Technical Brief

Microsoft® DirectX® 8: Raising the Ante for Realism in Graphics

DirectX 8: Raising the Ante for Realism in Graphics

The pace of innovation in PC consumer graphics continues unabated. With technology licensed from NVIDIA, Microsoft® has delivered DirectX® 8, which is poised to exploit the upcoming new generation of programmable graphics hardware. Key aspects of the DirectX 8 3D graphics application programming interface (API) are covered in this paper in order to understand this dramatic step forward in 3D graphics technology.

Programmability

DirectX 8 boasts a large list of new features, but *programmability* rises to the top of the list as the most significant new feature. Programmability allows 3D graphics developers to create effects and add features to graphics processing units (GPUs) that no amount of fixed-function hardware could hope to achieve. Previous versions of DirectX were fixed-function pipelines, meaning developers did not have the freedom to program their hardware and had to rely on static functions in the API.

Now, DirectX 8 introduces the era of programmable graphics hardware at the consumer level. Past efforts have been highly proprietary and not hardware independent. DirectX 8 now delivers standardized programmability by defining an API to program the hardware. NVIDA's GeForce3™ GPU is the most complete DirectX8 GPU available to consumer and developers, and will deliver cinematic effects to PC consumer graphics.

Programmability in graphics hardware isn't intended to replace a general-purpose CPU. Instead, the programmer works with code that specifically performs calculations on pixel or vertex data. The result that's returned is the actual value of the pixel or vertex data. The instruction set built into DirectX 8 is a very low-level language specifically architected for 3D graphics functionality.

The programming capability of DirectX 8 is split into two parts. The first operates on *vertex* (geometry) data and is referred to as programmable Vertex Shaders. The second aspect of programmability—programmable Pixel Shaders—affects the per-pixel level, allowing sophisticated custom effects to be applied to individual pixels. The next sections take a look at each of these and discuss their significance.

Programmable Geometry

A vertex is just the point where two lines meet—the corner of a triangle, for example. A single triangle has three vertices. Most geometry in 3D graphics eventually boils down to large masses of triangles linked together to construct realistic looking shapes. However, these objects and shapes aren't necessarily made up of discrete triangles. Instead, they're grouped together to make triangle strips or triangle fans. Using triangle strips or fans means that many triangles share vertices, which reduces the amount of data that the 3D graphics engine needs to worry about.

The triangle strip consists of 8 triangles. Normally, eight triangles would have 24 vertices, but the strip only has 10. Similarly, the fan has four triangles, but only six vertices

Vertex programs are short pieces of code (usually no more than 128 lines) that operate on individual vertices or on groups of vertices. A variety of effects, described on the following page, can be achieved with vertex operations.

Note the smooth, rounded joints on this character. When animated using matrix palette skinning, the joints move realistically without tears or breaks.

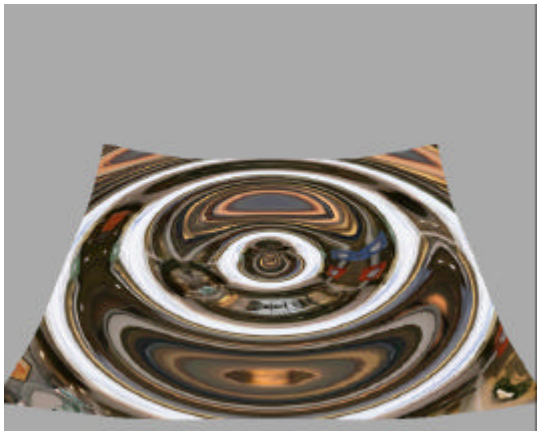
Matrix Palette Skinning

This technique lets programmers create realistic character animation. If you've ever seen characters in games move their joints, you may have noticed that the joints don't move smoothly—the motion appears robot-like and jerky. In addition, the math involved with creating the skeletal animation to bend tends to create unsightly gaps between the bones. Using vertex shaders, 3D programmers can create character animation with eight to ten "bones" per joint. This allows joints to move and flex convincingly.

