Technical Brief

Transform and Lighting

Executive Summary

NVIDIA has created a discontinuity in the PC market with the introduction of the GeForce 256™ GPU (Graphics Processing Unit). The introduction of the GPU with integrated hardware transform and lighting engines is a 3D graphics breakthrough that enables a new class of applications.

The GPU delivers a level of realism that was impossible without yesterday's multimillion -dollar graphics supercomputers. Animated characters running on a GPU become lifelike detailed characters with complex facial expressions with smooth movement. The world these characters inhabit are now able to be lush with organic life (trees, bushes, plants) and architectural and structural details that we take for granted in the real world.

In the pre-GPU era, walls for buildings and rooms were placed with the fewest details possible, which resulted in a 3D vacuum. This vacuum will now be filled by the GPU with the details of the real world: realistic furniture, appliances, lights, consumer items, clothing and vehicles. In short, the GPU world will approach what we see in the real world

- CPU power, which was previously held hostage to calculate the sparse pre-GPU world, will now be freed for other tasks, such as physics, inverse kinematics, sophisticated artificial intelligence etc. The combination of a GPU with a highly utilized CPU will enable a new class of user experience.
- This paper will cover the transform and lighting capabilities of the GPU and how this will be experienced by users.

Transform and Lighting

Transform and lighting (T&L) are the first two of the four major steps a GPU computes in the 3D graphics pipeline. They are highly compute intensive, with a set of very specific mathematical instructions performed billions of times per second to render a scene.

To understand the role of (T&L), it is helpful to have a general understanding of the entire process of creating 3D graphics starting with the application itself. The major functional steps are shown in Figure 1.

The Role of the Transform Engine

The process of describing and displaying 3D graphics objects and environments is complex. To reduce this complexity, it is useful to describe the 3D data according to different frames of reference, or different coordinate systems, at different times. These different frames of reference are referred to as "spaces" such as world space, eye space and screen space. Each of these is spaces is convenient for one or more operations that must be performed as a 3D image is created. World space is used for holding all of the 3D objects that are part of the 3D world. Eye space is used for lighting and culling and screen space is used for storing the scene in the graphics frame buffer. However, because these spaces use different coordinate systems, 3D data must be converted or "transformed" from one space to another as it moves through the 3D pipeline. The transform engine performs all of these mathematical transformations. Figures 2 and 3 give visual examples of world space and screen space.

