EUROGRAPHICS 2002 / I. Navazo Alvaro and Ph. Slusallek
(Guest Editors)

Improved Bump Mapping by using Quadratic Vector Interpolation

Anders Hast

Creative Media Lab
University of Gävle, Kungsbäcksvägen 47, S-801 76 Gävle, Sweden. aht@hig.se

Tony Barrera

Cycore AB
Dragarbrunnsgatan 35, P.O. Box 1401, S-751 44 Uppsala, Sweden

Ewert Bengtsson

Centre for Image Analysis
University of Uppsala, Lägerhyddsvägen 17. S-752 37, Uppsala, Sweden. ewert@cb.uu.se

Abstract

Bump mapping is a technique that perturbs normals used in the illumination computation in order to make the surface appear rough or wrinkled. Standard graphics hardware linearly interpolates rotated vectors in the direction to the light source over the polygon. Then, a cube map is used to normalize these vectors. We propose the use of quadratic interpolation instead, which will make the normalization stage unnecessary. The result is faster bump mapping with quality near the quality obtained by using vector interpolation with normalization. Quadratic interpolation is already used for ordinary shading but has to our knowledge not previously been used for bump mapping. The fast setup we use in our new algorithm can also be applied to Phong shading resulting in increased performance. Similarly the same approach can be used for efficiently setting up the linear interpolation.

1. Introduction

Bump mapping was introduced by Blinn [5] as a method for making surfaces appear rough or wrinkled without increasing the number of polygons. Instead, the normals used in the lighting computations are perturbed to achieve this effect. Bump mapping is often used in combination with texture mapping with the purpose of removing the flatness of the texture. Instead it enhances the natural roughness of the texture. Kautz et al [15] show how this can be achieved for arbitrary levels of detail.

Blinn uses an approximation for the perturbed normal, which depends on the surface normal and the height information in the bump map. The perturbed normal is calculated as:

$$\mathbf{N}' = \mathbf{N} + \frac{F_u(\mathbf{N} \times P_v) - F_v(\mathbf{N} \times P_u)}{||\mathbf{N}||}, \qquad (1)$$

where $\mathbf{N}$ is the surface normal, $\mathbf{P}_u$ and $\mathbf{P}_v$ are the partial derivatives of the surface in the u and v directions respectively. F_u and F_v are the gradients of the bump map. Peercy et al [23] use another approach for computing the normal. An orthonormal frame on the surface is used to rotate the vector in the direction to the light source $\mathbf{L}$ into that local frame. Hence, the diffuse intensity is computed as $I_d = \mathbf{N}' \cdot \mathbf{L}'$, where $\mathbf{N}'$ is the normal obtained from the bump map, and $\mathbf{L}'$ is the normal rotated by the frame $(\mathbf{T}, \mathbf{B}, \mathbf{N})$, where $\mathbf{T}$ is the tangent vector of the surface at the point with the surface normal $\mathbf{N}$ and finally $\mathbf{B}$ is the binormal, computed as $\mathbf{N} \times \mathbf{T}$. They are taking this approach a step further by precomputing bump normals over the whole object. Nonetheless, this is the main idea of moving frame bump mapping. Here, the perturbed normal is:

$$\mathbf{N}' = (-F_u, -F_v, 1), \qquad (2)$$

which is obtained by computing the cross product of the gradients as shown by Doggett et al [6].

Kugler [19] uses a different representation of the normal perturbation based on spherical coordinates. Sung Kim et al [26] elaborate this idea further by directly computing the inner products for the diffuse and specular light from the perturbed normal in this representation. A hardware implementation of a bump mapping chip is proposed by Ikedo et al [13], where vector normalization is avoided by using vectors in angular form. Miller et al [21] discuss how hardware accelerated bump mapping using standard texture-mapping hardware can be performed. An overview of other bump map approaches is given by Ernst et al [8] and Kilgard [16].