The ability to create realistic characters has been something of a holy grail for 3D content developers. With matrix palette skinning and GeForce3, characters can now move and bend in a life-like manner.

Deformation of Surfaces

In the real world, not all surfaces are perfectly smooth and flat. For example, water is not always perfectly flat and calm. There are deformation, ripples and waves in the water that previously have been difficult to model in real time on consumer graphics.



This is a simple sine wave being applied to a flat surface, but more elaborate deformation effects can be created easily using DirectX 8 Vertex Shaders.

With a DirectX 8 Vertex Shader, developers can now model dynamic changes to their models to create the look of ripples and waves on water. These techniques would not be possible without GeForce3 and DirectX 8 enabling these features.

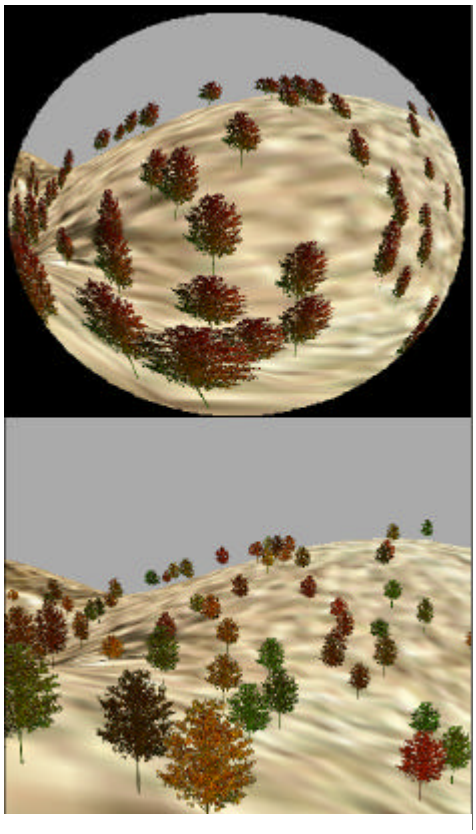
Vertex Morphing

The bending and twisting of a swimming dolphin can be created with vertex shader programs.

Vertex programs are used to morph triangle meshes from one shape to another. Consider the example of a swimming dolphin. The dolphin bends and twists to propel itself through the water. A Vertex Shader that works on the dolphin's skeleton results in a smooth animation.

Fisheye Lens

Another use for Vertex Shaders is the generation of specialized effects in a scene. For example, Vertex Shaders can create a fisheye lens effect. The Vertex Shader used for the images on the left creates a custom transformation matrix. This custom transform isn't possible with the fixed-geometry pipeline of previous versions of DirectX. Although this example shows a fisheye lens, the variety of possible effects is limited only by the imagination of the developer.



The bottom shows the undistorted image; the top shows the image through a fisheye lens.

Large Numbers of Vertex Lights

Previous NVIDIA GPUs only supported 8 hardware lights per vertex. There was no way to use more hardware lights in a single pass, due to the fixed nature of the DirectX geometry pipeline. GeForce3 GPUs are capable of using vertex programs that can theoretically use any number of lights in a single pass, giving developers the flexibility to create from their imagination instead of being limited by the API.

The lighting for this teapot includes 17 sources generated with a vertex shader.

Using Vertex Shaders with Per-Pixel Effects

Vertex Shaders can work in conjunction with Pixel Shaders, usually setting up the geometry for the pixel effects. The geometry calculations necessary for setting up key per-pixel effects can be performed on the GPU with a vertex program. Let's take a look at some examples.

