The Voodoo™ Architecture Advantage

GRAPHICS PERFORMANCE AND IMAGE QUALITY

Revision 1.2
February 5, 1999
Copyright © 1998 3dfx Interactive, Inc. All Rights Reserved
3dfx Interactive, Inc.
4435 Fortran Drive
San Jose, CA 95134
Phone: (408) 935-4400

Copyright © 1998 3dfx Interactive, Inc.

1. Introduction

This paper describes the Voodoo™ architecture Advantage, a combination of performance and quality advantages that makes the Voodoo3 3D architecture stand out from all other chips. There are two aspects to the Voodoo Advantage: performance and image quality. Certain features of the unique Voodoo architecture result in substantial performance increases, above and beyond what the normal benchmark numbers indicate. Other Voodoo 3D features result in improved image quality. Image quality differences can sometimes be subtle, and can sometimes be substantial.

This paper will describe each of the components of the Voodoo Advantage, taking an in depth look at the issues involved and the advantages that Voodoo 3D offers.

Advantages:

2. Multi-Texturing

Voodoo 3D, by means of it's patent-pending architecture, has the unique capability of rendering multiple textures onto a polygon in a single pass and single cycle. Employing multiple Texture Mapping Units (TMUs) that each render a completely independent texture onto a polygon, makes multi-texturing a standard feature of a consumer game platform.

While most of the components of the Voodoo Advantage are categorized strictly under performance or quality, multiple textures can improve **either** performance or image quality and sometimes both. Because of this, multiple textures are a unique advantage and deserve a dedicated category. Thus the existence of multiple TMUs can result in either improved image quality or increased performance, or in some cases, both.

2.1 performance

Multiple textures can result in a dramatic increase in performance. Increasingly, game content is being authored to make use of advanced multipass rendering techniques to improve the visual quality of the game. These multipass techniques simulate lighting, reflections, and detailed textures among other things. These multipass techniques essentially apply multiple textures to each pixel. Increasingly game performance will become dependent upon not only triangle rate and raw pixel fill rate, but on the texel fill rate. Texel fill rate is the rate at which the hardware can generate and fill texutres to pixels. 3dfx defines the texel fill rate as the number of bilinearly filtered textures per second, and will commonly be expressed as megatexels per second. It is important to note that the Voodoo 3D architecture supports high quality, filtered texels, which means not only high performance game play, but high quality images.

2.1.1 double the effective fill rate

Fill rate is often a limiting bottleneck in most games. With multiple TMUs two textures can be simultaneously rendered resulting in twice the effective fill rate. Voodoo3 for example, has a pixel fill rate of 183 Mpixels per second. Voodoo3 additionally has the ability to render two fully featured textures per pixel in a single cycle. This ability allows applications that make use of multipass rendering techniques to deliver an effective fill rate of 366 Mtexels per second. In other words, architectures that do not support multitexturing like Voodoo3 would require 366 Mpixels of real fill rate to match the performance of Voodoo3. Rendering multiple textures can often have a profound impact on games that are fill rate limited. Hence, Voodoo3 has 366 megatexels/sec fill rate.

A large number of recent games utilize lighting maps (or dark maps) to achieve a realistic lighting effect within a scene. The lighting maps are computed, either offline or in realtime by the game, and are typically multiplied by the

