

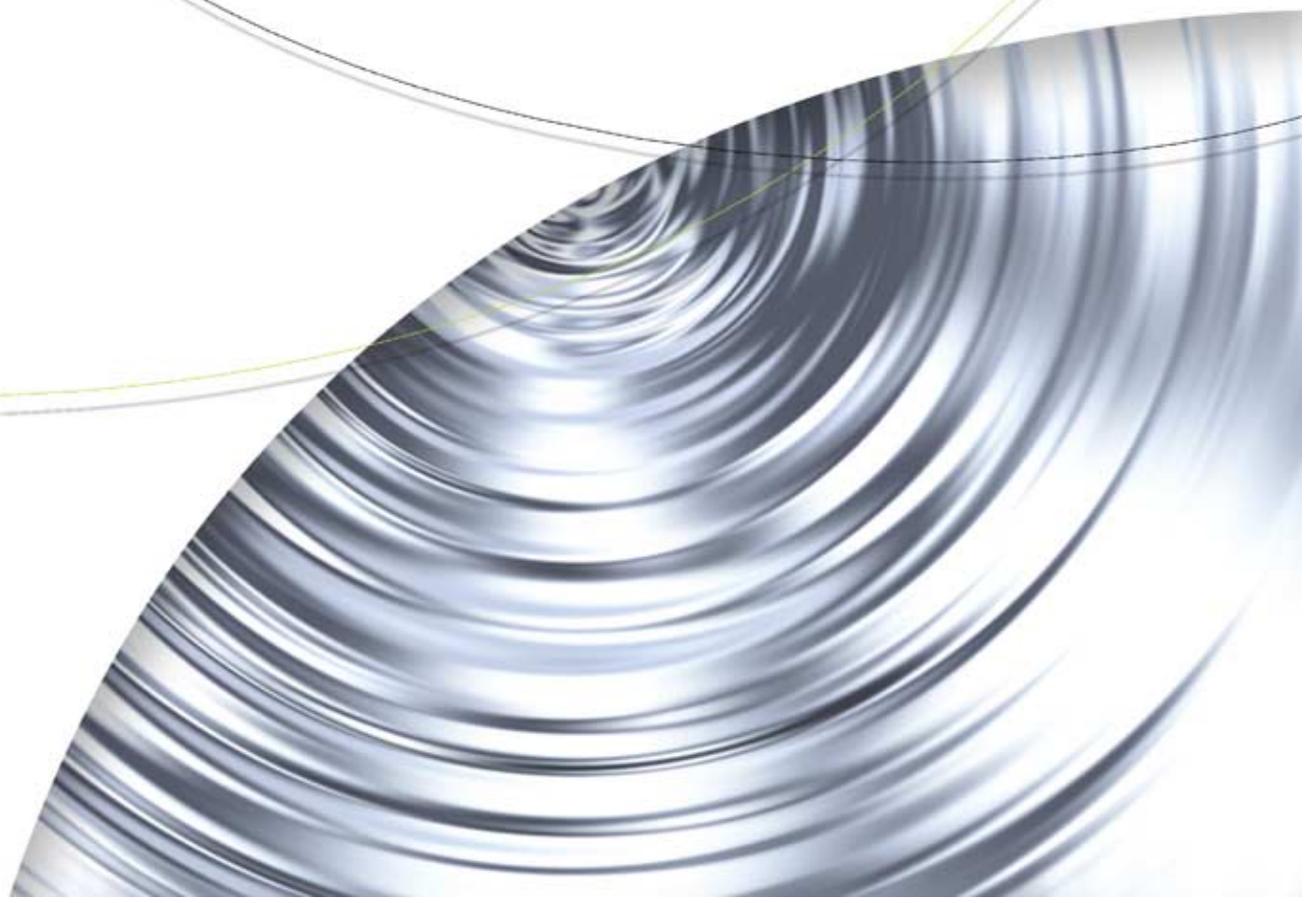


**NVIDIA.**

## Technical Brief

### **NVIDIA Lightspeed Memory Architecture II**

Breakthrough Memory Design for  
Industry-Leading Performance



The top of the page features the NVIDIA logo in a large, light blue, semi-transparent font. To the left of the logo is a circular graphic with concentric, wavy lines, resembling a lens flare or a stylized eye. The background is a light, textured blue.