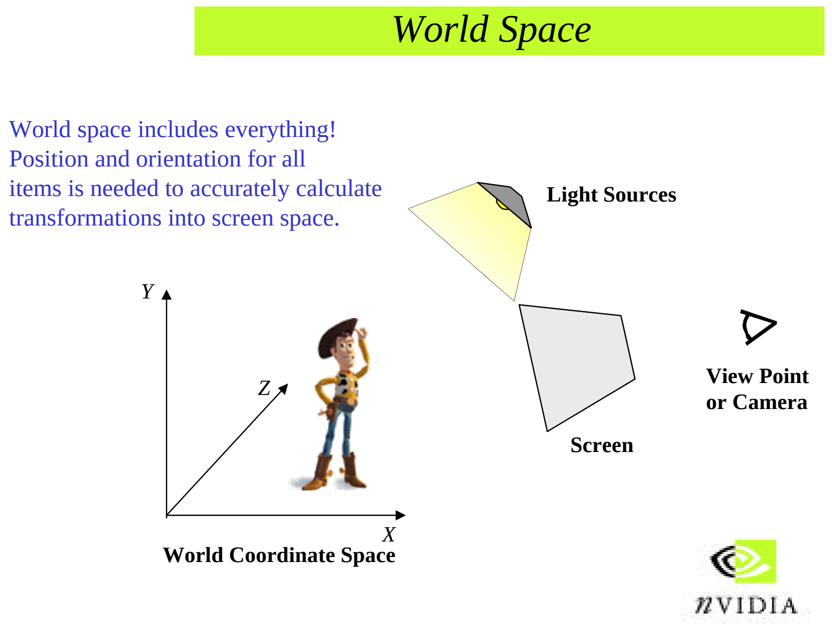
Figure 1
The 3D Graphics Pipeline

Application Tasks	The application controls the movement of objects (including the camera) and their interaction in the 3D world. Problems such as realistic physics calculations (objects have momentum) and collision detection (did my race car bump the wall or bump another car?) affect how an object moves in the scene.
Scene-level Tasks (culling, LOD, display list)	Scene-level tasks include object level culling (car #3 is completely behind the camera, so it doesn't need to be sent to the next stage), selecting the appropriate detail level (car #1 is far away, so a low-detail model is best), and creating the list of all objects in the current camera view.
Transform	The transform engine converts 3D data from one frame of reference to a new frame of reference. The system must transform the data to the current view before performing the following steps (lighting, triangle setup and rendering). Every object that is displayed and some objects that are not displayed must be transformed every time the scene is redrawn.
Lighting	Lighting is the next step in the 3D pipeline and provides high visual impact. Lighting effects are essential for enhancing the realism of a scene and bringing rendered images one more step closer to our perception of the real world.
Triangle Setup and Clipping	The triangle setup engine is a floating-point math processor that receives vertex data and calculates all of the parameters that the rendering engine will require. This unit 'sets up' and 'clips' the triangle for the rendering engine.
Rendering	Rendering is calculating the correct color for each pixel on the screen, given all of the information delivered by the setup engine. The rendering engine must consider the color of the object, the color and direction of light hitting the object, whether the object is translucent, and what textures apply to the object. If rendering is done in multiple stages, it is called multi-pass rendering.

3D Application and API

3D Graphics Pipeline

Figure 2
World Space

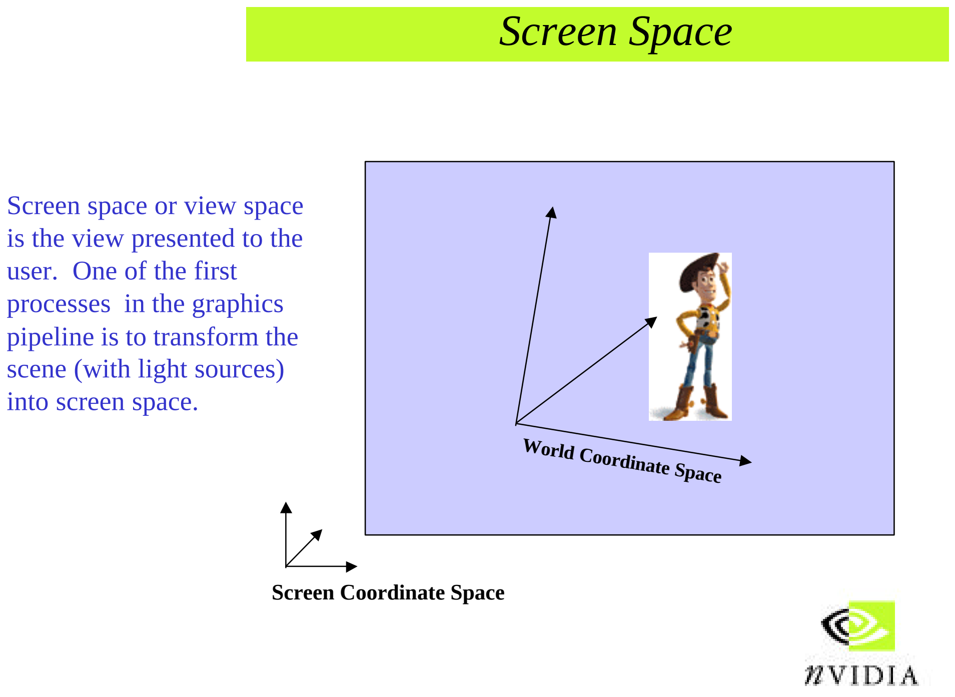


World Space

World space includes everything! Position and orientation for all items is needed to accurately calculate transformations into screen space.

Image courtesy of Pixar.

Figure 3
Screen Space



Screen Space

Screen space or view space is the view presented to the user. One of the first processes in the graphics pipeline is to transform the scene (with light sources) into screen space.

