1991

V.C.J. Disselkoen

Real-time quadratic shading

Computer Science/Department of Interactive Systems Report CS-R9123 April

Real-Time Quadratic Shading

Vincent C.J. Disselkoen

CWI
P.O. Box 4079, 1009 AB Amsterdam,
The Netherlands
e-mail: vincent@cwi.nl

This document describes a technique generating quadratic shading functions for scanlines when using a shading model closely related to Phong shading. The specular term is expressed using quadratic curves, obviating the need for pixel-time exponentiation. The method combines concepts from vector coordinate interpolation and vector rotation techniques.

CR Categories and Subject Descriptors:
I.3.1 Hardware Architecture: Raster Display Devices
I.3.3 Picture/Image Generation: Display Algorithms
I.3.7 Three-Dimensional Graphics and Realism: Shading
Key Words and Phrases:
Phong Shading, Quadratic Shading, Image Synthesis

1 Introduction

Many computer image generation systems represent curved surfaces using polygon meshes. In order to display these objects realistically, a shading function must be used that assigns intensity values to each of the visible pixels in such a way that the transitions at polygon boundaries are at least smooth. That is, the intensity should be continuous over the surface, and the direction of the tangent to the intensity curve on both sides of a boundary should be equal.

Phong shading [2] is a technique satisfying the above requirements. However, Phong shading of objects containing large numbers of polygons using multiple lightsources is usually too complex and costly to allow real-time generation of images. Real-time evaluation of the Phong shading function for one light source would require pixel-rate evaluation of a reciprocal square root, exponentiation and several elementary operations (sections 3 and 4). In contrast, the simple and cheap Gouraud shading method [1] which computes intensity values at polygon vertices and linearly interpolates these over the surface requires a huge amount of refinement, i.e. subdivision into smaller and smaller polygons, to reduce Mach banding and disappearance of highlights.

There have been several successful simplifications of the original Phong model that allow more efficient computation without sacrificing image quality. Duff [3] rewrote Phong's expression to allow more efficient evaluation and forward differencing (section 2). The method of Bishop and Weimer [4] uses a very cheap shading function that produces good results by approximating the Phong function using a two-dimensional quadratic Taylor series, but still requires pixel-rate exponentiation, which is impractical for hardware implementation.

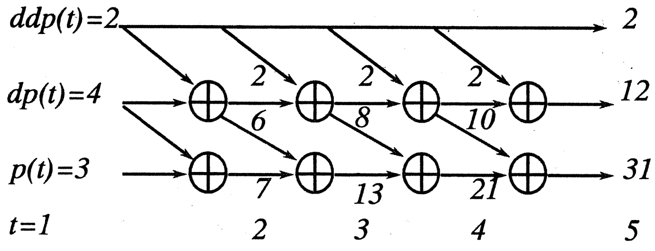
We propose a shading function [5] that allows use of quadratic curves to approximate the diffuse as well as the specular component, implying that exponentiation need only be performed once per scanline instead of once per pixel. All possible preprocessing of constants per scanline is per-

Report CS-R9123
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

formed at scanline level, and all possible preprocessing of constants per frame is performed at frame level.

2 Forward differencing

Forward differencing is an efficient method for evaluation of polynomials. A polynomial, e.g. $p(x)$, is specified using its initial value at the location $x=t$ and first forward difference, $dp(t)=p(t+1)-p(t)$. The evaluator then uses the initial value at t, and adds the first forward difference to the original value to arrive at the value $p(t+1)=p(t)+dp(t)$. A similar process is performed to compute the first forward difference. The n-th forward difference is constant. For example, evaluation of the function $p(t)=t^2+t+1$ over the range $t=1$ to $t=5$ is performed as follows: the first forward difference is $dp(t)=(\,(t+1)^2+(t+1)+1\,)-(\,t^2+t+1) = 2t+2$, and the second forward difference is $ddp=(2(t+1)+1)$ $-\ (2t+1)=2$. The initial values for the second order forward difference are then $p(1)=3$, $dp(1)=4$ and $ddp=2$.

