HWAccel

COLLABORATORS			
	TITLE : HWAccel		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 24, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

HWAccel

1.1 HWAccel

3D Hardware acceleration, although hardly available on the Amiga ↩
, has
been one of the buzzwords in gaming industry for about two years.
Modern 3D Hardware can virtually do miracles. This text should serve as
an introduction to 3D hardware with a bias towards the Warp3D
environment.

1.2 HWAccel/Introduction

Introduction
****

The ViRGE and Permedia2 are two examples of 3D Hardware on the Amiga. Although these are currently the only ones available, chances are that this will change with the arrival of the new machines, and the Permedia2 is a fairly modern and fairly fast chip. But so far, programming 3D graphics hardware has been difficult due to the fact that there wasn't any uniform API that supports all available cards.

This short introduction concentrates on the features of the available chips, with a bias towards the ViRGE chip, since the current Warp3D implementation directly supports it.

Basically, a 3D chip can draw Triangles. More modern chips can do other geometric shapes, like quadrangles or polygons, with the latest generation able to draw curved surfaces, but the Triangle remains the number one drawing primitive because

1. A triangle is always planar, meaning that all vertices of a triangle alway lie in the same plane

2. A triangle is always convex, meaning that any two points within the triangle can always be connected with one straight line that lies completely within the triangle.

Because of this, Warp3D basically supports triangles as a drawing primitive (besides points and lines). The following chapters describe what you can do with these triangles.

Two other "primitives" are supported by Warp3D, the Triangle Strip and the Triangle Fan. A detail description can be found in the OpenGL specification. Basically, a triangle strip is a series of vertices, with each three successive vertices defining one triangle that has one edge in common with the previous one. A triangle fan is a series of vertices describing triangles that always have the first vertex in common.

1.3 HWAccel/Texture Mapping

Texture Mapping
****

Texture mapping is the process of mapping a rectangular (or square) image map onto a triangle projected to the screen in such a way that the image seems to be stretched perspectively. There are two basic modes for texture mapping: Linear and Perspective texture mapping.

Linear texture mapping is faster and easier, since the perspective foreshortening is not taken into account, meaning that there is less calculations required for drawing the map. The result, however, tends

to "shift" and "wobble" at sharp viewing angles because of dramatic differences in vertex depth.

Perspective mapping, sometimes also shortly called "p-mapping", takes the depth of each texel into account, resulting in a steady, correctly looking texture map. It is a bit slower than linear mapping, since more calculation is required on behalf of the chip, but the dramatic increase in picture quality is often worth this price.

In order to know how to map a square or rectangular texture map onto a triangle, each vertex is assigned a pair of texel coordinates, known as the U/V coordinate pair. The U/V coordinates at a vertex correspond to the X/Y coordinates into the texture map, with 0/0 being the upper left corner of the map.

With this information per vertex, the 3D chip can "stretch" the image onto the triangle like a piece of cloth over a wooden frame. Unlike the cloth, however, the texture map can be repeated over the triangle for as many times as desired. This process is called Texture Wrapping, and can be enabled or disabled. If disabled, the parts of the triangle outside the texture map are filled with a special "border" color. This process is called Clamping.

If texture mapping is not active, the triangle might either be flat-shaded or Gouraud-Shaded. When Flat-Shading, the complete triangle is filled with one single color. The vertex colors (See
Blending
) are
ignored.

Gouraud-Shading works by interpolating the vertex colors linearly along the edges of the triangle, and also linearly along the scan-lines. This gives an effect that resembles lightning of the triangle. With Blending, this effect can be combined with texture mapping, resulting in textured triangles that are also illuminated. The game Descent makes heavy use of lit-textureing.

1.4 HWAccel/MIP-Mapping

MIP-Mapping
****

Apparently, the term MIP-Mapping comes from the latin word multum in parvo, meaning "many things in a small place". The term refers to a kind of anti-aliasing technique that almost all 3D chip can do.