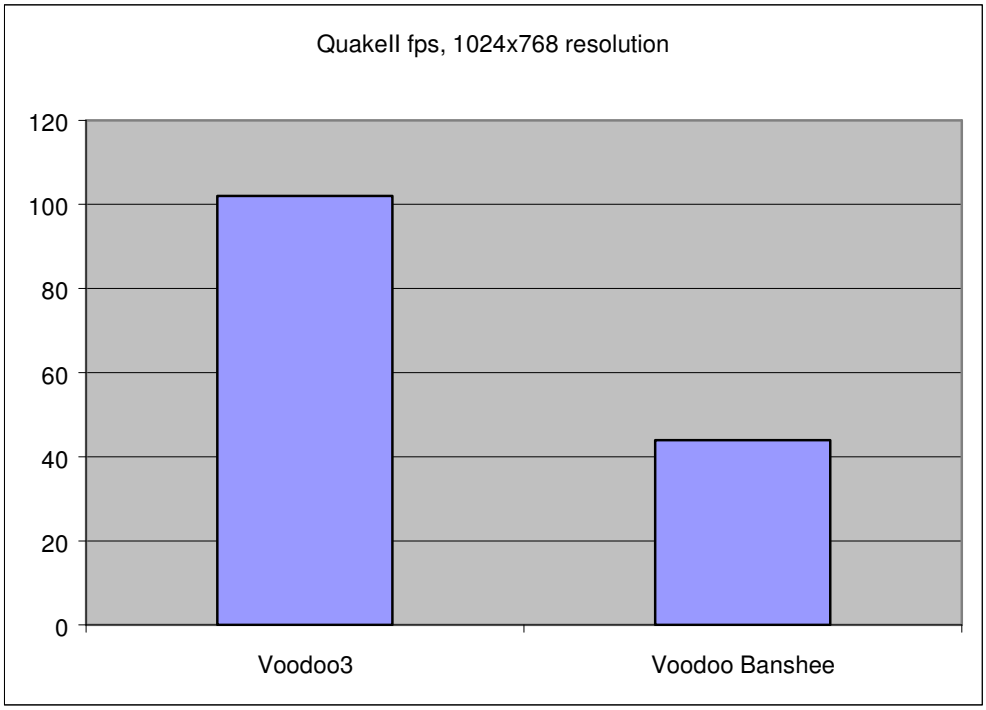
Copyright © 1998 3dfx Interactive, Inc.

base texture. The base texture is repeated across a polygon or many polygons at high resolution, while the lighting map is unique at every location at is very low resolution. Thus, the two textures cannot be combined into a single texture map beforehand (it would require many times the memory requirements). Lighting maps are a perfect application for multiple textures - both the lighting map and the base texture are rendered at the same time - effectively doubling the fill rate of the system.

2.1.2 One pass through data base

When a game requires two textures per polygon and the graphics only supports one texture, games often render the entire scene twice in order to avoid changing alphablending modes for every triangle (QuakeII). This requires two passes through the scene database and twice the number of triangles transformed, lit, clipped, and rendered. Voodoo3, with its ability to process two textures in a single cycle, can render such a scene in one pass, greatly reducing the load on the CPU, the IO bus, and the graphics subsystem.

2.1.3 Performance Gains



While all of the performance gains cannot be attributed to two textures per pixel, clearly the ability to eliminate the second lighting pass in QuakeII offers almost a doubling of performance, as well as reducing CPU loading due to the increased triangle rate requirements of the second (lighting) pass.

2.2 Quality

Multiple textures can result in improved quality as well as increased performance. Below are some examples of how multiple textures can improve image quality.

Copyright © 1998 3dfx Interactive, Inc.

2.2.1 Trilinear mipmapping

Trilinear mipmapping is one of the highest quality texture filtering methods available, requiring 8 texture samples and three linear interpolations (thus the name trilinear). Trilinear mipmapping looks better than bilinear mipmapping because it eliminates *mipmap bands* which appear within a polygon when the rendering engine switches from one mipmap level to another mipmap level. Trilinear mipmapping blends between mipmap levels, producing a smooth transition between mipmap levels with no banding. In many textures, mipmap bands are not noticeable, but in other textures they are very distracting. In the picture below the left image is bilinear filtered and the right image is trilinear filtered. Notice several visible bands along the tracks in the left image.

Each of the TMUs in Voodoo[3] 3D is capable of performing a bilinear filter operation on 4 texture samples per clock. Trilinear mipmapping thus requires two passes with one TMU. With two TMUs, trilinear mipmapping can be performed at full speed in a single pass and single cycle. Although trilinear mipmapping can be performed without multiple TMUs, doing so will result in a loss of performance. Since frame rate is of utmost importance in real-time applications, bilinear mipmapping is often chosen instead of trilinear mipmapping when there is a performance difference. Since two TMUs allows for trilinear mipmapping without compromising performance, higher image quality can be achieved with no performance degradation on Voodoo[2].

