

PLAYSTATION TECHNICAL NOTE

=====

Date:
Ref:
Author: Yuji Takahashi, Executive Vice President of Business
Affairs
Subject: Important notice on software development due to some
hardware parts changes

ABSTRACT

Thanks to all of you, we achieved the shipment of 1 million consoles at the end of May. The sales condition in Japan is very positive. The reason for the success is that various attractive software titles from you have been released continuously and that the PlayStation console is marketed at an attractive price. We highly appreciate your efforts and would like to ask for your further support to PlayStation.

Now, the sales in the US will be started on September 9th. European business will also be started in September. In each area, the number of licensees are already more than 100. The expectation from each market is also very good.

So, a very big demand for PlayStation is expected all over the world. SCE is aware that it is very important to keep supplying a very large number of PlayStation consoles to each market. As you know, the semiconductor industry is suffering from shortage of memory chips because of a worldwide increase in PC demand, especially in the US. SCE has been suffering and is going to continue to suffer from the shortage of high-performance chips for the PlayStation console. In order to solve this problem, we have reviewed our current parts design for PlayStation and have re-designed some of the LSIs.

We have now completed the development of the new LSIs and the production of PlayStation with the new parts will be started at the end of this year. This will be a running change.

We named the hardware with new parts the **"Revision-C system"**.

We are aiming at full compatibility within the **"Revision-C system"**. However, it is reported that very small/subtle differences in the parts specifications might cause some very subtle difference in the software operation/gameplay, even though it seems this is a very rare case.

When the software programs contain the following conditions, it is reported that the possibility of having slightly different gameplay in Revision-C systems from gameplay in current systems will be significant.

(1) The VSync() function is not properly called.

(2) MoveImage() for rectangular regions less than 32 dots wide is often used.

We'd like to ask for your cooperation to check all of your already-released titles and already-submitted master disks. If you find any suspicious title, please contact our Account executive immediately. As soon as we get your reports, we will also check the mentioned titles in detail.

As for titles under development, please refer to the attached technical advise to maintain full compatibility. If you think it's difficult for you to check your title by yourself because of tight schedules, or if you have any difficulty in finding any suspicious phenomena, please do not hesitate to contact our account executives. This information is also available in BBS.

As for yet-to-be developed titles, we'd like to do our best to provide the circumstances in which developers can start development without having to be aware of any hardware parts differences. For this purpose, we'd like to release a new library to suit the current situation.

Your kind cooperation and understanding to this matter will be highly appreciated.

TECHNICAL DETAILS

Let us inform you of some technical advise on further software development under the circumstance of the release of the Revision-C system. The Revision-C system is hardware where the graphics chips have changed. The main purpose of releasing the Revision-C system is to maintain stable memory supplies. Basically, it was designed under the concept with keeping full compatibility with current machine. However, there is a small difference between the Revision-C system and current models, Revisions A and B, as follows:

(Revision A is a hardware for the Japanese market, and Revision B is a hardware for the US/European market. There is no problem of compatibility between Revisions A and B.)

1) Semi-transparent drawing is faster. An application which uses many transparent drawings runs faster.

2) It runs slower if vertically thin drawings or MoveImage() are frequently used. This will be obviously observed with thin rectangular regions that have a width of less than 16 dots.

Fundamentally, the two points above are the differences between current system and the Revision-C system. The reasons are due to the differences of the memory systems which are used as frame buffers.

Following are the details of each point.

(1) The possibility that the problem occurs increases when the existence of the GPU-bottleneck (*1), the use of semi-transparency, and no proper control of frame rate simultaneously occur.

```
(Example) while (1) {  
            DrawSync(0);  
            VSync(0);  
            DrawOTag(ot);  
        }
```

In the example above, the following problem will happen in the case of a GPU-bottleneck.

