PC Graphics Reach New Level: 3D

PCs Will Become a Compelling Platform for 3D Applications in 1996

by Yong Yao

This article is part 1 of 2 on PC three-dimensional graphics. It discusses basic 3D architecture, techniques, and terminology. Part 2 (see **100304.PDF***)will compare recently announced 3D chips.*

Three-dimensional graphics is rapidly moving from the workstation world into the mainstream PC. Several low-cost 3D accelerators have recently appeared, and many PC makers have plans to put 3D capabilities into their systems this year. These chips and systems represent only the first wave: there is great headroom for further technological innovation as well as for rapid market growth.

Entering the 3D world brings exposure to a confusing welter of new concepts and buzzwords. From a software standpoint, 3D graphics involves APIs that allow applications to access hardware accelerators in a general way. The most important such API is the emerging Direct3D from Microsoft, but other software standards are being used today. In a 3D application, the CPU typically handles tessellation and geometry calculations, but rendering the image on the display is left to a hardware accelerator. These accelerators use a variety of techniques to improve the quality and speed of the displayed images.

3D Market Emerges for PCs

Various 3D-graphics products have been used for game consoles, set-top boxes, home video-game accessories, handheld portable systems, add-in graphics cards, multimedia PC motherboards, and coin-operated video-games. There is no doubt that multimedia PCs will be the fastest-growing market for 3D-related graphics products. For example, PC games today account for slightly more than 10% of the entire game market. Three-dimensional titles like Daytona, Virtua Fighter, Doom, and Mortal Kombat have been or soon will be ported to the PC platform. We expect PC game sales to reach 30% of total game sales by the end of 1997.

Most new computers come with CD-ROM drives and sound cards already installed. Most current titles are delivered on CD-ROM, as future titles will be. Microsoft has added an autoplay feature to Windows 95 so users can pop in a CD-ROM and immediately start playing a game, just as they would on a dedicated system. PCs will become the all-in-one game, information, and entertainment center for the home, rivaling the performance of electronic games developed by companies like Sony, Nintendo, and Sega.

High-quality 3D entertainment for the PC platform will help reshape the consumer multimedia PC market. Three-dimensional graphics will be a compelling feature in consumer PCs for Christmas 1996. The images you are used to seeing in Doom today will pale in comparison with the games that will appear in one year or less. The end result is that PCs will become the single most important platform for our information and entertainment.

The Biggest Challenge for Multimedia PCs

A full-featured multimedia PC requires 2D graphics acceleration, audio processing, video playback, and 3D graphics. Two-dimensional graphics, handled by low-cost GUI accelerators, is a mature technology. Audio is typically based on the original SoundBlaster (i.e., FM synthesis) from Creative Labs, but newer techniques such as wavetable, waveguide, 3D sound, and Dolby surround sound can be used to improve audio quality.

For video playback, full-screen MPEG-1 decoding at 30 frames per second has become popular. For example, companies like Compaq and Diamond are producing thousands of machines or add-in cards using the S3 MPEG-1 decoder. On the other hand, there is no urgent need for more advanced features such as MPEG-2 decoding or MPEG encoding in PCs today. Therefore, the biggest challenge for building high-performance multimedia PCs is to design affordable products that offer professional-level 3D graphics with full-screen resolution in real time.

Direct3D Encourages ISVs

One of the enabling technologies for pervasive 3D graphics on PCs is Microsoft's Direct3D programming model. It establishes a basic 3D macroarchitecture that allows ISVs to develop their 3D titles without being tied to a specific hardware implementation. In addition, Direct3D allows hardware vendors to create chips that accelerate 3D applications from various title developers. Direct3D supports both Microsoft's own RealityLab API, acquired from Rendermorphics, as well as third-party interfaces, making it a truly open 3D software standard. One drawback is that Microsoft has only recently released the specifications for Direct3D, and final code is not planned for shipment until this spring.