It is important to understand that Voodoo3 delivers 183 megapixels/sec in full trilinear filtered mode. Voodoo3 achieves this through single pass, single cycle trilinear filtering. Other architectures may offer trilinear filtering, but will do so at a performance penalty. In measuring trilinear performance one should measure real trilinear fill rates, not simply a claim of single cycle or single pass.

2.2.2 Full Speed Filtering

The Voodoo 3D architecture performs filtering operations at full speeds under all single texture per pixel conditions. In fact, Voodoo3 can perform trilinear filtering at full hardware speeds, suffering no performance degradation. The Voodoo platform has been designed for maximum game performance, which means sustained, high frame rates. Functions or filtering that have substantial performance penalties are essentially valueless to the game developer as their frame rates will suffer an unacceptable frame rate penalty. Many other graphics architectures claim advanced

Copyright © 1998 3dfx Interactive, Inc.

filtering techniques but perform them either so slowly in hardware, or perform them in software, that there is a 4x, 8x, or possibly even a 16x performance penalty. These operations may improve the visual quality of a single frame, but when enabled the game's frame rate my drop from 30 frames per second, to 2 frames per second, rendering the feature completely unusable. Voodoo performs advanced filtering operations at full speed, allowing both for advanced image quality as well as high frame rates.

2.2.3 Full 32-bit RGBA with 1 pass

One subtle benefit of being able to render multiple textures with one pass is that multiple passes can be avoided and color computations can be performed in full 32-bit RGBA precision. For example, to render a base texture combined with a lighting map in a graphics system that can only render one texture per pass, the results of the first rendering pass are typically truncated from 24-bit RGB to 16-bit RGB when stored in the framebuffer. When this truncation is followed by a second pass, visual anomalies often result, which can typically be seen as bands of discoloration.

2.2.4 Special effects

Although gratuitous special effects can have a negative effect on a game or movie, there are many special effects that are not gratuitous. For example, reflections in a mirror or off a sheet of glass or shiny surface can add a realistic touch to a scene. Many special effects can be implemented by rendering two textures onto a polygon.

Copyright © 1998 3dfx Interactive, Inc.

2.2.4.1 Detail Textures

A very useful technique is to use a detail texture to add high frequency noise to a texture. This prevents the base texture from becoming *blurry* when viewed at high magnification. For example, a detail texture like stucco can be combined with a wall texture. When the viewer is up close to the wall, the wall texture is very blurry, but the detail texture is not. Detail textures are typically noisy patterns and are typically repeated many times across a polygon. For example, a detail texture might be repeated 16 to 64 times as often as the base texture. This prevents the detail texture from becoming blurry when up close. Ordinary per-pixel mipmapping prevents the detail texture from aliasing or sparkling.

2.2.4.2 Reflection Maps

A simple reflection can be implemented using a reflection map. One example of a reflection map is the effect of clouds reflected in a car's rear window. When the rear window polygon is rendered, rays can be cast from the viewer towards the vertices and upwards into the sky, indexing into a sky texture. The sky texture and the texture for the rear window (streaky glass for example) are then rendered simultaneously. This same technique can also be applied to other shiny surfaces on the car, e.g. the car's roof. Mirrors can also be rendered using this technique assuming that a reflection map of the surrounding environment has been created beforehand.

Copyright © 1998 3dfx Interactive, Inc.

2.2.4.3 Projected Textures

Spot lights and head lamps can be rendered using projected textures. In this case, the light's texture is projected onto polygons in the scene, and a new set of texture coordinates for the projected are computed. The projected texture is rendered at the same time the base texture for the polygon.

2.2.4.4 Bump Mapping

Bump mapping adds lighting detail to an otherwise flat surface, giving the surface a "bumpy" look and feel. There are several methods for creating bump mapping, one involves using paletted textures and the other involves multi-pass rendering. Voodoo^3 3D supports both these methods at full rendering performance and with all filtering modes. In fact, Voodoo^3 supports bump mapping at full speeds, in a single pass and single cyle. This full speed approach to bump mapping makes Voodoo^3 unique among graphics architectures, offering full speed performance even while bump mapping.

