Hardware Supported Bump Mapping: A Step towards Higher Quality Real-Time Rendering

I. Ernst*, D. Jackèl**, H. Rüsseler*, O. Wittig*

*German National Research Center for Information Technology,
Institute for Computer Architecture and Software Technology (GMD FIRST)
**University Rostock, Hansestadt Rostock Germany

Abstract

Today's high-end Gouraud renderers produce nicely textured scenes by mapping two-dimensional images onto modeled objects in real-time.
We present a renderer which textures surfaces in the normal sense of the word using bump textures to simulate wrinkled or dimpled colorful surfaces. Using a simplified bump mapping method we first succeeded in designing a real-time bump mapping renderer based on the high-quality Phong shading model.
Applying several improvements to our former Phong shading hardware we are able to walk through perspectively correct bump mapped scenes illuminated by colored lightsources.
This paper describes the main building blocks of the overall architecture, including reflectance cubes to support a local viewer, a Taylor-series based division to calculate homogeneous coordinates and our hardware adapted bump mapping method.

Keywords: bump mapping, reflectance cube, normal vector interpolation, hardware division, local viewer, unlimited colored lightsources, Phong shading, real-time rendering

1. Introduction

A contemporary "high-end" graphics workstations has a rendering performance of typically one million polygons per second in conjunction with hardware supported transparency, anti-aliasing, perspective texture mapping, image compositing, etc. In spite of the fact that great steps have been taken to improve the realistic display of very complex objects and scenes, many problems remain to be solved in order to achieve

- image generation rate of complex scenes (>500.000 polygons) of at least 30-50 frames per second
- more photorealistic image generation by means of hardware supported "normal vector shading" as well as improved „anti-aliasing" or „bump mapping".
- linear scaleability of the rendering performance and an improvement of the cost-performance relationship.

In the past decade, many attempts have been undertaken to make „normal vector shading" applicable for hardware implementations. M. Deering et al.[6] presented a simulated VLSI-approach of a *normal vector shader* (NVS) chip suitable for Phong shading [4]. Bishop and Weimar [3] suggested a method which approximates Phong´s normal interpolation and light reflection equation by a Taylor series expansion. Bishop and Weimar´s *fast Phong shading* reduces the number of computation steps per pixel to only 5 additions and one memory access. Another alternative, suggested by U. Clausen [5], interpolates polar angles instead of normal vectors, which has the advantage that the normal vector unit length remains unchanged during the interpolation. Our normal-vector shading approach [11] is based on the approximation of the illumination model geometry by using the *reflection map* method. This shading technique, which is used since 1989 in our VISA-systems (VISA=VISualization Accelerator), operates similar to the reflection vector shading hardware of Voorhies and Foran [13]. Instead of the single precalculated map of the VISA-shader, Voorhies and Foran propose a six-sided cube, which incorporates the specular spread function values. Similarly to the reflection map method, each face of the cube is indexed by an unnormalized reflection vector.

An additional step towards photorealistic image generation can be achieved by means of bump-mapping. Half of the work of hardware supported bump-mapping is done when using a normal vector shader instead of a gouraud shader. This paper is focus on the extension of our normal vector shading method for achieving a simple and efficient hardware bump-mapping support.

First, we discuss the principle of hardware Phong-shading. In section 3 the extension of this shading technique to achieve bump-mapped surfaces is presented. Finally, the hardware implementation aspect of our bump-mapping method and some simulation results are discussed in more detail.

2. Hardware Phong Shading

Phong shading as proposed in [11], uses a planar reflectance map to "pick" the intensity.

The "picking" of the reflectance map is done as follows:

$$\alpha_1 = \operatorname{atan}\left(\frac{x_n}{z_n}\right), \alpha_2 = \operatorname{atan}\left(\frac{y_n}{z_n}\right)$$

Considering the simplified geometry using parallel light and parallel viewer the amount of light reflected by an object's surface is given by $r = f(\alpha_1, \alpha_2)$.

There are some disadvantages to our former approach using a reflectance map:

- visible artifacts caused by point sampling the reflectance map;
- visible distortions at the borders of the atan map (angle > 80 degrees);
- illumination model allows parallel viewer only;
- costly computation for reflectance map if viewer changes.

Solutions to reduce the size of the planar maps are given by for example in [12], [10]. To avoid distortions in the border area of the map, it is straightforward to use a hemi-cube over one polygon. This algorithm is commonly used for radiosity methods and to approximate normal vector directions. Distortions which occur at the corners of the hemi-cube will be reduced by using an arctan map in the range of -45 to 45 degrees, similar to the reflectance map approach. Bilinear interpolation of the four neighboring cube elements reduces point sampling artifacts.

