Shading by Quaternion Interpolation

Anders Hast
Creative Media Lab
University of Gävle, Sweden
aht@hig.se

ABSTRACT

The purpose of this paper is to show that linear interpolation of quaternions can be used for true Phong shading and also for related techniques that use frames, like bump mapping and anisotropic shading. Quaternion interpolation for shading has not been proposed in literature and the reason might be that it turns out to be mostly of academic interest, and it will here be explained why. Furthermore some pros and cons of interpolation using quaternions will be discussed. The effect of using this approach is that the square root in the normalization process disappears. The square root is now implemented in modern graphics hardware in such way that it is very fast. However for other types of platforms, especially hand held devices, the square root is computationally expensive and any software algorithm that could produce true Phong shading without the square root might turn out to be useful. It will be shown that linear interpolation of quaternion could be useful for bump mapping as well. However, quaternion arithmetic operations are not implemented in modern graphics hardware, and are therefore not useful until this is done.

Keywords
Phong Shading, Quaternion Interpolation

1. INTRODUCTION

Shading makes faceted objects appear smooth. Two widely used techniques are known as Gouraud [Gou71] and Phong Shading [Pho75]. Gouraud shading suffers from the Mach band effect and handles specular reflections poorly. This problem is diminished when Phong shading is used since the normals are interpolated instead of the intensities. However, the drawback with this approach is that all interpolated normals must be normalized. Otherwise, Phong shading will not be much different from Gouraud shading [Duf79].

This paper will show that it actually is possible to avoid the square root in the normalization process and still obtain normalized normals for Phong shading. This can be done when quaternions are used for the interpolation of the normals. If modern graphics hardware is used, then this is not really a problem anymore since these operations are as fast as multiplications and additions nowadays. However, quaternion arithemtics is not implemented in modern graphics hardware. If such arithmetics were implemented then it also could be used for bump mapping, as explained in this paper.

2. PREVIOUS WORK

Shoemake [Sho85] introduced spherical linear interpolation (slerp) to the computer grapics society. Kuijk and Blake [Kuij89] showed how such angular interpolation could be used for faster Phong shading. They use spherical trigonometry to derive an equation for how both the normal and the vector in the direction to the light source varies over the polygon. A cosine has to be evaluated for each pixel. However, they propose a quadratic approximation that will make the evaluation faster. Abbas et al. [Abb00] elaborates this idea further for a suitable hardware implementation. Barrera et al. [Bar04] showed that equal angle interpolation of normals could be done in a very efficient way, removing the need for the division and square root, since this approach yields normalized normals in the interpolation process. However, the setup for each scanline involves the computation of several trigonometric functions. Hence, the gain in speed will be diminished by the extra setup, unless tables are used.

Bump mapping was introduced by Blinn [Bli78] 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. Peercy et al. [Pee97] use an orthonormal frame on the surface to rotate the vector in the direction to the light source into that local frame. An overview of other bump map approaches is given by Ernst et al. [Ern98] and Kilgard [Kil00].

3. QUATERNIONS

Unit quaternions are related to orthonormal rotation matrices. The quaternion consists of four elements

$$q = (x, y, z, w) \qquad (1)$$

A more compact form is to describe the three first elements as a vector since they constitute the imaginary part of the quaternion. The fourth part is the real part

$$q = (\mathbf{v}, s) \qquad (2)$$

A quaternion may be used to rotate θ degrees around a vector **n** and it can be shown that the quaternion can be written as

$$q = (\mathbf{n}\sin(\theta/2), \cos(\theta/2)) \qquad (3)$$

The quaternion rotation [Sva00] of a point in 3D space is defined as

$$p' = qpq^{-1} \qquad (4)$$

where p is a vector in quaternion form

$$p = (\mathbf{r}, 0) \qquad (5)$$

such as $\mathbf{r}=(x_p,y_p,y_p)$ and q^{-1} is the same as the conjugate of a quaternion for unit quaternions

$$q^{-1} = (-\mathbf{v}, s) \qquad (6)$$

A quaternion can be transformed into a rotation matrix and vice versa [Wat92]. The rotation matrix corresponding to q in equation (1) is

$$\mathbf{M}=\begin{bmatrix} 1\text{-}2(y^2+z^2) & 2(xy+zw) & 2(xz+yw) \\ 2(xy+zw) & 1\text{-}2(x^2+z^2) & 2(yz+xw) \\ 2(xz+yw) & 2(yz+xw) & 1\text{-}2(x^2+y^2) \end{bmatrix} \qquad (7)$$

The conversion from a matrix to a quaternion is a bit messier.

Frames

Even though it is not mentioned in literature, quaternions could be used for bump mapping and anisotropic shading [Pou90, Ban94], since they use frames on the surface, and quaternions are just another representation of frames. And orthogonal frames are rotation matrices. This fact is used in moving frame bump mapping, where the bump normal is rotated by the frame. However, in order to use quaternions as frames, there are several obstacles that must be handled. A quaternion can not be rotated by a frame. Hence frames, or at least the normal and the tangent, must be stored per vertex, rotated and then converted into quaternions. When bump mapping is used, the interpolated quaternion must either be converted into a rotation matrix for the rotation of the bump map normal, or the bump map normal can be rotated directly by the quaternion. However, this operation is not directly supported by modern graphics hardware, as matrix multiplication is. The advantage by interpolating quaternions, over interpolating the frame itself is that the qutaernion will still always define an orthonormal frame as long as the quaternion is normalized. When a frame is interpolated, the normal and tangent are interpolated. Both vectors must be normalized for each pixel. The binormal can be computed as the cross product of these two vectors. The resulting frame will usually not be orthogonal, but close enough.