Instead of storing only one texture map of size n times n, additional texture maps are stored at decreasingly smaller sizes. For example, for a texture of size 128x128, additional maps would be stored with size 64x64, 32x32, 16x16, 8x8, 4x4, 2x2 and 1x1. Now, when a texture is mapped to the screen and made larger, the original image can be used and filtered (See
Filtering

```
                ); when it is made smaller, a mip-map is
selected accordingly.
```

MIP-Maps are either generated automatically by Warp3D, or can be pre-fabricated by a graphic artist. The latter method yields better results, since an artist can decide which details on the texture map are important enough to be "emphasised" on the smaller map. When MIP-Maps are automatically generated by Warp3D, an box filter is applied. Even with this simple process, the result looks much better with MIP-Mapping than with traditional mapping. For a good discussion of MIP-Mapping, see http://www.irth.net/milo/3dgames/mipmap.html

1.5 HWAccel/Filtering

```
                Filtering
*********
```

When walking close to a wall in about any Wolfenstein- or Doom-Style game, the first thing we notice is how blocky the graphics get. In the game Descent, for example, all texture maps are 64x64 pixels. When flying close to the wall, this texture map gets stretched to almost screen size, making each pixel four or five times larger than usual. Jaggies get very obvious at this size, making the overall impression look bad.

Filtering is a process for eliminating these jaggies. Imagine a black and a white pixel enlarged 5 times, resulting in two 5x5 pixel squares. Instead of drawing five black and five white pixels per scan-line, a filtering 3D chip draws only three or so black pixels, followed by a few medium grays, and finally three white pixels. This "ramping" of colors washes out the jaggies, anti-aliasing the texture. This really dramatically improves the look of the final image, as can be seen in the ViRGE version of ADescent...

There are a bunch of different filtering modes available. They can be divided into two main groups: Those that do mip-mapping(See

```
                MIP-Mapping
                ), and those that don't.
```

All filter modes that have the word MIP in them are MIP-Mapping modes. For example, the mode W3D_NEAREST means that no filtering and no MIP-Mapping is performed. The mode W3D_LINEAR means to do bilinear filtering, as described above. Finally, the mode W3D_LINEAR_MIP_LINEAR specifies tri-linear filtering, meaning that bilinear filtering is performed on two mipmap levels closest to the real point, and that this value is linearly interpolated afterwards.

1.6 HWAccel/Fogging

Fogging
****

An effect often seen in Doom- and Wolfenstein-Type engines is that the player is glowing. At least that is the impression one gets, because everything further away from the player gets darker and darker, until it's pitch black. This effect is called Depth Cueing, and in fact is only a special form of the more general technique called Fogging.

Fogging blends a pixel with a specified fog color depending on the depth of the pixel and on the specified fogging function. There are three fogging functions available in Warp3D, namely those that are defined by the OpenGL specification. Those modes are linear fogging (W3D_FOG_LINEAR), exponential fogging (W3D_FOG_EXP) and square exponential fogging (W3D_FOG_EXP_2). The first mode, linear fogging, is used for depth cueing. The two other modes are also collectively called non-linear fogging, and are most often used for atmospheric effects like, well, fog.

In order to use fogging, a fog start and end must be specified. Everything in front of the fog start is drawn normally, while everything behind the fog end is drawn entirely in the fog color. When a triangle lies partly or completely between the fog start and end, then the fog color is blended with the texture color depending on the depth and fogging function.

1.7 HWAccel/Alpha Blending

Alpha Blending
****

The Term Alpha Blending refers to the process of making textures transparent. When Alpha Blending is enabled, a texture does not simply replace screen pixels, but is rather blended with the pixel color, the amount of blending determined by the Alpha value of the texture pixel. An alpha value of 0 means the texture pixel is completely transparent, while a value of 1.0 means is is opaque.

Some 3D chips support source and destination functions. The combination of source and destination function determine how the alpha value of the new pixel (from the texture) and the alpha value of the old pixel (the screen pixel) are combined to form the final alpha value for the blending process. Once this alpha value is determined, blending works like the one for fogging.