Despite the progress on planar or hemi-cube maps the restriction to parallel viewers could not be alleviated. One reason for this is that all progress is focused on the use of normal vectors to address cube faces. In [8], [13] it is proposed to use the reflection of the eye vector on the surface to address a six sided cube. Since the reflection of the viewer is closely related to the specular component of the Phong shading algorithm, this method can also be applied to shading. Furthermore a six-sided cube contains all possible reflection vector directions around a fixed normal vector, the viewer can be local. As a consequence, this method allows for the view point to be changed without recomputation of the cube map.

As indicated in [13] it is possible to calculate the reflection vector R_U without normalization of the surface normal N_U and the eye vector by multiplying both sides by the length of the unnormalized normal vector squared:

$$R_U = 2 \cdot N_U \cdot (N_U \bullet E) - E \cdot (N_U \cdot N_U) \quad \text{(EQ 1)}$$

To calculate the diffuse component, another six sided cube map can be used which can be indexed by the unnormalized normal vector similar to the hemi cube approach. Calculating colors at the polygon vertices and bilinearly interpolating these colors over the polygon is not sufficient because highlights in the middle of a triangle cannot be generated. To reduce cube map space and calculation time methods proposed in [12], [10] are used. In Figure 1 the indexing of the cube's faces is illustrated.

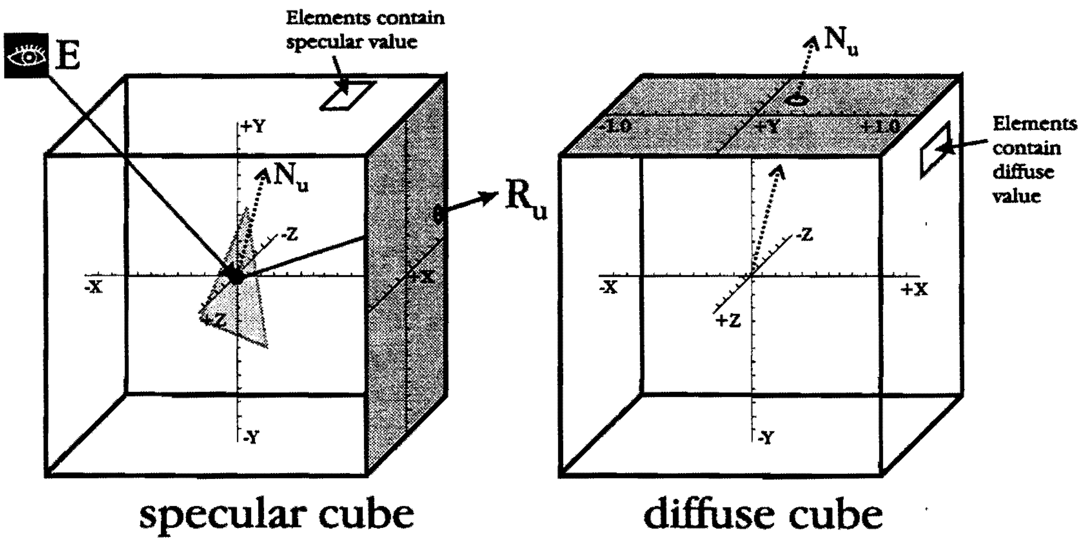

Figure 1: Application of cube faces to calculate phong illumination

3. Bump mapping

Bump mapping is an elegant technique to simulate wrinkled or dimpled surfaces without the need to model surface properties geometrically.

This is done by angular perturbing the surface normals according to information given in a two dimensional bump map.

Since intensity is mainly a function of the surface normal, the local reflection model produces local variations on a smooth surface dependent on the perturbation defined in the bump map.

Bump mapping is important because it textures a surface in the normal sense of the word rather than modulating the color of flat surfaces.

3.1. Classic Bump mapping

Blinn [1] first developed a scheme that perturbs the normal vector independently of the orientation and position of the surface. In the case of animations, otherwise it is obvious that the normal at a particular point must always receive the same rate of perturbation. Otherwise the bump map detail moves as the object moves. This is achieved by aligning the perturbation on a coordinate system based on local derivatives.

