1992

E.H. Blake, V.C.J. Disselkoen, A.A.M. Kuijk

Faster Phong shading

Computer Science/Department of Interactive Systems Report CS-R9228 July

Faster Phong Shading

E.H. Blake, V.C.J. Disselkoen and A.A.M. Kuijk

CWI
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands.
Email: edwin@cwi.nl

Abstract

Phong shading is a very well known shading algorithm for producing realistic images. Real-time Phong shading has on the whole remained too expensive for hardware implementations, either because of excessive computational complexity or because of large storage requirements. The algorithm presented here is designed to produce Phong shaded pixels with the smallest incremental cost using the least storage possible. Its incremental pixel cost is 5 additions with 3 registers to store and accumulate forward differences. This method is well suited to fast hardware implementations but, because of its accuracy and extensibility, it is also an attractive software solution. A quadratic expression based on angular interpolation for Phong shading is derived which involves neither further approximations beyond those implicit in the Phong model for vector interpolation nor table lookup for exponent evaluation. These benefits are paid for by increased polygon and scanline preprocessing compared to earlier methods.

The method can deal with the case where the viewer and light source are inside the scene, that is, when the light or viewer direction varies over the shaded surface. Extensions deal with pixel integration anti-aliasing and these require cubic rather than quadratic interpolation. The paper contains a detailed analysis of the costs and benefits of the new method and previous methods of optimizing or approximating the Phong shading algorithm.

1991 Mathematics Subject Classification : 65D20, 68Q25, 68U05.

Key Words and Phrases : angular interpolation, illumination model, quadratic approximation, shading method.

1. Introduction.

Phong shading is a very well known method for rendering objects which have been approximated with planar polygonal facets (see [5]; an excellent overview of the topic is given in [9]). This shading method gives a smooth rounded appearance to the objects and provides fairly good highlights corresponding to point light sources. This method is less likely to miss highlights and it suffers less from Mach banding than the more common shading method employed for high speed rendering: Gouraud shading.

Phong shading would be of real benefit in interactive applications, such as computer-aided design, if it could be done in (near) real-time (i.e., 12-30 complete pictures per second). More advanced algorithms such as ray tracing and radiosity would be better, but for interactively changing environments these methods are not yet remotely feasible computationally. The problem up to now has been that, even for Phong shading, the computational complexity has been too high for the kind of hardware found in graphics workstations.

Report CS-R9228
ISSN 0169-118X
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

The overall aim of our research, of which we report on one aspect here, is to develop a new architecture for an interactive workstation [11]. The graphics requirements are essentially real-time support of the Phigs+ standard, where Phong shading has been chosen as the best quality shading method. A novel feature of the architecture is that there is no frame buffer — this was done mainly to improve interactive performance, but one could also argue that if 24 frames per second are needed then one might as well do 60 frames per second and get rid of the frame buffer which becomes a bottle-neck at high update rates. This feature has a number of implications, for example, that the smallest pick primitive becomes a visible surface and not a pixel, and that object space hidden surface removal is required. It also means that fully shaded pixels for multiple light sources have to be produced every 12ns, depending on resolution. The algorithm presented here has been implemented partially in custom VLSI to achieve this rate.

1.1 Fast Realistic Rendering in Dynamic Interactive Environments.

Standards of realism are constantly being improved. We shall restrict ourselves to that degree of realism which can be achieved at more-or-less real-time rates, at least 12 frames per second and preferably 24 or more. This frame rate must be realised in an interactive setting where the user is free to alter the models being rendered, the position of the lights and the viewpoint.

The current state-of-the-art for fast high quality rendering is real-time Gouraud shading. The challenge is to do better than that. In spite of advances in radiosity algorithms the current aim there is low quality images during interaction with progressive refinement of static pictures (see for example [2], or other papers on progressive refinement radiosity in the same publication). Ray tracing approaches to rendering show no promise of real-time solutions.

The next hurdle to be taken in real-time realism is therefore the Phong illumination equation. It may be written as follows, if we use the **h** vector introduced by Blinn [4] instead of the original formulation:

$$I = \underbrace{a C_O C_L}_{\text{ambient}} + \underbrace{d C_O C_L (\mathbf{n} \cdot \mathbf{l})}_{\text{diffuse}} + \underbrace{s\, C_S C_L (\mathbf{n} \cdot \mathbf{h})^{\text{gloss}}}_{\text{specular}} \qquad (1)$$

Where colours are governed by the following terms:

a	coefficient of ambient reflection	C_O	object (diffuse) colour
d	coefficient of diffuse reflection	C_L	light source colour
s	coefficient of specular reflection	C_S	object specular colour
gloss	exponent of the Phong shading model — measures glossiness of the object.		

Geometry is specified by following unit vectors:

n	normal to the object
l	direction of the light source from the object
h	direction halfway between eye and light — direction of maximum highlight.

For Gouraud shading the components are evaluated at the vertices of the polygon which is to be shaded and the resultant colours are linearly interpolated over the facet. Clearly the values in the interior of the facet cannot be higher than those on the vertices, so if a maximum lies inside the facet it will be missed. For the Phong shading algorithm the normals themselves are linearly interpolated across the polygon and the shading Equation (1) evaluated at each interior point. If the light source and viewer are modelled as being within the scene then the **l** and **h** vectors are interpolated in the same way.

The major cost in Phong shading is the renormalization via a square root and a division which has to occur after each vector interpolation step. The best optimization still incurs this cost (see Section 2.1). Because pixel values need only limited accuracy look-up tables can be used for square roots and exponentiation. Real-time rates are possible if enough hardware is dedicated to the problem [6, 8].

Another approach to this problem is to simplify the Phong shading model. Here one should be beware of producing expensive Gouraud shaders due to over simplification, or of producing methods which cannot deal with the extreme cases which occur in practice. On the other hand Phong shading is itself nothing more (or less) than a very good practical approximation to real shading effects — it is not a holy Grail which may not be sullied.