The Role of the Lighting Engine

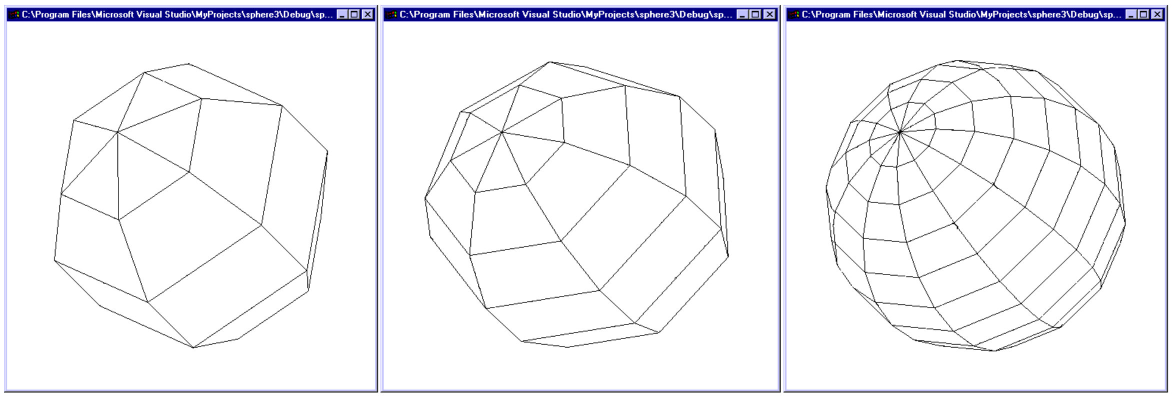
The lighting engine is similar to the transform engine because it has a set of mathematical functions that it must perform. The GeForce 256 GPU uses separate transform and lighting engines so that each engine can run at maximum efficiency. Without discrete engines, the transform performance is limited by sharing compute time with the lighting computation. This discrete engine calculates distance vectors from lights to objects in the 3D scene as well as distance vectors from objects to the viewer's eyes. A vector, by definition, contains information about direction and distance. The lighting engine must also separate the length or distance information from the direction information because that simplifies future steps in the 3D pipeline. Lighting calculations are used for vertex lighting, but they are also critical for other effects, such as advanced fog effects, that are based on the eye-to-object distance rather than just the Z-value of the object.

Why Users Want Integrated Transform Engines in a GPU

Separate transform and lighting engines integrated into one chip, such as in the GeForce 256 GPU, are absolutely necessary for graphics processors to continue to advance the user experience. 3D graphics performance has scaled rapidly over the past four years with the primary emphasis being placed on pixel fill rate and texture-mapping capabilities. This emphasis has reaped for users the rewards of fast frame rates for fluid navigation in 3D environments, but has left the task of the geometry transform and lighting calculations for the host CPU to do. The problem with this is the fact that CPUs only double in speed every 18 months while graphics processors advance eight times in the same period of time. Geometry processing (T&L) performance has now become the most significant barrier to more sophisticated 3D graphics on a typical PC. The graphics processor spends too much time waiting for the CPU! This bottleneck has forced software developers to limit the geometric complexity of their 3D characters and environments, sacrificing image quality and elaborate 3D environments in order to maintain expected performance. Integrated transform engines are the most cost-effective way to alleviate the geometry performance bottleneck and open up new opportunities for software developers to add greater geometric detail to their 3D worlds.

Transform performance dictates how finely software developers may tessellate the 3D objects they create, how many objects they can put in a scene and how sophisticated the 3D world itself can be. Tessellation is the process of converting curved lines into a series of line segments that approximate the original curve. This idea extends to three-dimensional objects as curved surfaces are converted into polygons that approximate the surface. This creates a classic performance-versus-quality trade-off for the software developer because finer tessellation will result in more polygons and slower performance, but with the reward of higher quality. Figure 4 shows a simple example of a sphere tessellated by different degrees.

Figure 4
Tessellated Spheres


C:\Program Files\Microsoft Visual Studio\MyProjects\sphere3\Debug\sp...
C:\Program Files\Microsoft Visual Studio\MyProjects\sphere3\Debug\sp...
C:\Program Files\Microsoft Visual Studio\MyProjects\sphere3\Debug\sp...

Each of the images in Figure 4 represents the same sphere, but the image on the far right is clearly the most realistic of the three. It also has almost five times as many polygons as the image on the far left and three times as many polygons as the middle image. This means that the 'sphere' on the right requires five times as much transform performance as the leftmost sphere to maintain a given frame rate for the user. Computer users prefer frame rates in the 30 fps to 60 fps range and will often adjust the 'quality' settings of the application (rendering features, resolution, etc.) to make sure that the frame rate is in this range.