Although Microsoft's flight simulator was written for PCs many years ago, until recently most 3D software titles were written for game consoles such as Sega and Nintendo boxes. Those titles that were ported to PCs perform better under DOS than under Windows 3.1 due to the lack of high-performance API support, interactivity, and real-time support in Windows 3.1. Before Windows 95, PC-based 3D APIs—such as RealityLab, Criterion RenderWare, and Argonaut BRender—were mainly DOS-based. On the workstation side, vendors have rallied around the OpenGL standard for CAD and other 3D applications.

Anticipating the coming boom in PC 3D applications, Microsoft plans to introduce the Direct3D API as part of a Windows 95 upgrade kit shipping in spring 1996. Direct3D, one of the most critical building blocks for forthcoming 3D PCs, is a 3D API under Windows 95. With the acquisition of Rendermorphics, Microsoft essentially defines the macro-architecture of PC 3D graphics using RealityLab as a high-level API and Direct3D as a low-level API, as illustrated in Figure 1.

In this five-layer model, the application layer allows independent software developers (ISVs) to create 3D graphics titles. The high-level API is for RealityLab or third-party APIs. The low-level API is Direct3D, which defines a standard interface for developers who need hardware access but still want hardware independence. The device-driver layer is the Direct3D hardware abstraction layer (HAL). It is hardware-specific, isolating a hardware device from the upper layers. The physical layer is the actual 3D hardware device. With Microsoft's 3D architecture, hardware vendors can develop their own innovative designs as long as they can provide Direct3D-compatible device drivers.

Direct3D allows third-party APIs to interface with mainstream hardware accelerators. The high-level API is the layer where third-party APIs can play a role, which allows existing applications and other third-party APIs to easily take advantage of hardware acceleration. This approach enables some differentiation in software rendering. Companies such as Criterion and Argonaut can continue to offer 3D title developers Windows 95–compatible APIs that deliver different features than RealityLab. This flexibility makes Direct3D a truly open standard.

The advantages to having an open standard are:

- New applications are immediately compatible with an existing and future hardware base.
- New hardware products are instantly compatible with an existing and future software base.
- Hardware and software solutions are both backward- and forward-compatible with changing standards.
- Applications written for compatible third-party APIs can be handled by either software or hardware rendering.

Direct3D is the layer for converging all the high-level APIs into a single platform. This way, a software developer can focus primarily on the dynamics of applications rather than putting effort toward creating and debugging a proprietary 3D engine. This is especially useful when porting an existing DOS-based 3D game to Windows 95. Modifying a small portion of a program not only allows the program to execute under Windows 95 but allows it to be accelerated by available hardware.

Direct3D can do everything in software if there is no hardware support available. Direct3D's built-in software emulation will perform 3D functions that are not accelerated by hardware.

It is worth pointing out that even with Direct3D, developing a device driver is still not easy. Most chip companies, such as 3Dlabs and 3Dfx, spend more resources to create support software than to complete their hardware designs. A good product needs to support popular operating systems like DOS, Windows 3.1, and Windows 95 and interface with key APIs such as DirectDraw, Direct3D, RealityLab, and RenderWare.

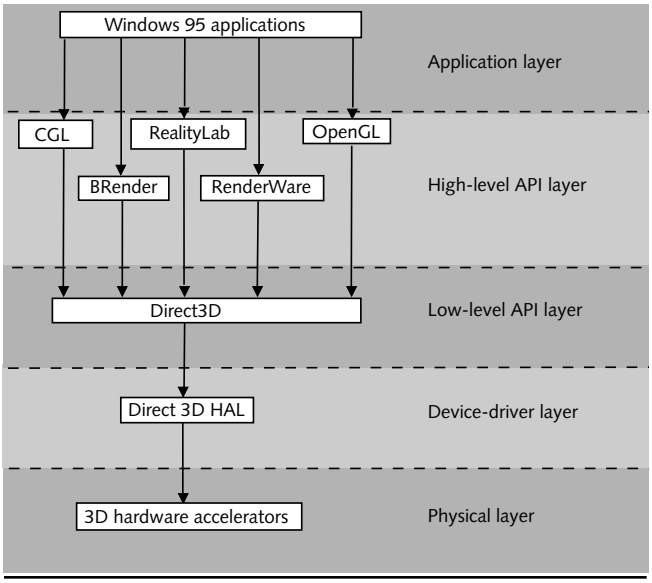

Figure 1. Microsoft 3D application programming interface and five-layer 3D graphics macroarchitecture. Microsoft's Direct3D works with a variety of high-level interfaces from Microsoft and from third parties.