The rest of the paper develops as follows: there is next (Section 2) a survey of previous optimizations and simplifications of the Phong method. In Section 3 we introduce our method, quadratic Phong shading via angular interpolation, which has the lowest pixel cost in terms of time and storage of any method we are aware of. Section 4 is a detailed comparison of the various methods in terms of their complexity and compares the visual appearance of the angular interpolation method with standard shading methods. Section 5 summarizes our conclusions. An appendix has been added with pseudo-code to guide readers who may wish to implement the angular interpolation method.

2. Optimizing the Phong Shading Model.

The Phong shading algorithm has been around long enough for it to have gone though several cycles of optimization, some of these were:

1. The original formulation [5], which can now be found in most graphics texts (e.g., [9]).
2. Introduction of the **h** vector which essentially replaces the reflection vector [4].
3. Duff's optimization which allowed incremental computation of a number of the terms [7].
4. Bishop and Weimer's quadratic Taylor series approximation, followed by explicit exponentiation of the specular term, for pixel evaluation [3].

The first two cycles of optimization have already been dealt with in the introduction. The other two will follow in the next subsections.

2.1 Incremental Calculation of the Phong Shading Terms.

For triangle primitives the linearly interpolated vectors can be written as the unique combination of three vectors, **a**, **b** & **c**:

$$\mathbf{n}(x,y) = \mathbf{a}x + \mathbf{b}y + \mathbf{c} \qquad (2a)$$

A typical calculation is then to find **l**·**n**, and Duff [7], pointed out that such expressions may be written in the form:

$$\mathbf{l}\cdot\mathbf{n}(x,y) = \frac{\mathrm{a}x + \mathrm{b}y + \mathrm{c}}{\sqrt{\mathrm{d}x^2 + \mathrm{e}xy + \mathrm{f}y^2 + \mathrm{g}x + \mathrm{h}y + \mathrm{i}}} \qquad (2b)$$

where the terms a ... i are various combinations of scalar products of the vectors involved (see [3] for more detail on this equation). The right hand side of Equation 2b can be evaluated by finite differences. This calculation involves some overhead to find the terms a ... i, and the differences on each scanline. For each pixel the evaluation of (2b) requires 3 additions, a reciprocal square root and a multiplication. The specular term also requires an exponentiation, but the denominator is shared between the terms.

If **l** (or **h**) is also to be interpolated then the terms become somewhat more complex. The numerator is a quadratic and the denominator the square root of the product of two quadratics. The specular term again requires a further exponentiation and only one of the quadratic terms under the square root can be shared between the diffuse and specular terms. Thus the shading equation has the following form:

$$I = \text{constant} + \frac{\text{quadratic}_1}{\sqrt{\text{quadratic}_2\,\text{quadratic}_3}} + \left(\frac{\text{quadratic}_4}{\sqrt{\text{quadratic}_2\,\text{quadratic}_5}}\right)^n \quad (2c)$$

2.2 Taylor Series Approximation.

The expense of the optimization presented in Section 2.1 lies mainly in the reciprocal square root. To avoid this the whole right hand side of Equation 2b can be expressed as a two-dimensional Taylor series approximation [3]. It turns out that using only the terms up to second order provides sufficient accuracy. The diffuse term has the form:

$$I = T_5x^2 + T_4xy + T_3y^2 + T_2x + T_1y + T_0 \quad (3)$$

The specular term has the same form except that it has to be raised to a power; the ambient term can be included in the diffuse calculation. This means that the cost per pixel comes down to evaluating two quadratics and raising the specular term to the required power, a cost of 5 additions and one exponentiation.

When the light or viewer directions are not constant then the somewhat more complex Duff terms (Equation 2c) have to be approximated* . Again it turns out that a quadratic approximation works quite well.

3. Quadratic Phong Shading using Angular Interpolation.

Our approach to quadratic Phong shading depends on two major results:

1. A piecewise quadratic expression for the cosine of an angle θ, $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$, raised to a power n, $1 \leq n \leq 125$.
2. A linear expression in terms of the pixel position, x, on a scanline for the angle between the interpolated normal vector, **n**, and the light or highlight vector, **l** or **h**, from Equation 1.

We shall provide a brief statement of the results, enough to indicate the derivation and allow implementation of the ideas. An introduction with more detailed derivations of some early results may be found in [13]†. The paper covers only the results along a single scanline: here we derive the optimized preprocessing algorithm to set up the parameters for scanline processing.

3.1 Quadratic Approximation for cos^n.

A piecewise quadratic expression for the cosine of an angle θ, $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$, raised to a power n, $1 \leq n. \leq 125$, was found by least square fitting:

* Apart from published errata, there are a couple of further minor corrections that have to be applied to the published version of the approximation, these are also listed under [3] in the references section.

† Please note the corrections given with the reference at the end of this paper.

$$\cos^n\theta \approx q(\theta, n) = \begin{cases} 0 & \text{if } -b \geq \theta \text{ or } b \leq \theta \\ \dfrac{(\theta+b)^2}{b(b-a)} & \text{if } -b < \theta < -a \\ 1-\dfrac{\theta^2}{ab} & \text{if } -a \leq \theta \leq a \\ \dfrac{(\theta-b)^2}{b(b-a)} & \text{if } a < \theta < b \end{cases} \tag{4}$$

where:

$$a = \frac{n+5.6}{n(0.09n+5.2)}$$

$$b = \frac{n+65.0}{5.0n+31.7}$$

This quadratic expression, $q(\theta, n)$ is parameterized on n, thus no tables are needed. It is certainly accurate in the range it was designed for and usable beyond that range up to about $n = 256$. At $n = 1$ it deviates from the cosine where its value approaches zero: the slope is much more gradual. This is useful as it removes yet another source of Mach banding found in diffuse illumination.

3.2 Angular Interpolation of Vectors Along a Scanline.

We shall be interpolating by a vector rotation rather than by linear interpolation and renormalization as in the standard Phong method. Clearly any two vectors to be interpolated define a plane. If we interpolate linearly in this plane and renormalize then the angular velocity of the vector is constantly changing, while if we perform equiangular interpolation then the angular velocity is constant. This is illustrated in Figure 1. This difference is of no importance as long as we remain consistent — both approaches are equally 'wrong' since both ignore the fact that equal steps along the display screen of a perspective transformed picture are not necessarily equal steps along the facet of the model being rendered.