This simple "performance-versus-quality" trade-off becomes more complex when one considers that a current 3D scene requires hundreds to thousands of objects that must share the transform and lighting ability of the CPU. Software developers must budget how much detail to put in each model and how many models to use in the scene so that the overall quality of the scene is maximized and performance is maintained within an acceptable range. If a jungle scene is the goal, lots of trees and bushes are necessary, having a single tree and bush does not convey the idea of a jungle.

The trade-off between performance and higher geometric complexity (to create higher image quality) would not be severe if the transform performance of today's PCs were dramatically higher. The GPU's goal is to deliver performance and quality that push the limits of human perception. The reality today is that even the fastest CPU forces software developers to make severe compromises in both performance and quality to deliver 30fps to 60fps.

Why Users Want Integrated Lighting Engines in a GPU

Integrated lighting is a critical requirement in 3D graphics, because it has such a dramatic impact on image quality. Lighting calculations are an effective way of adding subtle and not-so-subtle changes in brightness to 3D objects in a manner that mimics our real-world experiences. That ability to render realistic scenes is crucial because makes it 3D graphics appeal to a broad audience. The more realistic 3D graphics scenes become, the more impact and applicability they will have. 3D applications run the gamut from entertainment to education to science, industry, medicine, data analysis and more. Lighting effects add impact to each of those applications because they create realistic scene detail. The human eye is actually more sensitive to changes in brightness than it is to changes in color.

Simply put, lighting effects take greater advantage of the natural abilities of the human eye. This means that an image with lighting effects will communicate more information to the viewer in the same amount of time. Figure 5 shows an architectural scene that demonstrates the potential of lighting effects to add realism to a scene. Without integrated lighting, objects cannot react to complex changes in eyepoint relative to light sources accurately

Figure 5
Lighting Example

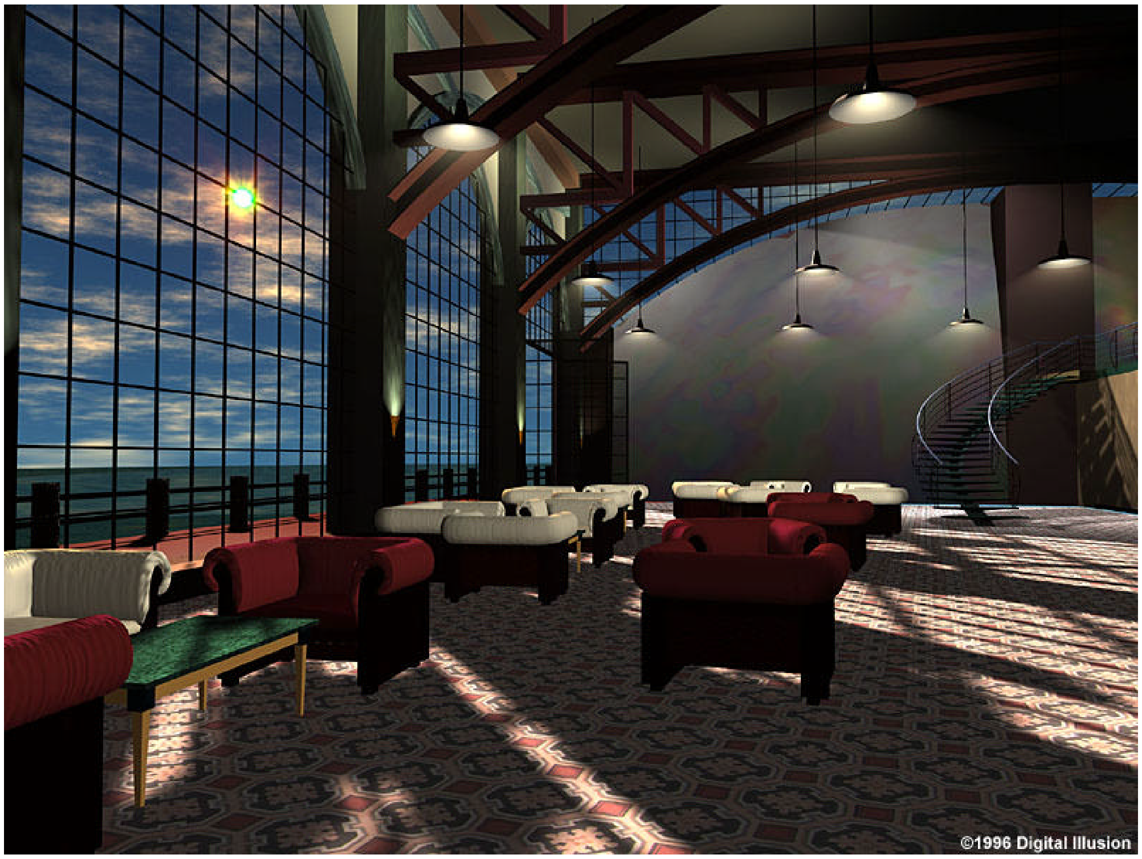
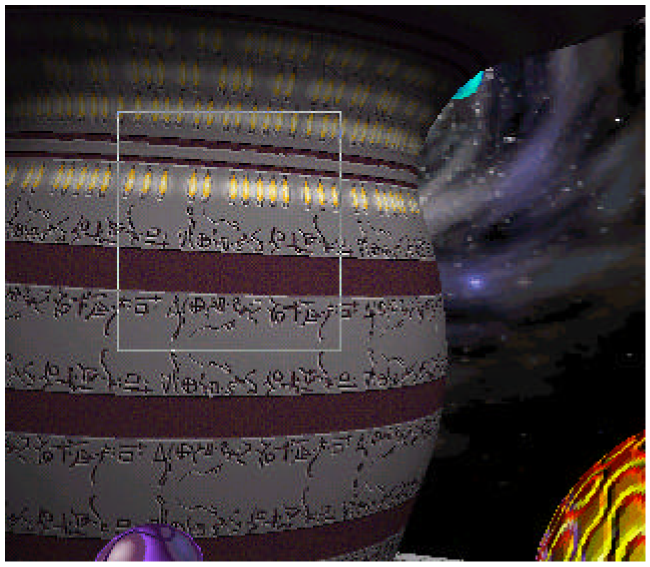