Elimination of the Square Root

Another advantage by using quaternions is that the square root disappears. This will not have much effect for modern graphics hardware since the square root nowadays is very fast. However, for software shading it will have a larger impact. Quaternions are normalized in the same way as vectors are

$$q' = \frac{q}{\sqrt{q \cdot q}} \qquad (8)$$

Nonetheless, equation (7) shows that each element of the normalized quaternion is multiplied with another element of the same quaternion, when the elements of the matrix is computed. For an example we have

$$q'_x q'_y = \frac{q_x}{\sqrt{q \cdot q}} \frac{q_y}{\sqrt{q \cdot q}} = \frac{q_x q_y}{q \cdot q} \qquad (9)$$

Hence, the square root disappears, but the division is still necessary. Moreover, it can be shown that the square root disappears for quaternion rotation as it is defined in equation (4). The rotation can be simplified as [Wat92]

$$qpq^{-1} = \mathbf{r}(ss - \mathbf{v}\cdot\mathbf{v}) + 2\mathbf{v}(\mathbf{v}\cdot\mathbf{r}) + 2s(\mathbf{v}\times\mathbf{r}) \qquad (10)$$

Clearly, any element of the quaternion q, will be multiplied with some other element of q and the square root can be removed. This fact is nothing new, but it is possible to utilize it for shading as is shown in the next section.

4. SHADING

Shading can utilize quaternion interpolation, since the normal can be obtained from the resulting matrix. Thus, it is not necessary to compute the tangent and binormal, in order to obtain the normal itself. The normal is found in the third column of the matrix in equation (7). Hence, the normal could be computed as

$$\mathbf{n}=\begin{bmatrix}2(xz+yw)\\2(yz+xw)\\1\text{-}2(x^2+y^2)\end{bmatrix} \tag{11}$$

Equation (11) can be compared to ordinary vector interpolation where each element is linearly interpolated and the dot product can be quadratically interpolated [Duf79], followed by a division and a square root. When quaternions are interpolated, there are four linear interpolations followed by equation (8) and (11). Even though multiplications and additions are several times faster than the square root on ordinary CPU's, there is not much time saved by this method. However, we can still do better.

Fast Normal interpolation

Note, that equation (11) would be much simpler if it would be possible to arrange so that the fourth element *w* is always set to zero. The normal becomes

$$\mathbf{n}=\begin{bmatrix}2xz\\2yz\\1\text{-}2(x^2+y^2)\end{bmatrix} \tag{12}$$

This would correspond to rotating the frame at the vertices in such way that the normal is still the same, but the tangent and binormal is rotated around the normal. This can be done. Since the quaternion should be normalized, but *w* is zero, then

$$z^2+x^2+y^2=1 \tag{13}$$

Thus

$$x^2+y^2=1-z^2 \tag{14}$$

Substitute equation (14) into equation (11) for computing $\mathbf{n}_z$

$$\mathbf{n}_z=2z^2-1 \tag{15}$$

Solving for z gives

$$z=\sqrt{\frac{\mathbf{n}_z+1}{2}} \tag{16}$$

Now, equation (16) could be used to compute *x* and *y* by substituting the expression for *z* into equation (12). Hence

$$x=\frac{\mathbf{n}_x}{2z} \qquad y=\frac{\mathbf{n}_y}{2z} \tag{17}$$

Equations (16) and (17) need to be computed per vertex. Hence, the square root is necessary three times per polygon instead of one square root per pixel. The following vertex shader code in the OpenGL shading language exemplifies the idea

```
varying vec3 q;
...
q.z=sqrt((normal.z+1.0)*0.5);
float t=1.0/(2.0*q.z);
q.xy=normal.xy*t;
```

Here, *q* is the quaternion that is interpolated over the polygon, *normal* is the normal at the vertices and *t* is temporary variable used to optimize the computation.

The normal at each pixel is computed by equation (12), where the interpolated quaternion first is normalized by equation (9). Using equation (15) instead of computing $\mathbf{n}_z$ with equation (12) gives the following fragment shader code in the OpenGL shading language

```
float t= 2.0*q.z/dot(q,q);
vec3 N=t*q.xyz;
N.z-=1.0;
```

Here, *q* is the quaternion and *t* is temporary variable used to optimize the computation.