If $O(u, v)$ is a function representing position vectors of points O on the surface of an object, we first add a small increment, derived from the bump map $\mathrm{B}(\mathrm{u}, \mathrm{v})$ to define $O'(u, v)$:

$$O'(u, v) = O(u, v) + B(u, v)\frac{N}{|N|}$$

Because the displacement function B is small compared with its spatial extend by differentiating we get:

$$\mathrm{N}'(u, v) = O_u \times O_v + B_u\left(\frac{N}{|N|} \times O_v\right) + B_v\left(O_u \times \frac{N}{|N|}\right) + B_u B_v\left(\frac{N \times N}{|N|^2}\right) \qquad \text{(EQ 2)}$$

The last term $B_u B_v\left(\frac{N \times N}{|N|^2}\right) = 0$ and can be dropped.

Note that the normal to a surface at a point is given by $N = O_u \times O_v$ where O_u and O_v are the partial derivatives of the surface at point O lying in the tangent plane (see Figure 2).

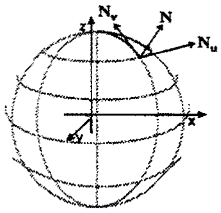

Figure 2: Tangent vectors O_u, O_v at a point on a sphere with normal vector N.

From equation (EQ 2) it can be seen that implementing bump mapping in hardware is not possible without simplifying the calculation scheme.

3.2. Hardware adapted Bump mapping

Using Phong vertex normal interpolation, we already know the surface normal at a point. This surface normal N represents the orientation of the surface independent of the rotation about N.

Our new renderer VISA II will use a bump mapping scheme closely related to texture mapping.

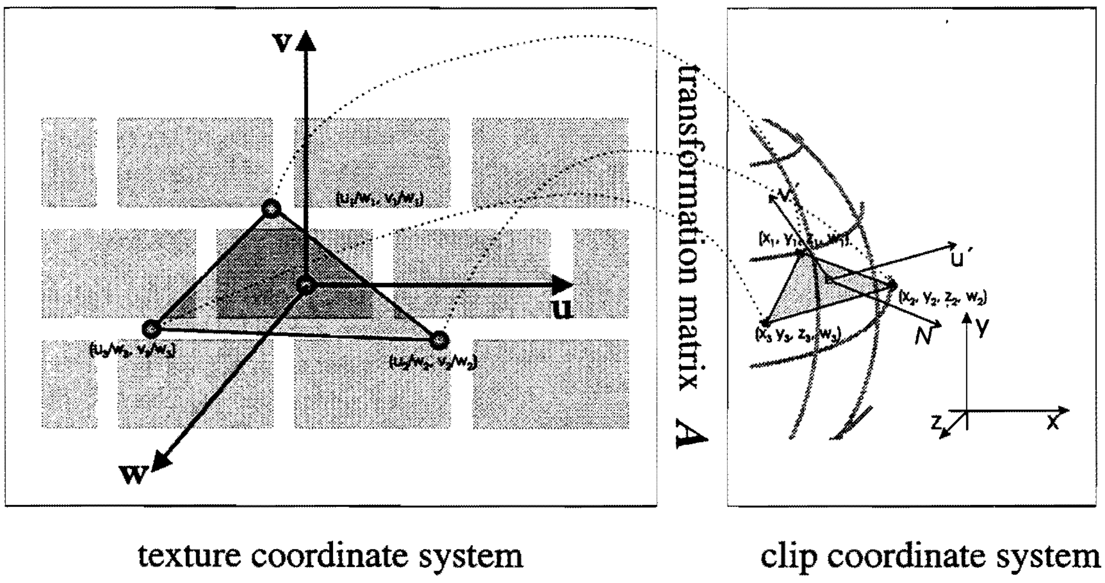

Figure 3: Mapping from clip space to texture space

The bump map describes the perturbation Δu, Δv of normal $[0, 0, 1]^T$ in texture space u, v, w.

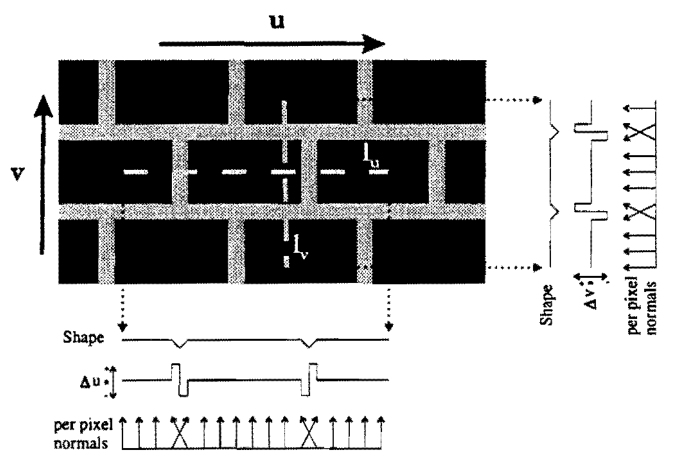

Figure 4: Effect of bump map on surface shape

Figure 4 shows a precomputed bump map to simulate brick structures. Notice the u and v projections of line l_u and l_v. The first line (*shape)* depicts surface orientations of the cut along l_u, l_v on the brick structure. The second line shows the displacement Δu, Δv. Positive Δu, Δv values represent a displacement of the underlying pixel normal in direction of u, v; negative values represent opposite directions. The third line shows the resulting surface shape and its perturbed surface normals. Notice the effect of the displacement between two brick stones to simulate the groove.