# GeForce4 LMA II Memory Performance

Creating real-time, lifelike 3D environments on desktop PCs is the driving ambition of thousands of hardware and software developers in the computer graphics industry. While developments over the past years have made incredible strides towards improving the quality of real-time 3D graphics, one of the fundamental challenges in delivering interactive 3D content remains: “How to increase graphics performance given the limited increases in memory speed?”

Memory bus bandwidth remains a critical factor in determining graphics performance and quality. The NVIDIA® Lightspeed Memory Architecture II (LMA II) incorporates a number of revolutionary advances in graphics architecture. These advances dramatically improve the memory bandwidth efficiency of the GeForce4 graphics processing units (GPUs), enabling them to deliver new levels of performance and image quality.

**Note:** Understanding the demands that pixel rendering places on the memory interface is key to understanding the various challenges of rendering realistic 3D worlds. Refer to the Glossary for a list of useful terms.

---

## Memory Bandwidth Challenges

Limited memory bandwidth is a huge barrier to increasing PC graphics performance. The memory interface gets inundated with multiple, continuous, high-bandwidth demands such as pixel writes, pixel reads, display refresh, AGP bus transactions, and texture reads. Unfortunately, end users notice a slowdown in graphics performance when one of their multiple demands gets bottlenecked by the memory interface.

## The Pixel Fill Rate and Memory Bandwidth Challenge

Traditional graphics architectures render pixels by reading to and from various buffers in the graphics memory. These buffers hold color data that represent the alpha, red, green, and blue (ARGB) components of the pixel color, and hold the Z-values that represent the depth or “visibility” values of the pixel too. Basic rendering involves calculating a color value for a pixel, reading the color and Z-data stored in the color and z-buffers for that pixel location, and calculating the new color and new Z-value for that pixel location—including blending and combining with any previous color values, as necessary. This calculation is affected by the Z-value and by the alpha values. Once the new color and Z-values are calculated, the new data must be written to the memory. For 32-bit color and 32-bit Z/stencil buffers, these transactions represent a significant amount of memory bus traffic per pixel:

In fact, this “basic” level of rendering represents 16 bytes of bandwidth per pixel:

Read 32-bit color + read 32-bit Z = 8 bytes,  
Write 32-bit color + write 32-bit Z = 8 bytes, or 16 bytes total

Texturing and depth complexity magnify the bandwidth required per rendered pixel by about 200 percent. Most games today use multitexturing (applying two or more textures to a single pixel) in combination with bilinear or trilinear texture filtering to achieve rich image detail and greater realism. This texture data must be read from the same graphics memory that is used to store the Z-buffer and color buffer. As an approximation of texture-read bandwidth, assume that each pixel has two textures (multitexturing) with trilinear filtering. Trilinear filtering uses 8 texture samples per texture per pixel and each texture sample is 32 bits (4 bytes) of data. Of course, this means that 32 bytes of data must be read per texture or 64 bytes of data per pixel for a two-texture scenario. Because all GPUs offer some level of texture caching, assume that half of the texture data, on average, must be transferred across the external memory interface. (This oversimplifies a complex, dynamic problem, but is a reasonable assumption to make.) In a high quality scenario with tri-linear filtering, an average of 32 bytes of information must be read from graphics memory for every pixel rendered.

Depth complexity multiplies the impact of all of the bandwidth demands described previously because, in a typical rendering system, each pixel is rendered several times because some objects are in front of others. Unfortunately, the GPU doesn’t know that a pixel is hidden until it has already done the work to render the hidden pixel, which wastes valuable bandwidth.

So, the new bandwidth demands for a 1280 x1024 x32bpp display mode with an application that has an average depth complexity of 2.5 and an antialiasing setting of 2 AA samples/pixel is:

Horiz. Res x Vert. Res x (Basic Rendering + Texture Reads) x Depth x AA factor =  
1280pixel/line x 1024 lines/frame x (16bytes/pixel+ 32bytes/pixel) x 2.5 x 2 =  
**315 million bytes per frame.**

If we render at 60fps, this results in a memory bandwidth demand of 315MB/frame x 60fps which equals **18.9GB/sec. bandwidth for pixel rendering alone!**

This demand becomes even larger if we consider 4x antialiasing or higher resolutions such as 1600x1200. Specifically, a 1600x1200 resolution with 4x AA requires 921 million bytes per frame or a mind-numbing 55.3GB/sec. of memory bandwidth.