The ViRGE chip cannot do all forms of alpha blending. Most notably, there's no difference between source and destination functions, the alpha value of the texture always completely determines the final alpha value.

Also, like most blending effects), alpha blending does not work on

LUT8 screens (i.e. screens that do have only 8 bit and a palette).

1.8 HWAccel/Blending

Blending
****

Although it sounds very much like Alpha Blending, the term Blending is a bit different, because it determines the way that the texture affects the primitive it is mapped onto. The most common use for this technique is lighting. A detailed description of Blending is given in the OpenGL specification.

For lighting, each vertex is assigned a color. This color defines the color of the light that this vertex receives. If the mode is W3D_REPLACE, those values are ignored since the texture color replaces the vertex color.

Again, the ViRGE chip does not support every blending mode, most notably, it only supports W3D_REPLACE and W3D_DECAL as blending modes, but these usually are sufficient for games and simulated lighting.

1.9 HWAccel/Z Buffering

Z Buffering
****

In most cases it is not sufficient to simply draw every polygon in a scene to obtain a three-dimensional image. At the first step, surfaces that are not visible must be culled away, but still this is not enough to get the 3D-Effect. The problem is that triangles that are further away from the viewer must not overdraw triangles that are closer to the viewer.

There are many techniques to achieve this. One is the famous Painter's Algorithm, where triangles are sorted by their depth and then drawn back to front. One of the inherent problems with this algorithm however is that this depth order might not be defined, for example when there is a cyclic overlap of polygons.

Another possibility is the use of BSP trees. Those have become famous since the game Doom, because Doom uses them effectively to solve the depth ordering problem.

A third possibility is Z-Buffering. The advantage of this technique is that is is implemented in most modern chips in hardware, making it quite fast.

Z-Buffering works by assigning each pixel on the screen a depth coordinate. When a primitive is to be drawn, each pixel to be drawn is

checked against the pixel already on screen. If the test succeeds, the pixel is drawn and it's depth coordinate
(1)
Z Buffering-Footnotes
replaces that
of the screen pixel. Thus, a primitive drawn later will fail to be drawn if its depth value does not succeed in the test.

The Test performed depends on the Compare Mode currently set. For example, it the mode W3D_Z_LESS is set, and the ZBuffer is initialized to all 1.0, each incoming pixel will be drawn that has a Z coordinate of less than 1.0, essentially overwriting the 1.0 with its own value. Each successive pixel will be discarded if its depth value is again closer to 1.0, essentially solving the depth sorting problem.

Note also that all kinds of interesting effects can be achieved. For example, setting the Compare Mode to W3D_Z_EQUAL and setting the Z-Buffer uniformly to one value will essentially cut a slice out of the scene at the given depth.

One of the problems with Z-Buffering is the precision. Most Z-Buffers are 16 bits wide, meaning there are 65536 different depth steps. When pixels are very close together, this precision might not be enough, resulting in "leaking" of pixels from other polygons. This problem is inherent to Z-Buffering, and might not be solved ad-hoc without careful consideration, but for games this effect is most likely not so dramatic.

1.10 HWAccel/Z Buffering-Footnotes

(1) Also called Z-Value, hence the term ZBuffer

1.11 HWAccel/Conclusion

Conclusion
****

It is the hope of the author of this text that it has served a bit to introduce the reader to the basics of 3D-Hardware acceleration. As mentioned before, it was not intended as a technical guide or anything like that.

For further reading, I recommend the following:

* The OpenGL specification, available on their web site at http://www.sgi.com

* John DiCammillo's Web Site at http://www.irth.net/milo

* Fast Rendering Techniques for Real-Time 3D Image Synthesis in an

interactive Environment, a master's thesis by Hans Holten-Lund, at the Technical University of Denmark (sorry, no URL)

* ... or one of the many 3D-Graphics related web sites out there