Transformation of the bump map onto an object is done similar to texture mapping. By using homogenous coordinates the resulting address $\left(\frac{u}{w}, \frac{v}{w}\right)$ is the perspectively correct pixel index on the bump map.

To guarantee a perturbation in the desired direction u, v, the object coordinate system must be transformed such that the w-axis equals the surface normal vector direction N and the u- and v- coordinate axis are rotated about N according to the polygon mapping defined by the modeler.

Figure 3 illustrates the mapping of the polygon $[x_1, y_1, z_1, w_1]^T$, $[x_2, y_2, z_2, w_2]^T$, $[x_3, y_3, z_3, w_3]^T$ in the clip coordinate system x, y, z, w to the edges $\left[\frac{u_1}{w_1}, \frac{v_1}{w_1}\right]^T$, $\left[\frac{u_2}{w_2}, \frac{v_2}{w_2}\right]^T$, $\left[\frac{u_3}{w_3}, \frac{v_3}{w_3}\right]^T$ in the texture coordinate system u, v, w.

The transformation matrix A carries out this mapping by projecting the local polygon coordinate system u', v', N on the texture coordinate system u, v, w.

To obtain the perturbed per pixel normal $N' = \left[n'_x, n'_y, n'_z\right]^T$, $\Delta u, \Delta v$ transformed by A is added to the surface normal $N = \left[n_x, n_y, n_z\right]^T$:

$$
\begin{aligned}
n'_x &= n_x + \Delta u \cdot A_{00} + \Delta v \cdot A_{01} \\
n'_y &= n_y + \Delta u \cdot A_{10} + \Delta v \cdot A_{11} \\
n'_z &= n_z + \Delta u \cdot A_{20} + \Delta v \cdot A_{21}
\end{aligned}
$$

The resulting normal is used to calculate the pixel color in the Phong illumination model described in the previous sections.

4. Proposed Architecture

The scan-converting procedure for triangles is described in detail in [11]. Given the triangle's start coordinate with its start attributes and slope increments, the *Scan Line Initializer* computes the relevant data at the start point of every new scanline, while the *Pixel Calculator* interpolates the coordinates and attributes within a scanline. This differs from the former design in the interpolation of an extended data set. In particular, we extend the interpolator to homogenous x, y, z, w space, which is necessary for achieving perspective texture mapping [9], texture-, bump-coordinates (u, v), eye vector $[e_x, e_y, e_z]^T$.

After rasterization, the obtained pixel fragment describes the position and orientation of small surface elements in 3D-space, the direction of reflection, the $\frac{u}{w}$, $\frac{v}{w}$ texture and bump coordinates.

Figure 5 shows a sketch of the proposed architecture.

The non-shaded boxes will be discussed in more detail. The operation principle of the interpolation-units is presented in [11]. To visit the pixels of adjacent triangles only once, the edge traverser was slightly modified [7]. This is essential for color blending.