Image courtesy of Digital Illusion.

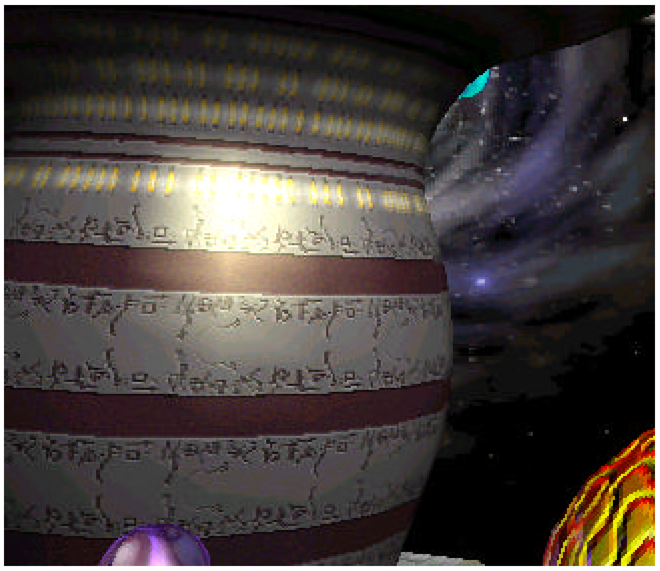
Lighting in 3D graphics is divided into two major components to simplify the task of modeling the physics of light. Those two components are diffuse lighting and specular lighting. Diffuse lighting assumes the light hitting an object scatters in all directions equally, so the brightness of the reflected light does not depend at all on the position of the viewer. Sunlight on a playground is an example in the real world of diffuse lighting. The brightness of an object in the scene is primarily determined by the diffuse lighting calculations. Specular lighting is different because it does depend on the position of the viewer as well as the direction of light and orientation of the triangle being rendered. Shining a spotlight into a dark corner of a room onto a TV set and looking at the hot spots on the picture tube will show you specular lighting. Specular lighting captures the mirror-like properties of an object so effects such as reflection and glare are achievable. Figure 6 shows two examples of a space station from the

3D WinBench® benchmark that demonstrate the differences between diffuse and specular lighting effects.

