Efficient Bump Mapping Hardware

Mark Peercy
John Airey
Brian Cabral
Silicon Graphics Computer Systems *

Abstract

We present a bump mapping method that requires minimal hardware beyond that necessary for Phong shading. We eliminate the costly per-pixel steps of reconstructing a tangent space and perturbing the interpolated normal vector by a) interpolating vectors that have been transformed into tangent space at polygon vertices and b) storing a precomputed, perturbed normal map as a texture. This represents a considerable savings in hardware or rendering speed compared to a straightforward implementation of bump mapping.

CR categories and subject descriptors: I.3.3 [Computer Graphics]: Picture/Image generation; I.3.7 [Image Processing]: Enhancement

Keywords: hardware, shading, bump mapping, texture mapping.

1 INTRODUCTION

Shading calculations in commercially available graphics systems have been limited to lighting at the vertices of a set of polygons, with the resultant colors interpolated and composited with a texture. The drawbacks of Gouraud interpolation [9] are well known and include diffused, crawling highlights and mach banding. The use of this method is motivated primarily by the relatively large cost of the lighting computation. When done at the vertices, this cost is amortized over the interiors of polygons.

The division of a computation into per-vertex and per-pixel components is a general strategy in hardware graphics acceleration [1]. Commonly, the vertex computations are performed in a general floating point processor or cpu, while the per-pixel computations are in special purpose, fixed point hardware. The division is a function of cost versus the general applicability, in terms of quality and speed, of a feature. Naturally, the advance of processor and application-specific integrated circuit technology has an impact on the choice.

Because the per-vertex computations are done in a general processor, the cost of a new feature tends to be dominated by additional per-pixel hardware. If this feature has a very specific application, the extra hardware is hard to justify because it lays idle in applications that do not leverage it. And in low-end or game systems, where every transistor counts, additional rasterization hardware is particularly expensive. An alternative to extra hardware is the reuse of existing hardware, but this option necessarily runs much slower.

* {peercy,airey,cabral}@sgi.com
2011 N. Shoreline Boulevard
Mountain View, California 94043-1389

Shading quality can be increased dramatically with Phong shading [13], which interpolates and normalizes vertex normal vectors at each pixel. Light and halfangle vectors are computed directly in eye space or interpolated, either of which requires their normalization for a local viewer and light. Figure 1 shows rasterization hardware for one implementation of Phong shading, upon which we base this discussion.[1] This adds significant cost to rasterization hardware. However higher quality lighting is almost universally desired in three-dimensional graphics applications, and advancing semiconductor technology is making Phong shading hardware more practical. We take Phong shading and texture mapping hardware as a prerequisite for bump mapping, assuming they will be standard in graphics hardware in the future.

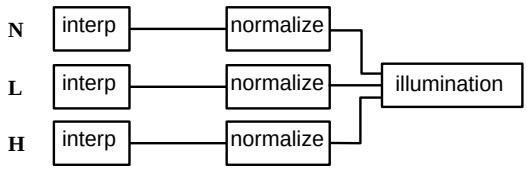

Figure 1. One implementation of Phong shading hardware.

Bump mapping [3] is a technique used in advanced shading applications for simulating the effect of light reflecting from small perturbations across a surface. A single component texture map, $f(u, v)$, is interpreted as a height field that perturbs the surface along its normal vector, $\mathbf{N} = (\mathbf{P}_u \times \mathbf{P}_v)/|(\mathbf{P}_u \times \mathbf{P}_v)|$, at each point. Rather than actually changing the surface geometry, however, only the normal vector is modified. From the partial derivatives of the surface position in the u and v parametric directions ($\mathbf{P}_u$ and $\mathbf{P}_v$), and the partial derivatives of the image height field in u and v (f_u and f_v), a perturbed normal vector $\mathbf{N}'$ is given by [3]:

$$\mathbf{N}' = ((\mathbf{P}_u \times \mathbf{P}_v) + \mathbf{D})/|(\mathbf{P}_u \times \mathbf{P}_v) + \mathbf{D}| \quad (1)$$

where

$$\mathbf{D} = -f_u(\mathbf{P}_v \times \mathbf{N}) - f_v(\mathbf{N} \times \mathbf{P}_u) \quad (2)$$