1.1. Illumination

The Phong-Blinn illumination model is often used for bump mapping. The intensity at each pixel on the polygons can be modeled by summing the intensities of the diffuse and specular components, as follows:

$$I = K_d(\mathbf{N}\cdot\mathbf{L}) + K_s(\mathbf{N}\cdot\mathbf{H})^n, \tag{3}$$

where K_d and K_s are material constants for the diffuse and specular properties of the surface, **N** is the normal vector, **L** is a vector pointing in the direction at the light source, **H** is the halfway vector introduced by Blinn [4], which is the direction of the normal that would give maximum highlight, and finally n is the shininess value which affects the size of the highlighted area.

Similar models can be used, and some include ambient light and the intensity of the light source. Sometimes a distance term is used, since the intensity of light becomes smaller with increased distance. Phong used $I_s = (\mathbf{R}\cdot\mathbf{V})^n$ for the specular intensity instead of $(\mathbf{N}\cdot\mathbf{H})^n$. However, since Blinn's equation is a dot product between the normal and a constant vector, it is similar to the computation of the diffuse intensity which also involves a dot product between the normal and a constant vector. Furthermore, Blinn's version turns out to be faster to compute. The formula for computing Phong highlight for bump mapping is shown in [13].

1.2. Interpolation

Peercy et al [23] state that it is not necessary to interpolate the whole frame over the polygon. Instead the computation of the frame is done on the vertices only. Since the frame is orthonormal, the resulting rotated normal will still have unit length. Therefore we can interpolate the rotated light vectors over the polygon. This works both for directional and point light sources.

It is often necessary to normalize the interpolated vectors, and hardware accelerated table lookups are often used. Kilgard [16] explains one method called cube maps. One disadvantage is that specular cube maps assumes that directional light sources are used. Another disadvantage with this approach is that one cube map is necessary for the diffuse light and one cube map is needed for each type of matter with different specular shininess. Nevertheless, it will be fast since the shininess is precomputed into the cube map. However, if the polygons are small we can use linear interpolation without normalization.

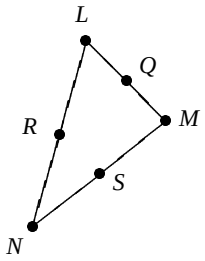

Figure 1: *Sample points*

If we could avoid table lookup normalization of the vectors we will have a more flexible bump mapping algorithm, allowing both directional and point light sources. We propose the use of quadratic interpolation of the vectors in this paper. This approach will give a visual result close to what vector interpolation with normalization will give, but is noticeable faster. It will also be shown in this paper how the setup variables for Phong shading can be computed faster.

1.3. Quadratic Interpolation

Several schemes using a bivariate quadratic polynomial for approximation of the Phong intensity have been developed over the years and are still being developed. The intensity will form a smooth surface in (x, y, i)-space and it is evaluated by only two additions per pixel. Bishop and Weimer [3] use a Taylor series expansion of Duff's equation to obtain a second order bivariate polynomial. Kappel [14] uses a surface fitting technique, where the purpose is to eliminate Mach bands by generating a C^1 continuous surface over the polygon. This is possible by subdividing the triangle into smaller triangles. The approach by Kirk et al [17] is to fit a surface to a triangle, where the intensity of the vertices of adjacent triangles are the same, and the first derivative of the intensity at the edges tend toward being continuous.

Kirk and Voorhies [18] adopt another approach to quadratic shading which is somewhat different from previous approaches. They show that quadratic interpolation could be setup by fitting a second order surface to six points, yielding a polynomial with six unknown coefficients which must be determined. The sample points are the vertices and edge midpoints of a triangle, as shown in Fig 1. A quadratic shading function is defined by:

$$\Phi = Ax^2 + By^2 + Cxy + Dx + Ey + F. \tag{4}$$

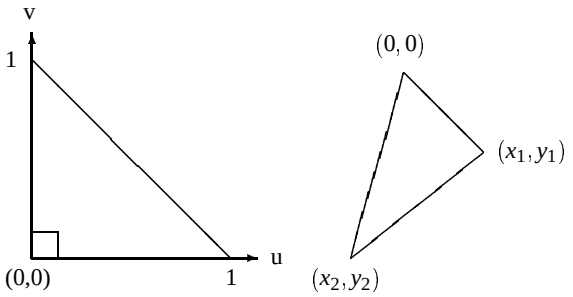

Figure 2: *To the left: Ortho-normalized triangle. To the right: a polygon with shifted vertices*

