addr)
unsigned long addr;
```

Arguments

addr Sound buffer transfer destination/transfer source start address

Explanation

Sets a starting address specified in *addr* for transferring from main memory to the sound buffer, and from the sound buffer to main memory.

However, the start address value must be

- In bytes.
- Divisible by 8.
- Greater than 0x100f and less than 512 KB for transfers to the sound buffer.
- Between 0x0-0xffff for transfers from the sound buffer.

For transfers from the 0x0 - 0xffff area, see `SpuReadDecodeData()`. 0x1000 - 0x100f is reserved for the system.

Return value

Set start address value.

If the value of the set address *addr* is not divisible by 8, the set value is advanced to the next value divisible by 8 and that value is returned.

For values smaller than 0x100f or greater than 512 KB, 0 is returned.

Remarks

See also: `SpuGetTransferStartAddr` (p. 15-41), `SpuWrite` (p. 15-142), `SpuWrite0 ()`, `SpuWritePartly ()`, `SpuRead` (p. 15-73), `SpuReadDecodedData ()`.

SpuSetVoiceADSR

Sets ADSR.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceADSR (  
int voiceNum,  
unsigned short AR,  
unsigned short DR  
unsigned short SR,  
unsigned short RR,  
unsigned short SL  
)
```

Arguments

<i>voiceNum</i>	Voice number (0 - 23)
<i>AR</i>	ADSR attack rate
<i>DR</i>	ADSR decay rate
<i>SR</i>	ADSR sustain rate
<i>RR</i>	ADSR release rate
<i>SL</i>	ADSR sustain level

Explanation

This function sets each ADSR attribute used in the S voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_ADSR_AR
SPU_VOICE_ADSR_DR
SPU_VOICE_ADSR_SR
SPU_VOICE_ADSR_RR
SPU_VOICE_ADSR_SL

For attack, sustain, and release rate, the rate modes are as follows:

Rate	Rate Mode
Attack Rate	SPU_VOICE_LINEARIncN (Linear Increase)
Sustain Rate	SPU_VOICE_LINEARDecN (Linear Decrease)
Release Rate	SPU_VOICE_LINEARDecN (Linear Decrease)

If you want to set multiple rate modes at the same time, use SpuSetVoiceADSRAttr.

Refer to SpuSetVoiceAttr for values that can be specified in each rate.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceAR(), SpuSetVoiceDR(), SpuSetVoiceSR(), SpuSetVoiceRR(), SpuSetVoiceSL().

SpuSetVoiceADSRAttr

Sets ADSR and each mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceARAttr (
int voiceNum,
unsigned short AR,
unsigned short DR
unsigned short SR,
unsigned short RR,
unsigned short SL
long ARmode,
long SRmode,
long RRmode
)
```

Arguments

<i>voiceNum</i>	Voice number (0 - 23)
<i>AR</i>	ADSR attack rate
<i>DR</i>	ADSR decay rate
<i>SR</i>	ADSR sustain rate
<i>RR</i>	ADSR release rate
<i>SL</i>	ADSR sustain level
<i>ARmode</i>	ADSR attack rate mode
<i>SRmode</i>	ADSR sustain rate mode
<i>RRmode</i>	ADSR release rate mode

Explanaton

This function sets ADSR attributes and mode. It performs a process which corresponds to SpuSetVoiceAttr() mask specifications

```
SPU_VOICE_ADSR_AR, SPU_VOICE_ADSR_AMODE
SPU_VOICE_ADSR_DR
SPU_VOICE_ADSR_SR, SPU_VOICE_ADSR_SMODE
SPU_VOICE_ADSR_RR, SPU_VOICE_ADSR_RMODE
SPU_VOICE_ADSR_SL
```

Refer to SpuSetVoiceAttr for values that can be specified in each rate and rate mode.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceADSR(), SpuSetVoiceAR(), SpuSetVoiceDR(), SpuSetVoiceSR(), SpuSetVoiceRR(), SpuSetVoiceSL(), SpuSetVoiceARAttr(), SpuSetVoiceSRAttr(), SpuSetVoiceRRAttr().

SpuSetVoiceAR

Sets ADSR attack rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceAR (
int voiceNum,
unsigned short AR
)
```

Arguments

voiceNum Voice number (0 - 23)
AR ADSR attack rate

Explanation

This function sets ADSR attack rate in voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_ADSR_AR

ADSR attack rate mode becomes SPU_VOICE_LINEARIncN (Linear increase mode) . If you want to set ADSR attack rate and ADSR attack rate mode at the same time, use SpuSetVoiceARAttr.

Refer to SpuSetVoiceAttr for values that can be specified in ADSR attack rate, AR.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceARAttr().

SpuSetVoiceARAttr

Sets ADSR attack rate / attack rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceARAttr (
int voiceNum,
unsigned short AR,
long Armode
)
```

Arguments

voiceNum Voice number (0 - 23)
AR ADSR attack rate
Armode ADSR attack rate mode

Explanation

This function sets ADSR attack rate / ADSR attack rate mode used in voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specifications

```
SPU_VOICE_ADSR_AR
SPU_VOICE_ADSR_AMODE
```

Refer SpuSetVoiceAttr for values that can be specified in ADSR attack rate AR and ADSR attack rate mode, ARmode.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceAR().

SpuSetVoiceAttr

Sets attributes for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

```
void SpuSetVoiceAttr (
SpuVoiceAttr *attr
)
```

Arguments

attr Pointer to voice attributes

Explanation

Sets attributes for each voice

Explicitly set the voices you wish to set by ORing together SPU_0CH, SPU_1CH, ...SPU_23CH in *attr.voice*.

You can set each attribute in *attr.voice* by ORing together the terms shown below.

Table 15–35

Attribute	Description
SPU_VOICE_VOLL	Volume (left)
SPU_VOICE_VOLR	Volume (right)
SPU_VOICE_VOLMODEL	Volume mode (left)
SPU_VOICE_VOLMODER	Volume mode (right)
SPU_VOICE_PITCH	Interval (pitch specification)
SPU_VOICE_NOTE	Interval (note specification)
SPU_VOICE_SAMPLE_NOTE	Waveform data sample note
SPU_VOICE_WDSA	Waveform data start address
SPU_VOICE_ADSR_AMODE	ADSR Attack rate mode
SPU_VOICE_ADSR_SMODE	ADSR Sustain rate mode
SPU_VOICE_ADSR_RMODE	ADSR Release rate mode
SPU_VOICE_ADSR_AR	ADSR Attack rate
SPU_VOICE_ADSR_DR	ADSR Decay rate
SPU_VOICE_ADSR_SR	ADSR Sustain rate
SPU_VOICE_ADSR_RR	ADSR Release rate
SPU_VOICE_ADSR_SL	ADSR Sustain level
SPU_VOICE_ADSR_ADSR1	ADSR adsr1 for 'VagAtr'
SPU_VOICE_ADSR_ADSR2	ADSR adsr2 for 'VagAtr'
SPU_VOICE_LSAX	Loop start address

If *attr.mask* is 0, all attributes will be set.

The individual settings are described below.

a) Volume and Volume Mode

Each Volume Mode and the range of possible volume settings for each Volume Mode are provided below.

Table 15–36: Volume Mode and Volume Setting Ranges

Mode (phase)	SPU_VOICE_VOLMODEx	SPU_VOICE_VOLx
Direct mode	SPU_VOICE_DIRECT	-0x4000 - 0x3fff
Linear inc. mode	SPU_VOICE_LINEARIncN	0x00 - 0x7f (normal)
Linear inc. mode	SPU_VOICE_LINEARIncR	0x00 - 0x7f (inverted)
Linear dec. mode	SPU_VOICE_LINEARDecN	0x00 - 0x7f (normal)
Linear dec. mode	SPU_VOICE_LINEARDecR	0x00 - 0x7f (inverted)
Expon. inc. mode	SPU_VOICE_EXPIncN	0x00 - 0x7f (normal)
Expon. inc. mode	SPU_VOICE_EXPIncR	0x00 - 0x7f (inverted)
Expon. dec. mode	SPU_VOICE_EXPDec	0x00 - 0x7f

- **Direct Mode**
Fixed volume mode. In normal usage, this mode produces sound. When the set volume is negative, its phase is reversed. In this situation, "inverted phase", described below, is valid.
 - **Linear Increase Mode (Normal Phase)**
When the current volume value is positive and this mode is specified as the sound production status, volume increases linearly from the current value to the maximum value.
 - **Linear Increase Mode (Inverted Phase)**
When the current volume value is negative (inverted phase) and this mode is specified as the sound production status, volume increases linearly from the current value to the maximum value, with phase inverted.
 - **Linear Decrease Mode (Normal Phase)**
When the current volume value is positive and this mode is specified as the sound production status, volume decreases linearly from the current value to the minimum volume value.
 - **Linear Decrease Mode (Inverted Phase)**
When the current volume value is negative (inverted phase) and this mode is specified as the sound production status, volume decreases linearly from the current value to the minimum volume value, with phase inverted.
 - **Exponential Increase Mode (Normal Phase)**
When the current volume value is positive and this mode is specified as the sound production status, volume increases exponentially from the current value to the maximum value.
 - **Exponential Increase Mode (Inverted Phase)**
When the current volume value is negative (inverted phase) and this mode is specified as the sound production status, volume increases exponentially from the current value to the maximum value, with phase inverted.
 - **Exponential Decrease Mode**
When this mode is specified as the sound production status, whether the current volume value is positive or negative, volume decreases exponentially from the current value to the minimum volume value.
- b) **Interval (set pitch, set note)**
Interval may be set by the two methods listed below.
- **Pitch specification**
Specify an interval in attr.pitch in the range 0x0000-0x3fff.
See Table 8-38 for an explanation of the meaning of these values. The only unit shown in the table is octaves, but any value in the range 0x0000-0x3fff may be set.

Table 15–37: Pitch Specification Values and Interval

Value Set	0x0200	0x0400	0x0800	0x1000	0x2000	0x3fff
Interval	- 3 oct.	- 2 oct.	- 1 oct.	tone	+ 1 oct.	+ 2 oct.

- Note specification
An interval is set in attr.note as follows, using a 16-bit value for note and cent (here, the value of a half tone divided by 128).
This setting cannot be used unless the waveform data sample note feature, described below, is set.

Table 15–38: Note Specification Values

Bit	Value Set
Upper 8 bits	MIDI note number
Lower 8 bits	Cent (expressed as a half tone divided by 128)

c) Waveform Data Sample Note

Sets interval in attr.sample_note at the time of sampling, using a 16-bit value for note and cent (here, the value of a half tone divided by 128). Setting this value makes it possible to set b) Interval--Note specification as above.

Table 15–39: Waveform Data Sample Note Specification Values

Bit	Value Set
Upper 8 bits	MIDI note number
Lower 8 bits	Cent (expressed as half tone divided by 128)

d) Waveform Data Start Address

The sound buffer starting address of the waveform data you want to produce in the voice is set in attr.addr.

e) Loop Start Address

If waveform data that generates sound in a voice is created with a loop specified, and if the waveform starting address is set, the loop start address is usually automatically identified and set. Explicit setting is unnecessary.

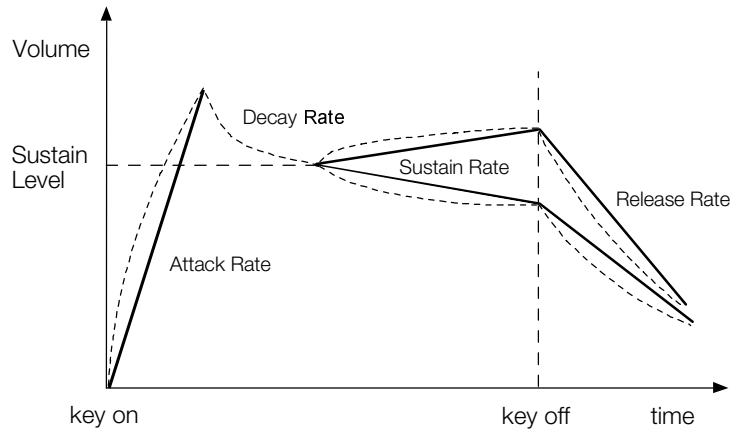
However, when you wish to set a loop start address dynamically at the time of execution, you must set the address that is the starting point of the loop in the sound buffer in attr.loop_addr.

If a loop was not set at the time of waveform data creation, even if SPU_VOICE_LSAX is specified and set in attr.loop_addr, that setting is invalid.

f) ADSR

A conceptual diagram of ADSR is shown below.

Figure 15-1: ADSR Conceptual Diagram



ADSR attributes are set by the structure members listed in Table 8-41; the range of these attributes is listed in Table 8-41.

Table 15-40: Parameters and Structure Members

	Attribute	Structure Member
Rate	Attack rate	attr.ar, attr.a_mode
	Decay rate	attr.dr
	Sustain rate	attr.sr, attr.s_mode
	Release rate	attr.rr, attr.r_mode
Level	Sustain level	attr.sl

Table 15-41: Rate and Level Setting Ranges

Attribute	Structure Member	Setting Range
Attack rate	attr.ar	0x00 - 0x7f
Decay rate	attr.dr	0x0 - 0xf
Sustain rate	attr.sr	0x00 - 0x7f
Release rate	attr.rr	0x00 - 0x1f
Sustain level	attr.sl	0x0 - 0xf

Rate curves may be set for Attack, Sustain, Release (see Table 8-43).

Because only exponential decrease may be used for Decay, that attribute cannot be set.

Table 15-42: ADSR Rate Modes

Attribute	Mode settable in attr.?._mode
Attack rate	SPU_VOICE_LINEARIncN (linear increase)
	SPU_VOICE_EXPIncN (exponential increase)
Decay rate	N/A
Sustain rate	SPU_VOICE_LINEARIncN (linear increase)
	SPU_VOICE_LINEARDecN (linear decrease)
	SPU_VOICE_EXPIncN (exponential increase)
	SPU_VOICE_EXPDec (exponential decrease)
Release rate	SPU_VOICE_LINEARDecN (linear decrease)
	SPU_VOICE_EXPDec (exponential decrease)

Also, data from structure VagAtr members adsr1 and adsr2 may be set directly in attr.adsr1 and attr.adsr2. In this case only SPU_VOICE_ADSR_ADSR1 and SPU_VOICE_ADSR_ADSR2 can be set for ADSR in attr.mask.

Return value

None

Remarks

See also: [SpuRSetVoiceAttr](#) (p. 15-78), [SpuGetVoiceAttr](#) (p. 15-42), [SpuSetKey](#) (p. 15-91), [SpuSetKeyOnWithAttr](#) (p. 15-92), [SpuVoiceAttr](#) (p. 15-10).

SpuSetVoiceDR

Sets ADSR decay rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceDR (
int voiceNum,
unsigned short DR
)
```