Figure 6 and 7
Specular Highlights



Diffuse Lighting Only



Diffuse and Specular Lighting

Specular highlights move on the object if the viewer or the object moves relative to the light source. For this reason they cannot be pre-computed or static. Specular lighting is particularly useful for two effects in a 3D scene: displacement mapping and creating the appearance of different materials for objects.

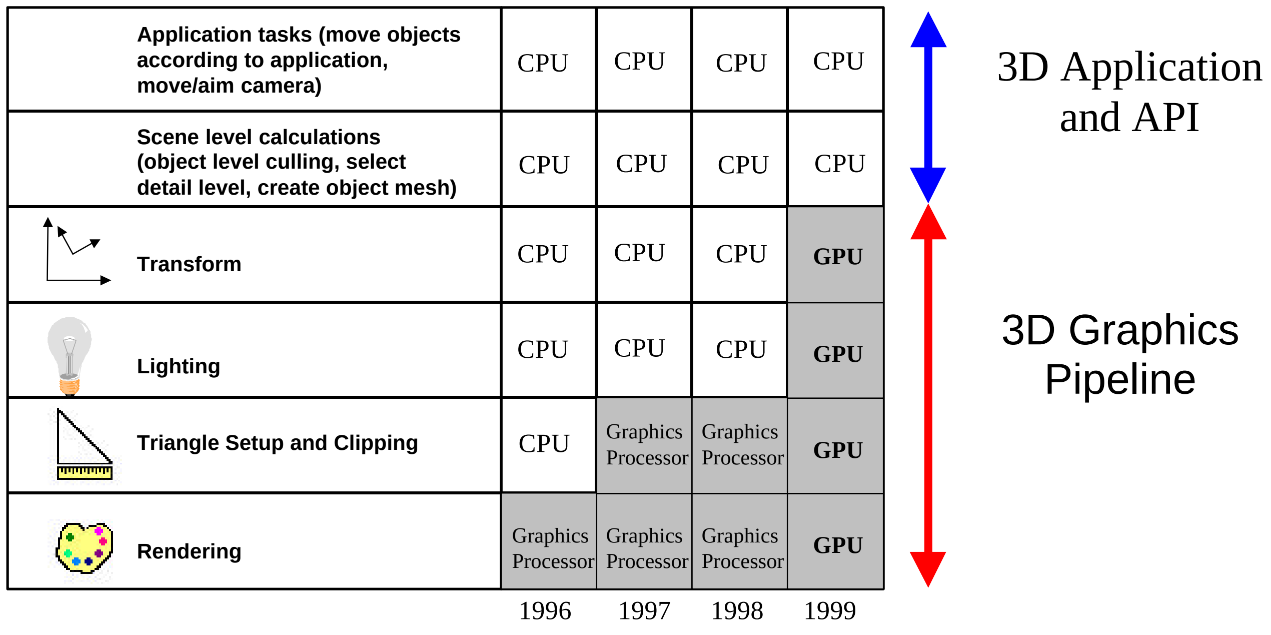
Specular lighting is also important for representing different materials for objects in a 3D scene. A silk shirt looks different than a cotton shirt, even if they are the same color. A major difference is how the two materials reflect light, which is captured with specular lighting. Another example is polished stone such as marble compared to the same material before polishing. The polishing doesn't change the color or pattern in the marble, but it does affect the way light is reflected. Specular lighting combined with texture mapping creates more realistic objects because they have the visual properties of real materials.

Specular lighting can be done without a dedicated hardware lighting engine, but only with a severe loss of performance. Texture mapping can be used for some specular lighting effects, but if the viewer moves around in the scene, then the environment maps must be re-calculated, which is a time-consuming process unless the graphics hardware supports cube environment mapping. Sphere mapping is the common alternative but it suffers from performance and quality problems as the viewer moves around in the scene, making it unattractive for interactive 3D environments.

Where Transform & Lighting (T&L) Work is Done: The Migration from CPU to GPU

All of the work in the 3D graphics pipeline is divided between the CPU and the graphics processor. The line that divides the CPU tasks from those performed on the graphics processor moves as the capabilities of the graphics processor continue to grow. 1999 is the year when graphics processors with integrated T&L engines can be sold at mainstream PC price points and create a compelling value proposition for any PC user. Figure 8 graphically shows the growing role in the last few years of the dedicated graphics processors for mainstream PCs. The complete graphics pipeline is now computed by the graphics processing unit, hence the term GPU.