Seiler [25] proposes a fast way to setup the coefficients for this type of quadratic shading. Lee and Jen [20] have elaborated this approach further. How the setup variables needed in the two inner shading loops are computed is shown by Abbas et al [1], [2].

2. Quadratic Vector Interpolation

In a previous paper [11] we have shown how ortho-normalized triangles can be used to simplify the computation of quadratic shading coefficients. The ortho-normalized triangle has vertices at: $p_1 = (0,1)$, $p_2 = (0,0)$ and $p_3 = (1,0)$, as shown to the left in in figure 2. The temporary coordinates used in the setup for the actual triangle has been obtained by first sorting the vertices, and then shifting them as shown in the same figure to the right. Screen space coordinates were used where the top left corner is $(0,0)$. The coefficients for a bivariate polynomial over the left, in some sense well behaved triangle, is first computed using the sample points in figure 1:

$$a = 2(L + M - 2Q), \tag{5}$$
$$b = 2(L + N - 2R), \tag{6}$$
$$c = 4(L + S - R - Q), \tag{7}$$
$$d = M - L - a, \tag{8}$$
$$e = N - L - b, \tag{9}$$
$$f = L, \tag{10}$$

where:

$$\Phi = au^2 + bv^2 + cuv + du + ev + f. \tag{11}$$

A mapping from the actual triangle to the ortho-normalized triangle is needed in order to obtain the coefficients for equation (4). This transformation is derived in [12].

For any point (x,y) on the actual triangle, a corresponding point (u,v) on the normalized triangle can be obtained by the following transformation:

$$u = x\alpha_u + y\beta_u, \tag{12}$$
$$v = x\alpha_v + y\beta_v, \tag{13}$$

where:

$$\alpha_u = y_2\omega, \tag{14}$$
$$\beta_u = -x_2\omega, \tag{15}$$
$$\alpha_v = -y_1\omega, \tag{16}$$
$$\beta_v = x_1\omega, \tag{17}$$

and:

$$\omega = \frac{1}{x_1y_2 - x_2y_1}. \tag{18}$$

The coefficients for equation (4) is obtained by substituting equation (12) and (13) into (11).

This fast version of quadratic interpolation will be used to compute the $\mathbf{L}'$ and $\mathbf{H}'$ vectors, which will be close to have unit length. For a torus with 288 polygons the difference in length between the quadratically interpolated vector and the unit length vector was on average 0.45% and maximum 3.2%.

Since we are quadratically interpolating vectors, we must interpolate each element separately. These vectors must be obtained at all six sample points. At the vertices they are obtained in the same way as previously described. But at the edge midpoints we have to construct new frames and compute both vectors again. The normals and tangents on midpoints can be computed as the normalized average of the normals and tangents on the corresponding vertices respectively. Whereas the binormal again can be computed as the cross product of the tangent and normal. Thus, a total of six normalizations are necessary. Another approach would be to compute L' and H' on the midpoint of the edges as the normalized average of the corresponding vectors on the vertices. Still six normalizations are necessary, but it is not necessary to compute the binormal for the edges.

Figure 3 shows a 512 polygon bump mapped Torus using quadratic interpolation.

3. Linear Interpolation

Linear interpolation is needed both for vector interpolation with normalization as well as for Gouraud shading [9] and Z-buffer interpolation. Here we can compute the gradients in the scan line direction and in the direction along the edges as shown by for example Narayanaswami [22]. We will now show how the previously described transformation method can be used for computing these gradients as well.

