Design Principles of Hardware-based Phong Shading and Bump Mapping

K. Bennebroek[&], I. Ernst[§], H. Rüsseler[§], O. Wittig[§]

[§]German National Research Center for Computer Science,
Institute for Computer Architecture and Software Technology (GMD FIRST)
[&]University of Twente, department of Electrical Engineering,
Networking Theory Group, the Netherlands

Abstract

The VISA+ hardware architecture is the first of a new generation of graphics accelerators designed primarily to render bump-, texture-, environment- and environment-bump-mapped polygons. This paper presents examples of the main graphical capabilities and discusses methods and simplifications used to create high quality images. One of the key concepts in the VISA+ design, the use of reflectance cubes, is predestined for environment mapping. In combination with bump- and texture-mapping it shows the strength of our new architecture.
Furthermore it justifies some of the decisions made during simulation and development of the complex VISA+ architecture.

Keywords: real-time bump mapping, environment mapping, environment bump mapping, reflection map, normal vector interpolation, Phong shading hardware.

1. Introduction

Even the fastest high quality graphics workstations like the recently announced InfiniteReality™ graphics accelerators are still based on common texturing in combination with Gouraud Shading. With continuously improving VLSI technologies in combination with CAD tools to support full custom chip design, new graphic architectures must evolve. As an important step towards higher realism, real-time Phong Shading will be the next technology-push in computer graphics.

Over the past decades, many attempts have been made, to make Phong shading practicable for hardware implementation. Bishop and Weimar [1] proposed a Taylor series approximation for $N.H$. For a curvature less than 60 degrees, forward differencing a quadratic polynomial fits quite well and reduces the computational efforts to five additions per pixel. Deering [2] presented a shader that interpolates the normal and eye vectors. In both approaches, exponentiation of the cosine function was done by table lookup, which leads to intolerable hardware size if a broad range of exponents is to be supported.

Another method uses angular interpolation techniques rather than vector interpolation. This has the advantage that the vector length remains unchanged during the interpolation. This way, normalization can be reduced to vectors at the vertices and calculated in software. While an approach suggested by U. Clausen [3] interpolates polar angles, Kuijk and Blake [4] expand this principle to angular interpolation on great-circles. The resulting expressions, however, are rather complicated. The angular interpolation approach can be important if diffuse intensities are calculated on an incremental basis.

The normal-vector shading approach of the GMD-FIRST is a reflection map method. This shading technique, implemented in the VISA system [5] operates in a similar way to the reflection-vector shading hardware of Voorhies and Foran [6].

A logical extension of the normal vector interpolation principle was to support bump mapping [9]. Bumps improve visual effects for low-end game, as well as high-end VR applications. A combination of texture and bump maps for extremely perspective and huge polygons reduces the total setup- and transfer-time for computer graphics scenes.

In this paper, the quality aspects of homogeneous versus non-homogeneous interpolation for Phong shading, bump mapping, environment and environment bump mapping, will be discussed. Focusing on efficient division schemes for perspective texturing and cube-reflectance mapping, a proposed hardware architecture is discussed in more detail.

2. Basic Fundamentals

2.1 Phong Shading

Phong Illumination was introduced by Tong Bui Phong in 1975 [7] as an empirical model to simulate highlights on shiny surfaces. The model has been accepted as a very good approximation of physical highlights. Phong introduced a specular exponent, reflecting material properties, and modelled the steep falloff of the highlight as the cosine of the angle between eye-vector and reflected light-vector, raised to the specular exponent. For the implementation of Phong Illumination on a per pixel basis, the normal vector has to be interpolated over a polygon. Normal Vector Interpolation Shaders are therefore called Phong Shaders, in contrast to the commonly used Gouraud Shaders that interpolate color-values over the polygon (that may have been calculated using Phong Illumination!). Phong's Illumination model consists of an ambient, a diffuse and a specular component, resulting in:

Equation 2-1:

$$I = a + \sum_i (d(N \bullet L) + s(R \bullet E)^n)$$

for all lightsources *i* in the scene, with ***a***, ***d*** and ***s*** ambient, diffuse and specular column-vectors $(R,G,B)^T$ respectively to depict material properties. ***N*** depicts the surface normal vector, ***L*** the vector pointing from the point on the surface to the light source, ***R*** the mirrored vector of ***L*** around ***N*** (reflected light vector) and ***E*** the reflected eye vector. All vectors need to be normalized.