Figure 8
The 3D Graphics Pipeline



	1996	1997	1998	1999
Application tasks (move objects according to application, move/aim camera)	CPU	CPU	CPU	CPU
Scene level calculations (object level culling, select detail level, create object mesh)	CPU	CPU	CPU	CPU
Transform	CPU	CPU	CPU	**GPU**
Lighting	CPU	CPU	CPU	**GPU**
Triangle Setup and Clipping	CPU	Graphics Processor	Graphics Processor	**GPU**
Rendering	Graphics Processor	Graphics Processor	Graphics Processor	**GPU**

The transform and lighting functions of the 3D pipeline traditionally have been performed by general purpose microprocessors such as those used as CPUs in PCs. This was driven by the traditionally high cost of duplicating the floating-point functions required for T&L in the graphics processor coupled with the fact that most CPUs already contained complete floating-point units (FPUs). The cost of implementation has been a traditional barrier to adding these features to graphics products in mainstream PCs.

The cost equation was different for exotic Unix workstations, where non-integrated transform and lighting processors were available to those with unlimited budgets. As costs declined in the past 2 years, Windows NT® workstations were the next business opportunity but those solutions were relegated to multi-board graphics add-in cards costing thousands of dollars. Prices continued to decline as semiconductor manufacturing allowed more transistors to be put on a single die. In 1999, the affordable GPU is possible with dedicated hardware T&L acceleration because it can now be integrated onto the same die with the other components of the 3D graphics pipeline. Silicon process technology has shattered this cost barrier and enabled NVIDIA's GPU architects to integrate all functions on a single die.

Now that the GPU is available to high-volume PC designs, system designers will choose this new path to deliver a stunning visual experience to all users. The two key architectural benefits are:

- 1) Higher graphics performance – GPUs with integrated T&L engines will process those functions at two to four times the speed of the leading CPUs.
- 2) CPU power can be better utilized for functions such as physics calculations, artificial intelligence, other application functions, and managing system resources

GPUs can easily outperform a CPU because a dedicated graphics engine focuses on the critical math functions only. The floating-point unit in a CPU must provide many other functions because its general-purpose nature requires that versatility. GPU's are not as versatile; however they are extremely efficient in executing a specific set of graphics functions.

The mathematics requirements for a GPU engine are specific. T&L calculations rely heavily on matrix mathematics, specifically 4x4 matrix multiplication, and a few other vector operations. The focused number of operations that a T&L engine must support makes it straightforward to optimize the design for maximum silicon efficiency. That efficiency makes the throughput of a GPU highly predictable too. This predictability stems from the fact that buffers and pipelines can be optimized for these very specific requirements. When the CPU performs T&L calculations, the throughput is more variable because it varies with the rest of the workload imposed on the CPU. Most CPUs today offer some type of matrix math extensions such as MMX, SSE or 3DNow!®, but these extensions of the instruction set are, to a degree, general purpose as well. Those extensions must be able to process a variety of multimedia functions including audio, video and communications algorithms.

A hit 3D game running on a GPU also requires more than pretty graphics, or "eye candy." It requires sophisticated artificial intelligence (AI), realistic physics and more complex game elements. The need for heavy artificial intelligence in strategy and sports games has limited pre-GPU titles to characters that are poorly defined. The developers have had to sacrifice realistic looking graphics for basic game play since the CPU is required to handle both AI and graphics tasks. The GPU allows realistic soccer, football and basketball teams to have players that look, run, jump and play like real players. Realistic physics modeling enhances 3D applications by more accurately mimicking the real world. Objects in the real world have momentum, so objects in a 3D application should too. These functions alone can overwhelm a leading-edge CPU today and the CPU still has all of its traditional functions to do as well, such as managing bus traffic, servicing the operating systems requirements, and handling all application level requirements. These functions cannot be offloaded to the graphics processor, but the GPU operations will be offloaded to improve the overall graphics performance and overall application performance too.