In these equations, $\mathbf{P}_u$ and $\mathbf{P}_v$ are not normalized. As Blinn points out [3], this causes the bump heights to be a function of the surface scale because $\mathbf{P}_u \times \mathbf{P}_v$ changes at a different rate than $\mathbf{D}$. If the surface scale is doubled, the bump heights are halved. This dependence on the surface often is an undesirable feature, and Blinn suggests one way to enforce a constant bump height.

A full implementation of these equations in a rasterizer is impractical, so the computation is divided among a preprocessing step, per-vertex, and per-pixel calculations. A natural method to support bump mapping in hardware, and one that is planned for a high-end graphics workstation [6], is to compute $\mathbf{P}_u \times \mathbf{P}_v$, $\mathbf{P}_v \times \mathbf{N}$, and $\mathbf{N} \times \mathbf{P}_u$ at polygon vertices and interpolate them to polygon interiors. The perturbed normal vector is computed and normalized as in Equation 1, with f_u and f_v read from a texture map. The resulting normal vector is used in an illumination model.

[1] Several different implementations of Phong shading have been suggested [11][10][4][5][7][2] with their own costs and benefits. Our bump mapping algorithm can leverage many variations, and we use this form as well as Blinn's introduction of the halfangle vector for clarity.

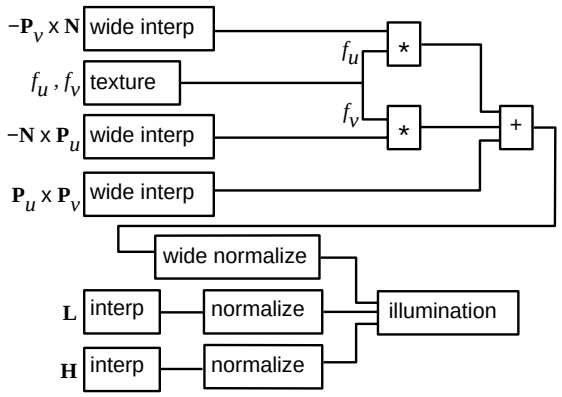

Figure 2. One possible implementation of bump mapping hardware.

Several implementations of this bump mapping method with minor variations are possible; the hardware for one approach based on Figure 1 is shown in Figure 2. Because $\mathbf{P}_u$ and $\mathbf{P}_v$ are unbounded, the three interpolators, the vector addition, vector scaling, and normalization must have much greater precision than those needed for bounded vectors. These requirements are noted in the figure. One approximation to this implementation has been proposed [8], where $\mathbf{P}_v \times \mathbf{N}$ and $\mathbf{N} \times \mathbf{P}_u$ are held constant across a polygon. While avoiding their interpolation, this approximation is known to have artifacts [8].

We present an implementation of bump mapping that leverages Phong shading hardware at full speed, eliminating either a large investment in special purpose hardware or a slowdown during bump mapping. The principal idea is to transform the bump mapping computation into a different reference frame. Because illumination models are a function of vector operations (such as the dot product) between the perturbed normal vector and other vectors (such as the light and halfangle), they can be computed relative to any frame. We are able to push portions of the bump mapping computation into a preprocess or the per-vertex processor and out of the rasterizer. As a result, minimal hardware is added to a Phong shading circuit.