2.2 Bump Mapping

In 1974, Blinn [8] developed a technique that enables a surface to appear wrinkled or dimpled without the need to model this roughness geometrically.

For this purpose, he defined a special texture map called bump map $B(u,v)$ containing displacement or height values to perturb the local normal vector.
The perturbed normal vector is given as $N' = N + D$ where D is the perturbation vector. To calculate D, he defined two other vectors lying in the tangent plane of the surface given by N. These vectors are defined as $\mathbf{A} = N \times O_v$ and $\mathbf{B} = N \times O_u$, where $O(u,v)$ is a parametrized function representing the position vectors of point O on the surface. Then, the components of D are given by $A = B_u\mathbf{A} - B_v\mathbf{B}$. This way, he defined a bump map function as a displacement but uses its derivatives at the point (u,v) to calculate D. D can be better expressed by

Equation 2-2:

$$D = B_u(N \times O_v) - B_v(N \times O_u)$$

where $N / |N|$ is written as N.

3. Simplifications

3.1 Hardware Bump Mapping

Hardware bump mapping needs to be much simpler than the classical approach (Equation 2-2).
Instead of constructing a perturbation vector out of local derivatives which have to be defined, we already store it in the bump map. Since we interpolate the normal over one triangle, we already have the tangent plane of the surface. Given the perturbation vector $P_i = [\Delta u \quad \Delta v]^T$ in WC, we might construct the resulting normal vector as $N' = N + P$. To guarantee a perturbation in the direction O_u, O_v of the tangent plane at point *i*, we have to
align the texture coordinate system u, v, w to the tangent plane system O_u, O_v, N. This transformation is done by an alignment matrix **A**. Then the resulting normal vector becomes $N' = N + \mathbf{A} \cdot P$. Since we have a homogeneous interpolated normal vector, the equation expands to

Equation 3-1

$$\frac{N'}{w} = \frac{N}{w} + \left[\mathbf{A} \cdot \frac{P}{w}\right] * BumpMorph.$$

Note the additional *BumpMorph* parameter which is used to control the amplitude of the bump perturbation (length of P). Since this parameter is constant over the polygon, it can be included in the matrix **A**. This way, Equation 3-1 can be simplified to:

Equation 3-2

$$\frac{N'}{w} = \frac{1}{w}(N + \mathbf{A}P)$$

To convert P_i from WC to ScreenSpace we need two multiplications for each of its components by $\frac{1}{w}$. However in section 3.2 we explain why we have to calculate the reflected ray in WC instead of ScreenSpace coordinates. Hence we must convert the normal N from ScreenSpace to WC too. This is done by three divisions ($\frac{1}{w}$).

The resulting formula, which is implemented in our hardware, than becomes:

Equation 3-3:

$$N' = N + [\mathbf{A} \cdot P]$$

Note that N is generated by interpolating the normalized edge normal vectors of the triangle. Hence N is unnormalized inside the triangle which leads to small highlight shifts on the surface. This artifact can not be recognized because there are no color differences near the edges of adjacent triangles [9].

The alignment matrix **A** can be used not only to get the correct perturbation vector, but also to rotate, stretch, mirror or shorten the perturbation vectors for one triangle.