3D Graphics Pipeline

Processing 3D graphics can be viewed as a three-stage pipeline: tessellation, geometry, and rendering. In the tessellation stage, a description of an object is created and the object is then converted to a set of polygons. The geometry stage includes transformation, lighting, and setup. The rendering stage, which is critical for 3D image quality, creates a two-dimensional display from the polygons created in the geometry stage.

Various techniques may be applied to the rendering stage to achieve photorealistic and interactive 3D graphics. Among them, flat and Gouraud shading, texture-mapping, Z- and double-buffering, and perspective correction are essential features for 3D accelerators in 1996. Advanced features include MIP mapping, texture filtering, anti-aliasing, subpixel correction, and special effects like fogging and depth cueing. We project that all these features will be available to PC OEMs in 2Q96.

Theoretically, 3D graphics can be handled entirely by software. But even a CPU with performance ten times better than that of a Pentium-133 might not be enough for professional-level 3D graphics or even high-quality game programs. Thus, for many years to come, 3D-graphics processing tasks will be shared between a host CPU and a dedicated hardware accelerator.

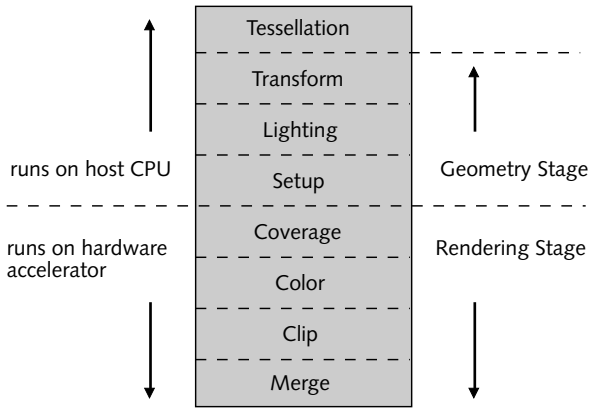

Figure 2. A logical way to divide 3D graphics processing tasks. The geometry stage involves steps two through four. The rendering stage involves the last four steps. Although current CPUs have enough power to handle 3D setup, it may be faster to execute it on a dedicated hardware accelerator.

There are different ways to divide tasks between a host CPU and a hardware accelerator. A logical way is to have the host CPU handle operations in both the tessellation stage and the geometry stage, as Figure 2 shows. The hardware accelerator then handles operations in the rendering stage. Therefore, the hardware accelerator can also be viewed as a rendering engine. The data streams in the tessellation and geometry stages have conventional data types, and the operations involve vector and matrix processing. A CPU with high floating-point performance is ideal for these tasks.

In the rendering stage, data is pixel-based. Rendering requires tremendous pixel-processing capabilities. The rendering engine must be able to process thousands of polygons per scene and construct a quality 2D representation in 1/30th of a second to produce real-time animation. This construction consists of many pixel reads and writes, which require a great deal of memory bandwidth. It makes sense for a dedicated rendering engine to deal with the pixel-related operations. Another advantage of sharing the tasks in this way is that while the hardware accelerator is producing the current frame, the host CPU is calculating the geometry for the next frame, maximizing performance.

Rasterization	points/lines/polygons/bitmaps
Anti-aliasing	4x4 or 8x8 subpixel sampling
Shading	flat/Gouraud shading
Fogging	per-pixel fogging
Texture mapping	map images on polygons
Alpha blending	add transparency
Z-buffering	compare against z-value/pixel write
Clipping	scissor clip/stenciling/stippling
Frame buffer ops	logic operations/dithering

Figure 3. The pipeline details of a typical rendering engine.