Let ϕ be a bivariate linear function:

$$\Phi = au + bv + c. \tag{19}$$

The coefficients for a bivariate polynomial over this triangle is first computed by setting up the following system of equations:

$$\Phi = M \cdot K, \tag{20}$$

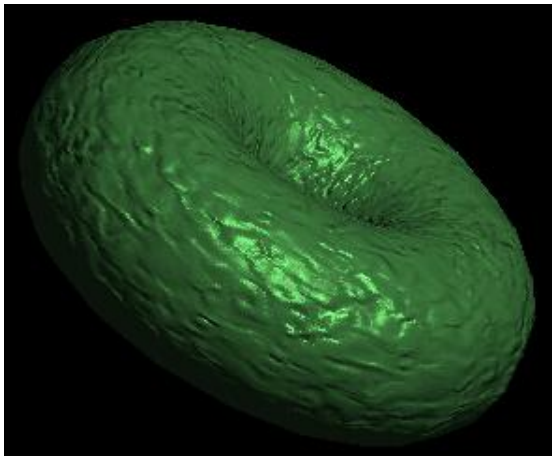

Figure 3: *Bump mapped 512 polygon Torus, using quadratic interpolation.*

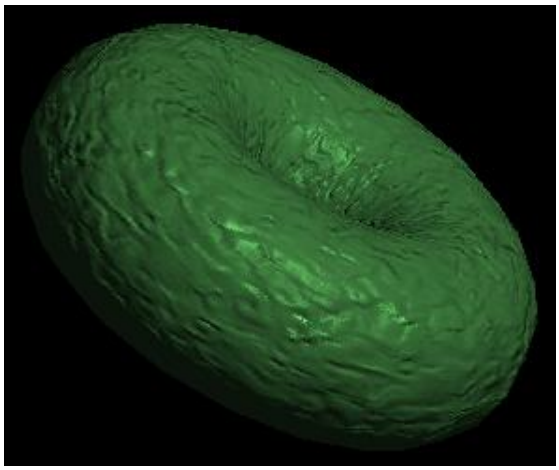

Figure 4: *Bump mapped 512 polygon Torus, using linear interpolation.*

where:

$$\Phi = [L, M, N]^T, \tag{21}$$

$$K = [a, b, c]^T. \tag{22}$$

The matrix M is:

$$M = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}. \tag{23}$$

The solution for $K = M^{-1}\Phi$ is:

$$a = M - L, \tag{24}$$

$$b = N - L, \tag{25}$$

$$c = L. \tag{26}$$

Use the basis vectors in order to do the transformation from (x, y) to (u, v) when using equation (19):

$$\phi = a(\alpha_u x + \beta_u y) + b(\alpha_v x + \beta_v y) + c \tag{27}$$

$$= (a\alpha_u + b\alpha_v)x + (a\beta_u + b\beta_v)y + c. \tag{28}$$

When moving down along the edges , equation (19) only requires one additions per scan line. While the y-value is increased by 1 for each new scan line, the x value must be increased by the slope of the edge. For the polygon in figure 2 the slope of the left edge is $k = x_2/y_2$. The slope of the upper right side is $k = x_1/y_1$ and the slope for the lower right side is $k = (x_2 - x_1)/(y_2 - y_1)$. For the edge of the n'th scan line the bivariate polynomial is:

$$\Phi_n = (a\alpha_u + b\alpha_v)nk + (a\beta_u + b\beta_v)n + c$$

$$= n(k(a\alpha_u + b\alpha_v) + (a\beta_u + b\beta_v)) + c. \tag{29}$$

The value of the polynomial for the m'th pixel along the n'th scan line beginning at nk is:

$$\Phi_{(n,m)} = a(nk + m) + bm + c$$

$$= a(\alpha_u(nk + m) + \beta_u n) + \tag{30}$$
$$\quad b(\alpha_v(nk + m) + \beta_v n) + c$$