2 OUR BUMP-MAPPING ALGORITHM

We proceed by recognizing that the original bump mapping approximation [3] assumes a surface is locally flat at each point. The perturbation is, therefore, a function only of the local tangent space. We define this space by the normal vector, $\mathbf{N}$, a tangent vector, $\mathbf{T} = \mathbf{P}_u/|\mathbf{P}_u|$, and a binormal vector, $\mathbf{B} = (\mathbf{N} \times \mathbf{T})$. $\mathbf{T}$, $\mathbf{B}$, and $\mathbf{N}$ form an orthonormal coordinate system in which we perform the bump mapping. In this space, the perturbed normal vector is (see appendix):

$$\mathbf{N}'_{TS} = (a, b, c)/\sqrt{a^2 + b^2 + c^2} \quad (3)$$

$$a = -f_u(\mathbf{B} \cdot \mathbf{P}_v) \quad (4)$$

$$b = -(f_v|\mathbf{P}_u| - f_u(\mathbf{T} \cdot \mathbf{P}_v)) \quad (5)$$

$$c = |\mathbf{P}_u \times \mathbf{P}_v| \quad (6)$$

The coefficients a, b, and c are a function of the surface itself (via $\mathbf{P}_u$ and $\mathbf{P}_v$) and the height field (via f_u and f_v). Provided that the bump map is fixed to a surface, the coefficients can be precomputed for that surface at each point of the height field and stored as a texture map (we discuss approximations that relax the surface dependence below). The texel components lie in the range -1 to 1.

The texture map containing the perturbed normal vector is filtered as a simple texture using, for instance, tri-linear mipmap filtering. The texels in the coarser levels of detail can be computed by filtering finer levels of detail and renormalizing or by filtering the height field and computing the texels directly from Equations 3-6. It is well known that this filtering step tends to average out the bumps at large minifications, leading to artifacts at silhouette edges. Proper filtering of bump maps requires computing the reflected radiance over all bumps contributing to a single pixel, an option that is not practical for hardware systems. It should also be noted that, after mipmap interpolation, the texture will not be normalized, so we must normalize it prior to lighting.

For the illumination calculation to proceed properly, we transform the light and halfangle vectors into tangent space via a 3×3 matrix whose columns are $\mathbf{T}$, $\mathbf{B}$, and $\mathbf{N}$. For instance, the light vector, $\mathbf{L}$, is transformed by

$$\mathbf{L}_{TS} = \mathbf{L} \begin{pmatrix} \mathbf{T} & \mathbf{B} & \mathbf{N} \\ \downarrow & \downarrow & \downarrow \end{pmatrix} \quad (7)$$

Now the diffuse term in the illumination model can be computed from the perturbed normal vector from the texture map and the transformed light: $\mathbf{N}'_{TS} \cdot \mathbf{L}_{TS}$. The same consideration holds for the other terms in the illumination model.

The transformations of the light and halfangle vectors should be performed at every pixel; however, if the change of the local tangent space across a polygon is small, a good approximation can be obtained by transforming the vectors only at the polygon vertices. They are then interpolated and normalized in the polygon interiors. This is frequently a good assumption because tangent space changes rapidly in areas of high surface curvature, and an application will need to tessellate the surfaces more finely in those regions to reduce geometric faceting.

This transformation is, in spirit, the same as one proposed by Kuijk and Blake to reduce the hardware required for Phong shading [11]. Rather than specifying a tangent and binormal explicitly, they rotate the reference frames at polygon vertices to orient all normal vectors in the same direction (such as $(0, 0, 1)$). In this space, they no longer interpolate the normal vector (an approximation akin to ours that tangent space changes slowly). If the bump map is identically zero, we too can avoid an interpolation and normalization, and we will have a result similar to their approximation. It should be noted that the highlight in this case is slightly different than that obtained by the Phong circuit of Figure 1, yet it is still phenomenologically reasonable.