3.2 Hardware Reflected Ray

Based on the $M \bullet L$ model [10], with homogeneous eye vector $\overline{E} = \frac{E}{w}$ and surface normal vector $\overline{N} = \frac{N}{w}$, the homogeneous reflected ray in ScreenSpace coordinates leads to

Equation 3-4:

$$\overline{R} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix} =$$

$$2N(NE) - (NN)E =$$

$$2\frac{1}{w}N(\frac{1}{w}N\frac{1}{w}E) - (\frac{1}{w}N\frac{1}{w}N)E =$$

$$2\frac{1}{w^3}N(NE) - \frac{1}{w^3}(NN)E =$$

$$\frac{1}{w^3}(2N(NE) - (NN)E) = \frac{1}{w^3}R$$

Since calculating R in ScreenSpace leads to higher precision multipliers and adders (extra

$$\overline{R} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix} =$$

bits for homogenous component), we convert E and N to WC to reduce the overall gatecosts for the Reflected Ray Unit.

3.3 Hardware Phong

There are two ways to calculate the color contribution of each lightsource.

⇒ Using reflection maps (VISA)
N and R are used as an index into the diffuse and specular reflection map respectively.
Diffuse:
$\begin{bmatrix} n_x & n_y & n_z \end{bmatrix} \rightarrow \max\{|n_x| \quad |n_y| \quad |n_z|\}$, $\mathrm{sign}(n_x \quad n_y \quad n_z)$

$$MapAddress = \begin{bmatrix} a_x & a_y \end{bmatrix} =$$

$$X-Faces: \begin{bmatrix} \frac{n_y}{|n_x|} & \frac{n_z}{|n_x|} \end{bmatrix}$$

$$Y-Faces: \begin{bmatrix} \frac{n_x}{|n_y|} & \frac{n_z}{|n_y|} \end{bmatrix}$$

$$Z-Faces: \begin{bmatrix} \frac{n_x}{|n_z|} & \frac{n_y}{|n_z|} \end{bmatrix}$$

Note that the homogeneous perturbed normal is $\frac{N'}{w}$ (see Equation 3-1) which in the case of *X-Faces* leads to

Equation 3-5

$$X-Faces: \begin{bmatrix} 1 & \frac{\frac{n_y}{w}}{\left|\frac{n_x}{w}\right|} & \frac{\frac{n_z}{w}}{\left|\frac{n_x}{w}\right|} \end{bmatrix} = \begin{bmatrix} 1 & \frac{n_y}{|n_x|} & \frac{n_z}{|n_x|} \end{bmatrix}$$

As we can see, the division by w is canceled out. This means that no conversion from and to homogeneous coordinates is necessary when indexing the maps.

Specular:
The specular index is calculated analogously to the diffuse index using R.

Equation 3-6

$$X-Faces: \begin{bmatrix} 1 & \frac{r_y}{|r_x|} & \frac{r_z}{|r_x|} \end{bmatrix}$$

For example the a_x index expands to

$$a_x = \frac{\dfrac{e_y \cdot \left(-n_x^2 + n_y^2 - n_z^2\right) + 2 \cdot n_y \cdot \left(e_x \cdot n_x + e_z \cdot n_z\right)}{w_i^3}}{\dfrac{e_z \cdot \left(-n_x^2 - n_y^2 + n_z^2\right) + 2 \cdot n_z \cdot \left(e_x \cdot n_x + e_y \cdot n_y\right)}{w_i^3}}$$

$$a_x = \frac{e_y \cdot \left(-n_x^2 + n_y^2 - n_z^2\right) + 2 \cdot n_y \cdot \left(e_x \cdot n_x + e_z \cdot n_z\right)}{e_z \cdot \left(-n_x^2 - n_y^2 + n_z^2\right) + 2 \cdot n_z \cdot \left(e_x \cdot n_x + e_y \cdot n_y\right)}$$

Again we can use the index division for the transformation from ScreenSpace coordinates to WC.

As mentioned in section 3.2 N and E are already converted to WC, therefore we only need a divider without the additional precision bits for the homogenous component.