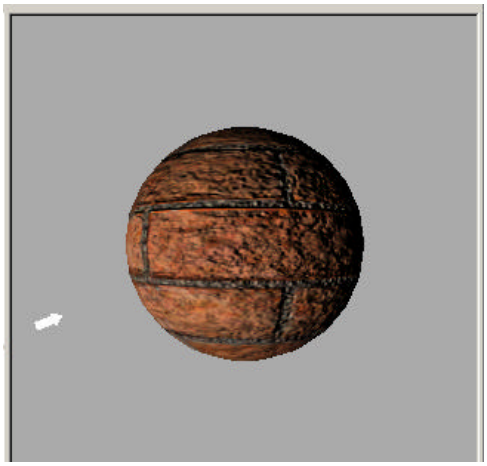
A cube environment map is applied to this teapot. The texture coordinates needed to apply the cube map to the teapot were generated with a Vertex Shader.

Cube Environment Mapping

One simple use for a vertex shader is to generate the texture coordinates for cube environment mapping, offloading this task from the CPU. The net result is more accurate than standard environment mapping, and better performance since the calculations are done on the GPU.

Setup for Dot-Product Bump Mapping

Dot-product bump mapping is potentially one of the most significant improvements in visual quality. This type of bump mapping was first available from NVIDIA on the NVIDIA GeForce™ 256, and was part of DirectX 7. However, setting up the geometry for dot-product bump mapping had to be performed on the CPU with DirectX 7 and the necessary calculations were computationally intensive.



This sphere has low geometric complexity, but dot product bump mapping makes it look realistic and complex. Dot product bump mapping also ensures realism with varying light angles.

Now with GeForce3 and DirectX 8, Vertex Shaders bring these calculations onto the GPU, where they really belong. The powerful processing capabilities of this GPU will enable widespread use of dot product bump mapping on a broad array of systems, creating more realistic, rough surfaces instead of smooth models that do not look real.

Layered Fog

Vertex Shaders can create realistic fog and smoke effects that stay low to a surface. Whether it's a fog bank moving over a hilltop or a layer of smoke in a room, fog effects add another level of realism.

Here's an example of a realistic-looking layer of smoke in a complex environment.

A vertex program can be written to create texture coordinates based on the geometry of the scene, with all the work being performed on the GeForce3 GPU. The net result is a realistic layer of fog or smoke that sits on the ground, without sacrificing performance.

Per-Pixel Reflection

When setting up an accurate reflection on a bump-mapped surface, a fairly complex set of calculations must be performed. In the past, this would have been done on the CPU, but performance was abysmal. This made per-pixel reflections impractical for real-time applications. By combining a series of Vertex Shaders on a GeForce3 GPU with powerful texture mapping techniques, visually stunning and complex effects can be created without sacrificing performance.

Here's a bump-mapped surface with reflections.

Programmable Pixel Shaders

Along with programmable Vertex Shaders, DirectX 8 enables programmable *Pixel Shaders*.

What is a Pixel Shader? Remember that the end result of computer graphics is to display an image. That image consists of a large number of pixels, each of which must have a color. The result of all the calculations to transform, light, and texture map an object is to create a set of pixels that, when displayed on a computer screen, looks correct to your eye. A Pixel Shader generates a color from texture coordinate information—the locations within a texture map that's been applied to a polygon. This color includes not only base color information, but also transparency and blending information. The output color is the color of the pixel as it finally appears on the screen.

DirectX 8 programmable Pixel Shaders and GeForce3 GPUs allow programmers to implement all the per-pixel effects of the fixed pipeline, with better performance. But more importantly, programmers can now create a virtually unlimited set of custom lighting and texture effects for their applications. Programmability gives developers tremendous flexibility to create their own methods for shading objects. The new effects developers can create will have material properties that will look and feel realistic.

The following paragraphs discuss some examples of this capability.

Using programmable Pixel Shaders, it's now possible to implement shiny bump maps that look correct from all angles and do not distort as the angle of lighting varies.