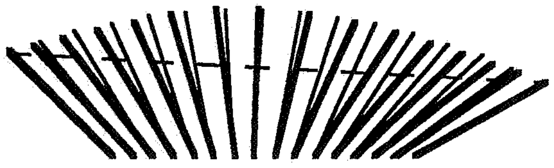

Figure 1. The different rates for linear and angular interpolation.
The progression of linearly interpolated vectors is indicated by the thick lines. Linear interpolation takes equal steps along the dashed line in the figure. The thinner lines are drawn at equal angles. In the extreme case illustrated above it is clear that the angular rate of change of a linearly interpolated vector is smaller near the end points than in the middle.

We shall draw all vectors with a common origin — the centre of a common unit sphere. This can be done because only the directions of the vectors are important, not their relative spatial position. In the derivation we shall use a few familiar results from spherical trigonometry. We adopt the convention that angles at the vertices of the spherical triangles are written as upper case Roman letters and that the angles w.r.t. to the centre of the sphere, the lengths of the sides, are written as lower case Greek letters.

We shall first consider the case of diffuse illumination with interpolated normal vectors and a fixed light direction: $\mathbf{n}_l$ and $\mathbf{n}_r$ are the normal vectors at the left and right ends of a scanline respectively, and **l** represents the light vector. The vectors $\mathbf{n}_l$ and $\mathbf{n}_r$ lie in a plane **P**.

Let **n** be the interpolated vector which lies between $\mathbf{n}_l$ and $\mathbf{n}_r$. We proceed in Δx equiangular steps of size $\frac{\Delta\theta}{\Delta x}$, where $\Delta\theta$ is the angle between $\mathbf{n}_l$ and $\mathbf{n}_r$, and Δx is the number of pixels along the scanline. On the unit sphere we construct a right-angled spherical triangle **n p l**, **p** being the perpendicular dropped from **l** onto the arc joining $\mathbf{n}_l$ and $\mathbf{n}_r$ (see Figure 2). The length of arc **p l** is γ, the constant angle between the plane **P** and the vector **l.** If we write α for the(negative) arc from **p** to $\mathbf{n}_l$, then by the cosine law for right-angled spherical triangles:

$$\cos\delta = \cos\gamma\cos\left(\frac{\Delta\theta}{\Delta x}x-\alpha\right) \tag{5a}$$

where cos δ is the required angle between the light vector **l** and the rotated normal vector **n**.

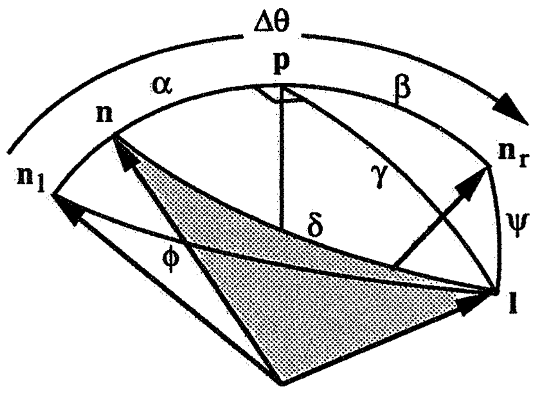

Figure 2. Angular relations within a scanline.
n is the interpolated vector which ranges in equal angular steps between $\mathbf{n}_l$ and $\mathbf{n}_r$ for a total angle of $\Delta\theta$; **p** is the perpendicular dropped from the light vector **l** onto the plane of $\mathbf{n}_l$ and $\mathbf{n}_r$. γ is the angle between the light vector and the plane of interpolation of **n**. Intensity depends on $\cos\delta = \cos\gamma\cos\left(\frac{\Delta\theta}{\Delta x}x-\alpha\right)$
ϕ and ψ are the initial and final angles between the light and the normal vectors on a scanline.

For the specular term **h** replaces the light vector in the above argument, and we get:

$$\cos^n\delta = \cos^n\gamma\cos^n\left(\frac{\Delta\theta}{\Delta x}x-\alpha\right) \tag{5b}$$

Since γ is constant and by substituting the expression $q(\theta, n)$ from Equation 4 we get a piecewise quadratic expression for Phong shading along a scanline.

3.3 Angular Interpolation Between Scanlines.

The vectors $\mathbf{n}_l$ and $\mathbf{n}_r$ used above as starting points for scanline angular interpolation are rather expensive to compute if we have to use an explicit rotation between scanlines as suggested in [13]. Although linear interpolation could be used between scanlines the resulting highlights would no longer be symmetric due to the different rates of angular and linear interpolation, even if the underlying geometry were symmetric.

Consider the vector $\mathbf{a} = \mathbf{n}_l \times \mathbf{n}_r$, the axis about which $\mathbf{n}$ rotates as it is interpolated on a scanline. Since all vectors except $\mathbf{a}$ are unit vectors (we assume angles are less than right angles):

$$\begin{aligned}
\cos\gamma = \sin\left(\frac{\pi}{2} - \gamma\right) &= \frac{|\mathbf{a} \times \mathbf{l}|}{|\mathbf{a}|} \\
&= \frac{\sqrt{(\mathbf{n}_l \cdot \mathbf{l})^2 - 2(\mathbf{n}_l \cdot \mathbf{n}_r)(\mathbf{n}_l \cdot \mathbf{l})(\mathbf{n}_r \cdot \mathbf{l}) + (\mathbf{n}_r \cdot \mathbf{l})^2}}{\sin\Delta\theta} \\
&= \sqrt{\frac{\cos^2\phi - 2\cos\phi\cos\Delta\theta\cos\psi + \cos^2\psi}{1 - \cos^2\Delta\theta}} \\
&= \sqrt{\frac{(\cos\phi - \cos\psi)^2}{1 - \cos^2\Delta\theta} + \frac{2\cos\phi\cos\psi}{1 + \cos\Delta\theta}} \qquad (6)
\end{aligned}$$

Where ϕ is the initial (left-hand) angle and ψ the final (right-hand) angle between the light and interpolated normal (Figure 2).