However, even the fastest graphics memory interfaces today only provide about 10.3GB/sec. of peak bandwidth. GPUs today are consuming virtually all of the peak bandwidth for pixel rendering alone. This doesn't even include going to higher screen resolution, using the maximum antialiasing settings, and performing a display refresh or other demands, such as AGP transactions. This level of bandwidth is needed on a sustained basis, not just as a theoretical peak. Clearly, memory bandwidth is a limiter of performance, so anything the GPU can do to get more work done with a fixed amount of bandwidth will result in a better experience for the end user ... along with richer graphics and faster frame rates.

---

## NVIDIA Lightspeed Memory Architecture II Memory Bandwidth Breakthrough

The NVIDIA Lightspeed Memory Architecture II implements many patent-pending technologies to improve the efficiency with which GPUs render pixels. These technologies include a crossbar-based memory controller; Quad Cache memory caching subsystem; lossless Z-buffer compression; a visibility subsystem; fast clearing of Z-buffers; and auto pre-charge.

### Crossbar Memory Controller

Because 3D graphics are dependent upon memory bandwidth, the memory controller is the crux of the bottleneck for improving performance. Aside from the GPU, the most significant component of a graphics system is the local graphics memory. The local memory holds all the various buffers: the frame buffer, Z/stencil buffer, video buffers, texture buffers, geometry buffers, and more. Because it has so many uses and demands placed on it, the local graphics memory is typically the highest bandwidth memory system in a PC. Hence, it is critical to make the most efficient use of the resources available.

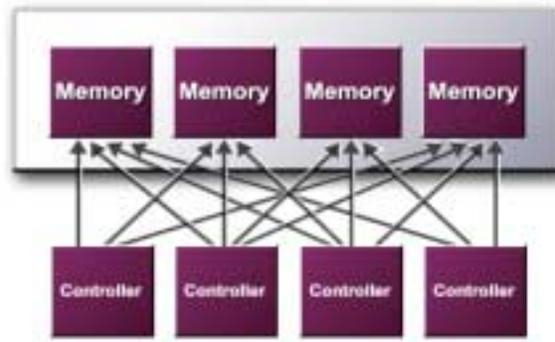
Traditional memory controllers have reached the point where they are reasonably efficient with basic loads, getting more than 50 percent of the peak memory bandwidth from the frame buffer under most conditions. In today's double data rate (DDR)-based designs, a typical 128-bit memory controller will actually access information in 256-bit "chunks" (because DDR transfers twice the information in a single access).

While it would seem that transferring large amounts of data in large blocks is generally optimal, in fact—for complex scenes with hundreds of thousands of polygons per frame—the reality is actually quite different. For leading-edge 3D applications, the size of the average triangle (the fundamental building block of all real-time graphics) can be very small—sometimes only a few pixels. If a triangle is perhaps two pixels in size, and is composed of 32 bits of color or Z/stencil data for each pixel, the total amount of data for that triangle would be 32 bits x 2 pixels, or 64 bits.

**Note:** Pixels typically have both color and Z/stencil data, but these are stored in different areas of memory. Thus, the color read would be 64 bits or a Z/stencil read would be 64 bits for the 2-pixel triangle in this scenario, rather than a single 128-bit read operation.

If memory controllers access information only in 256-bit “chunks,” then much of this access would be wasted, as this “payload” or amount of data being transferred would essentially waste much of the frame buffer’s potential bandwidth. In this example, a traditional 128-bit memory controller would be only 25 percent efficient, “wasting” 75 percent of the memory bandwidth.

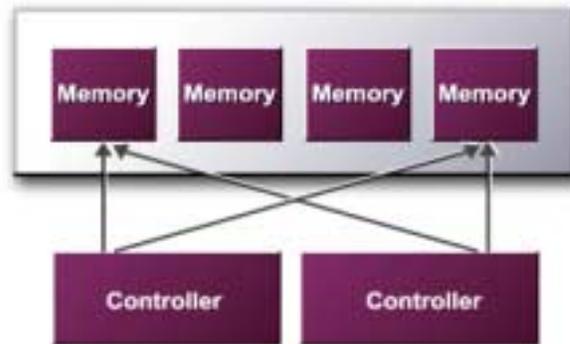
The GeForce4 GPU implements a radical crossbar memory controller (see Figure 1) that is optimized for accessing the frame buffer with a fine granularity access pattern. This ensures that each individual access is highly efficient. The GeForce4 Ti series has four independent controllers for the absolute highest performance possible. The benefit of the GeForce4 Ti’s four memory controllers is a 4x improvement in bandwidth efficiency.



