Parallel Architectures and Algorithms for Real-Time Synthesis of High Quality Images using Deferred Shading

TR92-034

September, 1992

Brice Tebbs
Ulrich Neumann
John Eyles
Greg Turk
David Ellsworth

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

Parallel Architectures and Algorithms for Real-Time Synthesis of High Quality Images using Deferred Shading

Brice Tebbs, Ulrich Neumann, John Eyles, Greg Turk and David Ellsworth

Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

High-end graphics workstations provide us with the ability to interactively display polygonal models of high geometric complexity. It is our hope that future graphics workstations will also provide us with shading models that go beyond Gouraud shading of polygons. This paper describes performance improvements that result from storing a polygon's shading parameters at each pixel and deferring the shading calculations until all the polygons have been processed. One benefit of this approach is that the shading calculations are only performed on the parts of a surface that are visible, which means that this method becomes more attractive as shading models become more complex and as the depth complexity of the scene increases. We also show how this technique maps onto different parallel architectures for high performance rendering. In particular, we describe how this method can be used to produce real-time images that incorporate Phong shading and procedural texture mapping on Pixel-Planes 5. Pixel-Planes 5 is a massively parallel SIMD machine that is under construction at UNC. Since the processing elements of Pixel-Planes are fully programmable, new shading models can be incorporated in the system without any hardware modifications.

Introduction

Both the complexity of geometric models and the surface properties of objects contribute to the visual richness of computer generated scenes. Polygons are the most commonly used geometric elements for graphical models and much effort has been directed towards building hardware to rapidly display large collections of polygons [Fuchs 85] [Akeley 88] [Apgar 88] [Potmesil 89]. However, most of the hardware architectures developed so far are unable to generate images with complex shading models in real-time.

This paper describes an approach to high-quality rendering that we call *deferred shading*. While this approach is not new, we would like to explore its application to the design of real-time graphics systems. To illustrate this technique, we describe how we have used this approach to improve the performance of Pixel-planes 5 [Fuchs 89].

Deferred shading involves two steps: rasterization and shading. For our purposes rasterization includes scan-conversion, visibility determination, and the loading of geometric information, such as surface normals, into each visible pixel. Storing this information requires additional memory per pixel. This may seem impractical at first but currently available graphics systems already have as many as 96 bits per pixel [Akeley 89] (A recently announced machine has >200 bits per pixel but full details are currently unavailable). Once rasterization is complete, the information stored at each pixel is used to compute the pixel's color based on the shading model. We believe this technique can be used effectively on any graphics system with the following characteristics:

- Deep frame buffer >100 bits per pixel
- High bandwidth into the frame buffer
- Pixel-level parallelism

Related Work

Deferred shading is not new to rendering software. In a scan-line polygon renderer [Watkins 70] the nearest surface at a given pixel is found and then only the shading for that front-most surface is computed at the pixel. Other examples of deferred shading include Andrew Glassner's Late-Binding Renderer [Glassner 88] and Ken Perlin's Pixel Stream Editor [Perlin 85], in which information about surfaces such as surface ID, depth and a normal vector are saved for the entire image. The only communication between renderer and shader is the image description that includes this surface information for each pixel.

Deferred shading was part of the triangle processor system proposed by Deering et. al. [Deering 88]. In this architecture polygon rasterization is performed by passing a stream of pixels through a pipeline of triangle processor chips that perform depth comparisons and scanning out the closest surface at each pixel. The system defers shading calculation until after scan conversion by passing the depth, a surface normal and color description for each pixel to a collection of chips called the Normal Vector Shaders. The Normal Vector Shaders compute the full Phong lighting model based on the surface normal and color. Unfortunately, the Normal Vector Shaders were to be hardwired for a specific lighting model and would have been unable to take full advantage of some of the deferred shading concepts and algorithms that we present in this paper.

Advantages of Deferred Shading

An obvious benefit of deferred shading is that only those pixels that are visible in the final image are shaded. Computing the Gouraud shading model is so fast that current graphics workstations can afford to shade all the pixels in a polygon even if many of the pixels will later be obscured. However, as scene depth complexity increases and as shading models become more complex, this becomes a substantial amount of wasted effort.

For certain classes of parallel hardware, deferred shading enjoys a second benefit. For many scenes the shading computation will be the same at each pixel. We call the degree to which this is true the *shading coherence* of the scene. For example, a scene in which all of the surfaces are shaded using the Phong lighting model would exhibit high shading coherence. Parallel machines achieve high utilization when performing a large number of similar computations. This means that scenes with high shading coherence can utilize massive parallelism at the pixel level even if the geometric operations required for rasterization are more difficult to parallelize.