$$= a(\alpha_u nk + \beta_u n) + \tag{31}$$
$$\quad b(\alpha_v nk + \beta_v n) + c + a\alpha_u m + b\alpha_v m$$

$$= \phi_n + m(a\alpha_u + b\alpha_u) \tag{32}$$

Pseudo code for a Gouraud shader using the described approach:

```
p=L;
k=x2/y2;
kk=x1/y1;

dr=a*alpha_u+b*alpha_v;
dp=k*dr+a*beta_u+b*beta_v;

 for(y=y0; y<y2; y++)
    r=p;
    for(x=xs; x<xe; x++)
       intensity=r;
       r+=dr;
    p+=dp;
    xs+=k;
    xe+=kk;
```

Note, that we have to change the value for k for the lower part of the triangle if the polygon is oriented so that it has two edges on the left. This is not done in the pseudo code above.

Figure 4 shows that the highlight obtained with linear interpolation is corrupted compared to the highlight obtained by using quadratic interpolation.

4. Vector interpolation with Normalization

Phong [24] uses bilinear interpolation of the normals at the edges, to obtain intermediate normals for each pixel. These normals are then used in the illumination computation. Duff [7] shows how the computation can be implemented in an efficient way, requiring only three additions, one division and one square root per pixel for Phong shading. However, we do not want to compute the dot product directly, since the interpolated vector should be multiplied with the perturbed normal. The required calculation is:

$$\frac{\mathbf{L}' \cdot \mathbf{N}'}{\| \mathbf{L}' \|} = \frac{(\mathbf{A}x + \mathbf{B}y + \mathbf{C}) \cdot \mathbf{N}'}{\sqrt{Dx^2 + Ey^2 + Fxy + Gx + Hy + I}}. \quad (33)$$

Note that $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ are vectors. Thus we have to compute three different linear equations. These can each be efficiently computed with one addition per pixel using a forward difference. The bivariate quadratic function in the denominator can be computed by two additions per pixel. The same calculation must be performed for the specular light as well.

Duff's method for vector interpolation with normalization can be used also for Bump mapping and we will compare this with both linear and quadratic interpolation. For the denominator in equation (33) we have:

$$\| \mathbf{L}' \| = \sqrt{\mathbf{L}' \cdot \mathbf{L}'} \quad (34)$$

$$= \sqrt{Dx^2 + Ey^2 + Fxy + Gx + Hy + I}. \quad (35)$$

It turns out that the method described in [12] makes the setup for Phong shading faster. Once again we do not need to make the transformation, instead we compute the setup variables directly, using the ortho-normalized triangle. If $\mathbf{L}'$ is linearly interpolated by $\mathbf{A}u + \mathbf{B}v + \mathbf{C}$, then $\mathbf{L}' \cdot \mathbf{L}'$ is:

$$\mathbf{A}u + \mathbf{B}v + \mathbf{C} = \quad (36)$$

$$\mathbf{A}^2u^2 + \mathbf{B}^2v^2 + 2\mathbf{AB}uv + 2\mathbf{AC}u + 2\mathbf{BC}v + \mathbf{C}.$$

The coefficients for phong shading thus is:

$$a = \mathbf{A}^2, \quad (37)$$

$$b = \mathbf{B}^2, \quad (38)$$

$$c = 2\mathbf{AB}, \quad (39)$$

$$d = 2\mathbf{AC}, \quad (40)$$

$$e = 2\mathbf{BC}, \quad (41)$$

$$f = \mathbf{C}. \quad (42)$$

The coefficients are computed in the same way as usual for Phong shading, except that the transformation is not done.

Timing of setup variables needed for the quadratic denominator for Phong shading was performed. The timing of quadratic setup variables described in [2] compared to setup variables described in [12] but modified for computation of the quadratic denominator is shown in table 1. The setup variables are different for a polygon with only one left edge compared to if there are two edges on the left side. Both cases are shown in the table. It can be added that the timing of an empty loop was 0.24 sec.