To avoid distortion, an arcus tangens table is used to correct the reflectance map address.

$$\begin{bmatrix} a_x & a_y \end{bmatrix}' = \begin{bmatrix} \tan^{-1}(a_x) & \tan^{-1}(a_y) \end{bmatrix}$$

⇒ using DirectPhong™ (VISA+)
The common problem for designers of hardware shaders is the exponentiation in the Phong formula. Gouraud Shaders calculate the intensities on polygon-vertices in software and interpolate these RGB values to produce pixelvalues. The flaw in this approach is that no real, focused highlights can be achieved if the highlight lies within a polygon. This reduces the realism of the rendered scene. Attempts to implement Phong Shading all use table lookups for the exponentiation. This results in intolerable hardware sizes if different values of n must be supported. The VISA system implements Phong Shading using reflectance maps. This is a fast solution to provide real-time Phong Shading with real highlights. The problem with this implementation is that it can only model one value of *n* per scene. Currently, the VISA+ renderer is being designed at the GMD-FIRST that supports DirectPhong™. The VISA+ system is based upon the concept that Phong's exponentiation-function is purely empirical. It implements an alternative function to simulate the shape of Phong's function and to provide real-time Phong Shading for a broad range of specular exponents, defined by the OpenGL™ specifications. It is beyond the scope of this paper to describe the VISA+ system in more detail. Future publications will focus on this subject.

3.4 Hardware Environment Mapping

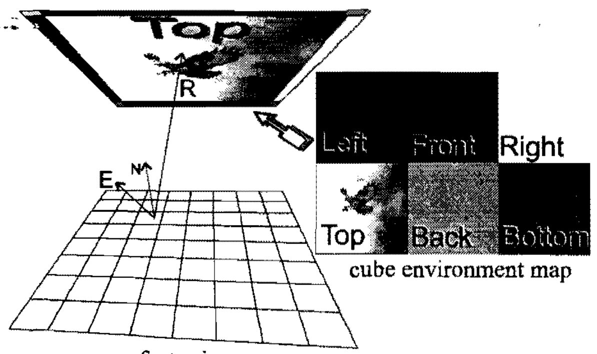


Figure 3-1: Principle of environment mapping

Metallic or mirror-like objects can easily be simulated using the reflectance cube approach. Since the reflected view-vector must be calculated for modeling specular highlights, Phong Shading can easily be extended to environment mapping. Closed room surroundings can be stored in the cube maps. Each cube side contains the image of the specified direction (up, down, left, right, front, back). The reflected view-vector will then be used for addressing these maps, corresponding to the address generation for the reflection maps (section 3.3).

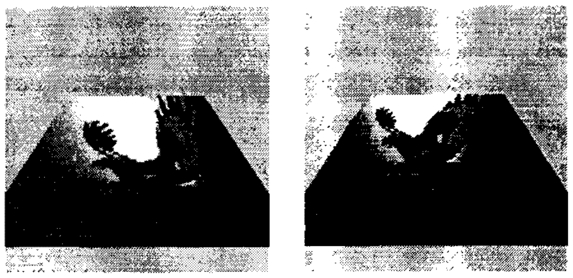


Figure 3-2: Effect on interpolation techniques using cube environment mapping

In the example shown in Figure 3-2, the surface (perfect mirror) reflects the sky (top side) of the environment. Figure 3-1 illustrates the principal of cube environment mapping.
Even the slightest change of the reflectance vector at the polygon edges produces visible artifacts. Therefore it is strongly recommended to use homogeneous interpolation for eye and normal vectors (see image on the right in Figure 3-2), to get undistorted, high quality results. The image on the left in Figure 3-2 is generated using linearly interpolated eye-vectors. As a result, the mirror image is shifted at the polygon edges. This results in missing perspective and crooked illumination.

3.5 Hardware Environment Bump Mapping

A powerful new feature in our hardware architecture is environment bump mapping. Now the modeler can incorporate structured mirrors into his or her virtual environments. To accomplish this task, bump mapping and cube environment mapping is combined.
The perturbed reflection vector is calculated on the basis of the perturbed normal vector. This normal is generated exactly in the same manner as if only bump mapping was used.