3. Texture Compression

3.1 Texture Compression

Voodoo3 supports texture compression in the form of palettized textures and a patent-pending proprietary Narrow Channel Compression format. Texture compression allows applications to have greater effective texture memory, making more efficient use of the available texture storage, as well as maximizing texturing performance as each texture downloaded can be smaller in size, minimizing the bandwidth impact.

3.1.1 Palletized Textures

Voodoo3 fully supports 8-bit paletted textures, offering both 24-bit RGB and RGBA formats. These formats are commonly used by game developers and provide for high-quality artwork while greatly reducing the texture memory requirements. Some competing designs either cannot filter paletted textures or convert them to 16-bit or 24-bit textures in their drivers. Voodoo^2 3D can perform advanced filter on paletted textures, providing both the texture memory savings and high-quality artwork needed by today's games.

3.1.2 NCC Textures

Voodoo3 also offers a patented proprietary Narrow Channel Compression (NCC) format for textures. NCC textures occupy 8 bits per texel just like palettized textures, but the decompression table is 20 times smaller. This makes switching textures much more efficient and allows applications to use a different table per texture. Several arcade games use NCC textures to offer the highest resolution textures possible without noticeable image quality loss.

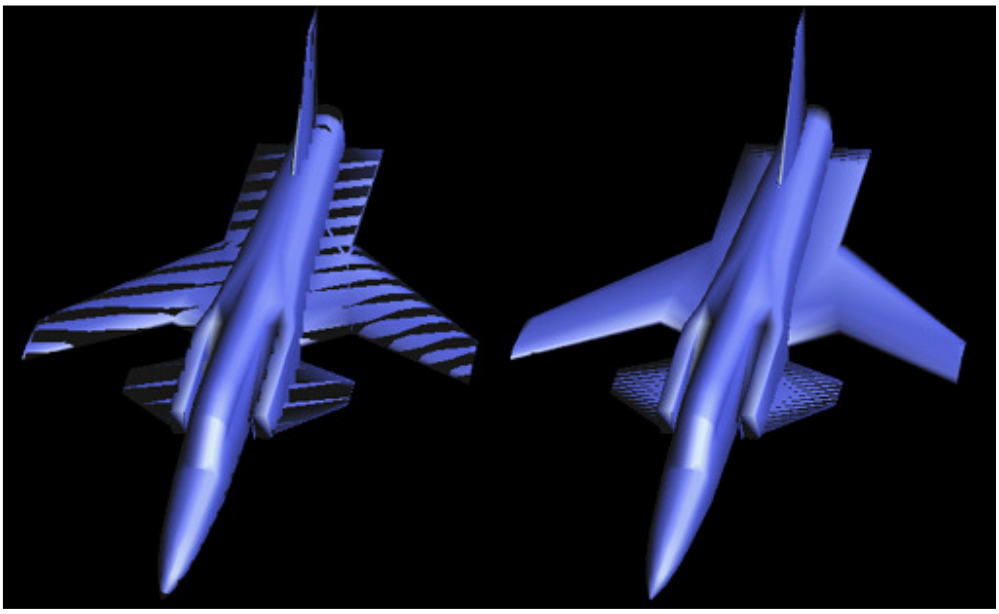
Copyright © 1998 3dfx Interactive, Inc.

4. Image Quality

4.1 Floating point depth buffering to eliminate Z aliasing artifacts

One of the fundamental problems in polygon based 3-d graphics rendering is rendering objects so as to assure that what is behind is actually behind and what is in front actually in front when it is drawn to the screen (typically called occlusion). To Z-buffer, one keeps a 2d-array of numbers the same dimensions as the viewport. (essentially a buffer of depth values equal in size (resolution) to the frame buffer). These values are typically integers, although the precision varies from 16 bits in consumer boards to 24 and 32 bits in professional boards. As each pixel is prepared for display, the Z value for the pixel is compared to the Z value already in the position in the Z-buffer array. If the new Z is closer than the one already in the buffer, the new pixel replaces the old pixel in both the color and depth buffers.