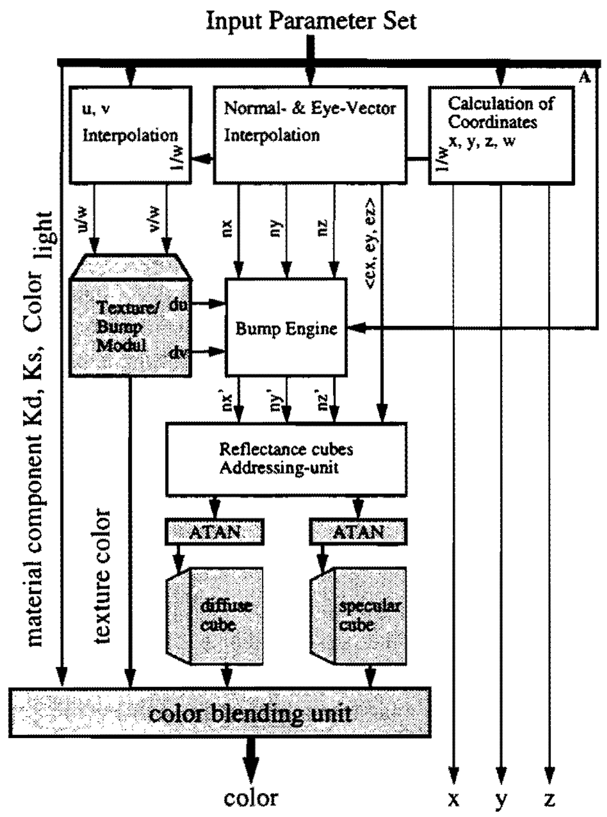

Figure 5: Overall blockdiagram of the rendering engine

The bump engine perturbs the pixel normal by the corresponding bump map entry $\Delta u, \Delta v$. The bump map is aligned perpendicular to the surface normal. This is done by multiplying the bump map entries by A:

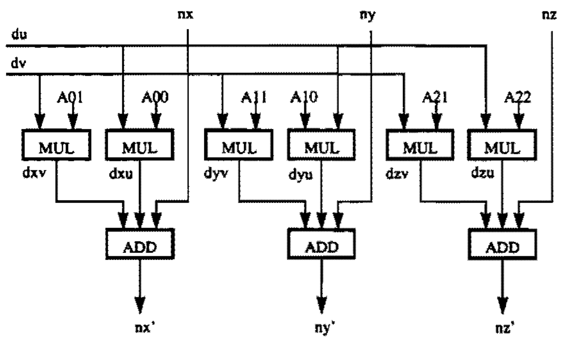

Figure 6: Bump Engine

$$\vec{N'} = \vec{N} + \left[\Delta u, \Delta v\right] \cdot A$$

The reflectance cubes are addressed by the normal for the diffuse color component or by the reflected ray for the specular component. The address calculation unit for the reflected ray can be found in [13]. The vector's major axis determines the corresponding reflectance cube face. Indexing this map is done by dividing the vector's minor axis values by the major axis value.

Selection of reflection map:

$$\left[n_x, n_y, n_z\right] \rightarrow \max\{|n_x|, |n_y|, |n_z|\}, \operatorname{sign}(n_x, n_y, n_z)$$

Address function:

$$X-\text{faces:}\left[1, \frac{n_y}{|n_x|}, \frac{n_z}{|n_x|}\right]$$

$$Y-\text{faces:}\left[\frac{n_x}{|n_y|}, 1, \frac{n_z}{|n_y|}\right],$$

$$Z-\text{faces:}\left[\frac{n_x}{|n_z|}, \frac{n_y}{|n_z|}, 1\right]$$

4.1. Division with on chip ROM

Perspective correct texturing and calculating the intersections of the normal and reflection vectors with the cube faces need one division per pixel. A common approach is to build the reciprocal $\frac{1}{w}$ followed by a multiplication. Convergence division algorithms normalize the divisor to a positive interval, typically $0{,}5 \le w < 1$; positive numbers are shifted up or down by a priority encoded shifter code. Negative numbers are first converted to signed positive values (complement +1). The result $\frac{1}{w}$ is corrected by the exponent of the normalization.

In the following we focus on the computation of the reciprocal $\frac{1}{w}$ by Taylor-series approximation [14] about $w = w_h$. After the normalization step, we can write:

$$w = 0{,}1w_2w_3\ldots w_{m-1}w_m\ldots w_{n-1}w_n$$

$$w_h = 0{,}1w_2w_3\ldots w_{m-1}w_m1\ldots1.$$

With $\Delta w = w - w_h \le 0$, one can develop

$$\frac{1}{w} = \frac{1}{w_h} + \frac{\Delta w}{w_h^2} + \frac{\Delta w^2}{w_h^3} + \frac{\Delta w^3}{w_h^4} + \ldots .$$

For a reasonable hardware realization, the number of terms in the sum has to be small and the on-chip ROM implementation has to be memory efficient.

Truncating after the third term of the sum and using B as the output of the look up table approximating $\frac{1}{w_h}$, one can rewrite the above formula:

$$\frac{1}{w} = B + B^2 \cdot \Delta w + B^3 \cdot \Delta w^2 + error$$

$$= B \cdot (1 + B \cdot \Delta w \cdot (1 + B \cdot \Delta w)) + error$$

The error is a function of the number of sum-terms t, the number of look-up-table entries $m - 1$ and b the number of bits per word B. (Remember: m are the relevant bits of $w_h = 0{,}1w_2w_3\ldots w_{m-1}w_m11$)

Figure 7 shows the division by the homogenous factor w. With technology currently available, first simulations estimate an area of ca. 25mm^2 for this unit. The division of the vector's major axis needs less accuracy, resulting in a smaller silicon area.