In the tessellation stage, a description of an object is created, and the object is converted to a set of polygons. Then, the geometry stage performs the following tasks:

- Transformation: Translating and rotating 3D objects, thereby positioning the viewer within the 3D space.
- Lighting: Determining lighting, shading, and texture characteristics of each polygon.
- Setup: Determining the shape of each polygon displayed.

Typically, the most compute-intensive task is setup, followed by transformation and then lighting. But the cost of lighting is very dependent on the lighting equation and can vary substantially. Setup depends heavily on the number of parameters (such as color, transparency, depth, etc.) iterated across a polygon as well as the rasterization algorithm and what inputs are required. Transformation is fairly predictable in all cases.

Consider a scene composed of 1,000 polygons for a 30-fps display. If these polygons are triangles, and half are meshed and half are independent, this results in 2,000 vertices that need to be transformed ($500 \times 1 + 500 \times 3$). Each vertex requires transformation through a 4×4 floating-point matrix, resulting in 16 multiplies and 12 additions, or 2,000 (vertices) $\times$ 28 (operations) $\times$ 30 (frames) = 1.7 MFLOPS. Doubling this figure to account for clip-testing and perspective adjustment yields 3.5 MFLOPS. Assuming that lighting and setup also average 3.5 MFLOPS each, the geometry stage needs a total of roughly 10 MFLOPS.

Thus, the geometry stage does not consume a significant portion of today's high-end CPU horsepower. A 90-MHz Pentium has a peak throughput of 90 MFLOPS and with good assembly code can sustain about a third of that, or 30 MFLOPS. Thus, according to our estimate, a 1,000-polygon scene requires only a third of the performance of a 90-MHz Pentium. A 166-MHz Pentium can handle perhaps 5,000 polygons per frame, although this would leave few, if any, cycles for other tasks.

It is better, however, to have a dedicated hardware accelerator handle the rest of the 3D processing, i.e., the rendering, which consists of:

- Coverage: Determining which pixels are covered
- Color: Determining the color of each pixel
- Clip: Determining which pixels are visible
- Merge: Writing pixels to the frame buffer

Figure 3 shows a rendering pipeline in more detail.

There are many techniques, such as texture mapping and perspective correction, that can be applied during the rendering stage. Other features, like atmospheric effects and anti-aliasing, can be used to further improve 3D realism.

Basic 3D Graphics Techniques

The features discussed in this section are mandatory for either software or hardware 3D implementation. They have

been recommended by Microsoft, 3Dlabs, and other industry leaders as features required by the lowest common denominator of games and APIs. These features also guarantee a minimum level of functionality to game developers.

Shading. Polygons can be rendered in one of many ways, such as flat shading, Gouraud shading, Phong shading, or texture mapping.

Flat shading assigns a uniform color throughout an entire polygon. This shading results in the poorest quality, creating images that look "blocky." Gouraud shading assigns color to every pixel within each polygon based on a linear interpolation from the polygon's vertices. This method removes the blocky look and provides an appearance of plastic or metal surfaces. Phong shading is based on a lighting equation at each pixel. It interpolates between the vertex normals to produce smooth shading, adding specular highlights to represent the effects of light sources.

Lacking computing power, most software 3D engines use the flat shading method. Most 3D hardware implementations, however, are based on Gouraud shading due to the intensive computing required by Phong shading. Texture mapping is used to more efficiently increase visual realism.

Texture mapping. Based on a stored bit-map consisting of texture pixels, or texels, texture mapping has the single biggest impact on bringing professional-level 3D graphics to the PC platform. It involves wrapping a texture image onto an object to create a realistic representation of the object in 3D space. As mentioned above, the object is represented by a set of polygons, usually triangles. Every vertex in each polygon references a UV address, called the texel address, as Figure 4 shows. A texture-mapping algorithm determines the texel addresses of the rest of the polygon.