Arguments

voiceNum Voice number (0 - 23)
DR ADSR decay rate

Explanation

This function sets ADSR decay rate used in the voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_ADSR_DR

Refer to SpuSetVoiceAttr for values that can be specified in ADSR decay rate, DR.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr().

SpuSetVoiceLoopStartAddr

Sets loop start address of waveform data in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceLoopStartAddr (
int voiceNum,
unsigned long loopStartAddr
)
```

Arguments

voiceNum Voice number (0 - 23)
loopStartAddr Loop start address

Explanation

This function sets start address of waveform data in the sound buffer. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_LSAX

Refer to SpuSetVoiceAttr for values that can be specified in the loopStartAddr, loop start address.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetTransferStartAddr().

SpuSetVoiceNote

Sets interval (note specification).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceNote (
int voiceNum,
unsigned short note
)
```

Arguments

voiceNum Voice number (0 - 23)
note Interval (note specification)

Explanation

This function sets the voice interval by note. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_NOTE

Thus prior to calling SpuSetVoiceNote, SpuSetVoiceAttr

SPU_VOICE_SAMPLE_NOTE

or the waveform data sample note feature for voice must be set. Refer to SpuSetVoiceAttr for values that can be specified in the interval by note specification.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceSampleNote().

SpuSetVoicePitch

Sets interval (pitch specification).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoicePitch (
  int voiceNum,
  unsigned short pitch
)
```

Arguments

voiceNum Voice number (0 - 23)
pitch Interval (pitch specification)

Explanation

This function sets the voice interval by pitch. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_PITCH

Refer to SpuSetVoiceAttr for values that can be specified in the interval by pitch specification.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr().

SpuSetVoiceRR

Sets ADSR release rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceRR (
int voiceNum,
unsigned short RR
)
```

Arguments

voiceNum Voice number (0 - 23)
RR ADSR release rate

Explanation

This function sets ADSR release rate used in the voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_ADSR_RR

ADSR sustain rate mode becomes SPU_VOICE_LINEARDecN (Linear decrease mode). If you want to set ADSR release rate and ADSR release rate mode at the same time, use SpuSetVoiceRRAttr.

Refer to SpuSetVoiceAttr for values that can be specified in ADSR release rate, RR.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceRRAttr().

SpuSetVoiceRRAttr

Sets ADSR release rate / release rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceRRAttr (
int voiceNum,
unsigned short RR,
long RRmode
)
```

Arguments

voiceNum Voice number (0 - 23)
RR ADSR release rate
RRmode ADSR release rate mode

Explanation

This function sets ADSR release rate / ADSR release rate mode used in the voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specifications

```
SPU_VOICE_ADSR_RR
SPU_VOICE_ADSR_RRMODE
```

Refer to SpuSetVoiceAttr for values that can be specified in ADSR release rate, RR and ADSR release rate mode, RRmode.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceRR().

SpuSetVoiceSampleNote

Sets waveform data sample note.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceSampleNote (
int voiceNum,
unsigned short sampleNote
)
```

Arguments

voiceNum Voice number (0 - 23)
sampleNote Sets waveform data sample note

Explanation

This function sets the waveform data sample note for voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_SAMPLE_NOTE

Refer to SpuSetVoiceAttr for values that can be specified in sampleNote.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceNote().

SpuSetVoiceSL

Sets ADSR sustain level.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceSL (
int voiceNum,
unsigned short SL
)
```

Arguments

voiceNum Voice number (0 - 23)
SL ADSR sustain level

Explanation

This function sets ADSR release rate used in the voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_ADSR_SL

ADSR sustain level mode becomes SPU_VOICE_LINEARDecN (Linear decrease mode). If you want to set ADSR sustain level and ADSR sustain level mode at the same time, use SpuSetVoiceSLAttr.

Refer to SpuSetVoiceAttr for values that can be specified in ADSR sustain level, SL.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceRRAttr().

SpuSetVoiceSR

Sets ADSR sustain rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceSR (
int voiceNum,
unsigned short SR
)
```

Arguments

voiceNum Voice number (0 - 23)
SR ADSR sustain rate

Explanation

This function sets ADSR sustain rate used in the voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_ADSR_SR

ADSR sustain rate mode becomes SPU_VOICE_LINEARDecN (Linear decrease mode). If you want to set ADSR sustain rate and ADSR sustain rate mode at the same time, use SpuSetVoiceSRAttr.

Refer to SpuSetVoiceAttr for values that can be specified in ADSR sustain rate, SR.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceSRAttr().

SpuSetVoiceSRAttr

Sets ADSR sustain rate / sustain rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceSRAttr (
int voiceNum,
unsigned short SR,
long SRmode
)
```

Arguments

voiceNum Voice number (0 - 23)
SR ADSR sustain rate
SRmode ADSR sustain rate mode

Explanation

This function sets ADSR sustain rate / ADSR sustain rate mode used in the voice. It performs a process which corresponds to SpuSetVoiceAttr() mask specifications

```
SPU_VOICE_ADSR_SR
SPU_VOICE_ADSR_SRMODE
```

Refer to SpuSetVoiceAttr for values that can be specified in ADSR sustain rate, SR and ADSR sustain rate mode, SRmode.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceSR().

SpuSetVoiceStartAddr

Sets start address of waveform data in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceStartAddr (
int voiceNum,
unsigned long startAddr
)
```

Arguments

voiceNum Voice number (0 - 23)
startAddr Waveform data start address

Explanation

This function sets start address of waveform data in the sound buffer. It performs a process which corresponds to SpuSetVoiceAttr() mask specification

SPU_VOICE_WDSA

Refer to SpuSetTransferStartAddr for values that can be specified in the startAddr, waveform data start address.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetTransferStartAddr().

SpuSetVoiceVolume

Voice volume set.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceVolume (
int voiceNum,
short volumeL,
short volumeR
)
```

Arguments

voiceNum Voice Number (0 - 23)
volumeL Volume (Left)
volumeR Volume (Right)

Explanation

This function sets the voice volume. It performs a process which corresponds to SpuSetVoiceAttr() mask specifications

```
SPU_VOICE_VOLL
SPU_VOICE_VOLR
```

Thus the Volume Mode will become "Direct Mode" and the range of value that can be specified to *volumeL* and *volumeR* is equivalent to "Direct Mode" of SpuSetVoiceAttr. If you want to specify both volume and volume mode at the same time, use SpuSetVoiceVolumeAttr. Refer to SpuSetVoiceAttr for values that can be specified in *volumeL* and/or *volumeR*.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceVolumeAttr().

SpuSetVoiceVolumeAttr

Voice volume/volume mode set.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	5/22/97

Syntax

```
void SpuSetVoiceVolumeAttr (
int voiceNum,
short volumeL,
short volumeR,
short volModeL,
short volModeR
)
```

Arguments

<i>voiceNum</i>	Voice Number (0 - 23)
<i>volumeL</i>	Volume (Left)
<i>volumeR</i>	Volume (Right)
<i>volModeL</i>	Volume mode (Left)
<i>volModeR</i>	Volume mode (Right)

Explanation

This function sets voice volume and/or volume mode. It performs a process which corresponds to SpuSetVoiceAttr() mask specifications