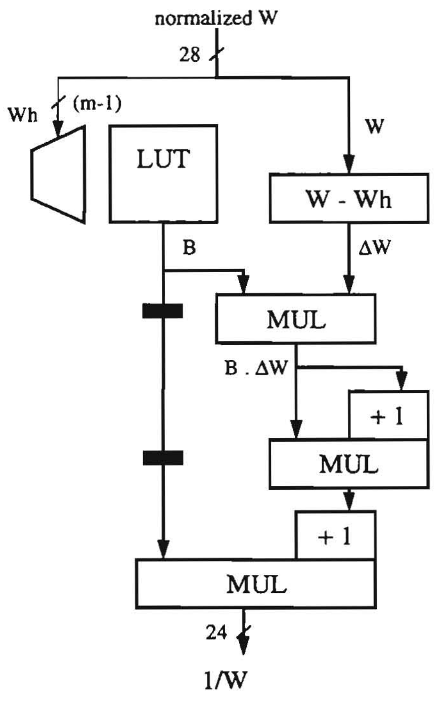

Figure 7: Divider for 1/w

5. Results

One limitation of the bump mapping method described above is that all normals on a polygon are perturbed relative to the surface normal direction. To calculate the correct perturbed normals for each per pixel normal, recalculation of transformation matrix A for each pixel is necessary. Although the perturbation of pixel normals varying from the surface normal is not correct, it causes only a slight movement of the centroid of the highlight.

In Figure 8, spherical interpolation of a per pixel normal (black) is shown. Given the shown lightsource and the viewer directions the centroid of the highlight lies exactly in the middle of the polygon. Applying linear interpolation (gray) the normal pointing exactly to the lightsource and the viewer is located slightly on the left causing the highlight to move to the left. In general this effect is not visible in animated scenes.