An additional advantage of this method has emerged as we developed the actual software for the Pixel-planes 5 machine. The separation of geometric computation from shading calculation means that we can implement new geometric primitives without having to write new shading code. If we are able to load the correct surface geometry information into the pixels, then all of the shaders we have written for polygons will work for other geometric primitives such as surfaces and spheres.

Architectural Considerations

We believe that as scene complexities increase and higher quality shading models are used, deferred shading will become more popular for real-time image synthesis. Our goal is to design systems which achieve 30Hz or greater update rates with advanced lighting models and texture maps. In this section we discuss ways this might be achieved using different organizations of processors for shading computations.

The shading computation for a pixel could be done by a pipeline of processors. Each processor would complete on step of the shading computation for each pixel. Unfortunately, a very deep pipeline would be required to perform complicated shading models on a >1M pixel frame buffer at refresh rates. Such a pipeline would be difficult to design and even harder to program for a variety of shading algorithms. This is important since highly realistic shading algorithms can be implemented with many different variations [Cook 84].

A MIMD array of processors where each processor shades a pixel at a time would allow for greater parallelism. We could continue to add more processors until we had enough get the shading rates we needed. This option would also allow for easier programming of new shading algorithms. A MIMD architecture could achieve good performance even for scenes with little or no shading coherence. However, we believe that the shading coherence in most scenes will make the cost of multiple control units for the MIMD processors an unnecessary expense.

A SIMD processor array could be used to exploit high shading coherence at a lower cost than a MIMD array. This is the architecture that has been used in both the Pixel-planes 4 and Pixel-planes 5 designs. The SIMD processor arrays can shade 256K pixels simultaneously. Since Pixel-planes 5 employs several SIMD renderers, it can be considered a hybrid MIMD-SIMD architecture. This enables it to more efficiently handle images with lower shading coherence.

Algorithms

We have been developing deferred shading algorithms for the Pixel-Planes 5 graphics architecture . Pixel-Planes 5 is a heterogeneous multiprocessor [Fuchs 89]: A fully configured system contains a MIMD set of 16 fast floating point engines called *graphics processors* for performing geometric transformations and has 16 SIMD arrays called *renderers* for performing polygon rasterization. Each renderer is a square array of 128x128 processing elements called *pixel processors*, and each of these pixel processors is a 1-bit ALU that runs at 40Mhz and has 208 bits of memory. On Pixel-Planes 5, deferred shading reduces the computational burden on the graphics processors and fully utilizes the quarter million pixel processors whose aggregate computing power completely overshadows that of the 16 graphics processors.

The remainder of this section describes two of the deferred shading algorithms that we have developed for Pixel-planes 5. They are all currently running on a software simulator.

Phong Shading

We defer the computation of a lighting model by storing the surface normal and shading parameters at each pixel during rasterization. We have found that 100 bits per pixel is adequate to store this information. For Phong shading the surface normals must be interpolated [Phong 73]. This interpolation process is simple and requires only 3 adds per pixel using forward differencing.

Before the actual lighting model can be evaluated the interpolated normal vectors must be normalized. This requires the computation of a square root and a divide at each pixel. The normal vector and eye vector are then used to compute the normalized reflection vector at each pixel. Evaluation of the lighting equation may now be performed. If, as is often done, we approximation the eye vector and light vector as constants across all pixels, each pixel requires 2 dot products, 3 additions, 12 multiplications plus the exponentiation for the specular term.

The general lighting model additionally supports positional and spot lights with soft edges:

$$\text{Intensity} = \left[\text{Amb} + (L \bullet N) * (L \bullet D)^{\text{cone}} * \text{Light}\right] * K_d + (L \bullet R)^{\text{spec}} * (L \bullet D)^{\text{cone}} * \text{Light} * K_s$$

Where:

Amb	=	Ambient light intensity	L	=	Direction to light source
Light	=	Light source intensity	D	=	Spot light direction
K_d	=	Surface diffuse coefficient	N	=	Surface normal vector
K_s	=	Surface specular coefficient	R	=	Reflection vector
spec	=	Surface specular power	cone	=	Spot light beam concentration

While infinite distance point light sources allow simple approximations, for positional and spot light sources, the correct light vectors at each pixel's surface must be computed. This can be done by applying the inverse of the perspective transformation to the screen-space coordinate of the surface sampled at each pixel. The true eye vector and any number of light vectors may now be computed at each pixel for use in the shading equation.