```
SPU_VOICE_VOLL
SPU_VOICE_VOLR
SPU_VOICE_VOLMODEL
SPU_VOICE_VOLMODER
```

Refer to SpuSetVoiceAttr for values that can be specified in volModeL, volModeR, volumeL and/or volumeR.

Return value

None

Remarks

See also: SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuSetVoiceVolumeAttr()

SpuStart

Starts SPU processing.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	7/31/96

Syntax

void SpuStart(*void*)

Arguments

None

Explanation

SpuStart() starts SPU processing. This function is also called by Spulnit(), so it is not necessary to call it when initializing, but SpuStart() must be called after calling SpuQuit() if you use SpuQuit() to turn functionality off.

In the current specification, DMA transfer initialization setting is performed after SpuStart() is called.

Return value

None

Remarks

See also: SpuQuit (p. 15-29), Spulnit (p. 15-47).

SpuStGetStatus

Determines the SPU streaming state.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	7/31/96

Syntax

long SpuStGetStatus(*void*)

Arguments

None

Explanation

It determines the state of the SPU streaming.

Return value

Table 15-43

Attribute	Description
SPU_ST_NOT_AVAILABLE	SPU streaming is not available; SpuStInit() has not been called.
SPU_ST_IDLE	Data transfer to the sound buffer has not been performed yet or all streams have terminated already.
SPU_ST_PREPARE	Transferring the first 1 buffer.
SPU_ST_TRANSFER	Transferring the data to the sound buffer. If SpuStTransfer (SPU_ST_PREPARE,) is executed in this state, the status does not change to SPU_ST_PREPARE.
SPU_ST_FINAL	Waiting for the end of the playback after transferring the last 1 buffer. SpuStTransfer() is not accepted in this state.

Remarks

See also: SpuStInit (p. 15-135), SpuStTransfer (p. 15-140), SpuStGetVoiceStatus (p. 15-134).

SpuStGetVoiceStatus

Determines the voices used for SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	6/22/98

Syntax

unsigned long SpuStGetVoiceStatus(*void*)

Arguments

None

Explanation

It determines the voices used for the SPU streaming.

Return Value

The value of the voices represented by the bit sum of SPU_0CH to SPU_23CH.

Remarks

See also: SpuStTransfer (p. 15-140), SpuStGetStatus (p. 15-133).

SpuStInit

Initializes SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	7/31/96

Syntax

```
SpuStEnv *SpuStInit (  
long mode  
)
```

Arguments

mode Not used under the current specification. Set "0".

Explanation

Initializes the streaming. *mode* is called only once in the executed program. SPU streaming is available after initialization.

Return Value

Pointer to the SPU streaming environment structure SpuStEnv.

Remarks

See also: SpuStQuit (p. 15-136), SpuStEnv (p. 15-10).

SpuStQuit

Completes SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	7/31/96

Syntax

long SpuStQuit(*void*)

Arguments

None

Explanation

Completes the SPU streaming. Prior to calling this function, the termination processing must be completed for all the streams.

Return value

SPU_ST_ACCEPT	Normal end
SPU_ST_WRONG_STATUS	SpuStQuit is not accepted. The cause is the current status is not SPU_ST_IDLE.

Remarks

See also: SpuStInit (p. 15-135), SpuStGetStatus (p. 15-133).

SpuStSetPreparationFinishedCallback

Sets the callback function called at the completion of the data transfer in the preparation for the stream in the SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	7/31/96

Syntax

```
SpuStCallbackProc SpuStSetPreparationFinishedCallback (
SpuStCallbackProc callback_proc
)
```

```
SpuStCallbackProc callback_proc (
unsigned long voice_bit,
long status
)
```

Arguments

callback_proc Pointer towards the callback function called at the completion of the data transfer in the preparation for the stream

Explanation

Sets the callback function called at the completion of the data transfer in the preparation for the stream in the SPU streaming.

When *callback_proc* is called, the value of the voices assigned for the stream where the data transfer is completed in the preparation is set for the argument *voice_bit* by the bitOR of SPU_0CH to SPU_23CH. The following value is set for "*status*" depending on the state of the streaming library.

Table 15-44

State	Status
SPU_ST_PREPARE	SPU_ST_PREPARE
SPU_ST_PLAY	SPU_ST_PLAY

Return Value

The pointer towards the previously set callback function called at the completion of the data transfer in the stream preparation.

NULL is returned if no callback function has been previously set.

Remarks

See also: SpuStTransfer (p. 15-140), SpuStSetTransferFinishedCallback (p. 15-139), SpuStSetStreamFinishedCallback (p. 15-138).

SpuStSetStreamFinishedCallback

Sets the callback function called at the completion of each stream processing in the SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	7/31/96

Syntax

```
SpuStCallbackProc SpuStSetStreamFinishedCallback (  
SpuStCallbackProc callback_proc  
)
```

```
SpuStCallbackProc *callback_proc (  
unsigned long voice_bit,  
long status  
)
```

Arguments

callback_proc The pointer towards the callback function called at the completion of each stream

Explanation

Sets the callback function called at the completion of each stream in the SPU streaming.

When *callback_proc* is called, the value of the voices assigned for the stream of which processing is completed is set for the argument *voice_bit* by the bitOR of SPU_0CH to SPU_23CH. The following value is set for "*status*" depending on the state of the streaming library.

Table 15-45

State	Status
SPU_ST_PLAY	SPU_ST_PLAY
SPU_ST_FINAL	SPU_ST_FINAL

Return Value

The pointer towards the previously set callback function called at the completion of each stream.

NULL is returned if no callback function has been previously set.

Remarks

See also: SpuStTransfer (p. 15-140), SpuStSetPreparationFinishedCallback (p. 15-137), SpuStSetTransferFinishedCallback (p. 15-139).

SpuStSetTransferFinishedCallback

Sets the callback function called at the completion of one transfer to the stream buffer for all the streams in the SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	7/31/96

Syntax

```
SpuStCallbackProc SpuStSetTransferFinishedCallback (  
SpuStCallbackProc callback_proc  
)
```

```
SpuStCallbackProc *callback_proc (  
unsigned long voice_bit,  
long status  
)
```

Arguments

callback_proc Pointer towards the callback function called at the completion of one transfer to the stream buffer for all the streams

Explanation

Sets the callback function called at the completion of one transfer to the stream buffer for all the streams in the SPU streaming.

When *callback_proc* is called, the value of the voices assigned for the stream where one transfer to the stream buffer is completed is set for the argument *voice_bit* by the bitOR of SPU_0CH to SPU_23CH.

SPU_ST_PLAY is always set for "*status*".

Return Value

The pointer towards the previously set callback function called at the completion of one transfer to the stream buffer for all the streams in the SPU streaming.

NULL is returned if no callback function has been previously set.

Remarks

See also: SpuStTransfer (p. 15-140), SpuStSetPreparationFinishedCallback (p. 15-137), SpuStSetStreamFinishedCallback (p. 15-138).

SpuStTransfer

Prepares for the stream, and provides the instruction for starting the stream in SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	7/31/96

Syntax

```
long SpuStTransfer (
long flag,
unsigned long voice_bit
)
```

Arguments

flag Stream state flag
voice_bit Streaming voices

Explanation

It prepares for the stream in the SPU streaming, and provides the instruction for starting it.

The voices where the stream is executed are set explicitly for *voice_bit* by the bitOR of SPU_0CH to SPU_23CH.

- (1) STREAM STATE FLAG values: SPU_ST_PREPARE=Preparation

Carries out stream preparation according to the attributes of the SpuStEnv structure returned by SpuStInit().

The end of the preparation is determined by the callback function set by SpuStSetPreparationFinishedCallback.

- (2) SPU_ST_PLAY=Start

The stream is started according to the attributes of the SpuStEnv structure returned by SpuStInit.

If the status of streaming is SPU_ST_PREPARE, the voice is keyed on promptly following analysis of the adequacy of *voice_bit* when this function is called.

If the status of SPU streaming is SPU_ST_TRANSFER, the transfer waits until processing is transferred to the latter part of the stream buffer of the currently processed streams. Consequently, the transfer is carried out with other stream processing when the latter part of the stream buffer is processed.

When one transfer to the stream buffer for all streams is completed, the callback function set by SpuStSetTransferFinishedCallback is called, and the attributes for the next transfer for each stream are set.

When a stream is completed, the callback function set by SpuStSetStreamFinishedCallback is called. (Precisely before the next transfer if other streams are processed.)

Return value

SPU_ST_NOT_AVAILABLE SPU streaming is not available. SpuStInit() has not been called.
SPU_ST_INVALID_ARGUMENTS The value of the arguments is not in the specification.
SPU_ST_WRONG_STATUS SpuStTransfer is not accepted.

The causes are:

- The current status is SPU_ST_FINAL without regard to flag.
- Flag is SPU_ST_PREPARE, and the current status is SPU_ST_PREPARE.
- Flag is SPU_ST_PLAY, and the current status is SPU_ST_IDLE.

SPU_ST_ACCEPT Processing is accepted.

Remarks

See also: `SpuStInit` (p. 15-135), `SpuStSetPreparationFinishedCallback` (p. 15-137), `SpuStSetTransferFinishedCallback` (p. 15-139), `SpuStSetStreamFinishedCallback` (p. 15-138).

SpuWrite

Transfers data from main memory to the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	6/22/98

Syntax

```
unsigned long SpuWrite(*addr, size)
unsigned char *addr;
unsigned long size;
```

Arguments

addr Pointer to transfer data start address in main memory
size Transfer data size (in bytes)

Explanation

Transfers size bytes of data from main memory *addr* to the sound buffer

The main memory address *addr* storing the transfer data must fulfill the following conditions.

- It is an address of an allocated variable that is a global variable
- It is an address of an allocated variable that is in the heap and is allocated by a function such as malloc.
- It does not address a stack area (address to an auto variable) declared in a function.

SpuWrite() does not perform sound buffer memory management, so real waveform data cannot be used if the user does not transfer to addresses which avoid the following areas.

- SPU decoded data transfer area: 0x0000-0xffff
- System reserved area: 0x1000-0x100f
- Addresses after the reverb work area offset (start) address

After calling, either call SpulsTransferCompleted () to confirm transfer completion or set the DMA transfer completion Callback function in advance using SpuSetTransferCallback.

Due to the limitations of the DMA transfer hardware, transfers are always performed in 64 byte units. When specifying values which are not multiples of 64 as secondary arguments, since the portion of the value which is a multiple of 64 will be transferred, there is the possibility of damaging the data in the SPU memory.

Return value

Transferred data size.

If the specified data size is larger than 512 KB, the actual transferred size is returned.

Remarks

See also: SpuWrite0 (p. 15-143), SpuWritePartly (p. 15-144), SpuRead (p. 15-73), SpuSetTransferMode (), SpuGetTransferMode (), SpuSetTransferStartAddr, SpuGetTransferStartAddr (p. 15-41), SpulsTransferCompleted (), SpuSetTransferCallback ().

SpuWrite0

Clears sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	6/22/98

Syntax