It is also clear from the cosine law for right-angled spherical triangles that:

$$\cos\gamma = \frac{\cos\phi}{\cos\alpha} = \frac{\cos\psi}{\cos\beta} \qquad (7)$$

We take a trapezium that has its parallel edges aligned with the x-axis of the display as a primitive polygon to be shaded. At the four vertices we have four normals: $\mathbf{n}_{l0}$, $\mathbf{n}_{r0}$, $\mathbf{n}_{l1}$, $\mathbf{n}_{r1}$. We again consider the case of diffuse illumination with a constant light direction $\mathbf{l}$ (Figure 3).

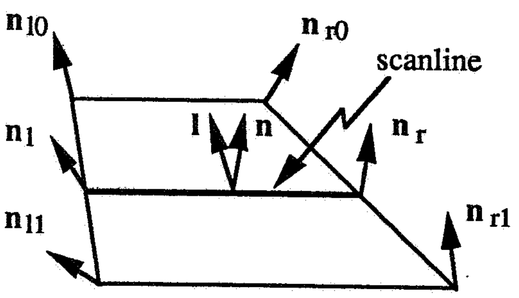

Figure 3. Vectors involved in diffuse shading of a trapezoidal facet.
The four normal vectors at the vertices are given as well as the constant light vector, if the light is not within the scene. The other normal vectors are obtained by interpolation.

Once again, since we are only interested in the directions of the vectors and not their relative position we can bring all of them to a common origin (Figure 4). In calculating both diffuse and specular terms there are two pairs of vectors between which we have to be interpolate and one constant vector with respect to which we shall be measuring angles. If we take the case of diffuse illumination we interpolate: $\mathbf{n}_{l0} \rightarrow \mathbf{n}_{l1}$, $\mathbf{n}_{r0} \rightarrow \mathbf{n}_{r1}$. The interpolation along the left edge ($\mathbf{n}_{l0} \rightarrow \mathbf{n}_{l1}$) proceeds in equal steps of angle σ_l, while on the right edge the steps are of size σ_r. S is the angle between the two arcs, the left hand and right hand ones, along which the vectors are interpolated. The angles of interest are now explicitly subscripted with the index 'i' to indicate the particular scanline.

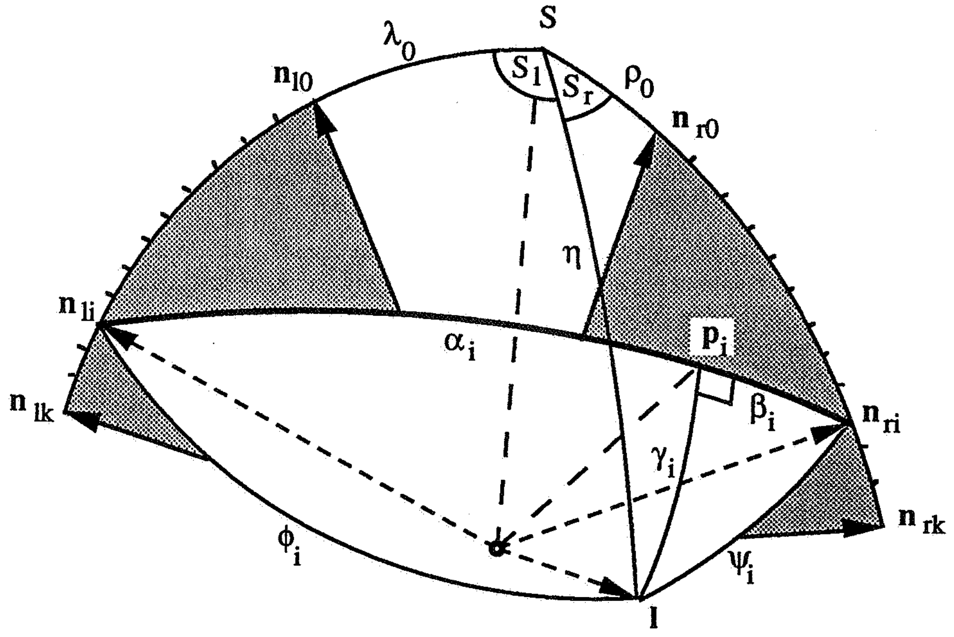

Figure 4. Angular relations over a trapeziodal facet.
All vectors drawn in Figure 3 are brought to a common origin, marked by a circle. The interpolation between scanlines is indicated on the left and right by the shaded arcs. The angular interpolation within a scanline is indicated by the thicker arc from $\mathbf{n}_{li}$ to $\mathbf{n}_{ri}$.

The cosine rule for spherical triangles immediately gives an expression for the range of interpolation as well as the start and end angles for interpolation on each scanline i:

$$\cos \Delta\theta_i = \cos \lambda_i \cos \rho_i + \sin \lambda_i \sin \rho_i \cos S \tag{8a}$$

$$\cos \phi_i = \cos \lambda_i \cos \eta + \sin \lambda_i \sin \eta \cos S_1 \tag{8b}$$

$$\cos \psi_i = \cos \rho_i \cos \eta + \sin \rho_i \sin \eta \cos S_2 \tag{8c}$$

where $\lambda_i = \lambda_0 + i\,\sigma_l$, $\rho_i = \rho_0 + i\,\sigma_r$, $S = S_1 + S_2$, and η is the arc between **l** and the intersection of the left and right interpolation ranges of the normals at S.

Combining Equations 6 – 8 yields expressions for α_i, $\Delta\theta_i$, and γ_i. These can be used to drive the quadratic expressions (Equations 4 and 5) for Phong shading along a scanline.

3.4 Practical Considerations

In implementing the above expressions a few points are important:

1. The degenerate cases when one or more of the vectors coincide, or lie in the same plane, or if the facet to be shaded is in fact a triangle, have to be detected and dealt with separately with simpler processing.
2. The crossover points in the piecewise quadratic approximation do not necessarily lie on an integer pixel boundary. If forward differences are used to calculate the pixel values then the intensity and the 1st and 2nd differences have to be computed at the ceiling of the crossover points.

Further details on the implementation may be found in the Appendix.

