
* Z-buffering example *

(c) Sony Computer Entertainment Europe
Introduction

This example performs Z-buffering on PSX. It consists of an assembly program which does the actual Z-buffering, and a C program that sets things up and does the controls and calls the drawing routines.

How it's done

The Z-buffering is carried out as follows:
Each poly is taken in turn, then for the area covered on screen by the bounding rectangle of the polygon the max and min z values are taken. If the three vertices of the polygon have a lower z than the min z value for the area then the polygon is drawn. If the three vertices have a larger z than the max z then the polygon is not drawn. If neither of these is the case the polygon is split in two and each of these polygons checked.

When a polygon is split some information is added to the stack, which in extreme cases may become large, however some reducing of the number added is done. If the poly has any area it is split into two, one of which overwrites the original poly, and the other is placed on the stack. To help reduce the amount of stack space used, checks are carried out to ignore any polys that have no area. This means it may be the case that one poly overwrites the original and nothing is added to the stack.

Drawing the scene

The scene is made up of a background and model. The whole thing runs in hi-res 640x480 interlaced. For the sake of speed the Z-buffer itself is only 320x240, thus making one Z value for four pixels.

The calculation of the Z-buffered polygons is separate from the drawing to allow the screen to be in interlaced mode, even though the calculation may take more than one frame (depending on the model, distance and Z values it can take six or more frames of calculation).

The background has been split into DR_LOAD chunks to allow the odd and even fields to be updated individually. Further to this only the lines altered by the drawing of the model are refreshed, thus saving more time.

The scene is also triple buffered allowing there to be one calculated, one being drawn and the next buffer to be calculated at once. Without this there is a delay when one is calculated and one is being drawn to wait for the drawing to finish before the next calculation starts. Triple buffering means that there is no delay waiting for a screen to be drawn before the next lot of calculations are started.

Although there's a field for OT position, because the model is drawn semi-transparent there is no need to use it. This saves a little time as no calculation of OT values is needed before the model is Z-buffered.

Limitations and alterations

Presently the code is set up to accept a model that is made up of only flat textured triangles. The triangle part cannot be changed in the Z-buffer

code, however quads can easily be split as the model is loaded.

In the assembly code there are some offsets. The pt offsets correspond to POLY_FT3 plus additional information for ot value and z values which are stored to stack when one of the polygons is split into two. The p offsets correspond to the POLY structure used in the C code.

To change the polygon type all that is needed is to change the info in the POLY structure (and therefore the p offsets in the assembly) and the pt offsets to match up with the new poly type. Then the information saved when either the a poly is added to the ot, saved or loaded from the stack needs to be changed. A further complication with Gouraud polygons is the subdividing of the colour information, which can be added to the section of assembly code dealing with vertex subdivision.

Parameters to the assembly routine

The function takes 5 parameters:

A pointer the polygons to be manipulated

A pointer to the Z-buffer

A pointer ot the ordering table

A pointer where the polys are to be stored for drawing

The number of polygons sent to the routine