The rasterization hardware required for our bump mapping algorithm is shown in Figure 3; by adding a multiplexer to the Phong shading hardware of Figure 1, both the original Phong shading and bump mapping can be supported. Absent in the implementation of Figure 2, this algorithm requires transforming the light and halfangle vectors into tangent space at each vertex, storing a three-component texture map instead of a two-component map, and having a separate map for each surface. However, it requires only a multiplexer beyond Phong shading, avoids the interpolation of $(\mathbf{P}_v \times \mathbf{N})$ and $(\mathbf{N} \times \mathbf{P}_u)$, the perturbation of the normal vector at each pixel, and the extra precision needed for arithmetic on unbounded vectors. Effectively, we have traded per-pixel calculations cast in hardware for per-vertex calculations done in the general geometry processor. If the application is limited by the rasterization, it will run at the same speed with bump mapping as with Phong shading.

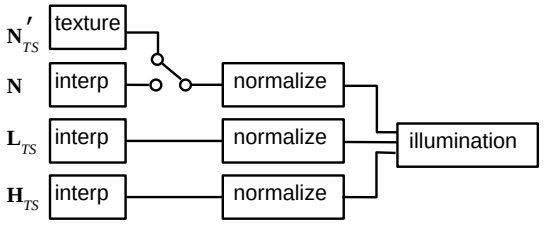

Figure 3. One implementation of our bump mapping algorithm.

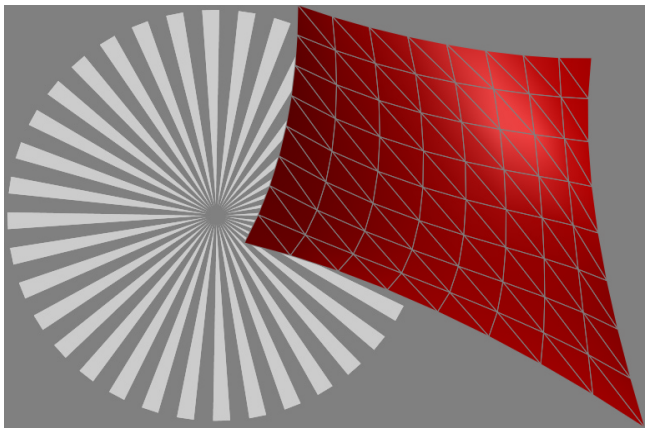

Figure 4.The pinwheel height field is used as a bump map for the tessellated, bicubic surface.

2.1 Object-Space Normal Map

If the texture map is a function of the surface parameterization, another implementation is possible: the lighting model can be computed in object space rather than tangent space. Then, the texture stores the perturbed normal vectors in object space, and the light and halfangle vectors are transformed into object space at the polygon vertices and interpolated. Thus, the matrix transformation applied to the light and halfangle vectors is shared by all vertices, rather than one transformation for each vertex. This implementation keeps the rasterization hardware of Figure 3, significantly reduces the overhead in the geometry processor, and can coexist with the first formulation.

2.2 Removing the surface dependence

The primary drawback of our method is the surface dependence of the texture map. The dependence of the bumps on surface scale is shared with the traditional formulation of bump mapping. Yet in addition, our texture map is a function of the surface, so the height field can not be shared among surfaces with different parameterizations. This is particularly problematic when texture memory is restricted, as in a game system, or during design when a bump map is placed on a new surface interactively.

The surface dependencies can be eliminated under the assumption that, locally, the parameterization is the same as a square patch (similar to, yet more restrictive than, the assumption Blinn makes in removing the scale dependence [3]). Then, $\mathbf{P}_u$ and $\mathbf{P}_v$ are orthogonal ($\mathbf{P}_u \cdot \mathbf{P}_v = \mathbf{T} \cdot \mathbf{P}_v = 0$) and equal in magnitude ($|\mathbf{P}_u| = |\mathbf{P}_v|$). To remove the bump dependence on surface scale, we choose