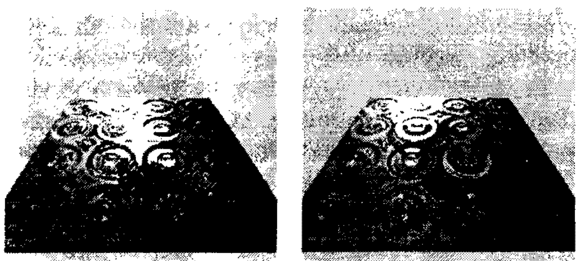


Figure 3-3: Linear (left) versus homogeneous (right) interpolation for bump-environment mapping

Figure 3-3 demonstrates the capabilities of this new graphics feature. For the images, we computed a bump map with a small depth structure on our perfect mirror surface. As a result, the sky is reflected perfectly with the bump structures taken into account.
The image on the left hand shows linearly interpolated eye-vectors and therefore the same artifacts as in the case of pure cube environment mapping. Homogeneously interpolated eye-vectors lead to artifact-free results (image on the right hand in Figure 3-3).

3.6 Simulated curvature

To reduce the complexity of curved objects course triangulation can be used. A typical example for simulated curvature using different directions for edge normals are tessellated hemispheres.
In *Figure* 3-4, a projection of a surface is shown with normals perpendicular to this surface. Assuming that this surface is a patch of a hemisphere with the shapes drawn below, normals are rotated about 45°, 30° and 10° respectively.
However to simulate a hemisphere by a cube's shape is only useful if it is very small. For example, it might be useful for a level of detail representation, required for far away objects.

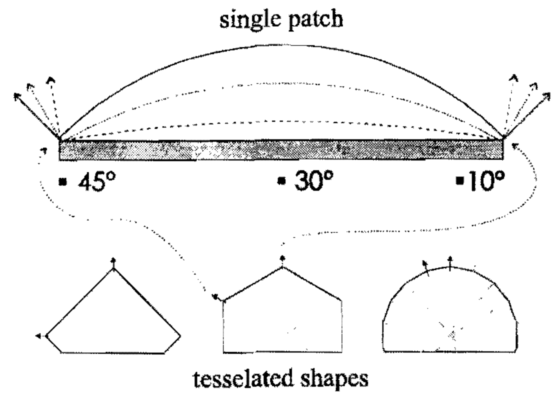


Figure 3-4: Simulating curvature using edge normals

Since we are interpolating normal vectors linearly over the triangles, the effects discussed in the previous sections will also occur.
Fortunately, the variation of vertex normal vectors over a triangle are not as big as on eye vectors. In our investigation on curvature we used large planes with at least 800x800 drawn pixels. Therefore shading results are extremely sensitive to interpolation artifacts. To ensure that even small artifacts can be recognized we used cube environment mapping with a grid texture.
In the table below, images with 5° steps of normal vector angles are shown. Nearly invisible artifacts are in the range of 0°-10°, which leads to over 18x18 patches per hemisphere (see marked region in the 10° case). Acceptable results for medium sized polygons lie in the range of 10°-30° (18x18-6x6). Angles greater than 30° are only useful for very small objects since they reveal severe artifacts (see marked regions in images). In Figure 3-5, environment mapping with homogeneously interpolated normal and eye-vectors is shown. It can be seen that there are no distortions on the polygon edges and the simulated curvature (45° normal vector angle) is expressed in the bowed grid of the texture.