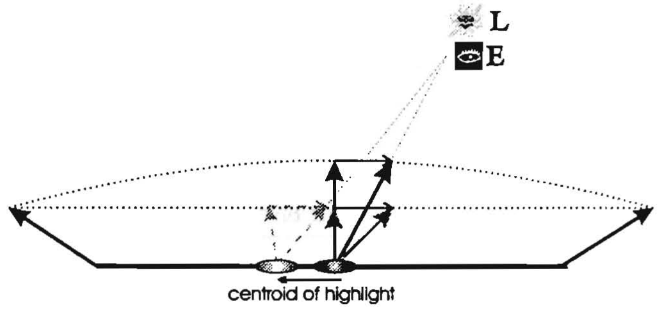

Figure 8: Spherical interpolation versus linear interpolation

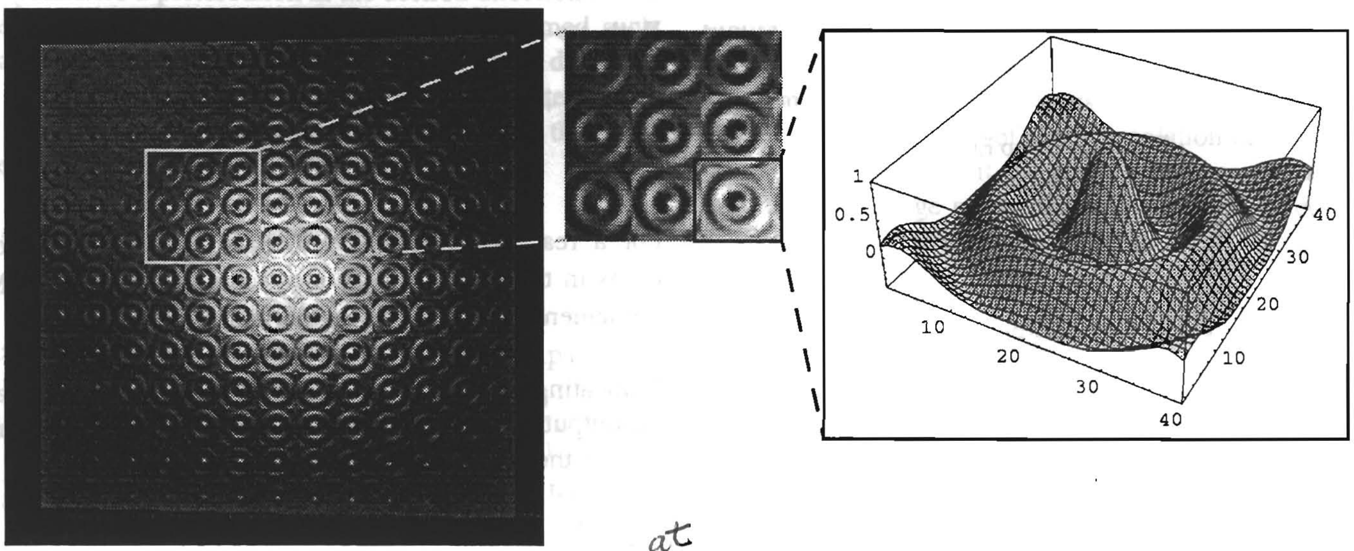

Figure 9: Mexican ~~Head~~ at Structure carved in tin plane

Another disadvantage caused by the calculation of A relative to the surface normal is shown in Figure 10:

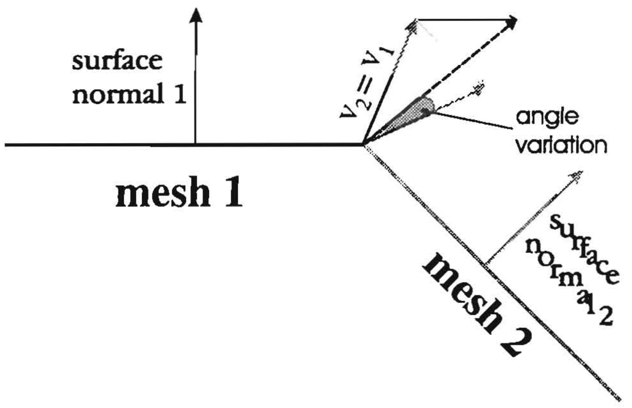

Figure 10: False illumination at edge of neighboring polygons.

Simulating extreme curvature of adjacent polygon meshes by modeling vertex normals leads to machband effects at the edges. The vertex normal v_1 of mesh 1 is perturbed dependent on the surface normal (dashed normal). The same rate of perturbation is applied to v_2 of mesh 2 perpendicular to the surface normal of mesh 2 (dashed gray normal). Assuming the vertex normal direction to be the perfect mirror direction, slight variations of vertex normals v_1 and v_2 generate different shading intensities. These shading intensity variations can be seen clearly along the edge revealing triangulation structures.

Therefore the modeler's responsibility is to avoid extreme curvature simulations or to use finer triangulation of these scene parts

Figure 9 shows a bluish tin plane illuminated by a purple (upper left) and a white (front) lightsource. The plane is assembled out of two bump textured triangles. A Mexican head function is used to calculate the normal perturbation for a 40x40 bump map (see Figure 9 right). Applying the bump map to the plane repeatedly leads to the desired carving. The zoomed patch in the middle of Figure 9 clearly shows correct purple and white highlights given by the illumination geometry.

In Figure 11 the creation of complex images using conventional textures and bump textures is shown. In our example we rebuilt a small part of the "Berlin Wall" painting a graffiti texture (Figure 11 middle). To simulate carving and rotting of the wall we created a greyscale image which was converted to a bump map. Notice the writing "Berlin Haup(t)stadt" chiseled in the wall (see Figure 11 left (bump sketch) and right (resulting image)). An animation of the "Berlin Wall" example scene generates the impression of moving sky trackers over the rotten wall.

6. Conclusions and Future Work

We have designed and simulated a real time renderer supporting unique features like Phong shading and bump textures. With the presented renderer we are capable of generating animated scenes of higher quality then with conventional gouraud shading renderers.

This technique could be improved further by:

- techniques to simulate spherical interpolation of unnormalized normal vectors to avoid slight translations of highlights on polygons;
- new hardware adapted methods to avoid artifacts caused by angle variations of neighboring polygons;
- artifact reduced MIP mapping for bump textures;
- implementation of "living" textures;