Voodoo 3D offers both a linear 1/z buffer (for compatibility with existing APIs) and a floating point Wbuffer for improved quality. This not only improves picture quality by increasing the effective resolution of the depthbuffer, but can also improve performance by reducing the data required from the CPU and the triangle setup math required. The picture below compares a linear depthbuffer on the left to a floating point depthbuffer on the right. The floating point depthbuffer has a better distribution of values and thus experiences less aliasing and errors.



The precision advantage of floating point can best be shown by a simple example. In the case where the near and far clipping planes are at [1…65,536] and Z values range from 1 to 65,536, the 1/z values range from 1/1 to 1/65,536. In practice, these values are multiplied by 65,536 resulting in a 16-bit integer 1/z value in the range 65,536 to 1. Note that because of the 1/z function (which is necessary to allow for linear screen-space interpolation), the Z value of 2 is represented in the depthbuffer as 32,768. Thus, half of all depthbuffer values (the values from 65,536 to 32,768) are used to represent the just the first unit of depth. More generally, half of the depthbuffer represents the range [near, 2*near]. On the other end of the spectrum, all Z values from in 32,768 to 65,536 (more generally [far/2,

Copyright © 1998 3dfx Interactive, Inc.

far] are represented by only two values in the depthbuffer: 2 and 1. Thus the 1/z linear depthbuffer deploys excessive resolution up close to the near clipping plane and too little resolution in the distance. This results in *z-aliasing*, where multiple pixels at different depths resolve to the same Z value in the depthbuffer. In these cases, either the first or the last pixel rendered is visible (depending on the Z compare function) and hidden surface removal is not accurate resulting in *z-poke-throughs*.

In contrast, a floating point depthbuffer results in an equal number of depthbuffer values for each power of two in depth. For example, Voodoo3 3D uses a 4.12 floating point representation, which means 4 bits are allocated for the exponent and 12 bits for the mantissa. This allows for 4096 unique mantissa values for each exponent. In the above example, the depth buffer range is from 2^0 to 2^{15}. A 4.12 floating point depthbuffer would have 4096 values in the range [near,2*near] or [1,2] and also have 4096 values in the range [far/2, far] or [32k,64k]. Thus, the values are distributed much more evenly than a 1/z linear depthbuffer.

Note that Voodoo3 3D offers both a linear depthbuffer and a floating point depthbuffer. This offers both compatibility with existing APIs as well as a choice by applications as to which to use. In some cases, extremely high precision up close may be desirable, and a linear depthbuffer can be used. In other cases, a more even distribution of depthbuffer values is more appropriate and a floating point depthbuffer can be used.

4.2 Table based fog

Voodoo implements both linear interpolated fog and table lookup fog. Linear interpolated fog is very limited in it's usefulness and can result in visual artifacts. In addition, fog is really an exponential type function and cannot be linearly interpolated in screen space. Because of the limitations of linearly interpolated fog, Voodoo implements a patent-pending fog table.

Fog is theoretically an exponential function: e^{-kz} where e is the natural logarithm base, k is a constant, and z is a pixel's distance into the scene. Sometimes, applications use e^{-kzz} because it results in a better looking picture. Regardless what equation is used, linearly interpolated fog cannot accurately represent the fog equation unless the database is such that all the polygons are extremely small. If all the polygons are small on the screen, then the fog equation will be computed at all the vertices and the linear interpolation between vertices will not result in significant error. However, if large polygons exist, then the linear interpolation will be inaccurate. For example, you cannot have a long runway or road in a scene and get linear interpolated fog to render the polygon accurately. The fog will be too dense or too light throughout most of the polygon, and the fog density will also not match up with the fog density of neighboring polygons, where the fog equation was evaluated at each vertex. Refer back to the figure in section 2.2.1 to see a good example of exponential fog.

Table based fog can represent any function, either linear or exponential, as well as functions that both increase and decrease with distance. The fog equation is evaluated for every rendered pixel and thus there is never an issue of interpolated values matching computed values. Voodoo uses a patent-pending technique for indexing into the fog table, to guarantee an optimal distribution of table entries in screen space. This distribution combined with linear interpolation between fog table entries results in minimal banding with a very cost-effective solution.

In addition to the quality difference, there is also a performance difference. Voodoo implements table based fog at no performance penalty and with no additional host information per vertex. Linearly interpolated fog requires a fog parameter per vertex and thus requires more setup computations and more data transfer from the host CPU to the graphics subsystem.