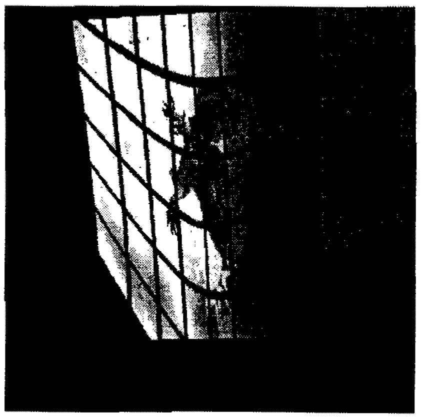
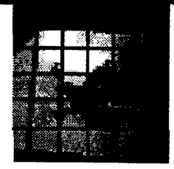
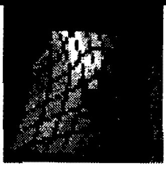
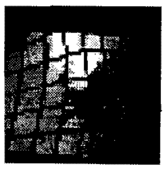
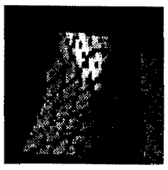


Figure 3-5: normal and eye-vectors interpolated homogeneously

perturbation (degree)		resulting image	
0°		30°	
10°		40°	

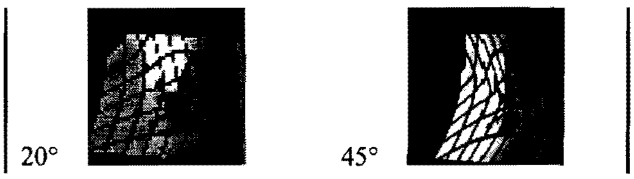


Table 1: Simulated curvature

4. Interpolation

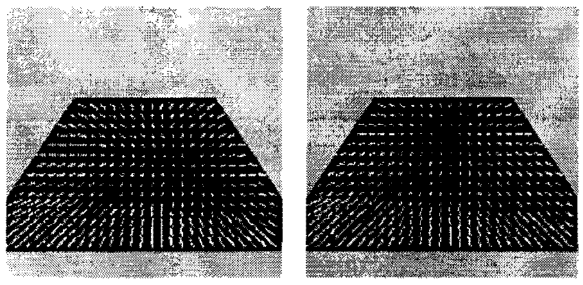


Figure 4-1: Effect on homogeneous and non homogeneous interpolation of eye- and normal vectors

Since the variation of the eye vector over the surface is immense compared to e.g. the normal vector, we generated Figure 4-1 to show the severe change in eye-directions.

Our investigations on interpolation techniques for all critical parameters of our renderer design lead us to the results depicted in Table 2.

parameter	interpolation technique	
normal-vector	homogeneous	recommended
eye-vector	homogeneous	strictly
texture	homogeneous	strictly
vertex color	linear	not critical
z value	homogeneous	recommended
misc. (fog etc.)	linear	not critical

Table 2: Interpolation technique for various parameters

5. Proposed Architecture

The VISA+ rendering engine scan-converts triangular datasets in a uniform manner [5]. Given a triangle vertex, the attached slope increments and their attribute parameters (normal, eye, fog etc.), the interpolators first compute the edge parameters per scanline and subsequently the interpolated attributes within a scanline. As the rasterization unit interpolates main shading parameters in homogenous space a division by w for u,v-texture and bump coordinates as well as for main vector entities (normal-, eye-, etc. vectors), is mandatory for screen space conversion. The division of w for the relevant parameters per pixel implies a screen-projective mapping. In contrast to bilinear mapping techniques, the above mentioned perspective-correct rendering generates artifact-free perspective scenes, no swimming textures, environments and bump-environments in an animation.