A Pixel-Planes 5 renderer can interpolate the three components of the surface normal for an entire polygon in 2.25 μs. (A complete triangle is rasterized in under 8 μs.) Normalization is done with a Newton iteration and requires 154 μs. The simple case of eye ray approximation and reflection ray computation requires 35 μs.

Evaluation of the shading equation for an infinite distance light source (light vector is constant) requires 229 µs for a specular power of 128. Adding an ambient light source adds 44 µs and brings the total cost for this case to 462 µs. For the above case, a fully configured Pixel-Planes 5 system can compute the phong shading for an entire 1280x1024 image in 2.31 msec. (Note that each renderer is computing the lighting model five times; once for each of five pixel regions.) Performance statistics for a Pixel-Planes 5 renderer (assuming spec and conc = 128) are broken down by task and summarized below.

Normalize surface normals	Compute eye and reflection vector	Evaluate lighting equation for each type of light
154 µs	Approx. eye vector = 35 µs	Ambient source = 44 µs
	True eye vector = 310 µs	Infinite point source = 229 µs
		Positional source = 504 µs
		Spot source = 640 µs

Texture Mapping

Texture mapping has been important in high-quality image synthesis for many years [Catmull 74]. With the exception of multi-million dollar flight simulators, few systems have incorporated the necessary hardware to do real-time texture-mapping. Since extensive calculations are necessary to render properly filtered textures, we believe that deferred shading is an ideal approach for texture mapping scenes with high depth complexity or for texturing any scene with procedurally defined textures. Other surface effects such as bump mapping can be done using deferred shading when the lighting calculations are also deferred.

The first step in texture-mapping is to compute the texture coordinates *u* and *v* for each pixel to be textured. *U* and *v* can be expressed as the ratio of two-bivariate linear expressions in screen space [Heckbert 86] [Fuchs 85]:

$$u = \frac{Ax + By + C}{Gx + Hy + I} \qquad v = \frac{Dx + Ey + F}{Gx + Hy + I}$$

With forward differencing, these equations can be computed on a scan-line basis with 3 adds and 2 divides per pixel. With deferred shading only the 3 adds are performed during rasterization; the divides can be computed for the visible pixels in parallel after rasterization.

We have concentrated on procedural texture mapping for several reasons. Procedural textures typically have small memory requirements [Perlin 85]; they are fast on our system as compared to image textures; they are easily extended to 3-d without a dramatic increase in memory usage; and they can be self-filtering [Gardner 88]. We hope to animate them to produce real-time moving scene backgrounds such as waving grass, moving clouds, or moving waves. Recently, procedural texture mapping has been used to generate high-quality images of intensely geometric objects such as fur [Perlin 89].

Gardner has used a product of two sets of cosine waves with cross-coupled phase offsets to generate realistic-looking textures for natural scene generation [Gardner 85]. We have implemented this technique using quadratic approximations to the cosine functions on our current Pixel-Planes 4 prototype and developed a tool that allows the user to explore the space of possible Gardner textures in real-time. A timing analysis of Gardner textures for Pixel-Planes 5 shows the advantage of massive parallelism at the pixel level. Typical interesting textures require 5 cosine calculations for each of the two sets of cosines comprising the texture function. Computing each cosine term in a sum requires 5 multiplies and 3 additions. Pixel-Planes 5 can compute the complete texture for a 128x128 renderer in about 0.5 msec. This means that we can do the shading calculations to texture map an entire 1280x1024 display in 2msec on a fully configured system. Since different textures are computed with the same formula we can compute textures for sky and terrain pixels simultaneously by loading different parameters for pixels with different textures. This allows us to maintain high shading coherence even though we may have several different texture maps.

Interactive displays are particularly sensitive to sampling artifacts, since artifacts sparkle and dance over moving polygons. Unfortunately, simple super-sampling in screen space is not sufficient to anti-alias texture maps [Heckbert 86]. The calculations to perform this filtering can also be deferred if enough information is saved in the frame buffer to compute an estimate of the relative size of texture pixels and screen pixels. This information could take the form of partial derivatives or values for *u* and *v* at the pixel corners. Image textures can then be anti-aliased using any of a number of standard techniques [Williams 83] [Crow 84]. Procedural textures can be anti-aliased when the texture samples are computed by omitting the high-frequency components of the function [Gardner 88]. We have concentrated on this and have achieved good results anti-aliasing Gardner textures using clamping [Norton 82].