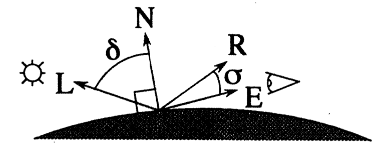


The method computes the values of the polynomial for each location in the range using a mere n additions per location.This implies that decomposing a shading function in a vertical component and a horizontal component which is constant per scanline, both of which can be expressed as polynomials only, would produce a very efficient shading technique.

3 Phong shading

Phong shading [2] is a method for smoothly shading curved surfaces. The curved surfaces are defined using a mesh of polygons of which normals, light vectors, eye direction vectors and reflection direction vectors are given at each vertex. The function that assigns intensity values to each pixel was established empirically and gives a realistic approximation of the actual lighting. The shading function used consists of an ambient term, and for each light source, a diffuse and specular term. Phong's method derives the intensity of a pixel due to a certain light source by adding the diffuse term $\cos\delta$ to the specular term $\cos^s\sigma$, where s is a constant defining the shininess of the surface. The angles δ and σ are the angle between the normal vector N of the surface and the direction of the light source L, and the angle between the viewing direction E and the direction of reflection R. Therefore, for each pixel, the expression that must be evaluated is

$$Intensity = \sum_{polygons} (ambient + \sum_{lights} (\cos\delta + \cos^s\sigma))$$



The methods used for interpolation of the normal, light, eye and reflection vector can be divided into

- ***vector coordinate interpolation techniques***, such as the original Phong [2] shading method and its optimizations by Duff [3] and Bishop and Weimer [4], where an interpolation is performed by linearly interpolating two vectors and normalization, and
- ***vector rotation techniques***, where the first of these two vectors is rotated towards the second one around their normal vector, such as the equi-angular interpolation method proposed by Ku-

ijk and Blake [5].

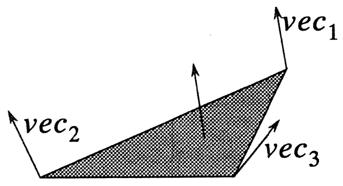
The four interpolated vectors are required to have unit length. The vector coordinate interpolation techniques have the advantage of being suited to producing a vector incrementally, while rotational techniques typically require matrix multiplications or angular updates. The vectors thus derived, however, do not require normalization while the vector coordinate interpolation techniques require a reciprocal square root and scalar product to arrive at a unit vector.

4 Vector Coordinate Interpolation techniques

Assuming that a polygon is defined by its vertices and the normal, light, eye and reflection direction vectors, *N* and *R* are expressed using a linear function in *x* and *y*. For the moment, *L* and *E* are assumed to be constant during interpolation, but these can be derived similarly when this is not the case.

For each light source, the original Phong shading method performs vector coordinate interpolation, normalization and scalar product with a light vector to obtain an intensity component, followed by an exponentiation of the specular component.

$$I = ambient + \frac{\vec{A}x + \vec{B}y + \vec{C}}{|\vec{A}x + \vec{B}y + \vec{C}|} \bullet \vec{L} + \left(\frac{\vec{S}x + \vec{T}y + \vec{U}}{|\vec{S}x + \vec{T}y + \vec{U}|} \bullet \vec{E}\right)^{s}$$



Given the normal and other vectors at the vertices of a triangle, we have to determine two triplets *A,B,C* and *S,T,U* to interpolate the four vectors. Suppose the normals are defined by (x_1,y_1,vec_1), (x_2,y_2,vec_2), (x_3,y_3,vec_3), then the triplets can be constructed using the following method:

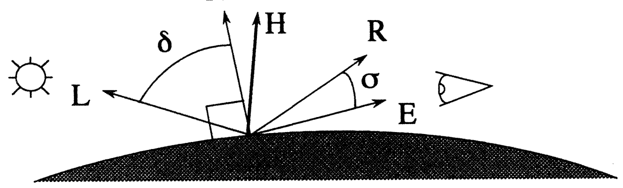
$$\begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}^{-1} \begin{bmatrix} vec_{11} \\ vec_{21} \\ vec_{31} \end{bmatrix} \quad \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} x_1 & y_2 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}^{-1} \begin{bmatrix} vec_{12} \\ vec_{22} \\ vec_{32} \end{bmatrix} \quad \begin{bmatrix} a_3 \\ b_3 \\ c_3 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}^{-1} \begin{bmatrix} vec_{13} \\ vec_{23} \\ vec_{33} \end{bmatrix}$$

It is easily seen that the complexity of evaluating the intensity expression used in this method for each pixel is four scalar products, two reciprocal square roots, two vector additions, an exponentiation and two additions. If light and eye vector are not at infinity, these have to be interpolated as well.

4.1 The H vector

A generally accepted simplification to the above method is to assume that the specular component can be rewritten as:

$$\vec{E} \bullet \vec{R} \cong \vec{N} \bullet \vec{H}, \vec{H} = \frac{\vec{L} + \vec{E}}{|\vec{L} + \vec{E}|}$$



This approach was suggested by Blinn [7]. *H* represents the surface normal that would produce the mirror reflection from the light to the eye. One can not simply assume that both methods are equivalent. For example, the shininess index must be larger in order to produce a comparible effect with this function. For more details, we refer to Hall [6], pages 76 through 78.

4.2 Duff's optimization

Duff [4] combined interpolation and evaluation of the components allowing optimizations and used forward differencing by rewriting Phong's expression as follows:

$$I = ambient + \frac{(\vec{A}\bullet\vec{L})x + (\vec{B}\bullet\vec{L})y + \vec{C}\bullet\vec{L}}{\left|(\vec{A}\bullet\vec{L})x + (\vec{B}\bullet\vec{L})y + \vec{C}\bullet\vec{L}\right|} + \left(\frac{(\vec{S}\bullet\vec{E})x + (\vec{T}\bullet\vec{E})y + \vec{U}\bullet\vec{E}}{\left|(\vec{S}\bullet\vec{E})x + (\vec{T}\bullet\vec{E})y + \vec{U}\bullet\vec{E}\right|}\right)^{s}$$

$$I = ambient + \frac{ax + bx + c}{\sqrt{dx^2 + exy + fy^2 + gx + hy + i}} + \left(\frac{\dot{a}x + \dot{b}x + \dot{c}}{\sqrt{\dot{d}x^2 + \dot{e}xy + \dot{f}y^2 + \dot{g}x + \dot{h}y + \dot{i}}}\right)^{s} L$$

The a through r are constants during interpolation, and can therefore be precomputed once per frame

$$d = \vec{A}\bullet\vec{A} \qquad \dot{d} = \vec{S}\bullet\vec{S}$$

$$a = \frac{\vec{L}\bullet\vec{A}}{|\vec{L}|} \quad b = \frac{\vec{L}\bullet\vec{B}}{|\vec{L}|} \quad c = \frac{\vec{L}\bullet\vec{C}}{|\vec{L}|} \qquad e = 2(\vec{A}\bullet\vec{B}) \qquad \dot{e} = 2(\vec{S}\bullet\vec{T})$$

$$f = \vec{B}\bullet\vec{B} \qquad \dot{f} = \vec{T}\bullet\vec{T}$$

$$\dot{a} = \frac{\vec{E}\bullet\vec{S}}{|\vec{E}|} \quad \dot{b} = \frac{\vec{E}\bullet\vec{T}}{|\vec{E}|} \quad \dot{c} = \frac{\vec{E}\bullet\vec{U}}{|\vec{E}|} \qquad g = 2(\vec{A}\bullet\vec{C}) \qquad \dot{g} = 2(\vec{S}\bullet\vec{U})$$

$$h = 2(\vec{B}\bullet\vec{C}) \qquad \dot{h} = 2(\vec{T}\bullet\vec{U})$$

$$i = \vec{C}\bullet\vec{C} \qquad \dot{i} = \vec{U}\bullet\vec{U}$$

Naturally, use of an H vector simplifies the above because $S=A$, $T=B$ and $U=C$.