Shiny Bumps

In the previous section, it was explained how developers can use Vertex Shaders to set up the geometry for shiny bump maps (per-pixel reflection). One method for generating these effects is environment-mapped bump mapping (EMBM), a feature introduced in DirectX 6. However, EMBM did not work correctly for many angles of light, and had limited use. There are now several ways to generate an accurate, shiny bump-mapped surface. In DirectX 7, the idea of register combiners and dot3 bump mapping was implemented. Dot-product bump mapping looks correct at all light angles. However, setup for dot3 was done on the CPU and as a result performance issues limited widespread use. NVIDIA's fully DirectX 8 compliant GeForce3 enables the full setup for dot3 bump mapping on the GPU. This will allow shiny, bumpy effects with outstanding performance.

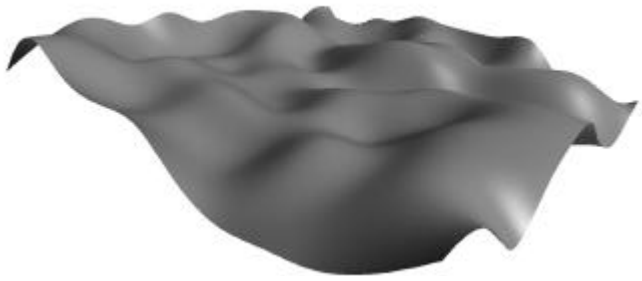
Dependent Texture Reads

Dependent texture reads have been available to developers as early as DirectX 6, but only with EMBM. The Pixel Shader interface of DirectX 8 now has a generic capability for dependent texture reads. Hardware that supports the DirectX 8 Pixel Shader interface, like the GeForce3, can now use dependent texture effects, such as EMBM. Other effects are possible, such as anisotropic lighting, used for effects like iridescence of shiny, grooved surfaces.

Higher-Order Surfaces

Creating 3D graphics using triangles as the base primitive is a difficult task since it's as if an artist was given a pile of small paper shapes and asked to create a great sculpture out of them. At the lowest level, triangles are fine, especially if you have enough of them and have a lot of pixel shading horsepower to make them look smooth and unfaceted. But developers want more intuitive tools for creating 3D graphics. One such tool is using *higher-order surfaces*.

GeForce3 and DirectX 8 enable higher-order surfaces, allowing developers to create objects by defining curves based on specified control points. A curve or surface defined with a set of control point is called a *spline*. While there are a number of different types of splines, all splines allow developers to use a few control points to create fairly complex, smooth curved surfaces. Joining splines together allows the formation of complex curved surfaces that are difficult to create with triangles.

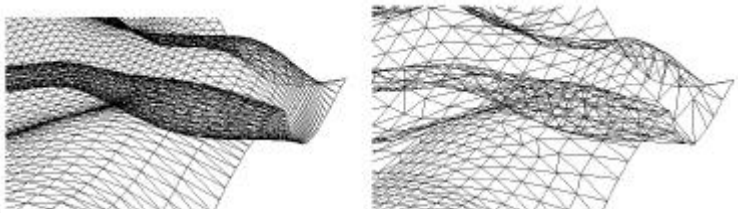


Complex object created using higher order surfaces.

The DirectX API supports two types of curved surfaces: polynomial surfaces and Npatches. In each case, the final object is *tessellated* before being rendered. This means that the curved surfaces are converted to a triangle mesh, but beyond that, the two techniques are quite different.

Npatches versus Polynomial Surfaces

Npatches have some advantages for content creators. They can be used in an existing 3D engine without making significant changes to the engine. Developers won't necessarily have to alter the way they work. However, Npatches do have some significant problems:



The left image uses fixed tessellation (like Npatches), the right image uses adaptive tessellation (polynomial surfaces). The triangles in the example on the right are all roughly the same size throughout the mesh. This means that the triangle density in the foreground is higher on the right-hand image, but the density drops off in the distance – where it's not needed anyway.

- Npatches don't support adaptive tessellation. For a large terrain mesh, the near part of the mesh usually consists of many triangles for a more accurate tessellation and a better rendering. The distant part can use fewer triangles.
- Image quality can suffer from cracks between polygons and lighting artifacts from inconsistent representation.
- Npatches are non-standard. Hardware support for Npatches will be very limited and there are no known content creation tools to support the easy creation of Npatch-based artwork.
- Collision detection can be difficult.