3.5 Light or Viewer in the Scene

If either the light source or the viewer are not at inifinite distance then we have to apply an approximation. In the case of diffuse illumination the approximation can be intuitively understood as replacing a curved surface with a light source near it with a more convex surface with the light source at infinity. The same can be done for the **h** vector if the viewer or the light is not in a fixed direction.

In order to apply the approximation consider diffuse shading of a trapezium where the light is within the scene. At each vertex of the trapezium two vectors are defined, a normal and a light direction, so we have the pairs: ($\mathbf{n}_{l0}$, $\mathbf{l}_{l0}$), ($\mathbf{n}_{l1}$, $\mathbf{l}_{l1}$), ($\mathbf{n}_{r0}$, $\mathbf{l}_{r0}$), ($\mathbf{n}_{r1}$, $\mathbf{l}_{r1}$).We pick one of the **l** vectors, $\mathbf{l}_{l0}$, say, or even any arbitrary vector. The **n** vectors are now rotated by the same amount as the rotation required to rotate their corresponding **l** vector to the single chosen vector. For example, if our chosen vector is $\mathbf{l}_{l0}$ then $\mathbf{n}_{l1}$ is rotated to give $\mathbf{n}'_{l1}$ by the same amount as that required to rotate $\mathbf{l}_{l1}$ onto $\mathbf{l}_{l0}$. Clearly the angle between $\mathbf{n}_{l1}$ and $\mathbf{l}_{l1}$ is the same as that between $\mathbf{n}'_{l1}$ andthe chosen constant vector $\mathbf{l}_{l0}$.

There are certain pathological cases involving large angles of interpolation and where the normals lie at about 90° to the light (or highlight) vectors where the approximation breaks down. This is because the angles between **n** and **l** at the vertices are correct but the approximated in between values may be different.

3.6 Anti-Aliasing.

A simple approximation to anti-aliasing in the form of area sampling may be applied. A pixel is considered to be a square. On the edges of trapezia the square may only be partly covered; the whole trapezium may even be smaller than a pixel. The facet contributes to the pixel in proportion to the area covered by the trapezium. This can be accumulated for all facets which overlap the pixel square if we are guaranteed that no obscured edges will be shaded, e.g., if an object space hidden surface algorithm is used. Transparent polygons are of course allowed.

The diagonal edges of a trapezium, which cut more than one pixel on a single scanline, may be considered a linear modulation of the quadratic shading function. The resultant is a cubic shading function. In other cases, except for pathological cases (single pixel wide trapezia etc.), a single constant factor suffices to multiply the edge pixels. Thus the additional benefit of quadratic shading can be edge anti-aliasing using third order forward differences.

4. Discussion and Comparison of Shading Methods.

It is rather difficult to compare the complexity of various algorithms, particularly if these comparisons are to be used for making practical decisions. We have have chosen to express our complexity in terms of the operations typically available on modern RISC architectures which have divisions, seeds for reciprocal square roots, etc. implemented. The salient feature of these architectures is that there is a single standard cycle time for the basic operations — all are equal to the time of a multiplication, say. To be specific we have chosen to express complexity in terms of the kinds of operations found on the Intel® i860™ microprocessor [10], which is designed for 3-D graphics workstations. Further costs of operations were found by referring to standard algorithms, such as those in [1] and [12].

Various typical operations were given the costs indicated in Table 1.

Mnemonic	Operation	Elementary Operations	Flops	Total Operations
acos	arc cosine	9+ 6* 1sqrt	13+ 20*	33
ang	angle	11+ 9* 1sqrt	15+ 23*	38
cos	cosine	8+ 8*	8+ 8*	16
crp	cross product	3+ 6*	3+ 6*	9
div	division	1div	2+ 5*	7
dot	dot product	2+ 3*	2+ 3*	5
exp	exponent int	7*	7*	7
expf	exponent float	10+ 8* 1rsr	14+ 22*	36
len	length	2+ 3* 1sqrt	4+ 14*	18
ncp	normalized crp	5+ 12* 1rsr	9+ 25*	34
rsr	reciprocal sqrt	1rsr	4+ 13*	17
sin	sine	7+ 8*	7+ 8*	15
sqrt	square root	1sqrt	4+ 14*	18

Table 1. Standard Operations and their Complexity.

The comparisons have been made between the following methods, for the cases where the light vector and **h** vector are constant (i.e., directional light source and viewer at infinity) and where the vectors vary over the facets being shaded (i.e., point light source and viewer within the scene):

1. Basic Phong. By this is meant a method which performs incremental interpolation between scanlines and then at each scanline recomputes the parameters for interpolation along scanlines.

2. Rotation Independent Phong. In the case of triangles a unique expression may be found for the linearly interpolated vectors. This can be evaluated more efficiently.

3. Duff's optimization applied to triangle primitives, without forward differences.
4. Duff's optimization applied to triangles, forward differences being used both between and along scanlines.
5. Bishop and Weimer's quadratic approximation, using forward differences in both x and y.
6. Angular Interpolation for trapezia, the primitive for which the version of the method presented here is aimed at.
7. Angular Interpolation for triangles. A number of terms may be removed when triangles are used. The per facet overhead is much reduced because, for example, no cross products are needed.

The costs were calculated by the most efficient means known, reusing results when possible. Operations were costed at the rates indicated in Table 1. It should be noted that costs of maintaining edges were included in all methods, these were estimated at 12 operations (1 division, 2 additions and 2 multiplications) for preprocessing, and 2 operations (additions) per scanline. For the case where the lights and/or viewer are not at infinity the calculation of the **h** vectors was included as an overhead (3 additions and 3 multiplications by a reciprocal square root of a scalar product) at 28 operations per **h** vector.

Method		Costs **l** & **h** constant			Costs **l** & **h** varying		
		Facet	Line	Pixel	Facet	Line	Pixel
1.	Basic Phong	24	21	46	144	45	64
2.	Rotation Indep. Phong	73	2	46	247	2	64
3.	Duff Calculated	130	19	32	426	67	57
4.	Duff using Differences	167	6	32	581	17	57
5.	Bishop & Weimer	273	8	12	732	8	12
6.	Angular, Trapezia	434	282	5	1080	353	5
7.	Angular, Triangle	198	282	5	578	353	5

Table 2. Costs in Elementary Operations for Various Shading Methods. The methods described in this paper are analysed in terms of their incremental costs incurred for preprocessing every facet (triangle or trapezium), and within a facet for preprocessing the scanlines and finally the incremental cost of generating pixels.