4.3 Per Pixel mipmapping

We believe the Voodoo architecture is in a class by itself when it comes to mipmapping. As far as we know, Voodoo is the only low-cost PC solution that performs accurate per-pixel mipmapping. While every chip claims to support mipmapping, none implement an accurate per-pixel mipmapping selection. Instead, they use a variety of short-cuts.

Copyright © 1998 3dfx Interactive, Inc.

Voodoo computes an extremely accurate Level-Of-Detail (LOD) value for every pixel rendered. This LOD value is used to select a mipmap for every single pixel rendered, and can freely change from one pixel to the next. It is very important to select the proper mipmap - an inappropriate selection results in either excessive blurring of the texture or excessive sharpness and therefore aliasing. This per-pixel computation requires absolutely no host CPU intervention or assistance.

There are many ways to approximate LOD. The common methods are described below and their disadvantages are discussed.

4.3.1 per-pixel interpolated LOD

The most accurate approximation to per-pixel mipmapping is per-pixel interpolated LOD. The host CPU typically computes an LOD for each vertex of the polygon, and then the graphics subsystem interpolates this LOD across the polygon. This imposes a severe computational load on the host CPU as well as additional parameter data transfer and setup requirements. Interpolation of LOD across a polygon is also inaccurate resulting in both excessive blurriness as well as sharpness (aliasing).

LOD can also be computed within the graphics processor, typically by the setup unit. This will impose an additional load on the setup processor, and typically result in decreased triangle throughput when mipmapping is enabled.

4.3.2 per-polygon LOD

The least accurate approximation to per-pixel mipmapping is per polygon LOD. In this scheme, the host CPU or graphics subsystem computes an LOD for each rendered polygon, and this one mipmap level is used for rendering the entire polygon. This results in substantial errors for larger polygons - the result being sections of the polygon that are either excessively blurry or sharp. Even worse is an artifact known as LOD “popping”. This occurs when the graphics code decides to change the LOD for a polygon for the new frame. When this occurs, the entire polygon changes LOD becoming either twice as blurry or twice as sharp. This is very noticeable and distracting, especially for large polygons. If the LOD computation is slightly unstable, or if the frame to frame changes are such that the LOD changes from one value to another and then back again, the polygon will repeatedly “pop” between mipmap levels and result in an extremely annoying visual artifact.

One technique to reduce the errors of per-polygon mipmapping is to subdivide the polygon into smaller polygons when it spans multiple LODs. This subdivision is typically performed in the driver or in graphics microcode. In either case, it can result in a substantial loss of performance, both for polygons that need subdivision - because of the creation of additional polygons to be rendered, as well as for polygons that do not need subdivision - because of the need to detect the cases where subdivision is necessary. This extra computational burden that is placed on either the host CPU or the graphics engine will ultimately limit performance.

Accurate trilinear mipmapping is nearly impossible to implement with per-polygon LOD. Since trilinear mipmapping requires blending between two mipmap levels based on the fraction of the LOD value, if an accurate LOD value cannot be computed per pixel, then there is absolutely no way to implement accurate trilinear mipmapping. Many vendors say they “support” trilinear mipmapping, but it is not known whether they accurately implement this feature or not.

Again, refer back to the figure in section 2.2.1: the railroad track spans many mipmap levels and is only comprised of a few polygons.

4.4 polygon cracks and subpixel correction

Voodoo uses a unique triangle filling algorithm that is infinitely precise. This guarantees that there will never be any cracks or gaps between adjacent polygons as long as the polygons share vertices. In addition, all parameters are sub-pixel corrected and adjusted to insure the highest quality rendering possible.

Copyright © 1998 3dfx Interactive, Inc.

Note that T-junctions can still result in occasional gaps between polygons, but this is unavoidable, even in software renderers. A T-junction is when two polygons abut a third polygon along one of it's edges.

4.5 alphablending

Voodoo contains a full alphablending implementation. A common myth is that alpha*src_color + (1-alpha)*dest_color is all there is to alphablending. In order to support the widest range of 3d applications and games the full complement of OpenGL and D3D alphablending modes are required. Although the above blending mode is very common, there are many other blending modes that are useful.