```
unsigned long SpuWrite0(size)
unsigned long size;
```

Arguments

size Clear area size (in bytes)

Explanation

Writes 0 in the sound buffer area.

This writing is done by DMA transfer, but is started synchronously.

The starting address of the area written is specified by `SpuSetTransferStartAddr()`, and its size is *size*.

Due to the limitations of the DMA transfer hardware, transfers are always performed in 64 byte units. When specifying values which are not multiples of 64 as secondary arguments, since the portion of the value which is a multiple of 64 will be transferred, there is the possibility of damaging the data in the SPU memory.

Return value

Returns the size of the area written with 0s.

If the data size set is larger than 512 KB, the actual written size is returned.

Remarks

See also: `SpuWrite` (p. 15-142), `SpuWritePartly` (), `SpuRead` (), `SpuSetTransferMode` (), `SpuGetTransferMode` (), `SpuSetTransferStartAddr` (p. 15-109), `SpuGetTransferStartAddr` ().

SpuWritePartly

Transfers data from main memory to the sound buffer (assuming the transfer is divided into sections).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	6/22/98

Syntax

```
unsigned long SpuWritePartly(*addr, size)
unsigned char *addr;
unsigned long size;
```

Arguments

addr Pointer to transfer data start address in main memory
size Transfer data size (in bytes)

Explanation

Transfers data from main memory to the sound buffer.

The main memory address storing the transfer data must fulfill the following conditions.

- It is an address of an allocated variable that is a global variable
- It is an address of an allocated variable that is in the heap and is allocated by a function such as malloc.

That is,

- It does not address a stack area (address to an auto variable) declared in a function.

Data is transferred from the address specified in SpuSetTransferStartAddr(), and after completion of the transfer specified by size, the starting address is incremented by size, and stored internally.

Normally, in the case of continuous transfer, the size of each transfer must be a number divisible by 8. But when transferring the final block of a continuous transfer, the size need not be divisible by 8.

If SpuSetTransferStartAddr() is called during continuous transfer processing, correct continuous transfer is not guaranteed.

SpuWritePartly() does not perform sound buffer memory management, so real waveform data cannot be used if the user does not transfer to addresses which avoid the following areas.

- SPU decoded data transfer area: 0x0000-0xfff
- System reserved area: 0x1000-0x100f
- Addresses after the reverb work area offset (start) address

After calling, either call SpulsTransferCompleted () to confirm transfer completion or set the DMA transfer completion Callback function in advance using SpuSetTransferCallback.

Due to the limitations of the DMA transfer hardware, transfers are always performed in 64 byte units. When specifying values which are not multiples of 64 as secondary arguments, since the portion of the value which is a multiple of 64 will be transferred, there is the possibility of damaging the data in the SPU memory.

Return value

Returns the size of the area written with 0s.

If the data size set is larger than 512 KB, the actual transferred size is returned.

Remarks

See also: `SpuWrite` (p. 15-142), `SpuWrite0` (), `SpuRead` (p. 15-73), `SpuSetTransferMode` (), `SpuGetTransferMode` (), `SpuSetTransferStartAddr` (p. 15-109), `SpuGetTransferStartAddr` (p. 15-41), `SpuIsTransferCompleted` (), `SpuSetTransferCallback` ().

Chapter 16: Serial Input/Output Library

Table of Contents

Functions	
AddSIO	16-3
DelSIO	16-4
Sio1Callback	16-5
_sio_control	16-6

AddSIO

Initializes SIO driver.

Library	Header File	Introduced	Documentation Date
<i>libsio.lib</i>	<i>libsio.h</i>	3.6	6/22/98

Syntax

```
long AddSIO(
int baud
)
```

Arguments

baud Communication speed (bps)

Explanation

Initializes the SIO driver at the specified communication speed.

Return value

Always returns 1.

Remarks

See also:

DeISIO

Deletes the SIO driver from the kernel.

Library	Header File	Introduced	Documentation Date
<i>libsio.lib</i>	<i>libsio.h</i>	3.6	6/22/98

Syntax

long DeISIO(*void*)

Arguments

None

Explanation

Deletes the SIO driver from the kernel.

Return value

Always returns 1.

Remarks

See also:

Sio1Callback

Sets SIO interrupt callback function.

Library	Header File	Introduced	Documentation Date
<i>libsio.lib</i>	<i>libsio.h</i>	4.0	5/22/97

Syntax

```
int Sio1Callback (
void(*func)()
)
```

Arguments

func Callback function

Explanation

When an interrupt has been generated by the interrupt factors (CR_DSRIEN, CR_RXIEN, CR_TXIEN) set by `_sio_control (1,1, param)`, the `func` function is called. When `func` is set to 0, a callback is not generated.

Return value

None

Remarks

When an interrupt is generated, the interrupt flag must be cleared using `_sio_control (2,1,0)` or `_sio_control (1,1,CR_ERRRST)`. The next SIO interrupt will not be generated unless the interrupt flag is cleared.

See also:

_sio_control

SIOBIOS

Library	Header File	Introduced	Documentation Date
<i>libsio.lib</i>	<i>libsio.h</i>	3.6	5/22/97

Syntax

```
long _sio_control (
unsigned long cmd
unsigned long arg
unsigned long param
)
```

Arguments

cmd Command
arg Subcommand
param Argument

Explanation

SIO driver control and information acquisition.

Used in detailed communication with the PC and also when the user wishes to suppress debugging data based on the standard output from the library, etc.

Return values

See remarks

Remarks

None

See also:

Table 16-1: Command Summary

cmd	arg	Function
0	0	Returns driver status (*1)
	1	Returns control line status (*2)
	2	Returns communications mode (*3)
	3	Returns communications speed (bps units)
	4	Reads 1 byte
1	0	System reservation
	1	Sets param value as control line status (*2)
	2	Sets param value as communications mode (*3)
	3	Sets param value as communications speed (bps units)
	4	Writes 1 byte
2	0	Resets driver
	1	Clears driver status error-related bits

Table 16-2: Driver Status

bit		Contents
31-10		Undecided
9	SR_IRQ	1: interrupt on
8	SR_CTS	1: CTS is on
7	SR_DSR	1: DSR is on
6		Undecided
5	SR_FE	1: frame error occurs
4	SR_OE	1: overrun error occurs
3	SR_PERROR	1: parity error occurs
2	SR_TXU	1: no communications data
1	SR_RXRDY	1: able to read communications data
0	SR_TXRDY	1: able to write communications data

Table 16-3: Control Register

bit		Contents
31-13		Undecided
12	CR_DSRIEN	1: DSR Interrupt Permission
11	CR_RXIEN	1: Receive Interrupt Permission
10	CR_TXIEN	1: Transmission Interrupt Permission
9,8		0 Fixed
7		Undecided
6	CR_INTRST	1: SIO1 Reset
5	CR_RTS	1: RTS is on
4	CR_ERRRST	1: Interrupt and error flag clear
3		Undecided
2	CR_RXEN	1: Receive permission
1	CR_DTR	1: DTR is on
0	CR_TXEN	1: Transmission permission

Table 16-4: Mode Register

bit		Contents
31-8		Undecided
7,6		stop bit length
	MR_SB_01	01: 1
	MR_SB_10	10: 1.5
	MR_SB_11	11: 2
5	MR_P_EVEN	parity 2 1: odd 0: even
4	MR_PEN	parity 1 1: exists
3,2		character length
	MR_CHLEN_5	00: 5 bit
	MR_CHLEN_6	01: 6
	MR_CHLEN_7	10: 7
	MR_CHLEN_8	11: 8
1		Always 1
0		Always 0

Chapter 17: HMD Library

Table of Contents

Structures

GsARGUNIT	17-3
GsARGUNIT_ANIM	17-4
GsARGUNIT_GND...	17-6
GsARGUNIT_IMAGE	17-8
GsARGUNIT_JntMIMe	17-9
GsARGUNIT_NORMAL	17-10
GsARGUNIT_RstJntMIMe	17-11
GsARGUNIT_RstVNMIMe	17-12
GsARGUNIT_SHARED	17-13
GsARGUNIT_VNMIMe	17-14
GsCOORDUNIT	17-15
GsRVIEWUNIT	17-16
GsSEH	17-17
GsSEQ	17-18
GsTYPEUNIT	17-20
GsUNIT	17-22
GsVIEWUNIT	17-23
GsWORKUNIT	17-24

Functions

GsGetHeadpUnit	17-25
GsGetLsUnit	17-26
GsGetLwsUnit	17-27
GsGetLwUnit	17-28
GsInitRstNrmMIMe	17-29
GsInitRstVtxMIMe	17-30
GsLinkAnim	17-31
GsMapCoordUnit	17-32
GsMapUnit	17-33
GsScanAnim	17-34
GsScanUnit	17-35
GsSetRefViewLUnit	17-36
GsSetRefViewUnit	17-37
GsSetViewUnit	17-38
GsSortUnit	17-39
GsU_00000008...	17-40
GsU_03000000	17-42
GsU_03000001...	17-44
GsU_03010110...	17-46
GsU_04010010...	17-49

GsARGUNIT

Primitive driver argument area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
} GsARGUNIT;
```

Members

<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when assigning OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area

Explanation

This is the common argument area transferred to the primitive driver called from GsSortUnit(). The pointer for the primitive to which characteristic information has been added is transferred to the primitive driver. Due to usual high speed operation, use scratch pad.

Remarks

See also: GsSortUnit(), GsU_00000008...().

GsARGUNIT_ANIM

Animation driver argument area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
    long header_size;
    unsigned long *htop;
    unsigned long *ctop;
    unsigned long *ptop;
} GsARGUNIT_ANIM;
```

Members

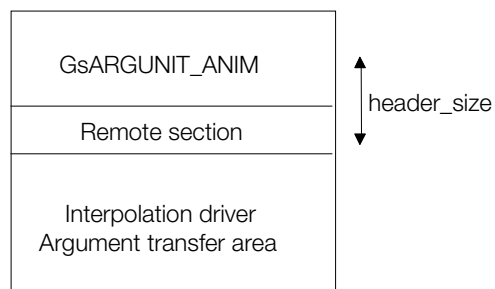
<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when assigning OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area
<i>header_size</i>	Size of primitive header used in animation
<i>htop</i>	Start address of interpolation function table section
<i>ctop</i>	Start address of sequence control section
<i>ptop</i>	Start address of parameter section

Explanation

This is the argument area transferred to the animation primitive driver called from GsSortUnit(). In addition to GsARGUNIT, the start address for each section needed for the primitive header size and animation is transferred. Due to the usual high speed operation in the argument area, use scratch pad.

The argument transfer area is larger than this member area. The start address of the section under GsARGUNIT_ANIM where rewriting is carried out is listed. This list number plus the combined numbers of htop, ctop and ptop are included in the header_size.

The transfer area for the interpolation function is taken following the start address of the rewriting section.



Although the number and meaning of the interpolation driver transfer area varies depending on the type of interpolation function, the following four parameters are contained in the linear interpolation:

1. Start address sequence pointer.
2. Interpolation source parameter address.

3. Interpolation destination parameter address.
4. Address which remembers parameters after interpolation.

Remarks

See also: GsSortUnit(), GsU_03000001...(), GsU_03000000().

GsARGUNIT_GND...

HMD ground primitive driver argument area.

Library	Header File	Introduced	Documentation Date
libgs.lib	libgs.h	4.1	10/01/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
    unsigned long *polytop;
    unsigned long *boxtop;
    unsigned long *pointtop;
    SVECTOR *nortop;
} GsARGUNIT_GND;
```

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
    unsigned long *polytop;
    unsigned long *boxtop;
    unsigned long *pointtop;
    SVECTOR *nortop;
    unsigned long *uvtop;
} GsARGUNIT_GNDT;
```

Members

<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when sorting OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area
<i>polytop</i>	Pointer to start of ground POLYGON section
<i>boxtop</i>	Pointer to start of ground box section
<i>pointtop</i>	Pointer to start of ground vertex section
<i>nortop</i>	Pointer to start of NORMAL section
<i>uvtop</i>	Pointer to start of UV section

Explanation

This is the argument area that is passed to the ground primitive driver which is called from GsSortUnit(). GsARGUNIT_GND does not use texture while GsARGUNIT_GNDT uses texture.

Pointers to the start of the ground POLYGON section, the box section, and the vertex section within HMD data are passed, in addition to the parameters which are passed in GsARGUNIT.

Use the scratch pad to improve performance.

Remarks

See also: GsARGUNIT, GsSortUnit(), GsU_00000008...().

GsARGUNIT_IMAGE

Image primitive driver argument area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
    unsigned long *imagetop;
    unsigned long *cluttop;
} GsARGUNIT_IMAGE;
```

Members

<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when assigning OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area
<i>imagetop</i>	Pointer to pixel data start
<i>cluttop</i>	Pointer to clut data start

Explanation

This is the argument area transferred to the image primitive drive called from GsSortUnit(). In addition to GsARGUNIT, the pointer to the start of the clut data and texture image pixel data is transferred. Due to usual high speed operation, use scratch pad.