We are currently implementing image textures using the MIPMAP anti-aliasing technique [Willams 83], which should give interactive speeds for small textures (33 ms for one 64x64 texture). We plan to implement at least two other algorithms with deferred shading: bump mapping [Blinn 78] and fog.

Summary

We have described an approach to designing graphics systems for real-time high quality image generation. The technique has the advantage that complicated shading algorithms need only be computed for the visible pixels. We have shown that SIMD parallel architectures work well with a deferred shading system, and show how to implement Phong shading and procedural texture mapping.

References

[Akeley 88] Akeley, Kurt and T. Jermoluk, "High-Performance Polygon Rendering," *Computer Graphics*, 22(4), (Proceedings of SIGGRAPH '88), pp. 239-246.

[Apgar 88] Apgar, B., B. Bersack, A. Mammen, "A Display System for the Stellar Graphics Supercomputer Model GS1000," *Computer Graphics*, 22(4), (Proceedings of SIGGRAPH '88), pp. 255-262.

[Blinn 78] Blinn, J. F., "Simulation of Wrinkled Surfaces," *Computer Graphics*, 12(3), (Proceedings of SIGGRAPH '78), pp. 286-292.

[Catmull 74] Catmull, Ed, "A Subdivision Algorithm for Computer Display of Curved Surfaces," Ph.D. Dissertation, University of Utah, December 1974.

[Cook 84] Cook, Robert L., "Shade Trees," *Computer Graphics*, 18(3), (Proceedings of SIGGRAPH '84), pp. 223-232.

[Crow 84] Crow, F., "Summed-Area Tables for Texture Mapping," *Computer Graphics*, 18(4), (Proceedings of SIGGRAPH '84), pp. 207-212.

[Deering 88] Deering, M., S. Winner, B. Schediwy, C. Duffy, N. Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics," *Computer Graphics*, 22(4), (Proceedings of SIGGRAPH '88), pp. 21-30.

[Fuchs 85] Fuchs, H., J. GoldFeather, J.P. Hultquist, S. Spach, J. Austin, F.P. Brooks, Jr., J. Eyles, and J. Poulton, "Fast Spheres, Textures, Transparencies, and Image Enhancements in Pixel-Planes," *Computer Graphics*, 19(3), (Proceedings of SIGGRAPH '85), pp. 111-120.

[Fuchs 89] Fuchs, H., J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs and L. Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *Computer Graphics*, 23(3), (Proceedings of SIGGRAPH 89), pp. 79-88.

[Gardner 85] Gardner, Geoffry Y., "Visual Simulation of Clouds," *Computer Graphics*, 19(3), (Proceedings of SIGGRAPH 85), pp. 297-304.

[Gardner 88] Gardner, G., "Functional Modeling of Natural Scenes, Functional Based Modeling," *SIGGRAPH Course Notes*, vol. 28, 1988, pp. 44-76.

[Glassner 88] Glassner, Andrew, "Algorithms for Efficient Image Synthesis," Ph.D. Dissertation, University of North Carolina at Chapel Hill, 1988.

[Heckbert 86] Heckbert, Paul S., "Survey of Texture Mapping," *IEEE Computer Graphics and Applications*, 6(11), pp. 56-67.

[Norton 82] Norton, Alan, "Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space," *Computer Graphics*, 16(3), (Proceedings of SIGGRAPH '82), pp. 1-8.

[Perlin 85] Perlin, Ken, "An Image Synthesizer," *Computer Graphics*, 19(3), (Proceedings of SIGGRAPH '85), pp. 151-159.

[Perlin 89] Perlin, Ken and Eric M. Hoffert, "Hypertexture," *Computer Graphics*, 23(3), (Proceedings of SIGGRAPH '89), pp. 253-262.

[Phong 73] Phong, B.T., "Illumination for Computer-Generated Pictures," Ph.D. Dissertation, University of

Utah, Salt Lake City, 1973.

[Potmesil 89] Potmesil, Michael and Eric M. Hoffert, "The Pixel Machine: A Parallel Image Computer," *Computer Graphics*, 23(3) (Proceedings of SIGGRAPH '89), pp. 69-78.

[Watkins 70] Watkins, G., "A Real-Time Visible Surface Algorithm, " University of Utah Computer Science Department, UTEC-CSc-70-101, June 1970, NTIS AD-762 004.

[Williams 83] Williams, Lance, "Pyramidal Parametrics," *Computer Graphics* 17(3) (Proceedings of SIGGRAPH '83), pp. 1-11.