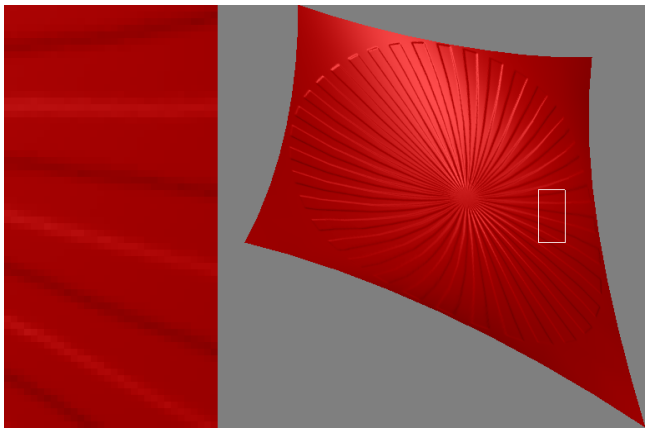

Figure 5. Bump mapping using the hardware implementation shown in Figure 2.

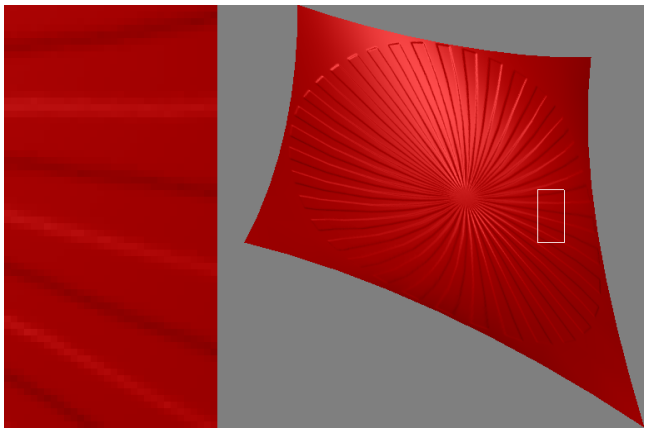

Figure 6.Bump mapping with the hardware in Figure 3, and the texture map from Equations 3-6.

$|\mathbf{P}_u| = |\mathbf{P}_v| = k$, where k is a constant giving a relative height of the bumps. This, along with the orthogonality condition, reduce Equations 3-6 to

$$\mathbf{N}'_{TS} = (a, b, c)/\sqrt{a^2 + b^2 + c^2} \quad (8)$$

$$a = -kf_u \quad (9)$$

$$b = -kf_v \quad (10)$$

$$c = k^2 \quad (11)$$

The texture map becomes a function only of the height field and not of the surface geometry, so it can be precomputed and used on any surface.

The square patch assumption is good for several important surfaces, such as spheres, tori, surfaces of revolution, and flat rectangles. In addition, the property is highly desirable for general surfaces because the further $\mathbf{P}_u$ and $\mathbf{P}_v$ are from orthogonal and equal in magnitude, the greater the warp in the texture map when applied to a surface. This warping is typically undesirable, and its elimination has been the subject of research [12]. If the surface is already reasonably parameterized or can be reparameterized, the approximation in Equations 8-11 is good.

3 EXAMPLES

Figures 5-7 compare software simulations of the various bump mapping implementations. All of the images, including the height field, have a resolution of 512x512 pixels. The height field, Figure 4, was

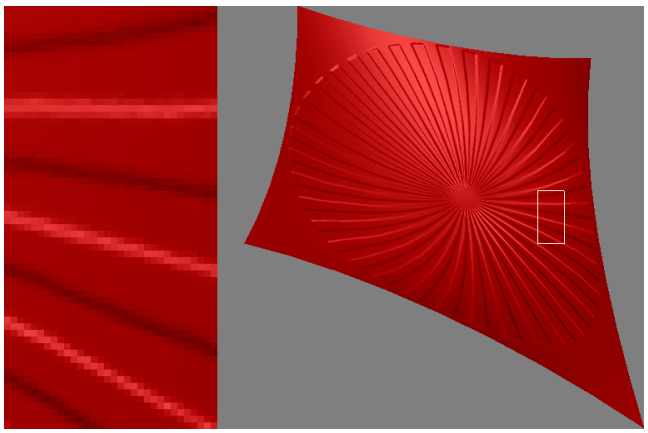

Figure 7.Bump mapping with the hardware in Figure 3, and the texture map from Equations 8-11.

chosen as a pinwheel to highlight filtering and implementation artifacts, and the surface, Figure 4, was chosen as a highly stretched bicubic patch subdivided into 8x8x2 triangles to ensure that $\mathbf{P}_u$ and $\mathbf{P}_v$ deviate appreciably from orthogonal. The texture maps were filtered with trilinear mipmapping.