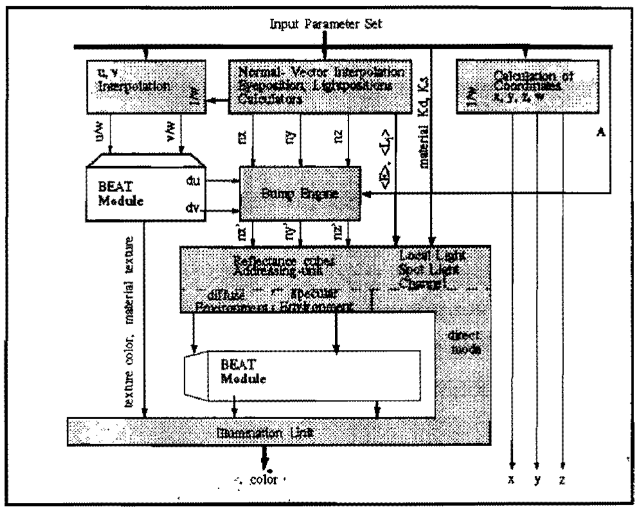


Figure 5-1:Overall Architecture

The above figure gives an impression of the architecture. The shaded boxes show the functional complexity of the VISA+ rasterizer engine, the BEAT (= Bump Environment And Texture) modules implement the functions necessary for memory relevant graphic operations with high bandwidth requirements. (The BEAT module will be described in a further report.) We will focus on the functional units of the VISA+ rasterizer.

The **bump engine** perturbs the pixel normal by the corresponding bump map entries Δu and Δv. In order to align the bump-map vectors perpendicularly to the surface normal, a transformation process must be executed by this functional unit. This is done by multiplying the bump-map entries by matrix **A.**

The **reflectance cubes** are addressed for the determination of the diffuse component by the normal *N* and for the determination of specular component by the reflected ray *R*. The vector's major axis determines the corresponding reflectance cube face. Indexing of this map is done by dividing the vector's minor-axis values by the major-axis value.

As one can see, there is no special unit left for **environment bump mapping.** The VISA+ engine will work in this operation mode by storing the environment of an object in the reflectance cubes. The object's structure is stored as bump-texture in the bump-module, the object's texture in the texture module. The color blending unit combines all these different color sources to the blended color of a pixel.

For each pass up to four spot lights or up to eight local lights can be generated according to the OpenGL illumination equation.

6. Conclusions and future work

The division by w of the relevant parameters implies a screen-projective mapping. In contrast to bilinear mapping techniques, perspective-correct rendering generates artifact-free perspective scenes and prevents swimming textures, environments and bump-environments from occurring in an animation.

Currently, the VISA+ architecture, supporting real-time Phong Shading with realistic texture-, bump- and environment-bump-mapping, is being designed at the GMD-FIRST. The rasterization unit will be fabricated in 0.35 micron technology. The datapath is implemented in a full custom design style. We believe that full custom datapath design style will be a must for high complexity designs minimizing the power consumption [11]. The design has a complexity of circa one million gates (spot light channels inclusive). By the end of 1996, we expect a working prototype. Using this rendering-system, we are able to generate animated scenes that by far surpass the realism of conventional Gouraud-shaders.

Acknowledgments

We are grateful to S. Budianto, K. Tehrani and our former colleague T. Le Vin for their suggestions and contributions to this work.

Literature

[1] Bishop G., Weimar D. M., "Fast Phong Shading", Computer Graphics 20, 4 (April 1986), pp. 255-262.

[2] 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.

[3] Clausen U., "Reducing the Phong Shading Method", Eurographics'89, pp. 333-344.

[4] A.M. Kuijk and E.H. Blake, "Faster Phong Shading via Angular Interpolation", Computer Graphics Forum, Vol. 8, 1989, pp. 315-324.

[5] D. Jackèl, H. Rüsseler, "A Real Time Rendering System with Normal Vector Shading", 9th EuroGraphics Workshop on Graphics Hardware, Oslo, Norway, 1994

[6] D. Voorhies and J. Foran, "Reflection Vector Shading Hardware", Computer Graphics (Proc. SIGGRAPH'94), pp. 163-166.

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

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

[9] I. Ernst, D. Jackèl, H. Rüsseler, O. Wittig, "Hardware Supported Bump Mapping: A Step towards Higher Quality Real-Time Rendering", In 10th EuroGraphics Workshop on Graphics Hardware (August 1995), EuroGraphics, pp. 63-70.

[10] P. S. Heckbert, Graphic Gems, ISBN 0-12-336156-7

[11] J. Smit, M. Bosma, "On the energy complexity of Algorithms realized in CMOS, a Graphics Example", EuroGraphics Workshop on Graphics Hardware, 1996, Poitiers, France.