The GeForce4 Ti memory crossbar interface with four independent controllers delivers four times the memory bandwidth efficiency of standard architectures

Figure 1. **The GeForce4 Ti Memory Crossbar Interface**

The GeForce4 MX GPU Crossbar Memory Interface incorporates a more economical version, using two independent controllers (see Figure 2). This version still delivers huge benefits by being up to 2x more efficient in transferring data as a non-crossbar implementation.



The GeForce4 MX memory crossbar interface with two independent controllers delivers twice the memory bandwidth efficiency of standard architectures

Figure 2. **The GeForce4 MX Memory Crossbar Interface**

The controllers of both memory crossbar implementations can also work together. In this case, the GPU is still capable of accessing 256 bits of information in an individual clock cycle.

The advantage of the crossbar memory interface is the flexibility to access information in 64-bit chunks, 128-bit chunks, or 256-bit chunks—whichever access pattern would be most efficient at that instant in time. This makes the efficiency of each of those accesses nearly perfect, keeping all aspects of the graphics processor and its frame buffer fully utilized for maximum performance.

This complex system load-balances continuously to ensure that every aspect of the memory system is balanced and that all memory requests by the graphics processor are handled properly. Under complex loads, typical of next-generation content, the LMA II memory crossbar architecture really shines, delivering two to four times the memory bandwidth efficiency of other standard architectures.

## Quad Cache

The LMA II also includes cache subsystems that are collectively known as Quad Cache. Caches are high-speed access buffers that store small amounts of data and operate at tremendously high bandwidth. These special buffers ensure that data is queued and ready to be written to the memory, and they provide a place to buffer data being read from the memory.

Quad Cache has four independent buffers optimized to improve the overall graphics pipeline performance. Individual caches store primitive, vertex, texture, and pixel information. These caches are individually optimized for the specific information they deal with. Results or data that have been previously stored in these caches can be retrieved almost instantaneously (as opposed to having to retrieve them from memory or recalculate them). The result is high-speed retrieval of key data that enables maximum performance from the graphics pipeline.

**Example:** In the pixel fill rate and memory bandwidth section, it was shown that for a tri-linearly filtered scene with two textures per pixel the calculation, based on 1280 x 1024 resolution and 16-byte read/write cycle, would be:

$$1280 \text{ pixel/line} \times 1024 \text{ lines/frame} \times (16 \text{ bytes/pixel} + 32 \text{ bytes/pixel}) \times 2.5 \times 2 = 315 \text{ million bytes per frame.}$$

The Quad Cache texture cache alone reduces the number of bytes for textures per pixel from 32 to 8! This brings the bandwidth requirement per frame from 315 million to 157 million bytes per frame ( $1280 \times 1024 \times (16 + 8) \times 2.5 \times 2 = 157 \text{ million}$ ).

Quad Cache reduces overall memory bandwidth used, allowing more beautiful and complex scenes to be rendered.

## Lossless Z Compression

The Z-buffer represents the depth or visibility information for the pixels that ultimately will be displayed after being rendered. Traditional GPUs read and potentially write Z-data for every pixel they render, making Z-buffer traffic one of the largest “consumers” of memory bandwidth in a graphics system. Through an advanced proprietary form of lossless data compression with a 4:1 compression ratio, the memory bandwidth consumed by Z-buffer traffic can be reduced by a factor of four. This Z-compression is implemented in hardware and is completely transparent to applications, with both compression and decompression taking place in real time by the LMA-II’s Z-compression/decompression engines. Because this compression is completely lossless, there is no reduction in image quality or loss of precision. The result of this technology is a dramatically more efficient memory subsystem that delivers higher performance with no compromise in image quality.

## Visibility Subsystem: Z-Occlusion Culling

As previously discussed, traditional graphics architectures render every pixel of every triangle as it receives them, accessing the frame buffer with each pixel to determine the appropriate values for the color and Z (or depth) for each of those pixels. This method produces correct results, but requires all the pixels to be rendered, regardless of whether they are visible or not. Typical content today has an average depth complexity of two, which means that two pixels must be rendered for every pixel that ends up getting displayed. This means that for every visible pixel, the graphics processor is forced to do twice the work, as the GPU must access the frame buffer twice, spending valuable frame buffer bandwidth rendering pixels that will not be seen.