?? No natural evolution of the content to superior capabilities later.

Npatches are more of a nod to existing game engines, letting developers add curved surfaces without having to alter their engines significantly. Polynomial surfaces are a more mature, forward-looking technology that puts significantly more power and flexibility into the hands of the applications developers. Advantages of polynomial surfaces include:

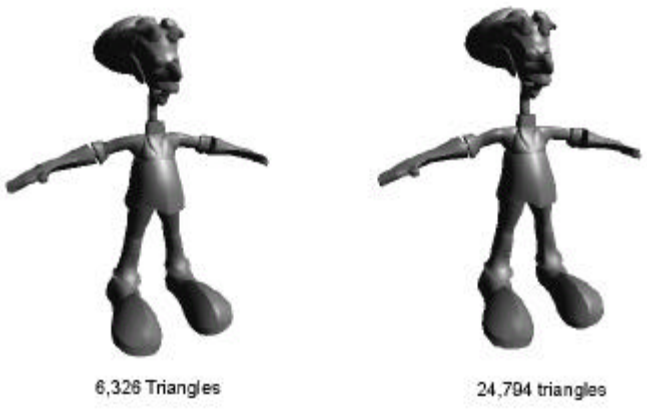
?? Adaptive tessellation (see above). This can considerably reduce the impact on overall triangle budget without sacrificing image quality.

?? Better image quality (no polygon cracking or lighting defects).

?? Direct support for continuous level of detail.

?? Easy migration to future technologies, such as subdivision surfaces.

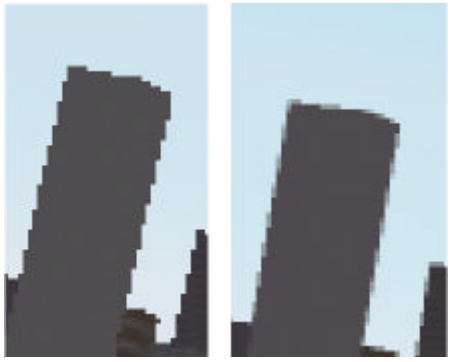
?? Better character models due to adaptive tessellation (see figure below).



Both characters look equally good, but the model on the left has far fewer triangles, and uses adaptive tessellation. Adaptive tessellation allows the modeler to put details (via more patches) where needed (nose, cheeks, mouth) and use bigger patches where there is less detail (the torso).

Multisample Rendering (Antialiasing, Depth of Field, and More)

Multisampling is a technique where the same pixel data is rendered into multiple locations offset by a small amount (less than a pixel in size). The number of times this *subpixel rendering* occurs is variable. More samples generate better image quality.



Antialiasing blends colors of adjacent pixels to minimize "the jaggies".

Multisampling has a number of different uses. Aliasing results from sampling each pixel at a single point, and assigning the pixel the color of one surface, even if other surfaces are partially visible in the pixel. This results in the stair-step or "jaggy" artifacts familiar in rendered images. With multisampling, the scene is sampled at multiple sample points per pixel. If more than one surface is visible, the color of the various surfaces will be blended proportionately, eliminating jaggy or crawling artifacts. GeForce3 supports advanced antialiasing techniques that for the first time allow for high quality, high-resolution antialiasing, without sacrificing performance.

Multisampling has other uses, too, such as depth-of-field effects. The depth-of-field concept is common to camera lenses, for which the near field may be sharply in focus while objects in the distance are blurry (or vice versa). Depth of field is a great tool for focusing the eye on the particular part of a scene that's in focus, and reinforces the intent of the designer.

Another use for multisampling is motion blur, which occurs when a moving object is slightly blurred along its axis of motion. One of the reasons that film looks so good at 24 frames per second is the natural motion blur that occurs in each frame.