Figure 5 shows the image computed from the implementation of bump mapping from Figure 2. The partial derivatives, f_u and f_v, in this texture map and the others were computed with the derivative of a Gaussian covering seven by seven samples.

Figures 6 and 7 show our implementation based on the hardware of Figure 3; they differ only in the texture map that is employed. Figure 6 uses a texture map based on Equations 3-6. Each texel was computed from the analytic values of $\mathbf{P}_u$ and $\mathbf{P}_v$ for the bicubic patch. The difference between this image and Figure 5 is almost imperceptible, even under animation, as can be seen in the enlarged insets. The texture map used in Figure 7 is based on Equations 8-11, where the surface dependence has been removed. Minor differences can be seen in the rendered image compared to Figures 5 and 6; some are visible in the inset. All three implementations have similar filtering qualities and appearance during animation.

4 DISCUSSION

We have presented an implementation of bump mapping that, by transforming the lighting problem into tangent space, avoids any significant new rasterization hardware beyond Phong shading. To summarize our algorithm, we

- precompute a texture of the perturbed normal in tangent space
- transform all shading vectors into tangent space per vertex
- interpolate and renormalize the shading vectors
- fetch and normalize the perturbed normal from the texture
- compute the illumination model with these vectors

Efficiency is gained by moving a portion of the problem to the vertices and away from special purpose bump mapping hardware in the rasterizer; the incremental cost of the per-vertex transformations is amortized over the polygons.

It is important to note that the method of transforming into tangent space for bump mapping is independent of the illumination model, provided the model is a function only of vector operations on the normal. For instance, the original Phong lighting model, with the reflection vector and the view vector for the highlight, can be used instead of the halfangle vector. In this case, the view vector is transformed into tangent space and interpolated rather than the halfangle. As long as all necessary shading vectors for the illumination model are transformed into tangent space and interpolated, lighting is proper.

Our approach is relatively independent of the particular implementation of Phong shading, however it does require the per-pixel illumination model to accept vectors rather than partial illumination results. We have presented a Phong shading circuit where almost no new hardware is required, but other implementations may need extra hardware. For example, if the light and halfangle vectors are computed directly in eye space, interpolators must be added to support our algorithm. The additional cost still will be small compared to a straightforward implementation.

Phong shading likely will become a standard addition to hardware graphics systems because of its general applicability. Our algorithm extends Phong shading in such an effective manner that it is natural to support bump mapping even on the lowest cost Phong shading systems.

5 ACKNOWLEDGEMENTS

This work would not have been possible without help, ideas, conversations and encouragement from Pat Hanrahan, Bob Drebin, Kurt Akeley, Erik Lindholm and Vimal Parikh. Also thanks to the anonymous reviewers who provided good and insightful suggestions.

APPENDIX

Here we derive the perturbed normal vector in tangent space, a reference frame given by tangent, $\mathbf{T} = \mathbf{P}_u/|\mathbf{P}_u|$; binormal, $\mathbf{B} = (\mathbf{N} \times \mathbf{T})$; and normal, $\mathbf{N}$, vectors. $\mathbf{P}_v$ is in the plane of the tangent and binormal, and it can be written:

$$\mathbf{P}_v = (\mathbf{T} \cdot \mathbf{P}_v)\mathbf{T} + (\mathbf{B} \cdot \mathbf{P}_v)\mathbf{B} \quad (12)$$