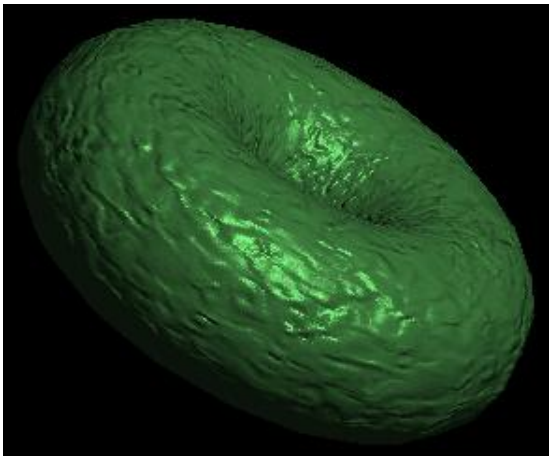

Figure 5: *Bump mapped 512 polygon Torus, using vector interpolation with normalization.*

Table 1: *Timing in seconds of one hundred million triangles, comparing the computation of just the coefficients and the computation of setup variables using the actual triangle and ortho-normalized triangle, for both orientations of triangles, on a 1.8 GHz Pentium 4 processor.*

Edges	Actual triangle	Ortho-norm. triangle
1	24.7	16.1
2	26.2	16.4

It is clear from the result in the table that the setup variables needed for the quadratic denominator is computed faster with the proposed method. However, this is mostly of academic interest since Phong shading is seldom used anyway. Nevertheless, the ortho-normalized triangle setup is used in this paper so that we use the same approaches when comparing quadratic interpolation with vector interpolation using normalization.

The computation needed for bump mapping using Vector interpolation with normalization is easy to derive. The three rotated vectors in the direction to the light source will be denoted $\mathbf{L}'_1$, $\mathbf{L}'_2$ and $\mathbf{L}'_3$. Let:

$$\mathbf{A} = \mathbf{L}'_2 - \mathbf{L}'_1, \quad (43)$$

$$\mathbf{B} = \mathbf{L}'_3 - \mathbf{L}'_1, \quad (44)$$

$$\mathbf{C} = \mathbf{L}'_1. \quad (45)$$

Then use equation (37) to equation (42) for computing the coefficients.

Figure 5 shows the Torus again with vector normalization.

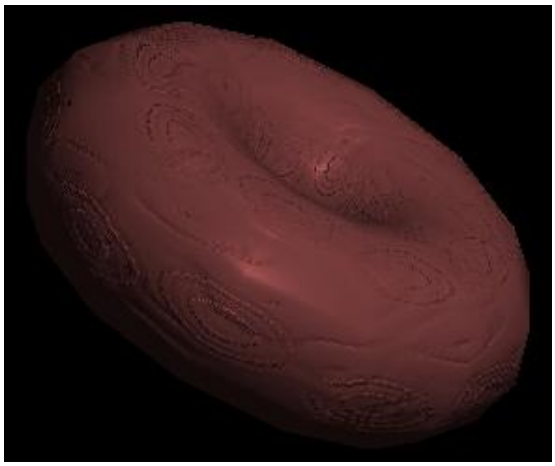

Figure 6: *Bump mapped 288 polygon Torus, using linear interpolation.*

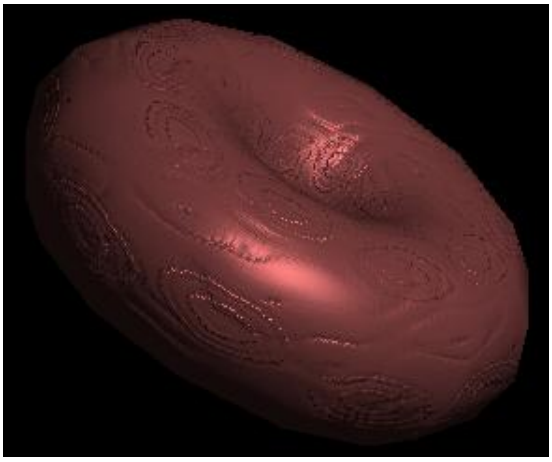

Figure 7: *Bump mapped 288 polygon Torus, using quadratic interpolation.*

the highlights are very much the same as for quadratic interpolation.