Once the costs of the various methods has been presented as above, the next question is the break even point. There is a break even point because cheaper methods have greater scanline and facet overheads. This question is also difficult to answer since shading quality and the basic primitives differ. However to give an indication break even points were calculated by asking at what point the facet overhead was outweighed by the linear accumulation of benefits per scanline and the quadratic accumulation of benefits per pixel.

	Break even points for **l** and **h** constant.					Break even points for **l** and **h** both varying.				
Method	1.	2.	3.	4.	5.	1.	2.	3.	4.	5.
2.	3					3				
3.	8	8				66	133			
4.	8	8	3			38	65	4		
5.	7	7	6	6		10	10	5	4	
6.	59	64	117	124	1578	55	61	53	60	2528
7.	49	53	100	107	1511	41	46	36	41	2385

Table 3. Break even points in terms of pixels for the various methods. The methods numbers are the same ones used in the previous table. For example, the entry in the row labelled 7 in the first column of the second table is 41. This means that for facets of average size 41 pixels or more the cost of angular interpolation is less than simple Phong shading.

It can be seen that for constant light and view vectors the Duff optimization using forward differences (method 4) has a low break even point and because of its greater simplicity and accuracy is probably the better method. The break even of Bishop and Weimer's method (method 5) is also low.

In the case of the point light source or viewer within the scene, the method of choice is clearly the Taylor series approximation of Bishop and Weimer (method 5). It has a break even of 10 pixels which is only surpased by the relatively inefficient rotation independent Phong method (number 2).

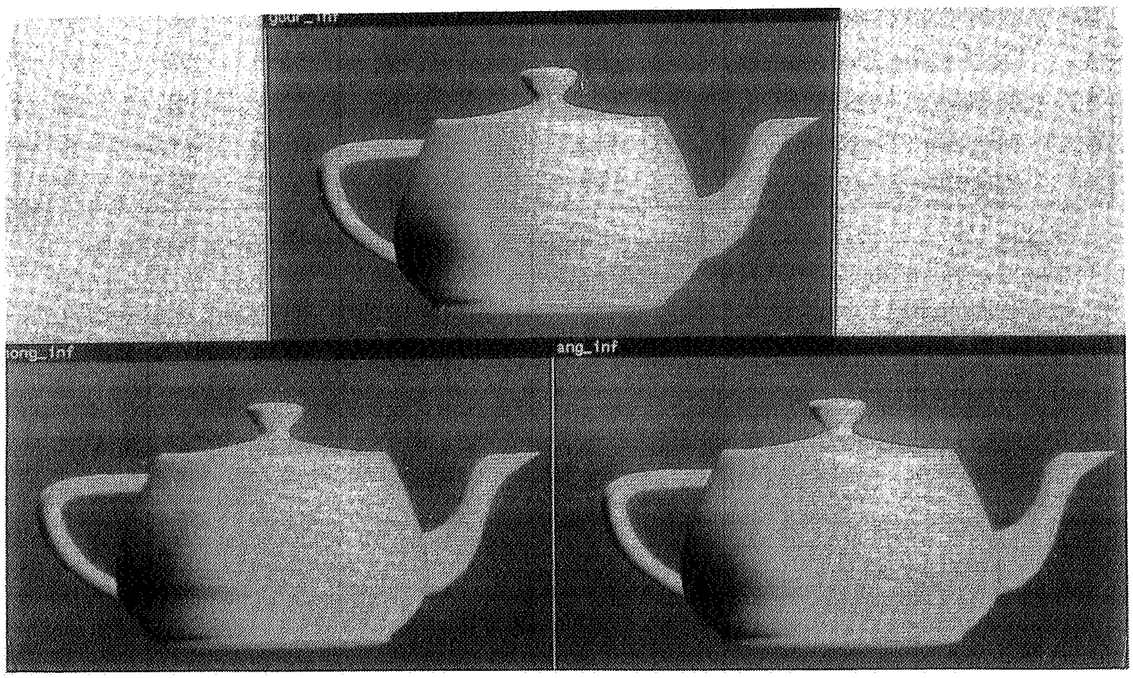

Figure 5. Comparison of Gouraud, Phong and Angular Shading. The objects were all shaded with constant light and viewer directions. The top picture is Gouraud shaded, the left hand is Phong and the right hand is angular interpolation. There is no difference between the results for Phong and angular shading and both are a improvement over Gouraud shading.

It is clear that the very low pixel cost of the angular interpolation method incurs too high an overhead to make it a general solution, this is particularly the case with the simplest situation: light and view direction constant. The prime virtue of this method is its ultimate simplicity and extremely low complexity which make it an ideal solution for hardware implementation. As a software method it is attractive if the facets and their overhead can be farmed out to a number of independent processors.

As far as the appearance of the angular interpolation method is concerned no visible differences could be detected in comparison to standard Phong shading (Figures 5 & 6). The method dealt successfully with light and viewers within the scene. Naturally the shading effects were much better than for Gouraud shading.