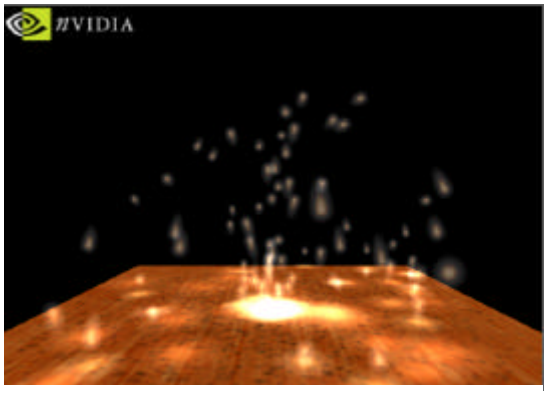
There are two uses for motion blur. One is to create flashy effects, such as demonstrating that a creature or vehicle is moving extremely fast. Another is to create the impression of smooth animation, even with a relatively slow frame rate, analogous to what occurs with film.

GeForce3 supports all DirectX 8 multi-sampling techniques, bringing cinematic effects to consumer level graphics hardware.

Point Sprites (Particles)

Swirling smoke, falling snow, sparks off a welder, and other similar effects have been traditionally difficult to render and performance intensive on PC systems. The effects have also lacked realism.

The technology required to create smoke and similar effects involves particle systems. Particle systems manage the behavior of numerous, small objects which DirectX calls *point sprites.* GeForce3 supports DirectX 8 point sprites effects that look superb and



Particle systems for generating showers of sparks are easy to do with DirectX 8 point sprites.

realistic.

Compliance versus Compatibility

DirectX 8 is a major advance in 3D technology for the consumer market. Because it's such a radical departure from historical methods, the issue of DirectX 8 compatibility versus DirectX 8 *compliance* becomes key.

Almost any current generation 3D graphics hardware can be DirectX 8 *compatible* once the company ships a DirectX 8 driver. An updated driver on current generation hardware could support particle effects, faster runtime processing and accelerated index buffers (for faster vertex processing).

However, to be DirectX 8 *compliant*, the 3D hardware must support the programmable features of DirectX 8. There are two levels of compliance: 1.0 and 1.1. Programmable Pixel Shaders come in two possible versions: 1.0 and 1.1. The key difference is that version 1.1 compliance requires the hardware to support eight blending operations in a single pass. Programmable Vertex Shaders come in *three* versions: version 0, version 1.0 and version 1.1. Version 0 refers to DirectX 7 hardware; the programmable Vertex Shader operations would be actually performed on the CPU, with a performance penalty. Versions 1.0 and 1.1 specify hardware-accelerated Vertex Shaders. However, version 1.1 adds an address index that makes matrix palette skinning much more viable.

For the most extensive DirectX 8 compliance, the hardware needs to support version 1.1 Pixel and Vertex Shaders. GeForce3 is the only DirectX 8 1.1 compliant GPU available. It is the best solution for developers and consumers looking for real-time cinematic effects for PC consumer graphics.

NVIDIA's Role in DirectX 8

NVIDIA played a key role in the development of DirectX 8. Consulting closely with Microsoft, NVIDIA's world-class team of 3D architects contributed heavily to the development of this groundbreaking technology. Microsoft licensed key pieces of NVIDIA-developed technology for DirectX 8, specifically, significant portions of the DirectX 8 programmable Pixel and Vertex Shaders, as well as the implementation of polynomial surfaces.

Conclusion

DirectX 8 is a watershed in computer graphics. It puts an unprecedented level of graphics programming power directly into the hands of the applications developer. The power of programmable shaders will yield new effects and cool applications undreamed of before today.

NVIDIA is proud to have worked closely with Microsoft on the creation of DirectX 8. This successful collaboration between two industry-leading companies has brought DirectX 8 into existence. Because of this close cooperation, NVIDIA's GeForce3 GPU will be the most robust implementation of DirectX 8 available. Once again, NVIDIA is poised to reshape what people think of when they hear the term "3D graphics."