The texture mapping of the single triangle requires determining the UV address associated with the pixels in the triangle. Once the host CPU computes the U and V values for each vertex, two other parameters, known as *du* and *dv* and called texture slopes, are also generated. As the triangle is being rendered, the *du* and *dv* parameters determine the rate of change of the texture address.

Let us consider the edge AB in the triangle shown in Figure 4. As X is increased by one pixel position, the U texture address is increased by *du*. When a sufficient number of *du* terms are accumulated, the address of the texel in the U direction will change. If the addition of a *du* or a *dv* does not cause the UV address to change, the same texel is mapped onto the triangle; if a change is made, a new texel is mapped onto the triangle. The *du* and *dv* values thus determine the particular set of texels that map to the edge of the triangle. Y and V are handled in a similar fashion. This processing is repeated for the other two edges. After that, the texture mapping determines texels on each horizontal span.

Point sampling is the simplest, lowest-cost, and lowest-quality method of applying textures onto an object. The rendering engine bases the texture on a one-to-one relationship between a texel on the rendered polygon and the texture map. The advantage is the speed of rendering, since only one texel read is required for each pixel being written to the frame buffer. The disadvantage is the blocky image that results when the object moves. More advanced texture-mapping algorithms include bilinear filtering and MIP mapping.

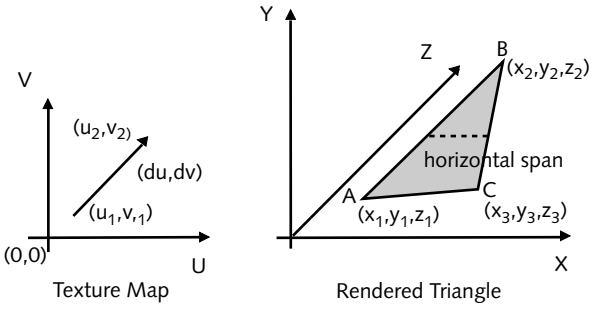

Figure 4. Basic texture mapping of a single triangle. The two-dimensional texture map is applied to the rendered triangle with a three-dimensional perspective, creating a more realistic representation of a 3D object.

Z-buffering. When objects are rendered into a 2D frame buffer, the rendering engine must remove hidden surfaces. The two common techniques used to accomplish this are Z-sorting and Z-buffering.

Z-sorting algorithms sort the polygons in back-to-front order prior to rendering. Thus, when the polygons are rendered, the forwardmost surfaces are rendered last. The rendering results are correct unless objects are close to or intersect each other.

Z-buffering algorithms store a depth value for every pixel in a buffer, known as the "Z-buffer." The Z-buffer has the same size as the main frame buffer. Before bringing in a new frame, the rendering engine clears the buffer, setting all Z values to "infinity." When rendering objects, the engine assigns a Z value to each pixel: the closer the pixel to the viewer, the smaller the Z value. When a new pixel is rendered, its depth is compared with the stored depth in the Z-buffer. The new pixel is written into the frame buffer only if its depth value is less than the stored one.

The number of bits used for the Z value is usually 16, 24, or 32 bits. An accuracy of 16 bits is good enough for most entertainment applications; 24-bit accuracy is adequate for the majority of professional applications. If the scene contains objects that are close together over a wide range, 32-bit accuracy is required.

Z-sorting algorithms have the advantage of not requiring RAM for storing depth values, but at the cost of accuracy. Z-buffering algorithms have the advantage of accuracy at the cost of RAM storage. Even with 16-bit accuracy, support of Z-buffering is very memory-bandwidth intensive. Both algorithms involve a great deal of computing.

Perspective correction. A particular way to do texture mapping, perspective mapping is extremely important for creating a realistic image. It takes into account the effect of the Z value in a scene while mapping texels onto the surface

3D Graphics Glossary

This glossary briefly defines acronyms, abbreviations, and terms frequently used in 3D literature. Several of these terms are discussed in more detail in the main text; these concepts are only summarized here.

Alpha-blending—Technique for adding transparency information for translucent objects.