5. Comparison of Algorithms

Timing was conducted of the three main algorithms as shown in table 2. First an empty loop was timed where the reading of the bump map and rotation of an 800 polygon torus are performed, as well as calling the shading function which immediately returns. For the three main methods no frame buffer update and lighting calculations are done. Only the setup and the computation in the two inner loops as well as Z-buffer computations are done.

Taking the empty loop into account the vector normalization algorithm takes about 37% longer time to run than the

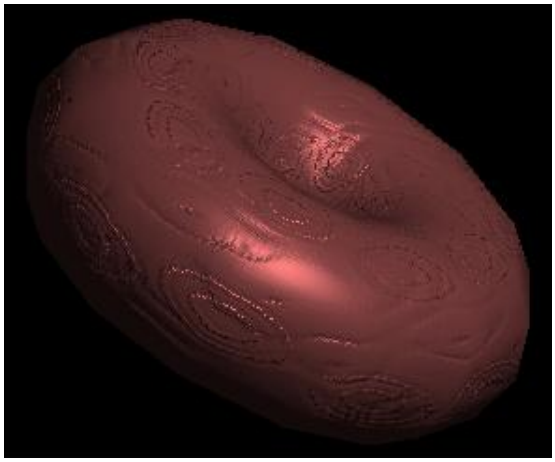

Figure 8: *Bump mapped 288 polygon Torus, using vector interpolation with normalization.*

Table 2: *Timing of the rotation of a bump mapped torus comparing the execution of different methods*

Empty	Linear	Quadratic	Vector norm.
8.0	15.7	18.1	22.0

quadratic interpolation algorithm. If the Z-buffer computations are removed, the difference would be even larger. It is quite clear that quadratic interpolation is a computationally more attractive alternative. Also when it comes to quality it is a good alternative. As stated earlier, the difference between the quadratically interpolated normal and a unit length normal is in most cases very small. This can be clearly seen when comparing figure 7 and 8. Note that the highlights are almost identical on the flat part of the torus, while the highlight is corrupted for linear interpolation in figure 6.

6. Discussion

We did not use cube maps in our tests with linear interpolated bump mapping. Table look-ups are surprisingly slow in software and it would not be fair to compare it with quadratic interpolation. We did not use hardware accelerated cube maps in our tests, since our intent is not to propose algorithms that work well with existing hardware. Instead we propose an algorithm that could be implemented in future hardware. Quadratic interpolation should be attractive to implement since there are no divisions nor square roots in the two inner loops. Instead there are a number of additions that could be parallelized in hardware. Hence, quadratic interpolation becomes very fast.

6.1. Future Work

The proposed algorithm should be implemented in hardware in order to make a fair comparison.

Other quadratic interpolation schemes should be investigated. The schemes chosen in this paper are all based on per vertex computations. It should be examined if scan line setup algorithms like the one proposed in [10] are faster in some situations.