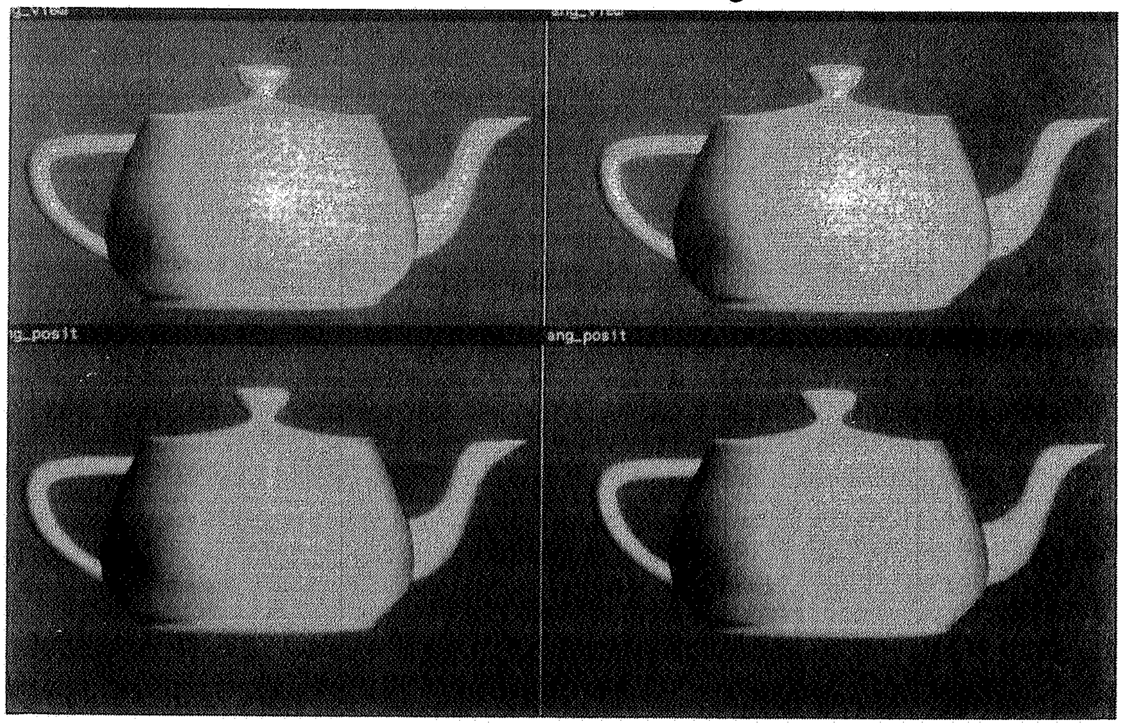

Figure 6. Phong vs. Angular Shading for point lights and viewer in scene. The Phong shaded images are on the left and the angular interpolated images on the right. In the top pair the light source is within the scene and in the bottom pair the viewer approaches the object more closely. The changing lighting effects are accurately modelled, again there is no difference between the results for Phong and angular shading.

5. Conclusion.

The angular interpolation method is prefered when low incremental pixel costs are essential. It may be attractive as a general shading if the overheads for the facet shading can be distributed amongst parallel processors. The Taylor series approximation of Bishop and Weimer is a good software solution to fast Phong shading. The overhead of the Taylor series approximation compares very well with those of other optimizations of the Phong shading model.

Appendix: Pseudo Code for Angular Interpolation.

The following pieces of (pseudo-)code present the details of implementing the general case of shading a scanline aligned trapezium. The variables used correspond with those employed in Section 3. The various tests to detect degenerate cases have been symbolically brought into various cases of a case statement. In fact it is more efficient to use results produced during normal processing for the tests.

```
procedure shader( Vector  $\mathbf{n}_{l0}, \mathbf{n}_{lk}, \mathbf{n}_{r0}, \mathbf{n}_{rk}$, light
                  Float    gloss,
                  Integer upperScanline, lowerScanline )

  height := lowerScanline - upperScanline
  case checkForDegeneracy( $\mathbf{n}_{l0}, \mathbf{n}_{lk}, \mathbf{n}_{r0}, \mathbf{n}_{rk}$ ) of
  allEqual:   ... inOnePlane: ... leftConstant: ... rightConstant: ...
  generalCase:
    $\mathbf{axis}_{nl}$ := normalizedCrossProduct( $\mathbf{n}_{l0}, \mathbf{n}_{lk}$ )      /* axes about which vectors   */
    $\mathbf{axis}_{nr}$ := normalizedCrossProduct( $\mathbf{n}_{r0}, \mathbf{n}_{rk}$ )      /* rotate between scanlines   */
    intersection := normalizedCrossProduct( $\mathbf{axis}_{nl}, \mathbf{axis}_{nr}$ )
    cosd := dotProduct( intersection, light )    /* Reverse intersection if cosd < 0 */
    dPlane := crossProduct( intersection, light )
    $\text{triple}_l$ := dotProduct( dPlane, $\mathbf{axis}_{nl}$ )        /*  scalar triple product=sin d * cosSl */
    $\text{triple}_r$ := dotProduct( dPlane, $\mathbf{axis}_{nr}$ )        /*  scalar triple product=sin d * cosSr *
    cosS := dotProduct( $\mathbf{axis}_{nl}, \mathbf{axis}_{nr}$ )   /* S= angle between planes of interpolation. */
    /***   Calculate angles on left interpolation ***/
    $\text{cosStart}_l$ := dotProduct( intersection, $\mathbf{n}_{l0}$ )
    $\text{cosEnd}_l$ := dotProduct( intersection, $\mathbf{n}_{lk}$ )
    $\text{cosRange}_l$ := dotProduct( $\mathbf{n}_{l0}, \mathbf{n}_{lk}$ )
    $\text{lambda}_0$ := arccos( $\text{cosStart}_l$ )
    $\text{lambda}_{step}$ := arccos( $\text{cosRange}_l$ )/height
    if $\text{cosEnd}_l$ < minimum( $\text{cosStart}_l$, $\text{cosRange}_l$ ) then $\text{lambda}_{step}$ := -$\text{lambda}_{step}$
    $\text{sinStart}_l$ := sin( $\text{lambda}_0$ )
    $\text{cosStep}_l$ := cos( $\text{lambda}_{step}$ )
    $\text{sinStep}_l$ := sin( $\text{lambda}_{step}$ )
    /***   Calculate angles for right interpolation: same as left above ***/
    /***   $\text{cosStart}_r$, $\text{cosEnd}_r$ $\text{cosRange}_r$ $\text{rho}_0$ $\text{rho}_{ste}$ $\text{sinStart}_r$ $\text{cosStep}_r$$\text{sinStep}_r$ ***/
    /*** Preprocessing for trapezium done, now we can proceed with the scanlines ***/
    scanline := upperScanline
    while scanline <= lowerScanline do
      cosPhi := cosd*$\text{cosStart}_l$+ $\text{sinStart}_l$*$\text{triple}_l$
      cosPsi := cosd*$\text{cosStart}_r$+ $\text{sinStart}_r$*$\text{triple}_r$
      cosDeltaTheta :=  $\text{cosStart}_l$*$\text{cosStart}_r$+$\text{sinStart}_l$*$\text{sinStart}_r$*cosS
      gamma := (cosPsi - cosPhi)*(cosPsi - cosPhi) + 2.0*(1.0 -
cost)*cosPhi*cosPsi
      gamma := sqrt( gamma/(1.0 - cosDeltaTheta *cosDeltaTheta ) )

      deltaTheta :=  arccos( cosDeltaTheta )
      cosAlpha := cosPhi/gamma
      cosBeta := cosPsi/gamma
      if(cosBeta < cost) && (cosBeta < cosAlpha) then  alpha := -arccos( cosAlpha )
      else  alpha := arccos( cosAlpha )
      $\text{cosNext}_l$ := $\text{cosStart}_l$*$\text{cosStep}_l$- $\text{sinStart}_l$*$\text{sinStep}_l$
      $\text{cosNext}_r$ := $\text{cosStart}_r$*$\text{cosStep}_r$- $\text{sinStart}_r$*$\text{sinStep}_r$
      $\text{sinNext}_l$ := $\text{sinStart}_l$*$\text{cosStep}_l$+ $\text{cosStart}_l$*$\text{sinStep}_l$
```

```
            sinNext_r := sinStart_r*cosStep_r+ cosStart_r*sinStep_r
            cosStart_l := cosNext_l
            cosStart_r := cosNext_r
            sinStart_l := sinNext_l
            sinStart_r := sinNext_r
            gamma := exponentiate( gamma, gloss)
            call yProc( alpha, deltaTheta, gamma)
        scanline++
    od /* while scanline <= lowerScanline */
esac /* anyDegeneracy */
end procedure

procedure yProc( Float alpha, deltaTheta, gamma)
/*
    Assume a,b, parameters of the quadratic approximation to cos^n are set.
    The denominators used below are sd13 = 2/(b*(b-a)),  sd2 = -2/(b*a).
*/
    width := xr - xl
    dtdx := deltaTheta/width
    dxdt := 1.0/dtdx
    dtdx := dtdx*dtdx*gamma
    alphadxdt := alpha*dxdt
    middle := alpha*dxdt + xl
    xbl := int( middle - b*dxdt)      /* Boundaries of the various ranges of the      */
    xal := int( middle - a*dxdt)      /* quadratic approximation to cos^n in terms   */
    xar := int( middle + a*dxdt)      /* pixels rather than angles                    */
    xbr := int( middle + b*dxdt)
    if xr > xbl then
        if xl < xbl then left := xbl
        else left := xl
        if left < xal then
            call eval( left, min( xr, xal), sd13, 0.0, xbl )
            left := xal+1
        fi
        if xr > xal then
            if left < xar then
                call  eval( left,min( xr, xar), sd2, gamma, xal)
                left := xal+1
            fi
            if left < xbr && xr > xar  then call eval( left, min( xr, xar), sd13, 0.0, xar)
        fi
    fi
end procedure

procedure eval(    FixedPoint     xstart, xend, ddi, fnExtreme, offset )
    deltaX := xend - xstart
    i := fnExtreme + ddi*offset*offset/2            /*Start Intensity */
    di := ddi/2 + ddi*offset                            /* First difference */
    call generatePixels( xstart, deltaX, i, di, ddi)   /* Calculate via forward differences */
end  /* procedure eval */
```