Remarks

See also: GsARGUNIT, GsSortUnit(), GsU_00000008...().

GsARGUNIT_JntMIMe

Joint MIMe primitive driver argument area.

Library	Header File	Introduced	Documentation Date
libgs.lib	libgs.h	4.0	10/01/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
    u_long *coord_sect;
    long *mimepr;
    u_long mimenum;
    u_short mimeid, reserved;
    u_long *mime_diff_sect;
} GsARGUNIT_JntMIMe;
```

Members

<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when assigning OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area
<i>coord_sect</i>	Pointer to COORDINATE section top
<i>mimepr</i>	Pointer to mime coefficient area
<i>mimenum</i>	Number of mime keys
<i>mimeid</i>	MIMeID
<i>mime_diff_sect</i>	Pointer to MIMe DIFF section

Explanation

This is the argument area transferred to the joint mime primitive driver called from GsSortUnit(). In addition to GsARGUNIT, the pointer to the coordinate section start within the HMD data, the pointer to the mime coefficient area, the number of mime keys, the MIMeID, and the MIMe DIFF section pointer are all transferred. Due to usual high speed operation, use scratch pad.

Remarks

See also: GsARGUNIT, GsSortUnit(), GsU_.....(), GsU_04.....().

GsARGUNIT_NORMAL

Independent primitive driver argument area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
    unsigned long *primtop;
    SVECTOR *vertop;
    SVECTOR *nortop;
} GsARGUNIT_NORMAL;
```

Members

<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when assigning OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area
<i>primtop</i>	Pointer to POLYGON section top
<i>vertop</i>	Pointer to VERTEX section top
<i>nortop</i>	Pointer to NORMAL section top

Explanation

This is the argument area transferred to the independent primitive driver called from GsSortUnit(). In addition to GsARGUNIT, the POLYGON section and VERTEX sections within the HMD data, and the pointer to the NORMAL section start are transferred. Due to usual high speed operation, use scratch pad.

Remarks

See also: GsARGUNIT, GsSortUnit(), GsU_.....().

GsARGUNIT_RstJntMIMe

Joint mime reset primitive driver argument area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	10/01/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
    u_long *coord_sect;
    u_short mimeid, reserved;
    u_long *mime_diff_sect;
} GsARGUNIT_RstJntMIMe;
```

Members

<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when assigning OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area
<i>coord_sect</i>	Pointer to COORDINATE section top
<i>mimeid</i>	MIMeID
<i>mime_diff_sect</i>	Pointer to MIMe DIFF section

Explanation

This is the argument area transferred to the joint mime reset primitive driver called from GsSortUnit(). In addition to GsARGUNIT, the pointer to the start of the COORDINATE section within the HMD data, the MIMeID and the MIMe DIFF section pointer are transferred. Due to usual high speed operation, use scratch pad.

Remarks

See also: GsARGUNIT, GsSortUnit(), GsU_.....(), GsU_04.....().

GsARGUNIT_RstVNMIMe

Argument area of vertex normal mime reset primitive driver.

Library	Header File	Introduced	Documentation Date
libgs.lib	libgs.h	4.0	10/01/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
    u_short mimeid, reserved;
    u_long *mime_diff_sect;
    SVECTOR *orgs_vn_sect;
    SVECTOR *vert_sect;
    SVECTOR *norm_sect;
} GsARGUNIT_RstVNMIMe;
```

Members

<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when assigning OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area
<i>mimeID</i>	MIMeID
<i>mime_diff_sect</i>	MIMe DIFF section pointer
<i>orgs_vn_sect</i>	OrgsVN section pointer
<i>vert_sect</i>	Vertex section pointer
<i>norm_sect</i>	Normal section pointer

Explanation

This is the argument area transferred to the vertex normal primitive driver called from GsSortUnit. The configuration is the same as GsARGUNIT. In addition to GsARGUNIT, the MIMeID within the HMD data, the MIMe DIFF section pointer, the OrgsVN section pointer, the vertex section pointer, and the normal section pointer are transferred. Due to usual high speed operation, use scratch pad.

Remarks

See also: GsARGUNIT, GsSortUnit(), GsU_.....(), GsU_04.....().

GsARGUNIT_SHARED

Shared primitive driver argument area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    unsigned long *primp2;
    SVECTOR *vertp;
    GsWORKUNIT *vertp2;
    SVECTOR *nortop;
    SVECTOR *nortop2;
} GsARGUNIT_SHARED;
```

Members

<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when assigning OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area
<i>primp2</i>	Pointer to POLYGON section top
<i>vertp</i>	Pointer to shared VERTEX section top
<i>vertp2</i>	Pointer to area storing the calculation results of shared vertex section
<i>nortop</i>	Pointer to shared NORMAL section top
<i>nortop2</i>	Pointer to area storing the calculation results of the shared NORMAL section

Explanation

This is the argument area transferred to the shared primitive driver called from GsSortUnit. In addition to GsARGUNIT, the POLYGON section and VERTEX section within the HMD data, the pointer to the NORMAL section start, and the pointer to the area storing those calculation results are transferred. Due to usual high speed operation, use scratch pad.

Remarks

See also: GsARGUNIT, GsSortUnit(), GsU_.....().

GsARGUNIT_VNMIMe

Vertex normal mime primitive driver argument area.

Library	Header File	Introduced	Documentation Date
libgs.lib	libgs.h	4.0	10/01/97

Structure

```
typedef struct {
    unsigned long *primp;
    GsOT *tagp;
    int shift;
    int offset;
    PACKET *out_packetp;
    long *mimepr;
    u_long mimenum;
    u_short mimeid, reserved;
    u_long *mime_diff_sect;
    SVECTOR *orgs_vn_sect;
    SVECTOR *vert_sect;
    SVECTOR *norm_sect;
} GsARGUNIT_VNMIMe;
```

Members

<i>primp</i>	Primitive start address
<i>tagp</i>	Pointer to GsOT_TAG currently located at the start
<i>shift</i>	Number of bits to shift when assigning OT
<i>offset</i>	OT screen coordinate system Z-axis offset
<i>out_packetp</i>	Pointer to unused packet area
<i>mimepr</i>	Pointer to mime coefficient area
<i>mimenum</i>	Number of mime keys
<i>mimeID</i>	MIMeID
<i>mime_diff_sect</i>	MIMe DIFF section pointer
<i>orgs_vn_sect</i>	OrgsVN section pointer
<i>vert_sect</i>	Vertex section pointer
<i>norm_sect</i>	Normal section pointer

Explanation

This is the argument area transferred to the vertex normal mime primitive driver called from GsSortUnit(). In addition to GsARGUNIT, the MIMeID within the HMD data, the MIMe DIFF section pointer, the OrgsVN section pointer, the vertex section pointer, and the normal section pointer are transferred. Due to usual high speed operation, use scratch pad.

Remarks

See also: GsARGUNIT, GsSortUnit(), GsU_.....(), GsU_04.....().

GsCOORDUNIT

Matrix type coordinate system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	10/01/97

Structure

```

struct _GsCOORDUNIT {
    unsigned long flg;
    MATRIX matrix;
    MATRIX workm;
    SVECTOR rot;
    struct _GsCOORDUNIT *super;
} GsCOORDUNIT;

```

Members

flg Flag indicating whether matrix was rewritten
matrix Matrix
workm Result of multiplication from this coordinate system to the WORLD coordinate system
rot Rotation vector for creating matrix
super Pointer to superior coordinates

Explanation

GsCOORDUNIT has superior coordinates and is defined by the MATRIX type matrix. *workm* retains the result of multiplication of matrices performed by the GsGetLwUnit() and GsGetLsUnit() functions in each node of GsCOORDINATE2 using the WORLD coordinates. However, this result is not stored in the *workm* of the coordinate system directly connected to the WORLD coordinate system.

flg is referenced to omit calculations for a node for which calculations were already made, during GsGetLw() calculations. 1 means the flag is set; 0 clears the flag. When the programmer has changed the contents of *matrix*, he has the responsibility to clear this flag. If it is not cleared, the GsGetLwUnit() and GsGetLsUnit() functions will fail to execute normally.

Remarks

See also:

GsRVIEWUNIT

HMD viewpoint position (Reference type).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
struct GsRVIEWUNIT {
    long vpx, vpy, vpz;
    long vrz, vry, vrz;
    long rz;
    GsCOORDUNIT *super;
};
```

Members

<i>vpx, vpy, vpz</i>	Viewpoint coordinates
<i>vrz, vry, vrz</i>	Reference point coordinates
<i>rz</i>	Viewpoint twist
<i>super</i>	Pointer to coordinate system which sets viewpoint (GsCOORDUNIT type)

Explanation

GsRVIEWUNIT retains the viewpoint information and is set in libgs by the GsSetRefViewUnit() function. The viewpoint coordinates in the coordinate system displayed by super are set in vpx, vpy and vpz. The reference point coordinates in the coordinate system displayed by super are set in vrz, vry and vrz.

When the z axis a vector from the viewpoint to the reference point, rz specifies the screen inclination against the z axis in fixed decimal format, with 4096 set to one degree. Viewpoint and reference point coordinate systems are set in super. For an example of the use of this function, an airplane cockpit view can be realized simply by setting super to the airplane coordinate system.