LMA II implements a sophisticated Z-occlusion culling technology that determines if a pixel is going to be visible. If the Z-occlusion culling unit determines that the pixel will not be seen, the pixel is not rendered, the frame buffer is not accessed, and the frame buffer bandwidth is saved. Depending on the depth complexity of the scene, this can mean tremendous improvements in efficiency. With today's content averaging a depth complexity of two, this technique could reduce bandwidth requirements by up to 50 percent. With next-generation content approaching depth complexities of four or more, the benefits can be tremendous, with up to a 4x improvement in memory bandwidth efficiency.

An additional technique that can be employed is an "occlusion query." Essentially, the application makes a request of the graphics processor to render a bounding box or region to test for visibility. If the GPU determines that the region is going to be occluded, all the representative geometry and rendering for that region can be skipped over, potentially offering an order of magnitude increase in fill rate. Characters behind walls, or scenery outside of a tunnel, can simply be "occlusion queried" and skipped over, without spending precious memory bandwidth or GPU processing time to render them.

Z-occlusion and occlusion query are the two key technologies that effectively amplify the bandwidth of a GPU. These technologies dramatically increase efficiency for the memory bandwidth offered by the frame buffer, and make more efficient use of the frame buffer. In some cases, each benefit can demonstrate as much as four times the performance of previous architectures, while in practice the typical benefit of these memory bandwidth amplification technologies averages a 50-100 percent improvement.

## Auto Pre-Charge

Another hidden 'tax' on memory bandwidth often overlooked is the various delays due to page management inside the memory chip itself. NVIDIA GPUs equipped with LMA II have a special feature, called auto pre-charge, to reclaim this lost bandwidth. To appreciate the benefits of auto pre-charge, it is important to understand that Dynamic Random Access Memories (DRAMs) are organized in rows, columns, and "banks" as a way to simplify the structure of the memory device and the memory interface. This organization allows the PC to access data in very large memories (a 4Mx32 memory equals 16MB of memory on a single memory chip) with relatively few control wires, reducing overall costs for both the memories and the GPU.

The drawback is that only the current row and column of an active bank can be accessed immediately. If the GPU wants to read from (or write to) a different area of the memory chip, it must tell the memory to close the current bank, and then activate the bank with new row and column settings. This process can take as much as 10 DRAM clock ticks because a bank must be "pre-charged" after it is closed and before it can be reactivated. This often means that no data is moving across the bus while the GPU waits for the memory to get ready. This 10-clock penalty must be paid every time a row and column change is necessary.

GeForce4 GPUs have the ability to tell the memory device proactively to ‘pre-charge’ areas of the memory that are not in use but are likely to be used in the very near future. (Very near future, in memory terms, is generally measured in millionths of a second, or microseconds.) The benefit is that the GPU won’t have to wait for the pre-charge step when it needs to access those areas. It only has to wait for the activation step, which varies by memory vendor and memory device, but is generally 2 or 3 clocks. The result is that the GPU spends less time waiting on the memory and more time rendering pixels, and the computer user enjoys faster graphics performance.

## Fast Z-Clear

Clearing the Z-buffer of “old” data is another task that consumes precious memory bandwidth. Because the Z-buffer data is used to determine if pixel data should be “overwritten,” it must be accurate or the user will see disconcerting artifacts. Unfortunately, clearing the Z-buffer (writing 0s to all of the locations to erase the existing data) takes time and memory bandwidth. The GeForce4 family of GPUs includes Fast Z-Clear technology to minimize the time it takes to clear the “old” data in the Z-buffer. The potential bandwidth savings for a 1600x1200 display with a 32-bit-per-pixel Z-buffer rendering at 60Hz is 460MB/sec. (1600 x 1200 x 60Hz x 32bit). Fast Z-Clears by themselves can boost frame rates up to 10 percent without any compromise in image quality!

---

## Conclusion

To attack the pixel bandwidth problems, LMA II introduces an array of technologies to maximize memory bandwidth efficiency and provide a tremendous leap in PC graphics performance. The combination of the most efficient and sophisticated crossbar-based memory architecture, Quad Cache, advanced lossless Z-compression for reduced bandwidth consumption, auto pre-charge, fast clearing of the Z-buffer, and a highly advanced method to avoid rendering and spending bandwidth on non-visible pixels has a tremendous impact. It means that GPUs equipped with LMA II make more efficient use of memory bandwidth than any other previous graphics architectures.

The advances offered by LMA II pave the way for increasingly dynamic and visually rich real-time 3D graphics experiences. By improving the efficiency of communication between the host and graphics, content developers can continue to increase the geometric richness and visual complexity of their scenes to new levels. This provides end users with the most realistic, life-like images, environments, and effects they have ever seen. Let the fun begin!