References

1 Abramowitz, M., and Stegun, I.A. (eds.) *Handbook of Mathematical Functions*. Dover Publications, inc. , New York, 1970.

2 Baum, D.R. & Winget, J.M. Real time radiosity through parallel processing and hardware acceleration. *Computer Graphics* 24, 2 (March 1990), 67-75 & 261.

3 Bishop, G., and Weimer, D.M. Fast Phong shading. *Computer Graphics* 20, 4 (Aug. 1986), 103-106. Corrections appeared in *Computer Graphics* 21, 1 (Jan. 1987) p. 53. In the expression for T_4 on p. 105 it appears the following terms should also be replaced: $-fjlqr$ with $+fjlqr$ and $-2f^2nr$ with $-2fl^2nr$.

4 Blinn, J.F. Models of light reflection for computer synthesized pictures. *Computer Graphics* 11, 2 (1977) 192-198.

5 Bui Tuong Phong. Illumination for computer generated pictures. *Comm. ACM* 18, 6 (June 1975) 311-317.

6 Deering, M., Winner, S., Schediwy, B., Duffy, C., and Hunt, N. The triangle processor and normal vector shader: A VLSI system for high performance graphics. *Computer Graphics* 22, 4 (Aug. 1988), 21-30.

7 Duff, T. Smoothly shaded renderings of polyhedral objects on raster displays. *Computer Graphics* 13, 2 (Aug. 1979), 270-275.

8 Fuchs, H., Poulton, J., Eyles, J., Greer, T., Goldfeather, J., Ellsworth, D., Molnar, S., Turk, G., Tebbs, B., and Isreal, L. Pixel-Planes 5: a heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics* 23, 3 (July 1989), 79-88.

9 Hall, R. *Illumination and Color in Computer Generated Imagery*. Springer-Verlag, Berlin, 1989.

10 Intel Corporation. *i860™ Microprocessor Data Sheet*. Order Number 240296. Intel Corporation, Santa Clara, Ca., 1989.

11 Jayasinghe, J.A.K.S., Kuijk, A.A.M., and Spaanenburg, L. A display controller for a structured frame store system. In *Advances in Graphics Hardware III*, A.A.M. Kuijk (ed.) Springer-Verlag, Berlin, 1991.

12 Knuth, D.E. *Seminumerical Algorithms*, 2nd ed., vol 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Mass., 1981.

13 Kuijk, A.A.M., and Blake, E.H. Faster Phong shading via angular interpolation. *Computer Graphics Forum* 8, 4 (Dec. 1989), 315-324. Please note that on p. 321 the definitions of a and b should be swapped.