Remarks

See also:

GsSEH

HMD animation sequence header.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    short idx;
    unsigned char sid;
    unsigned char pad;
} GsSEH;
```

Members

idx Indexes the sequence control descriptor of the sequence header and represents it in the index within the sequence control section

sid Sequence ID

pad System reservation

Explanation

GsSEH retains the sequence information and when multiple sequences are present, it then stores the sequence pointer as the GsSEH array.

Idx provides the index to the sequence control descriptor of the sequence playback start. This index is set to GsSEQ *tidx*. By setting GsSEH *SID* to GsSEQ *SID*, playback of the sequence can be started.

When multiple sequences are present, select one from the GsSEH array and update the value to GsSEQ.

Remarks

param may be freely used if TOD animation is not used.

See also: GsLinkAnim(), GsU_03000000(), GsU_0300....().

GsSEQ

HMD animation sequence pointer.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	10/01/97

Structure

```
typedef struct {
    unsigned long rewrite_idx;
    unsigned short size, num;
    unsigned short ii;
    unsigned short aframe;
    unsigned char sid;
    char speed;
    unsigned short srcii;
    short rframe;
    unsigned short tframe;
    unsigned short ci, ti;
    unsigned short start;
    unsigned char start_sid;
    unsigned char pad;
} GsSEQ;
```

Members

<i>rewrite_idx</i>	Stores the sections and offsets which are updated by the animation. The superior eight bits are the offset within the primitive header and specify the start address of the section to be updated. The subordinate 24 bits are the relative offset within that section. (word units).
<i>size</i>	Size up to area where next sequence pointer is stored
<i>num</i>	Number of sequences stored in this sequence pointer
<i>ii</i>	Index displaying area which remembers parameters after interpolation (Relative offset within parameter section)
<i>aframe</i>	This member is automatically decreased by the frame update pointer. The sequence will stop at 0. The sequence END attributes also become 0. The decrease can be prohibited by setting 0xffff. This is used for endless playback.
<i>sid</i>	Current status sequence ID of the currently playing sequencer. Matching can be achieved when performing sequence jump and END operations with the sequence ID value set from 0 to 127. The sequence flow can be controlled using SID. When performing a jump description with the sequence control descriptor, it defines the match SID and the SID of the location to be jumped to. By setting the match SID to 0, it is possible to match all SIDs.
<i>speed</i>	Controls sequence playback speed. The two's complement of the sign bit indicates the direction of playback. Standard playback speed is 0x10. 0x7f will increase speed eight times, and 0x01 will make speed 1/16. The sign bit is the bit used for reverse playback. The packaging is carried out by subtracting speed from rframe frame by frame to make a new RFRAME value.
<i>srcii</i>	Retains data which should be specified in ii. The area for parameter saving already included in the data is retained by the index.
<i>rframe</i>	In the interpolation coefficient 0 between the key frames SRC FRAME, rframe==tframe becomes DST FRAME. It is displayed as a 12 bit fixed point
<i>tframe</i>	Displays the distance between key frames. Displayed as a 12 bit fixed point. The tframe in the sequence control descriptor is 8 bits and shifts four bits to the left when updated to GsSEQ tframe. When tframe is 0, the unobtained frame update driver passes the next key frame in DST KEYFRAME (the interpolated target key frame).

<i>ci</i>	Index to the parameter which retains source key frames. (Word offset within parameter section)
<i>ti</i>	Parameter index which retains destination keyframes (Word offset within parameter section)
<i>start</i>	Retains the sequence start index. Copies start to til and inserts 0 in rframe in order to begin sequence. The work area indicated for Bspline and related interpolation is set in this member.
<i>start_sid</i>	Retains the sequence start time SID. Copies start_sid to sid when sequence begins.
<i>pad</i>	Clears to 0 when switching key frames. This member can be used by the programmer for various purposes.

Explanation

GsSEQ retains the frame update driver internal status. By rewriting this member during execution, animation can be dynamically controlled.

GsSEQ exists within HMD data and HMD data entities can be indicated using GsLinkAnim(). Members automatically updated by the frame update driver and those only referred to are differentiated. Members only referred to are: rewrite_idx, size, num, ii, speed, srcii, start and start_sid. All others are rewritten in the frame update driver.

Remarks

See also: GsU_03000000(), GsU_03000001...(), GsLinkAnim(), GsSEH().

GsTYPEUNIT

Type storage area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	3/25/98

Structure

```
typedef struct {
    unsigned long type;
    unsigned long *ptr;
} GsTYPEUNIT;
```

Members

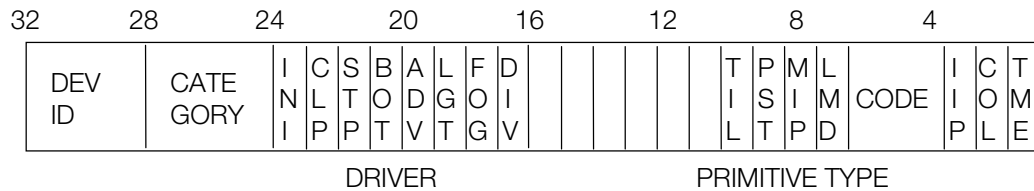
<i>type</i>	Primitive type
<i>ptr</i>	Address of area where type is stored

Explanation

In the `GsScanUnit()` function, `GsTYPEUNIT` stores the area which stores the type.

The user views that information and by assigning the subordinate functions corresponding to that type to ptr, the type will be overwritten on the subordinate functions pointer. GsSortUnit() calls that subordinate function.

The types are as follows:



DEV ID

DEV ID(Developer ID)
0000: SCE Reserved

CATEGORY

CATEGOR(Category)

0000:	Polygon driver
0001:	Shared Primitive data
0010:	Image data
0011:	Animation
0100:	MIMe
0101:	Surface

DRIVER

INI(init)	0: None 1: COORDINATE initialization is needed
CLP(clip)	0: No clipping 1: No clipping on left and top of screen
STP(Semi-trans)	0: Semi-transparent 1: Semi-transparent
BOT(both-side)	0: One-sided polygon 1: Two-sided polygon
ADV(active-div)	0: No automatic division 1: With automatic division
LGT(light)	0: With light source calculation 1: Without light source calculation

PRIMITIVE TYPE	FOG(fog)	0: FOG off 1: FOG on
	DIV(divide)	0: Without division 1: With division
	TILE	0: No information for continuous texture 1: With information for continuous texture
	PST(preset)	0: No presets 1: With presets
	MIP(mip-map)	0: No mip-map 1: With mip-map
	LMD(light-mode)	0: With normals 1: No normals
	CODE	000: Reserved 001: Triangle 010: Quadrangle 011: strip mesh 100-111: Reserved
	IIP	0: Flat 1: Gouraud
	COL(colored)	0: 1 quality color within identical polygon 1: Color for every vertex
	TME	0: Texture mapping OFF 1: Texture mapping ON

Remarks**See also:**

GsUNIT

Three-dimensional object handler for GsSortUnit.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    GsCOORDUNIT *coord;
    unsigned long *prmtop;
} GsUNIT;
```

Members

coord Pointer to local coordinate system
prmtop Pointer to primitive block header

Explanation

GsUNIT exists in every HMD data primitive block and allows movement of three dimensional models. GsSortUnit() is used to register GsUNIT to the ordering table. Coord is the pointer to the primitive block individual coordinate system. By setting the matrix in the coordinate system pointed to by coord, the object's location, inclination and size are reflected.

The primitive block start address is transferred in prmtop.

Remarks

See also: GsSortUnit().

GsVIEWUNIT

HMD viewpoint position (matrix type).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
struct GsVIEWUNIT {
    MATRIX view;
    GsCOORDUNIT *super;
};
```

Members

view Matrix used to change from superior coordinates to viewpoint coordinates
super Pointer to coordinate system which sets viewpoint

Explanation

This structure sets the viewpoint coordinates used by HMD. It directly specifies the matrix used to change from superior coordinates to viewpoint coordinates. The function used to set GsVIEWUNIT is GsSetViewUnit().

Remarks

See also: GsSetViewUnit().

GsWORKUNIT

Calculation result storage area for shared primitive.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Structure

```
typedef struct {
    DVECTOR vec;
    short otz;
    short p;
} GsWORKUNIT;
```

Members

vec Area which stores the three-dimensional coordinate values and the two-dimensional coordinate values after perspective conversion.

otz Area which stores the OTZ value obtained when *vec* is requested

p Area which stores the interpolation value obtained when *vec* is requested

Explanation

When using a shared primitive, first a perspective conversion of each three-dimensional coordinate value is carried out by its matrix to convert it into a two-dimensional coordinate. After perspective conversion is completed by the matrices, the shared primitive refers to the converted two-dimensional coordinate and creates a packet. This structure is the area which stores the results of that calculation.

Remarks

See also: GsARGUNIT_SHARED().

GsGetHeadpUnit

Obtains the current HMD header address.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
u_long *GsGetHeadpUnit (
void
)
```

Arguments

None

Explanation

This function returns the current type header address when using GsScanUnit() to scan HMD data.

Return value

None

Remarks

See also: GsScanUnit().

GsGetLsUnit

Calculates a local screen matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
void GsGetLsUnit (
GsCOORDUNIT *coord,
MATRIX *m
)
```

Arguments

coord Local coordinates
m Matrix

Explanation

This function calculates a local screen perspective transformation matrix from the GsCOORDUNIT structure pointed to by the *coord* argument and stores the result in the MATRIX structure pointed to by the *m* argument.

For high speed operation, the function retains the result of calculation at each node of the hierarchical coordinate system. When the next GsGetLsUnit() function is called, calculations up to the node to which no changes have been made are omitted. This is controlled by the GsCOORDUNIT flag. *libgs* replaces 1 in flags already calculated by GsCOORDUNIT. If the contents of a superior node are changed, the effect on a subordinate node is handled by *libgs*, so it is not necessary to clear the flags of all subordinate nodes of the changed superior node.

Return value

None

Remarks

See also:

GsGetLwsUnit

Calculates the local world and local screen matrices.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
void GsGetLwsUnit (
GsCOORDUNIT *coord,
MATRIX *lw,
MATRIX *ls
)
```

Arguments

coord Pointer to local coordinates
lw Matrix which stores local world coordinates as the result
ls Matrix which stores local screen coordinates as the result

Explanation

This function calculates both local world coordinates and local screen coordinates. It is faster than calling GsGetLwUnit() and then calling GsGetLsUnit(). Since LightMatrix must specify a local world matrix when light source calculations are being carried out, a local world matrix is necessary. In such cases it is faster to request one using GsGetLwsUnit().

Return value

None

Remarks

See also: GsGetLwUnit(), GsGetLsUnit().