Alpha buffer—An extra channel to hold transparency information; pixels become quad values (RGBA).

Anti-aliasing—Subpixel interpolation, which makes the edges appear to have better resolution.

Atmospheric effect—The result of adding one more layer of reality, such as fog and depth cueing, to an image.

Bilinear filtering—A method of anti-aliasing texture maps that averages four adjacent pixels.

Bilinear MIP mapping—A texture-mapping process combining bilinear filtering and MIP mapping (see below).

Blending—Combining two or more images by adding them on a pixel-by-pixel basis.

Decal texturing—The process of applying a texture to a surface without lighting.

Depth cueing—The lowering of intensity as objects move away from the viewpoint.

Direct3D—Microsoft's low-level primitive-based 3D API. It includes optional control for low-level transformations, lighting, and rasterization.

Dithering—A technique for achieving 24-bit quality in 8- or 16-bit frame buffers.

Double buffering—A method of using two buffers, one for display and the other for rendering. When the new frame is rendered, the two buffers are switched.

Fast-clear buffer—Used to quickly clear the frame buffer and the Z- buffer.

Flat shading—Rendering a polygon with a single color and uniform intensity.

Fog—Blending an object with a fixed color as its pixels increase in distance away from the viewer.

Frame buffer—Holds display pixels, either indexed or RGB triplets.

Geometry engine—The stage in the 3D pipeline for transformation and lighting.

Gouraud shading—Rendering a polygon through linear interpolation using a specified color at each vertex.

Lighted textures—A method of rendering that takes account of lighting source(s).

Meshed triangles—see Tristrips.

MIP mapping—*Multum In Parvum* (Latin) means "many in one." A method of increasing the quality of a texture map by applying different-resolution texture maps for different objects in the same image, depending on their size and depth.

of polygons. As a 3D object goes away from the viewer, the length and height of the object become compressed, making it appear shorter. This compression of the texture is achieved by a perspective algorithm. Its basic idea is that the rate of change of the UV texture address is proportional to the depth. Let a variable W be inversely proportional to the Z value. When the UV addresses are divided by W, the UV addresses will be changed in larger steps as polygons move away from the viewer. This results in a shrinking of the texture. Since it requires a division per pixel, perspective correction is very computing intensive.

Enhanced 3D Graphics Techniques

Features discussed in this section are for advanced high-performance implementations. Some of them are considered beyond mainstream PC applications in the near future.

MIP mapping. If a texture-mapped polygon is smaller than the texture image itself, the texture map will be undersampled during rasterization. As a result, the texture mapping will be noisy and "sparkly." The purpose of MIP mapping is to remove this effect.

This algorithm stores a number of different sizes of the texture map in memory, representing different resolutions. When a 3D object is large, due to its proximity to the viewer, a correspondingly large texture map is used. As the object moves away from the view point, the rendering engine switches to a smaller texture size.

Another texture-aliasing artifact occurs due to sampling on a finite pixel grid. Point-sampled texels jump from one pixel to another at random times. This aliasing is very noticeable on slowly rotating or moving polygons. The texture image jumps and shears along pixel boundaries. Bilinear filtering eliminates this problem.

Bilinear filtering takes a weighted average of four nearby texture pixels to create a single texel. This form of texture mapping typically requires four times the memory bandwidth of point-sampled texture mapping. Bilinear MIP mapping simply combines MIP mapping and bilinear texturing in which the four sampled texels are fetched from one of several versions of the original texture map.

Since the number of texture maps is limited, one artifact possible in a MIP-mapping approach occurs when an object is moving toward or away from the view point. As the object crosses a "MIP boundary," a change can sometimes be noticed when switching from one texture map to another.

Trilinear MIP mapping solves this problem by interpolating between MIP map levels. The texel value is computed by performing a bilinear filter on four texels each from the

3D Graphics Glossary (Cont.)

OpenGL—Silicon Graphics' high-level API, containing many useful graphics building blocks. It is designed for CAD, not games.