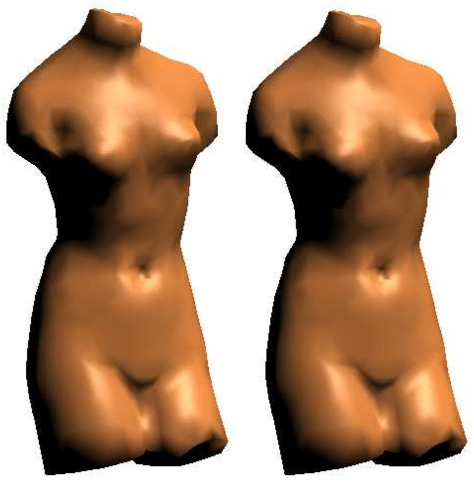

Figure 1. Left: Phong shading. Right: Fast linear quaternion interpolation

Figure 1 shows the famous Venus De Milo statue Phong shaded to the left. It can be compared to the image to the right, which is shaded using linear interpolation of quaternions. The figures are indistinguishable form each other. It can be shown that the normals produced by this type of interpolation, does not necessarily lie in the same plane, as normals obtained by linear interpolation does. However, it is not a requirement that the normals along a scanline should lie in the same plane. If the object is slightly rotated, then the scanline will be a new cross section of the surface that the polygon covers, which did not have their normals in the same plane in the previous frame.

5. DISCUSSION

Even though the square root is replaced by just a few simple arithmetic operations, the gain in speed when using this type of interpolation is small. It is not in the scope of this paper to implement quaternion interpolation on different platforms, since the calculation speed of different operations may vary quite a lot among platforms.

It should be noted that is possible to compute the inverse square root [Tur95] instead of normalizing a vector using division and the ordinary square root. This algorithm is based on the Newton-Raphson method and involves no divisions, but the proposed algorithm has one division. On the other hand it is possible to compute the division using the Newton-Raphson method more effectively than the inverse square root is computed. Hence, the proposed method may turn out to be faster on some platforms.

Quaternion arithmetic operations are not implemented in modern programmable graphics hardware. Therefore, bump mapping will not be faster using quaternion interpolation. However, if these operations were implemented, the conversion between quaternions and rotation matrices would be fast. Moreover, quaternion rotation could be very fast if it was implemented in hardware. If this is done, then quaternion interpolation could be standard procedure in the future, for bump mapping and anisotropic shading.

6. CONCLUSIONS

It is possible to perform true Phong shading by linear interpolation of quaternions representing the frame of the surface. The square root disappears in this process. A faster version can be obtained of this scheme, by rotating the frame in such way that the fourth element is always zero. This scheme may be useful for some software implementations.

It was also discussed that linear interpolation of quaternions could be useful for bump mapping and anisotropic shading if quaternion arithmetics were implemented in modern graphics hardware in the future.

7. REFERENCES

[Abb00]A. M. Abbas, L. Szirmay-Kalos, T. Horvath, Hardware Implementation of Phong Shading using Spherical Interpolation, Periodica Polytechnica, Vol. 44, Nos 3-4, 2000.

[Ban94] D. Banks. Illumination in Diverse Codimensions. In Proceedings SIGGRAPH (July 1994), pp. 327–334.

[Bar04] T. Barrera, A. Hast, E. Bengtsson, Faster shading by equal angle interpolation of vectors, IEEE Transactions on Visualization and Computer Graphics, pp. 217-223, 2004.

[Bli78] J. F. Blinn, Simulation of Wrinkled Surfaces, Proceedings SIGGRAPH 1978: pp. 286-292.

[Duf79] T. Duff, Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays, ACM, Computer Graphics, Vol. 13, pp. 270-275, 1979.

[Ern98] I. Enrst, H. Rüssler, H. Schultz, O. Wittig. Gouraud Bump mapping. Workshop on Graphics Hardware, pp. 47-53. 1998.

[Gou71] H. Gouraud, Continuous Shading of Curved Surfaces, IEEE transactions on computers vol. c-20, No 6, June 1971.

[Kil00] M. J. Kilgard A Practical and Robust Bump-mapping Technique for Today s GPUs Game Developers Conference, Advanced OpenGL Game Development. 2000.

[Kui89] A. A. M. Kuijk, E. H. Blake, Faster Phong Shading via Angular Interpolation, Computer Graphics Forum, vol. 8, No 4, pp. 315-324 1989.

[Pee97] Peercy, A. Airey, B. Cabral, Efficient Bump Mapping. Hardware In proceedings of SIGGRAPH, pp. 303-306. 1997.

[Pho75] B. T. Phong, Illumination for Computer Generated Pictures, Communications of the ACM, Vol. 18, No 6, June 1975.

[Pou90] P. Poulin, A. Fournier, A model for anisotropic reflection, Proceedings ACM SIGGRAPH, Vol. 24 No 4, pp 273 - 282 , 1990.

[Sho85] K. Shoemake, Animating rotation with quaternion curves, Proceedings ACM SIGGRAPH, Vol. 19 No 3, July 1985.

[Sva00] J. Svarovsky. Quaternions for Game Programming, Game Programming Gems. Charles River Media, pp. 195-299. 2000.

[Tur95] K. Turkowski, Computing the Inverse Square Root. Graphics Gems V, Academic Press, pp. 16-21. 1995.

[Wat92] A. Watt, M. Watt. Advanced Animation and Rendering Techniques - Theory and Practice. Addison Wesley, pp. 363-364.