The PC User Is the Ultimate Winner

PC users are the ultimate winners as T&L operations move from the CPU to GPU because the new architecture will deliver dramatically better performance at the same price. The higher performance comes from more efficient partitioning of tasks between the different processors in the PC system. The benefits are significant and include:

- Existing 3D applications can run faster.
- Software developers can use more advanced effects and graphics techniques, so the graphics in their applications will be more informative and more captivating.
- New applications using 3D as a medium emerge in the same way that the two-dimensional bit-mapped Graphical User Interface (GUI) enabled WYSIWYG desktop publishing rather than just word processing with text-based interfaces.

These benefits increase with the overall graphics performance of the PC. GPUs like the GeForce 256 will create a new level of price/performance for the typical PC user. This trend starts this year, in 1999, and will continue growing as every graphics processor includes integrated geometry acceleration.

Appendix A

How Transform and Lighting Works: The Mathematics Behind the Magic

Transform and lighting operations are similar because they both require a set of mathematical functions that are repeated over and over again millions of times each second. The actual math capabilities required in a GPU are described here so the reader can appreciate why these functions lend themselves to implementation in highly optimized, dedicated processors rather than general purpose CPUs.

A transform operation is a 4x4 matrix-multiply operation. A vector representing 3D data, typically a vertex or a normal vector, is multiplied by a 4x4 matrix called the transform matrix and the result is the transformed vector. To do this, the transform engine uses standard linear algebra to multiply the matrices. Figure 9 shows a generic example of this operation.

Figure 9
Matrix Multiplication

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax+by+cz+dw \\ ex+fy+gz+hw \\ ix+jy+kz+lw \\ mx+ny+oz+pw \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

Transform Matrix | Original Vector | Transformed Vector

Before a vector can be transformed, a transform matrix must be constructed. This matrix contains all of the information necessary to convert vector data to the new coordinate system. An interim matrix must be created for each action (scaling, rotation and translation) that should be performed on the vector and those interim matrices are multiplied together to create a single matrix that represents the combined effect of all of those actions. That single matrix is called the transform matrix and it can be re-used without being re-created. For example, once the transform matrix is constructed, it can be used to transform one vector or one million vectors. This ability to re-use the transform matrix effectively amortizes the setup time required to create the matrix over all the times it is used.

Intuitively, one might think that a 3x3 matrix is sufficient because this is a 3D graphics calculation. The fourth component in the original vector is *w*. *W* is used as a scaling factor for perspective correction and drives the transform matrix to be 4x4. A thorough discussion of *w* is beyond the scope of this paper.

Note that the number of actual mathematics operations for a single transform is substantial: 16 multiplication operations and 12 addition operations. The mathematics required for transforms are also very simple, only multiplication and addition are used.

Part of the magic of using matrix multiplication for transform operations is that scaling, rotation and translation all take the same amount of time to perform. They also can be done simultaneously. This makes the performance of a dedicated transform engine predictable and consistent. Predictability allows software developers to know how their application will perform with a given amount of geometry, allowing them to make informed decisions regarding performance and quality.

Lighting calculations are computationally expensive. Typical lighting calculations are computing distances and directions between light sources and objects. Lighting engines also perform some data conversion functions such as creating the normal vectors for triangles and vertices as well as "normalizing" those vectors. Normalizing a vector is the process of converting it to a new vector with a length of one unit and points in the same direction as the original vector. A 'normal' vector to a vertex or to a triangle by definition is a vector that is perpendicular to the triangle or the vertex described. Perpendicularity to a vertex is defined as perpendicular to the surface that the vertex describes.

The mathematics in a simplified lighting model is described here. The calculation of difference vectors involves only addition (subtraction is simply adding a negative value) of the xyz data for each of the objects and therefore requires three addition operations. For example the difference vector between light source #1 at position (x_1, y_1, z_1) and object #2 at position (x_2, y_2, z_2) is:

$$\text{Difference Vector} = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

These vectors must be processed further into a scalar distance value and a normalized vector before leaving the lighting engine. These operations are more complex but very necessary because representing the difference vectors as separate values for distance and direction simplifies future steps in the 3D pipeline. Calculating the distance is a matter of squaring the x,y and z values adding them and taking the square root of the sum, as shown in Equation 1.

Equation 1 $\quad$ $\text{distance} = (x^2+y^2+z^2)^{1/2}$

Normalizing the vector requires dividing each of the x,y and z values by the distance. So the lighting engine requires the ability to add as well as divide.