Perspective correction—The process of adding realistic convergence to three-dimensional objects as they move away from the viewpoint.

Phong shading—Rendering polygons based on a given lighting equation at each pixel.

Point-sampled texture mapping—Texture mapping based on a one-to-one relationship between texels on a rendered polygon and a texture map.

Rasterization—Translating an image into pixels.

RealityLab—Microsoft's high-level object-based 3D API, originated by Rendermorphics. It directly manipulates objects, lights, and cameras.

Rendering—The stage of the 3D pipeline that creates a 2D display from a set of 3D polygons fed from the geometry engine.

Scissor clip—Tests pixel coordinates against clip rectangles and rejects them if outside.

Stencil buffer—A screen-sized buffer holding a pixel mask.

Stenciling—Testing pixels against bit masks, rejecting them if masked.

Subpixel correction—A form of rendering based on subpixel interpolation.

Tessellation—The first stage of the 3D pipeline, which converts a description of an object into a set of three-dimensional polygons.

Texture filtering—Removing aliasing artifacts, such as sparkles and blockiness, through interpolation of stored texture images.

Texture mapping—Wrapping textures around objects to add realism or to reduce complexity.

Transformation—Translation and rotation in a 3D environment, positioning the viewer.

Trilinear MIP mapping—A method of reducing aliasing artifacts within texture maps by applying a bilinear filter on four texels from the two nearest MIP maps and then interpolating between the two.

Tristrips—A series of triangles where each triangle shares two vertices with at least one other triangle. It increases the rendering rate by eliminating the processing of two additional points per triangle.

Z-buffer—Holds the distance from the viewpoint for each pixel to ensure that only the nearest pixel is visible.

Z-buffering—A process of removing hidden surfaces using the depth value stored in the Z-buffer.

Z-sorting—A process of removing hidden surfaces by sorting polygons in a back-to-front order prior to rendering. Less accurate than Z-buffering.

two nearest MIP maps and then interpolating between the two to determine the output texel. This method typically requires twice the memory bandwidth of bilinear texture mapping and eight times the bandwidth of point mapping, as well as far more computational power.

Edge anti-aliasing. Rasterization of polygon edges results in "jaggies" due to sampling a geometric figure on a finite pixel grid. Edge anti-aliasing solves this problem by computing the percentage of a pixel that is covered by a polygon and then using this value to blend the color of the pixel with that polygon's color. The coverage can be computed in a variety of methods, including subpixel point-sampling (typically on a 4×4 or 8×8 grid) and area sampling.

Atmospheric effects. Effects such as fog and depth cueing improve the rendering of real-world environments. Fog blends an object with a fixed color as its pixels become further away from the view point. Depth cueing lowers the intensity as objects move away from the viewpoint.

Lighting effects. One lighting method is called lighted texture mapping. It takes place as the last step before pixels are written to the frame buffer. The texture is modulated with respect to the lighting source(s). Lighted texture mapping is a feature beyond Gouraud shading with an intensity ramp to achieve lighting effects.

3D Adds Complexity

There are additional 3D techniques that go beyond those described here, but this article outlines the basic techniques used by most hardware and software implementations expected this year. Just as the GUI model increased the complexity of both software and hardware compared with simple text-mapped displays, moving to 3D adds another layer of complexity. Yet with increasing transistor budgets and the availability of inexpensive DRAM and hard drives, the cost of this complexity is small.

The value perceived by the end user, however, will be significant. 3D graphics is one of the most important emerging technologies for PCs. Microsoft has provided an operating system, Windows 95, along with the Direct3D API and necessary development tools. Applications are being developed by various ISVs and will be widely available for PC users in 2H96.

Intel's high-end Pentium and Pentium Pro are capable of handling 3D graphics tessellation and geometry processing. More than a dozen vendors have, or will soon have, 3D graphics rendering engines. We expect 3D graphics will be the hottest thing for the 1996 Christmas season. ♦

Part 2 (see **100304.PDF***)will describe and compare several recently announced 3D graphics accelerators for PCs.*