GsGetLwUnit

Calculates local world matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
void GsGetLwUnit (
GsCOORDUNIT *coord,
MATRIX *m
)
```

Arguments

coord Local coordinates
m Matrix

Explanation

This function calculates a local screen perspective transformation matrix from the GsCOORDUNIT structure pointed to by the coord argument and stores the result in the MATRIX structure pointed to by the m argument.

For high speed operation, the function retains the result of calculation at each node of the hierarchical coordinate system. When the next GsGetLwUnit() function is called, calculations up to the node to which no changes have been made are omitted. This is controlled by the GsCOORDUNIT flag.libgs replaces 1 in flags already calculated by GsCOORDUNIT. If the contents of a superior node are changed, the effect on a subordinate node is handled by libgs, so it is not necessary to clear the flags of all subordinate nodes of the changed superior node.

Return value

None

Remarks

See also:

GsInitRstNrmMIMe

HMD MIME driver initialization function.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	10/01/97

Syntax

```
void GsInitRstNrmMIMe (
  unsigned long *prmtop,
  unsigned long *hp
)
```

Arguments

prmtop Primitive start address
hp Primitive header start address

Explanation

This is the HMD library normal MIME driver initialization function.

When using the normal MIME reset driver (GsU_04010029), it is necessary to initialize data using this function. When the address of the primitive driver is placed in the ptr of the GsTYPE structure which is returned by GsScanUnit(), GsInitRstNrmMIMe() is called.

Return value

none.

Remarks

Refer to:

psx\sample\graphics\hmd\mime\.

See also: GsInitRstVtxMIMe(), GsU_04010010....

GsInitRstVtxMIMe

HMD MIME driver initialization function.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	10/01/97

Syntax

```
void GsInitRstVtxMIMe (
  unsigned long *prmtop,
  unsigned long *hp
)
```

Arguments

prmtop Primitive start address
hp Primitive header start address

Explanation

This is the HMD library vertex MIME driver initialization function.

When using the vertex MIME reset driver (GsU_04010028), it is necessary to initialize data using this function. When the address of the primitive driver is placed in the ptr of the GsTYPE structure which is returned by GsScanUnit(), GsInitRstNrmMIMe() is called.

Return value

none.

Remarks

Refer to:

psx\sample\graphics\hmd\mime\.

See also: GsInitRstNrmMIMe(), GsU_04010010....

GsLinkAnim

Initializes a libgs ordering table structure.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
int GsLinkAnim (
GsSEQ *seq,
unsigned long *p
)
```

Arguments

seq GsSEQ structure array header address
p Address where the first type animation section exists

Explanation

This function retains sufficient information to control one sequence. It already exists within the HMD data and contains a value which allows the GsSEQ structure array to access the sequence pointer in HMD by means of GsLinkAnim(). As with GsScanAnim(0, GsLinkAnim()) must be activated during global SCAN.

By using *seq*, the programmer can control animation playback in real time. With GsLinkAnim(), specific members of specific sequences such as *seq[3]->aframe = 100* can be operated after switching to GsLinkAnim().

Information on each individual sequence can be accessed in *seh*. For example, assuming that ten sequences are defined in the fifth sequence, in order to play back the fourth, the fourth start index can be transmitted in the sequence pointer as follows:

```
seq[5]->ti = ((GsSEH *)&seq[5].start)+3->idx;
```

Return value

Returns the linked number.

Remarks

See also: GsScanAnim().

GsMapCoordUnit

Maps COORDINATE within the HMD data to actual address.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
GsCOORDUNIT *GsMapCoordUnit (
  unsigned long *base,
  unsigned long *p
)
```

Arguments

base HMD data start address
p Type of address where INI (init) bits are set up

Explanation

In cases where COORDINATE exists within the HMD data, one type in which INI bits are set up exists in GsTYPEUNIT type when GsScanUnit() is carried out. In such cases, by transferring that type address to GsMapCoordUnit(), the COORDINATE TOP within the data and super within COORDINATE are converted to the actual address.

Return value

COORDINATE section start address.

Remarks

See also: GsMapUnit(), GsScanUnit().

GsMapUnit

Maps HMD data to actual address.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
void GsMapUnit (
  unsigned long *p
)
```

Arguments

p HMD data start address

Explanation

A section referred to by a pointer exists in the HMD data. When creating HMD data, since the location in the memory where it will be loaded is not confirmed, the offset address from the HMD data start is stored by the number of words. The GsMapUnit() function converts the offset address into the actual address and in order to use the HMD data, this actual address conversion must take place first.

Return value

None

Remarks

Since a flag is attached to HMD data which had been converted to an actual address, there will be no adverse effects even if GsMapUnit() is called again.

See also:

GsScanAnim

Dedicated SCAN function for HMD animation.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
unsigned long *GsScanAnim (
  unsigned long *p,
  GsTYPEUNIT *ut
)
```

Arguments

point Address where animation section first type exists
 When p is 0, the type following that previously called by GsScanAnim() is examined
ut Pointer to area in which the type and its address are stored

Explanation

HMD data scan is divided into two: GsScanUnit() which carries out a common scan of all sections (known as a global scan), and a dedicated scan function which carries out a local scan of each section (known as a local SCAN). GsScanAnim() is of the latter type and is a dedicated scan function for HMD animation.

Scanning should be carried out when the animation type has first been global scanned. Since a bit is attached to the INI field of the animation type which was first globally scanned at the authoring level, timing for performing a local scan can be gauged by closely watching that bit.

GsScanAnim() is called to scan the types of all animation sections using the following procedure:

If the category of the type selected in the global SCAN is 3 (animation category) and INI is 1, a local SCAN should be performed.

type=0x03800000.

GsScanAnim() issues the above-mentioned type address as p.

If the return value check is 0, data is abnormal.

If it is 1, issue GsScanAnim() once more.

An interpolation primitive driver function pointer corresponding to the selected ut.type is assigned as ut.ptr.

Return to #4 and repeat until 0 is returned.

Return value

0 type to be scanned does not exist
 1 Scan successful, type exists

Remarks

With the primitive driver pointer which was written over by the HMD data, there will be no adverse effects even if GsScanAnim() is called again.

See also: GsScanUnit(), GsLinkAnim().

GsScanUnit

Examines types within HMD data.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
int GsScanUnit (
  unsigned long *p,
  GsTYPEUNIT *ut,
  GsOT *ot,
  u_long *scratch
)
```

Arguments

<i>p</i>	Block start address When <i>p</i> is 0, the type following that previously called by GsScanUnit() is examined
<i>ut</i>	Pointer to area where the type and its address are stored
<i>otp</i>	Pointer to the OT
<i>scratch</i>	Specifies scratch pad address

Explanation

HMD data is divided by type and calls the primitive driver according to that type. By overwriting the pointer to the primitive driver in the area where the type is located, the primitive driver is called when GsSortUnit() is called. On examination of the contents of GsTYPEUNIT returned by GsScanUnit() the pointer to the primitive driver is replaced by one which can be freely used by the user.

GsScanUnit() internally copies the type corresponding to the header to scratch. By transferring this scratch as a primitive driver argument, the primitive driver can be activated. Generally, image primitives which are only activated once (such as GsUIMGO and GsUIMG1) are activated at this initialization.

Initially, *p* is inserted as the block start address and called, then after that 0 is inserted as *p* and called, but it will automatically scan to the end of the block. This operation is carried out for every block.

Return value

0 Reached the end of the block, type does not exist
1 Type exists

Remarks

Since a flag is attached to the primitive driver pointer which was written over by the HMD data, there will be no adverse effects even if GsScanUnit() is called again.

See also:

GsSetRefViewLUnit

Sets HMD viewpoint position (High precision).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
int GsSetRefViewLUnit (
GsRVIEWUNIT *pv
)
```

Arguments

pv Viewpoint position information (viewpoint observation point type)

Explanation

This function calculates the WSMATRIX from viewpoint information. Its parameter is the GsRVIEWUNIT structure. If the viewpoint is not moved, the WSMATRIX will not change, so in such cases there is no need to call each frame.

However, if the viewpoint is moved, the changes will not be reflected unless each frame is called. Although the number of calculation mistakes using this function is smaller than with the GsSetRefViewUnit() function, the execution time is double. When the GsRVIEWUNIT "super" member is set to anything besides WORLD, even if the other parameters remain unchanged, the viewpoint will move if the parent coordinate system parameters are changed. In such cases, GsSetRefViewLUnit() must be called for each frame.

Return value

If viewpoint setting is successful:0; if unsuccessful:1.

Remarks

See also:

GsSetRefViewUnit

Sets HMD viewpoint position.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
int GsSetRefViewUnit (
GsRVIEWUNIT *pv
)
```

Arguments

pv Viewpoint position information (viewpoint observation point type)

Explanation

This function calculates the WSMATRIX from viewpoint information. Its parameter is the GsRVIEWUNIT structure. If the viewpoint is not moved, the WSMATRIX will not change, so in such cases there is no need to call each frame.

However, if the viewpoint is moved, the changes will not be reflected unless each frame is called. When the GsRVIEWUNIT "super" member is set to anything besides WORLD, even if the other parameters remain unchanged, the viewpoint will move if the parent coordinate system parameters are changed. In such cases, GsSetRefViewUnit() must be called for each frame..

Return value

If viewpoint setting is successful:0; if unsuccessful:1.

Remarks

See also:

GsSetViewUnit

Sets HMD viewpoint.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
int GsSetViewUnit (
GsVIEWUNIT *pv
)
```

Arguments

pv Viewpoint position information (matrix type)

Explanation

This function directly sets the WS matrix. If you use GsSetRefViewUnit() to determine the WS matrix from the viewpoint and the focal point, insufficient precision may cause errors when you move the viewpoint; it is more effective to use GsSetViewUnit(). When the GsVIEWUNIT "super" member is set to anything besides WORLD, even if the other parameters remain unchanged, the viewpoint will move if the parent coordinate system parameters are changed. In such cases, you must call GsSetViewUnit() for each frame. If GsIDMATRIX2 is used as the base matrix, then the aspect ratio of the screen will be adjusted automatically.

Return value

If setting is successful: 0; if unsuccessful: 1.

Remarks

See also:

GsSortUnit

Allocates an object to the ordering table.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
void GsSortUnit (
GsUNIT *objp,
GsOT *otp,
unsigned short *scratch
)
```

Arguments

objp Pointer to an object
otp Pointer to the OT
scratch Specifies scratch pad address

Explanation

This function performs perspective transformation and light source calculation on a three dimensional object handled by GsUNIT. The rendering command is generated in the packet area specified by GsSetWorkBase(). The rendering command generated is then Z-sorted and allocated to the OT displayed by otp. When scratch calls the subordinate functions, it is used in the transfer of variables. Usage volume is as follows:

Type	Scratch area usage (Unit: Byte)
Independent polygons	32
Shared polygons	40
Fixed division independent triangles	568
Fixed division independent quadrangles	852

Return value

None

Remarks

See also:

GsU_00000008...

GsSortUnit()primitive driver group (Polygon, Shared Polygon, Image).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	3/25/98

Syntax

```
unsigned long *GsU_00..... (
GsARGUNIT *ap
)
```

```
unsigned long *GsU_01..... (
GsARGUNIT *ap
)
```

```
unsigned long *GsU_02..... (
GsARGUNIT *ap
)
```

```
unsigned long *GsU_05..... (
GsARGUNIT *ap
)
```

Arguments

ap Start address for arguments transfer area

Explanation

This is the GsSortUnit()primitive driver group. The second letter of the header name is the category. The usual polygon primitive driver is GsU_00.....(). The image primitive driver is GsU_02.....(). The earh primitive driver is GsU_05.....().

When initializing HMD data for use, the GsTYPE structure type returned by GsScanUnit() must be checked and the primitive driver address must be placed in the pointer. The primitive drivers currently supported by libgs are listed in Appendix A of the HMD section in the File Formats manual.

Return value

Start address of the next primitive driver.

Refer to the sample program: (psx\sample\graphics\hmd\pdriver\00000008.c).