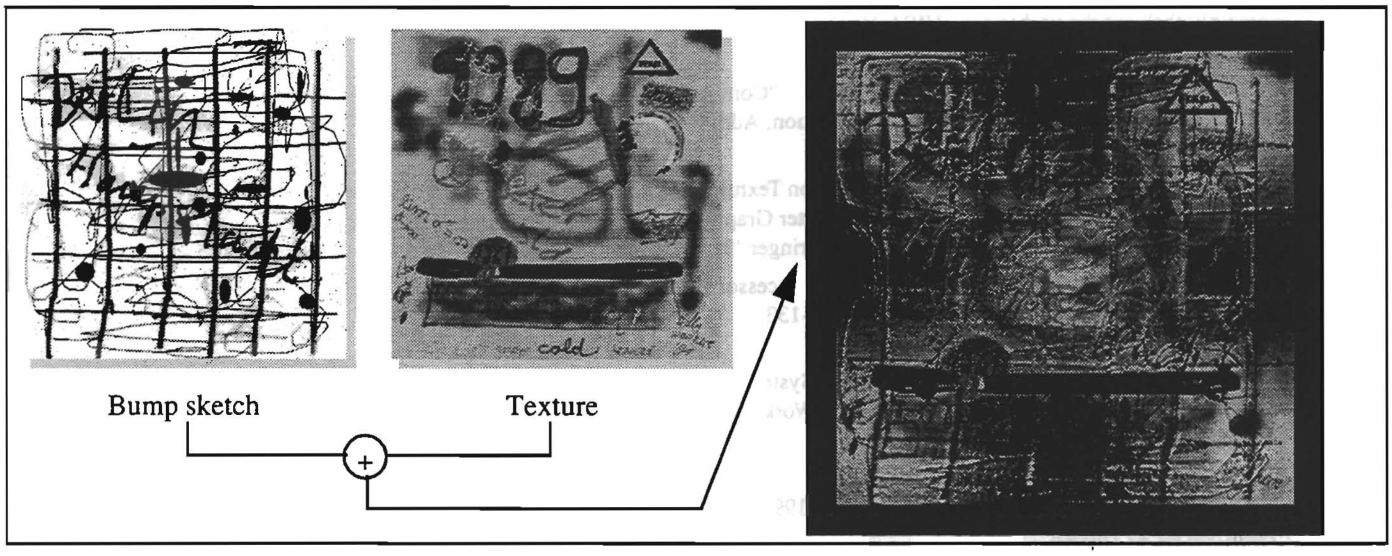

Figure 11: Simulation of graffiti on rotten wall.

- specification of a set of extensions for the OpenGL graphics language to support our unique graphic features.

Furthermore, it is desirable to switch between conventional texturing, bump texturing, reaction diffusion texturing and displacement mapping [2].

7. Acknowledgments

We would like to thank the following people for their suggestions and contributions to this work: J. Duenow (scanliner and parameter generation), L. Guangming (error analysis on division algorithms), T. Le Vin (Verilog simulation of arithmetic units, including silicon real estate estimation), and S. Budianto (Register Transfer Level Simulation). In addition, we would like to thank the many people who have commented on this paper.

Literature

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

[2] B.G. Becker, N.L. Max, "Smooth Transitions between Bump Rendering Algorithms", SIGGRAPH '93, pp 183 - 190

[3] G. Bishop, D. M. Weimar, "Fast Phong Shading", Computer Graphics, Vol. 20, No. 4 , pp. 103 - 106, 1986

[4] Phong, Bui Thong. "Illumination for Computer Generated Pictures". *Communications of the ACM*, Vol. 18, No. 6 (1975), pp. 311-317

[5] U. Clausen, "Reducing the Phong Shading Method", Proc. Eurographics 89, Eds. W. Hansmann, F.R.A Hopgood, W. Straßer; pp. 333 - 344, North-Holland, 1989

[6] M. Deering, S. Winner, B. Schediwy, C. Duffy, N. Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics", Computer Graphics, Vol. 22, No. 4; pp 21-30, 1988

[7] J. Duenow, "Strategien zur Abbildung der OpenGL Softwarebibliothek auf die vorhandene VISA Hardware", internal Report, GMD First, Spring 1995

[8] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes: "Computer Graphics: Principles and Practice", 2nd Edition, Addison-Wesley, 1990

[9] Heckbert, H. P. Moreton; "Interpolation for Polygon Texture Mapping and Shading", in State of the Art Computer Graphics: Visualization and Modeling; pp 101 - 111, Springer '91

[10] T. Ikedo, "A scalable high-performance graphics processor: GVIP" , The Visual Computer (1995), 11 pp. 121-133, Springer 1995.

[11] D. Jackèl, H. Rüsseler, "A Real Time Rendering System with Normal Vector Shading", 9th Eurographics Workshop on Graphics Hardware, Oslo, Norway, 1994, pp 48-57

[12] J. T. van Scheltinga, J. Smit, M. Bosma, "Design of an on-chip reflectance map" , EG Hardware Workshop 1995, Maastrich, 28-29 August

[13] D. Voorhies and Jim Foran, "Reflection Vector Shading Hardware", Siggraph Proceedings 1994, pp 163-166

[14] D. Wong, M. Flynn, "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations", IEEE TRANSACTION ON COMPUTERS, VOL.41, NO. 8, pp. 981 - 995, August 1992