Therefore

$$\mathbf{P}_v \times \mathbf{N} = (\mathbf{B} \cdot \mathbf{P}_v)\mathbf{T} - (\mathbf{T} \cdot \mathbf{P}_v)\mathbf{B} \quad (13)$$

The normal perturbation (Equation 2) is:

$$\mathbf{D} = -f_u(\mathbf{P}_v \times \mathbf{N}) - f_v(\mathbf{N} \times \mathbf{P}_u) \quad (14)$$
$$= -f_u(\mathbf{P}_v \times \mathbf{N}) - f_v|\mathbf{P}_u|\mathbf{B} \quad (15)$$
$$= -f_u(\mathbf{B} \cdot \mathbf{P}_v)\mathbf{T} - (f_v|\mathbf{P}_u| - f_u(\mathbf{T} \cdot \mathbf{P}_v))\mathbf{B} \quad (16)$$

Substituting the expression for $\mathbf{D}$ and $\mathbf{P}_u \times \mathbf{P}_v = |\mathbf{P}_u \times \mathbf{P}_v|\mathbf{N}$ into Equation 1, normalizing, and taking $\mathbf{T}_{TS} = (1, 0, 0)$, $\mathbf{B}_{TS} = (0, 1, 0)$, and $\mathbf{N}_{TS} = (0, 0, 1)$ leads directly to Equations 3-6.

References

[1] AKELEY, K. RealityEngine graphics. In *Computer Graphics (SIGGRAPH '93 Proceedings)* (Aug. 1993), J. T. Kajiya, Ed., vol. 27, pp. 109–116.

[2] BISHOP, G., AND WEIMER, D. M. Fast Phong shading. In *Computer Graphics (SIGGRAPH '86 Proceedings)* (Aug. 1986), D. C. Evans and R. J. Athay, Eds., vol. 20, pp. 103–106.

[3] BLINN, J. F. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)* (Aug. 1978), vol. 12, pp. 286–292.

[4] CLAUSSEN, U. Real time phong shading. In *Fifth Eurographics Workshop on Graphics Hardware* (1989), D. Grimsdale and A. Kaufman, Eds.

[5] CLAUSSEN, U. On reducing the phong shading method. *Computers and Graphics 14*, 1 (1990), 73–81.

[6] COSMAN, M. A., AND GRANGE, R. L. CIG scene realism: The world tomorrow. In *Proceedings of I/ITSEC 1996 on CD-ROM* (1996), p. 628.

[7] DEERING, M. F., WINNER, S., SCHEDIWY, B., DUFFY, C., AND HUNT, N. The triangle processor and normal vector shader: A VLSI system for high performance graphics. In *Computer Graphics (SIGGRAPH '88 Proceedings)* (Aug. 1988), J. Dill, Ed., vol. 22, pp. 21–30.

[8] ERNST, I., JACKEL, D., RUSSELER, H., AND WITTIG, O. Hardware supported bump mapping: A step towards higher quality real-time rendering. In *10th Eurographics Workshop on Graphics Hardware* (1995), pp. 63–70.

[9] GOURAUD, H. Computer display of curved surfaces. *IEEE Trans. Computers C-20*, 6 (1971), 623–629.

[10] JACKEL, D., AND RUSSELER, H. A real time rendering system with normal vector shading. In *9th Eurographics Workshop on Graphics Hardware* (1994), pp. 48–57.

[11] KUIJK, A. A. M., AND BLAKE, E. H. Faster phong shading via angular interpolation. *Computer Graphics Forum 8*, 4 (Dec. 1989), 315–324.

[12] MAILLOT, J., YAHIA, H., AND VERROUST, A. Interactive texture mapping. In *Computer Graphics (SIGGRAPH '93 Proceedings)* (Aug. 1993), J. T. Kajiya, Ed., vol. 27, pp. 27–34.

[13] PHONG, B.-T. Illumination for computer generated pictures. *Communications of the ACM 18*, 6 (June 1975), 311–317.