---

## Bit Depth

The bit depth refers to the number of bits of precision for the color and z-values associated with each pixel on the screen. More bits of precision improve the visual realism and accuracy of the rendered frame. The two most common bit depths in modern graphics hardware are 16-bit and 32-bit. Each of these values can be associated with color or Z-values. Color that is 32 bit (for example) typically is used to represent red, green, blue and alpha (or transparency) values with up to 8 bits per component, or 256 “values” for each of those components. A 32-bit z-value is typically allocated as 24 bits of Z precision (or depth precision) and 8 bits of stencil or “mask” precision.

---

## Depth Complexity

Depth complexity is a measure of the complexity of a scene. It refers to the number of times any given pixel must be rendered before the frame is done. For example, a rendered image of a wall has a depth complexity of one. An image of a person standing in front of a wall has a depth complexity of two. An image of a dog behind the person but in front of the wall has a depth complexity of three, and so on. As depth complexity increases, more rendering horsepower and bandwidth is needed to render each pixel or scene. The average depth complexity of today’s graphics applications is two to three, meaning that for every pixel you end up seeing, it gets rendered two or three times by the graphics processor.

---

## Fill Rate

Fill rate is the rate at which pixels are drawn into the screen memory. Fill rate is a common measure used to illustrate the pixel processing capabilities of today’s 3D graphics processors. Fill rate is usually measured in millions of pixels/sec. (Mpixels/sec.) In 1997, 50-70 Mpixels/sec. was considered state of the art. In 2002, the leading 3D graphics processors will be capable of more than 1200 Mpixels/sec. While this improvement is an incredible achievement, it is still barely enough to create a compelling 3D environment. Rendering pixels at such a high rate consumes enormous amounts of memory bandwidth.

---

## Frames per Second

Frames per second (fps), or frame rate, refers to how many times per second the scene is updated by the graphics processor. Higher frame rates yield smoother, more realistic animation. It is generally accepted that 30fps provides an acceptable level of animation, but increasing the performance to 60fps results in significantly improved interaction and realism. Beyond 75fps it is difficult to detect any performance improvement. Displaying images faster than the refresh rate of the monitor results in wasted graphics computing power, because the monitor is unable to update its phosphors (or display) that fast, wasting frame rate beyond its refresh rate.

---

## Memory Bandwidth

Memory bandwidth refers to the rate at which data is transferred between the graphics processor and graphics memory. Memory bandwidth limitations are one of the key bottlenecks that must be overcome to deliver truly realistic 3D environments. To deliver truly stunning 3D requires high-resolution, 32-bit color depth at high frame rates, with rich geometry, sophisticated texture mapping, and complex vertex and pixel shading.

---

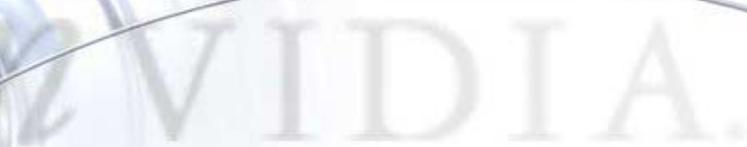
## Resolution

Resolution is the number of pixels on a screen. Higher resolutions can create a more realistic 3D environment because more scene detail can be displayed. Most modern displays are capable of at least 1280 horizontal pixels x 1024 vertical pixels, while many larger or more expensive displays are capable of 2048x1536 pixels. Most graphics applications support a variety of resolutions, allowing the end user to run at higher resolutions (and hence higher level of detail) with the trade-off being increased load on the graphics processing system.

---

## Texture Mapping

Texture mapping is the technique of projecting a 2D image (typically a bitmap) onto a 3D object. Texture mapping allows substantial increases in visual detail without significant increases in polygon count. Because of the improved realism that can be obtained with a very small increase in computational cost, texture mapping is one of the most common techniques for displaying realistic 3D objects. In order to render a texture-mapped pixel, the texture data for that pixel needs to be read into the graphics processor, consuming memory bandwidth.



Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

**Trademarks**

NVIDIA, GeForce4, GeForce4 Ti, and the NVIDIA logo are trademarks of NVIDIA Corporation.

Other company and product names may be trademarks of the respective companies with which they are associated.

**Copyright**

Copyright NVIDIA Corporation 2002.



**NVIDIA.**

NVIDIA Corporation  
2701 San Tomas Expressway  
Santa Clara, CA 95050  
[www.nvidia.com](http://www.nvidia.com)