- Semi-transparent drawing is faster
- > DrawSync(0) ends earlier.
- > VSync(0) is reached earlier.
- > If the drawing ends near a 1/30 (or 1/15) second mark, VSync(0) returns one frame earlier.
- > A frame rate increase (the opposite of a frame rate decrease) occurs
- > Movement on the screen seems to be quicker occasionally.

The rate of speeding up of the drawing will depend on how many semi-transparencies are used.

(*1) GPU-bottleneck is where the drawing by the GPU is the slowest when one frame is generated and displayed.

[Countermeasure]

Control the frame rate precisely by using VSync(n) function.

For example, in a scene in which more than 80% are running with 30 frames, please set the frame rate 1/30 sec fixed. Use VSync(2) instead of VSync(0) to make sure.

Related to this, the following points are about frame rate synchronization.

[About Frame Rate Synchronization]

In PlayStation, the display region on the frame buffers can be switched asynchronous to the video frame rate (1/60 sec). However, if the display region is switched at an irregular rate which is not a multiple of 1/60 sec, the switching is not done in the vertical retrace period and a phenomenon in which flicker can be seen will occur. This may misguide users to think that that an application has some inferiority.

Therefore, in the normal process, switching of the buffers should be synchronized with vertical synchronization (V-BLANK). In (A), the switching of buffers depends on the slower one, either display or drawing, and the switching becomes

asynchronous with V-BLNK. Therefore, if VSync(0) is not intentionally targeting special effects, please execute it to synchronize with the switching of buffers.

<pre>(A) while(1){ ... DrawSync(0); swap_buffer(); DrawOTag(ot); }</pre>	<pre>(B) while(1){ DrawSync(0); swap_buffer(); VSync(0); DrawOTag(ot); }</pre>
---	--

But, when the switch is forced to be synchronized with V-BLNK, the movement of objects will be awkward because the frame rate is frequently changed between 1/60 sec and 1/30 sec, in case the transaction ends around 1/60 sec. This may cause users to complain as well.

In such a case, please fix the frame rate to 1/30 sec by using VSync(2).

```
while(1) {
    ....
    DrawSync(0);
    VSync(2);          /* set to 1/30 sec fixed rate */
    swap_buffer();
    DrawOTag(ot);
}
```

Thus, as far as possible, please keep the frame rate constant by using VSync(n).

But depending on the application, it may not be better to set the worst frame rate. Even in the worst case, the internal clock of the program should not time buffer switching. Instead an absolute counter, such as VSync(-1), RCnt3, etc., should be used.

<pre>(A) while(1){ DrawSync(0); swap_buffer(); Vsync(0); frame++; DrawOTag(ot); }</pre>	<pre>(B) while(1){ DrawSync(0); swap_buffer(); VSync(0); frame = VSync(-1); DrawOTag(ot); }</pre>
---	---

By the means of a frame counter counting as shown in (B), the internal counter will not delay if the frame rate decreases momentarily due to a calculation or drawing overflow. If the counter is used to update the position of objects, the movement of objects can be kept natural, even if the frame rate drops.

(2) MoveImage() is slower than current system when it is executed for a thin rectangular region of width less than 32 dots. For example, this would be the problem when the MoveImage() is frequently used to move a thin rectangular region such as 8 by 240.

[Countermeasure]

- By using ResetGraph(1) just after VSync(0) to stop MoveImage(), the influence of slowness should be confined within the frame, and avoid affecting other frames.

- In the case of revision-C, the narrower the width of rectangular region is, the more prominent the difference from the current system is. Therefore, please make the width of the rectangular regions wider.

Both of the approaches above can be considered, but the former would be better.

More care is needed if the interlace mode is used with this function. Even if the frame rate is slower than 1/60 sec, the screen will be disordered. For example, when the drawing is done with the interlace mode (640x480 etc) and a single buffer, please use ResetGraph(1) instead of DrawSync(0).

For your reference, the following chart show how much slower MoveImage() is in Revision-C when compared to current system.

WIDTH	Revision-C / Revision-A (%)
32	100 %
16	85 %
8	77 %
4	70 %