Remarks

When using a primitive driver which performs division, the number of divisions must be set to the significant 8bits of the area in which the primitive index is stored as follows:

Type	
# of polygons	Size
# of divisions	Primitive index

In order to specify the number of divisions, a macro such as the one following is provided in libgs.h.

Macro	#of divisions
GsUNIT_DIV1	2x2 divisions
GsUNIT_DIV2	4x4 divisions
GsUNIT_DIV3	8x8 divisions
GsUNIT_DIV4	16x16 divisions
GsUNIT_DIV5	32x32 divisions

Furthermore, when using automatic division, it is necessary to use `GsSetAzwh()` and to set the division condition for the z value, polygon size, etc.

See also: `GsSortUnit()`, `GsScanUnit()`.

GsU_03000000

HMD animation frame update driver.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	5/22/97

Syntax

```
unsigned long *GsU_03000000 (
GsARGUNIT_ANIM *sp
)
```

Arguments

sp Argument transfer area
 Due to normal high speed operation, specify scratch pad

The argument information built on *sp* is as follows:

primtop
tag(OT)
shift(OT)
OUTP(packet area)
Animation header size
Interpolation function header
CONTROL TOP pointer
PARAMETER TOP pointer
COORDINATE TOP pointer
VERTEX TOP pointer

Sections containing color are always set. Since the primitive headers of sections containing no color are copied as is, if the header shape changes, it will be altered. The animation header size retains the amount of animation information which is copied to the arguments area. In the case of the above example, 6.

Explanation

The frame update driver function performs the activities of interpreting the sequence descriptor and calling the appropriate interpolation function. Sequence jumps, reverses, etc. are carried out by means of the frame update driver. Since the internal status of the frame update driver is stored in each sequence pointer area, real time control of the sequence is possible if the programmer rewrites this area when executing the program.

Refer to the explanation in GsSEQ for details on the sequence pointer.

The frame update driver is linked to the HMD primitive block PRE-PROCESS.

GsSortUnit() is called when processing the first primitive block (PRE-PROCESS). The frame update driver specifies the sequence descriptor which should be referred to by the sequence pointer. By interpreting that sequence pointer the sequence progression is controlled. Finally, the sequence descriptor which refers to the key frame performs interpretation, a suitable interpolation driver (SRC FRAME or DST FRAME) is attached and called.

Return value

The area in which the start address of the next primitive driver is stored.

Refer to the sample program, (PS-X/sample/grapihcs/hmd/pdriver/GsU_00000008.c).

Remarks

See also: GsSEQ, GsSortUnit(), GsScanUnit().

GsU_03000001...

HMD animation interpolation function (alignment, COORDINATE).

Library	Header File	Introduced	Documentation Date
libgs.lib	libgs.h	4.0	3/25/98

Syntax

```
int GsU_03000001... (  
GsARGUNIT_ANIM *sp  
)
```

Arguments

sp Argument transfer area
 Due to normal high speed operation, specify scratch pad

Information on arguments contained in sp is as follows:

primtop
tag(OT)
shift(OT)
offset(OT)
OUTP(packet area)
Animation header size
Interpolation function table pointer
CONTROL TOP pointer
PARAMETER TOP pointer
COORDINATE TOP pointer
VERTEX TOP pointer
Sequence pointer address
Pointer to src parameters
Pointer to dst parameters
Write pointer after interpolation

The parameters in the interpolation function which are always fixed are the last four.

- Sequence pointer address:
Designates the section which rewrites data from the rewrite index after interpolation, and the rewrite address.
- Pointer to src parameter:
Pointer which indicates the src keyframe within the parameter section.
- Pointer to dst parameter:
Pointer which indicates the dst keyframe within the parameter section.

Pointer which writes after interpolation:

There are cases when you wish to retain the parameter value after interpolation for the Realtime Motion Switch. The area retained for that purpose is transferred as a pointer which indicates the keyframe within the parameter section. If 0, writing is not performed.

Explanation

This function is the interpolation driver for parameter interpolation. The argument area pointer is transferred as the argument. The interpolation coefficient is calculated from the TFRAME and RFRAME stored in the sequence pointer and interpolation is performed.

If src is 0 and dst is 1, 1-RFRAME/TFRAME becomes the interpolation coefficient.

Since the name of the function in the interpolation type indicates the HMD type in interpolation types, refer to the HMD format

The interpolation algorithm can be either LINEAR, BEZIER, or BSPRINE.

LINEAR

Interpolate the interval between SRC KEYFRAME and DST KEYFRAME as a straight line.

BEZIER

The KEY FRAME has 3 control points. One control point in the DST KEYFRAME and 3 control points in the SRC KEYFRAME comprise the 4 control points used for BEZIER interpolation.

BSPRINE

The KEY FRAME has one control point. SRC-2, SRC-1, SRC, and DST together comprise 4 control points that are used for BSPRINE interpolation.

A WORK area is maintained for the sequence history of SRC-2 and SRC-1. The WORK area is set from the START IDX member of the sequence pointer. 4 x 32 bits are used in the WORK area. At the start of the sequence, there is no history of SRC-2 and SRC-2 so it is necessary to arrange the first 3 key frames so that TFRAME = 0.

The primitive drivers currently supported by libgs are listed in Appendix A of the HMD section in the File Formats manual.

Return value

Returns 0 after normal interpolation.

If the next key frame is read unconditionally, 1 is returned and in the same frame, the destination key frame is once again called back as an argument. This case allows the sequence to instantly advance with TFRAME=0.

Remarks

Refer to:

```
psx\sample\graphics\hmd\pdriver\03000001.c
psx\sample\graphics\hmd\pdriver\03000002.c
psx\sample\graphics\hmd\pdriver\03000003.c
psx\sample\graphics\hmd\pdriver\03000010.c
psx\sample\graphics\hmd\pdriver\03000020.c
psx\sample\graphics\hmd\pdriver\03000030.c
```

See also: GsU_03000000().

GsU_03010110...

HMD animation interpolation function (general).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	3/25/98

Syntax

```
int GsU_03010110... (
GsARGUNIT_ANIM *sp
)
```

Arguments

sp Argument transfer area
 Due to normal high speed operation, specify scratch pad

Information on arguments contained in *sp* is as follows:

prmtop
tag(OT)
shift(OT)
offset(OT)
OUTP(packet area)
Animation header size
Interpolation function table pointer
CONTROL TOP pointer
PARAMETER TOP pointer
COORDINATE TOP pointer
VERTEX TOP pointer
Sequence pointer address
Pointer to src parameters
Pointer to dst parameters
Write pointer after interpolation

The parameters in the interpolation function which are always fixed are the last four.

Sequence pointer address:

Designates the section which rewrites data from the rewrite index after interpolation, and the rewrite address.

Pointer to src parameter:

Pointer which indicates the src keyframe within the parameter section.

Pointer to dst parameter:

Pointer which indicates the dst keyframe within the parameter section.

Pointer which writes after interpolation:

There are cases when you wish to retain the parameter value after interpolation for the Realtime Motion Switch. The area retained for that purpose is transferred as a pointer which indicates the keyframe within the parameter section. If 0, writing is not performed.

Explanation

This function is the generic interpolation driver for parameter interpolation. The numerical values that are interpolated can be packaged as 3-element vectors of 8-bits, 16-bits, or 32-bits, or as a scalar. The argument area pointer is transferred as the argument. The interpolation coefficient is calculated from the TFRAME and RFRAME stored in the sequence pointer and interpolation is performed.

If src is 0 and dst is 1, 1-RFRAME/TFRAME becomes the interpolation coefficient.

Since the name of the function in the interpolation type indicates the HMD type in interpolation types, refer to the HMD format

The interpolation algorithm can be either LINEAR, BEZIER, or BSPRINE.

LINEAR

Interpolate the interval between SRC KEYFRAME and DST KEYFRAME as a straight line.

BEZIER

The KEY FRAME has 3 control points. One control point in the DST KEYFRAME and 3 control points in the SRC KEYFRAME comprise the 4 control points used for BEZIER interpolation.

BSPRINE

The KEY FRAME has one control point. SRC-2, SRC-1, SRC, and DST together comprise 4 control points that are used for BSPRINE interpolation.

A WORK area is maintained for the sequence history of SRC-2 and SRC-1. The WORK area is set from the START IDX member of the sequence pointer. 4 x 32 bits are used in the WORK area. At the start of the sequence, there is no history of SRC-2 and SRC-2 so it is necessary to arrange the first 3 key frames so that TFRAME = 0.

The primitive drivers currently supported by libgs are listed in Appendix A of the HMD section in the File Formats manual.

Return value

Returns 0 after normal interpolation.

If the next key frame is read unconditionally, 1 is returned and in the same frame, the destination key frame is once again called back as an argument. This case allows the sequence to instantly advance with TFRAME=0.

Remarks

Refer to:

HMD Format Specification

HMD Animation Section

psx\sample\graphics\hmd\pdriver\03000001.c

psx\sample\graphics\hmd\pdriver\03000002.c

psx\sample\graphics\hmd\pdriver\03000003.c

psx\sample\graphics\hmd\pdriver\03000010.c

psx\sample\graphics\hmd\pdriver\03000020.c

psx\sample\graphics\hmd\pdriver\03000030.c

See also: GsU_030000000().

GsU_04010010...

HMD mime driver.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.0	10/01/97

Syntax

```
unsigned long *GsU_04010010... (
GsARGUNIT *ap
)
```

Arguments

ap Start address of argument transfer area
Due to usual high speed processing, use scratch pad

Explanation

This function is the driver group for mime animation from the HMD library. Although the *ap* argument is GsARGUNIT type, it is handled internally as a different type. Mime category drivers currently supported by libgs are as follows:

Driver name	Type name macro	ap processing type	Explanation
GsU_04010010	GsJntAxesMIMe	GsARGUNIT_JntMIMe	Axis interpolation joint mime
GsU_04010018	GsRstJntAxesMIMe	GsARGUNIT_RstJntMIMe	Axis interpolation joint mime reset
GsU_04010011	GsJntRPYMIMe	GsARGUNIT_JntMIMe	RPY value interpolation joint mime
GsU_04010019	GsRstJntRPYMIMe	GsARGUNIT_RstJntMIMe	RPY value interpolation joint mime reset
GsU_04010020	GsVtxMIMe	GsARGUNIT_VNMIMe	Vertex mime
GsU_04010021	GsNrmMIMe	GsARGUNIT_VNMIMe	Normal mime
GsU_04010028	GsRstVtxMIMe (*)	GsARGUNIT_RstVNMIMe	Vertex mime reset
GsU_04010029	GsRstNrmMIMe (*)	GsARGUNIT_RstVNMIMe	Normal mime reset

(*) Initialization is needed for GsInitRstVtxMIMe, GsInitRstNrmMIMe

Return value

Start address of next primitive driver:

Refer to sample program (psx\sample\grapihcs\hmd\pdriver\00000008.c).

Remarks

See also: GsARGUNIT_JntMIMe(), GsARGUNIT_RstJntMIMe(), GsARGUNIT_VNMIMe(), GsARGUNIT_RstVNMIMe(), GsInitRstNrmMIMe(), GsInitRstVtxMIMe().