One useful blending mode is src_color*dst_color. This blending mode is used for colored lighting maps where the base texture is one of the colors and the other color is the lighting map. Some chips do not implement this blending mode and thus can only implement monochromatic light. This greatly reduces the quality of the lighting effect, especially in games, where non-white lights are extremely prevelant.

Another useful mode is src_color*dst_color + dst_color*src_color which results in 2*src_color*dst_color. While not immediately obvious what this is good for, the use for representing overbright colors, which are colors that are brighter than 1.0. One of the problems with multiplying colors together (texture modulation) is that when the colors are all in the range [0…1], colors can only get darker. This alphablending mode effectively allows one of the colors to be interpreted such that its range is [0…2] and thus can result in brightening of another color during multiplication. This results in more realistic lighting effects, especially specular highlights.

In addition to lighting, alphablending is one of the most useful techniques for special effects. Smoke, fire, explostions, clouds, motion blur and trails, shadows, reflections, and lens flare are just some of the special effects that rely on alphablending. In summary, the simple truth is that alphablending is not a luxury, it is an absolute necessity.

4.6 rectangular textures

Voodoo supports rectangular textures as opposed to only square textures. This allows texture memory to be more efficiently allocated for an application's textures. Efficient allocation of texture memory results in more effective texture memory and thus higher quality rendering.

In addition, Voodoo supports mipmap sizes down to 1x1 texel large. Some chips support textures only down to 32x32 minimum which results in wasted texture memory. Wasted texture memory decreases the effective texture memory that is available to an application and thus lowers the overall graphics quality.

4.7 texture formats

Voodoo3 supports 14 different texture formats, allowing for very efficient use of texture memory. This results in increased image quality. Of particular interest, Voodoo Graphics fully supports the popular 8-bit paletted texture format and does so with full-speed bilinear interpolation.

4.8 video output

Voodoo3 contains special circuitry to reduce any artifacts that can result from truncating 24-bit color to 16-bit color. This includes special processing during multi-pass rendering when 16-bit colors are read from the framebuffer, as well as the final display of 16-bit colors on the monitor. These proprietary features result in unsurpassed visual quality without the cost of 24-bit framebuffers.

Copyright © 1998 3dfx Interactive, Inc.

5. Performance

5.1.1 Driver concurrency for execution overlap

One of the main arguments in favor of bus mastering is that it decouples the host CPU from the graphics subsystem, allowing each to queue up and process commands relatively independent of one another. The goal is to avoid stalling the CPU when a large polygon is being rendered. DMA and bus mastering implementations often allow for this parallelism. While most programmed I/O implementations do not allow for this kind of parallelism, Voodoo 3D does.

The Voodoo architecture has a unique memory-backed FIFO that effectively enlarges its already large PCI fifo to the size of unused framebuffer memory. Commands are automatically placed into this memory fifo whenever the PCI fifo becomes full, and are read from this memory fifo whenever the PCI fifo becomes empty. This fifo overflow operation is totally transparent to software running on the host CPU. This scheme decouples the host CPU and graphics subsystem as effectively as a DMA-based implementation.

Voodoo3 also uses a framebuffer based command fifo similar to Voodoo Graphics. The command fifo on Voodoo3 has all the advantages of the memory-backed FIFO on Voodoo Graphics and in addition is optimized for Pentium Pro and PentiumII processors.

5.2 Triangle setup

Triangle setup in hardware helps to offload much of the floating point computation from the host, allowing for more cycles to be spent on gameplay, game AI, physics and the like. The truth is that not everyone's triangle setup is the same.

Voodoo3 has complete triangle setup in hardware. These chips are very unique in that they do not require gigaflops of floating point performance for outstanding performance - they require only hundreds of megaflops. There can be 2-5x difference in floating point computational requirements between various triangle engines. The entire Voodoo 3D line contains an ultra-efficient triangle engine that requires minimal floating point computations for triangle setup. Do not measure chip performance on megaflops - measure instead actual triangles/second.

Copyright © 1998 3dfx Interactive, Inc.