References

1. A. M. Abbas, L. Szirmay-Kalos, G. Szijarto, T. Horvath, T. Foris *Quadratic Interpolation in Hardware Rendering* Spring Conference of Computer Graphics, 2001.
2. A. M. Abbas, L. Szirmay-Kalos, T. Horvath, T. Foris *Quadratic Shading and its Hardware Implementation*, Machine Graphics and Vision, Vol. 9, No. 4, pp. 825-804, 2001.
3. G. Bishop, D. M. Weimer, *Fast Phong Shading* Computer Graphics, vol. 20, No 4, pp. 103-106, 1986.
4. J. F. Blinn, *Models of Light Reflection for Computer Synthesized Pictures* In Proceedings SIGGRAPH, pp. 192-198. 1977.
5. J. F. Blinn, *Simulation of Wrinkled Surfaces* In Proceedings SIGGRAPH, pp. 286-292. 1978.
6. M. Doggett, A. Kugler, W. Strasser, *Displacement Mapping using Scan Conversion Hardware Architectures* Computer Graphics Forum, Vol. 20 No 1. pp 13-26. 2000.
7. T. Duff, *Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays* ACM, Computer Graphics, Vol. 13, 1979, 270-275.
8. I. Enrst, H. Rüssler, H. Schultz, O. Wittig *Gouraud Bump mapping* Workshop on Graphics Hardware, pp. 47-53. 1998.
9. H. Gouraud, *Continuous Shading of Curved Surfaces*, IEEE transactions on computers vol. c-20, No 6, June 1971.
10. A. Hast, T. Barrera, E. Bengtsson, *Approximated Phong Shading by using the Euler Method*, Eurographics01, short paper pp. 43-48, 2001.
11. A. Hast, T. Barrera, E. Bengtsson, *Faster Computer Graphics - by Reformulation and Simplification of Mathematical Formulas and Algorithms*, IMAGINE2001, 2001.
12. A. Hast, T. Barrera, E. Bengtsson, *Faster Bivariate Quadratic Shading through Simplified Setup*, Submitted for publication, 2002.
13. T. Ikedo, E. Ohbuchi, *A Realtime Rough Surface Renderer*, Proc. of Computer Graphics International 2001 (CGI2001), IEEE Computer Socity Press, July 2001.
14. M. R. Kappel, *Shading: Fitting a Smooth Intensity Surface* Computer-Aided Design, Vol. 27, No. 8, pp. 595-603, 1995.
15. J. Kautz, W. Heidrich, H.-P. Seidel, *Real-Time Bump Map Synthesis*, Proceedings of the Eurographics/SIGGRAPH Workshop on Graphics Hardware 2001, pages 109-114, August 2001.
16. M. J. Kilgard *A Practical and Robust Bump-mapping Technique for Today s GPUs* Game Developers Conference, Advanced OpenGL Game Development. 2000.
17. D. Kirk, O. Lathrop, D. Voorhies, *Quadratic Interpolation for Shaded Image Generation* Patent Nr: US5109481, 1992.
18. D. Kirk, D. Voorhies, *The Rendering Architecture of the DN10000VS* Computer Graphics vol. 24, pp. 299-307, August 1990.
19. A. Kugler *IMEM: An Intelligent Memory for Bump- and Reflection-Mapping* Workshop on Graphics Hardware, pp. 113-122. 1998.
20. Y. C. Lee, C. W. Jen, *Improved Quadratic Normal Vector Interpolation for Realistic Shading* The Visual Computer, 17, pp. 337-352, 2001.
21. G. Miller, M. Halstead, M. Clifton *On-the-Fly Texture Computation for Real-Time Surface Shading*, IEEE Computer Graphics and Applications, Vol. 18, No. 2, March-April 1998
22. C. Narayanaswami, *Efficient Parallel Gouraud Shading and Linear Interpolation over Triangles*, Computer Graphics forum, Vol. 14, No. 1, pp. 17-24, 1995.
23. M. Peercy, A. Airey, B. Cabral, *Efficient Bump Mapping Hardware* In proceedings of SIGGRAPH, pp. 303-306. 1997.
24. B. T. Phong, *Illumination for Computer Generated Pictures* Communications of the ACM, Vol. 18, No 6, June 1975.
25. L. Seiler, *Quadratic Interpolation for Near-Phong Quality Shading* Proceedings of the conference on SIGGRAPH 98: conference abstracts and applications, Page 268, 1998.
26. J. Sung Kim, J. Hyun Lee, K. Ho Park *A Fast and Efficient Bump Mapping Algorithm by Angular Perturbation* Computers and Graphics No. 25, pp. 401-407, 2001.