4.3 Bishop and Weimer's approximation

To avoid the considerable overhead involved to compute Duff's expression, Bishop and Weimer [4] coded the above expressions using two-dimensional Taylor series expanded to the quadratic terms. Bishop and Weimer propose a quadratic function for both components which can easily be encoded using forward differencing. Therefore, the update overhead for both the diffuse term and the specular term drops to a mere two additions per pixel, but the specular term still has to be exponentiated for each pixel that is evaluated. Both expressions have the following form:

$$I_{term} = T_5x^2 + T_4xy + T_3y^2 + T_2x + T_1y + T_0 \qquad where$$

$$T_5 = \frac{3cg^2 - 4cdi - 4agi}{8i^2\sqrt{i}} \qquad T_3 = \frac{3ch^2 - 4cfi - 4bhi}{8i^2\sqrt{i}} \qquad T_1 = \frac{3bi - ch}{2i\sqrt{i}}$$

$$T_4 = \frac{3cgh - 2cei - 4bgi - 2ahi}{4i^2\sqrt{i}} \qquad T_2 = \frac{2ai - cg}{2i\sqrt{i}} \qquad T_0 = \frac{c}{\sqrt{i}}$$

Exponentiation is performed by table lookup. We noticed that the table used should be either large or nonlinear. If we use a linear table in both intensity value and exponent, we need a 512 by 128 entry table in order for the linear interpolation to be accurate to 1%. Leaving out all trivial entries still requires 10k. A table with better resolution when the intensity exceeds 0.95 and the exponent 100, say, dramatically reduces the size, but makes the solution even more complicated to realize in (fast) hardware.

The results of the method are surprisingly good for surfaces that are not too strongly curved. Similarly, an expression is given for the case in which E and L are not at infinity, which uses the Duff variables a through r. We refer to Bishop and Weimers article [4]. Using this method, the diffuse term can be computed accurately with extremely low overhead. As desired, it can be efficiently evaluated by a pipeline performing second order forward differencing. The remaining problem is therefore to find an expression for the specular term that obviates the exponentiation.

4.4 Curve fitting

Consider the horizontal component of a shading function. Fitting one or more quadratic curves over the existing shading function implies sampling data and approximating the original shading function as accurately as possible. This can be done using several criteria, such as minimizing the maximum difference over a range, minimizing the absolute value of the area or the area itself between both functions over the range, or the well-known least squares fit method where the square of the difference between both functions over the range is minimized.

Our problem is, however, more demanding, because we must also minimize the discontinuities and differences in derivative at the edges. In fact, we should guarantee that these do not cause visible discontinuities or Mach bands, which practically means that they should equal *0*. This demand is also applicable to the intensity of the highlight at x coordinate *xh*.

More accurately, we desire a quadratic approximation function q of the shading function s which allows

$$\begin{array}{ll} q(xl) \approx s(xl) & \frac{\partial}{\partial x} q(xl) \approx \frac{\partial}{\partial x} s(xl) \\ q(xh) \approx s(xh) & \\ q(xr) \approx s(xr) & \frac{\partial}{\partial x} q(xr) \approx \frac{\partial}{\partial x} s(xr) \end{array}$$

Given either Duff's or Bishop and Weimer's expression, we can accurately compute the specular intensity component of several pixels on a scanline. The intensity of the highlight is not easily computed, for we must either locate it using the complex Duff expression or the approximation used by Bishop and Weimer, which is too inaccurate outside the range [*xl*...*xr*].

Suppose that q is a single quadratic function in x. It is easily seen that theoretically, these demands generally do not have an exact solution. Our experiments show that practically, relaxing any of the above requirements to allow better approximation of other demands tends to severely deteriorate picture quality. As is apparent, matching derivatives will in most cases produce discontinuities in the intensity. Matching intensities will force us to use some technique minimizing the error made in one or both of the derivatives, and will generally produce visible Mach bands. We had no difficulty whatsoever to find a situation for each technique where either intensities where visibly discontinuous, Mach bands appeared or the highlight was displaced or had a visibly different intensity.

Therefore, we will use a set of several curves to approximate the scanline intensity function. A similar function must be devised that computes the vertical component of the intensity function, including the continuity constraints, but we can allow a more complex algorithm for this, so that we do not have to find a method that incrementally updates the information for the horizontal shading function, but can spend some time actually computing it. We will first describe some properties of the actual shading function that become apparent when we use a vector rotation technique.

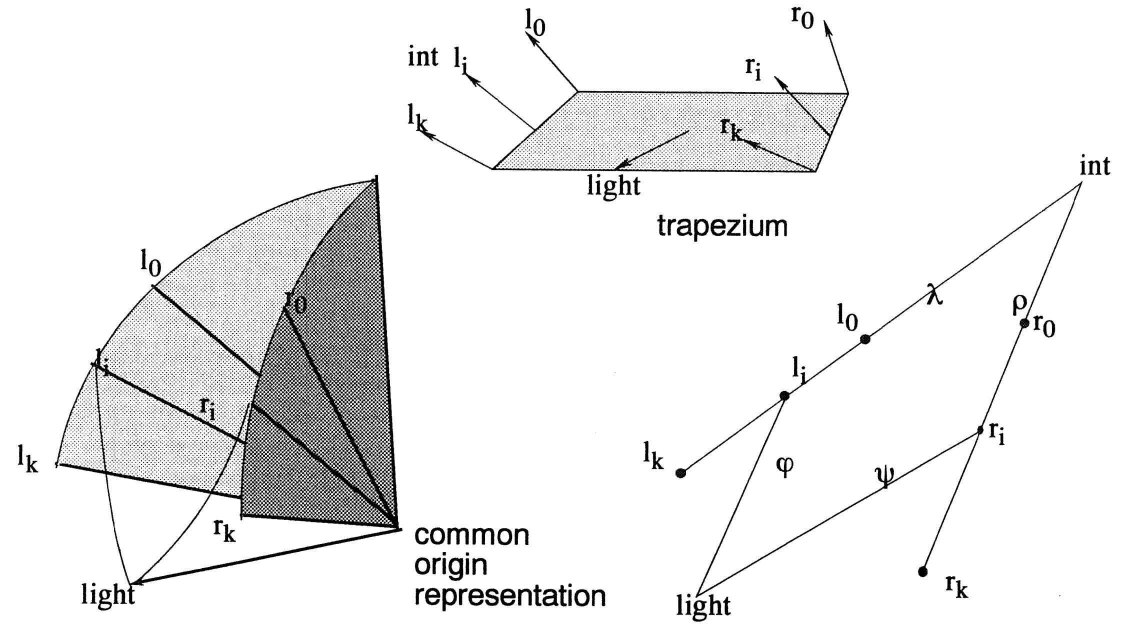
5 A vector rotation method

Without loss of generality, assume that the shape of the polygon to be shaded is a trapezium[(1)], where a normal, light and H vector are defined at each vertex. We assume the vectors are defined with respect to a common origin, although they are associated with the vertex of the trapezium.

Instead of using a linear expression to compute the normal vectors and normalizing them, we calculate a normal vector of the plane defined by the left vector pair and rotate a "left" vector over the total angle between these two vectors in a number of steps depending on the number of scanlines covered. Similar rotations are used for the right vector and the light and H vector. Given these left and right normal vector at a certain scanline, we again compute the normal vector of these two, and rotate a vector over the total angle between these two vectors in a number of steps depending on the number of pixels covered. It is easily seen that using similar rotations for the light and H vectors produces the desired vectors for each pixel. For the moment, assume the *light* and H vectors fixed.

The complexity of the method described above is roughly equal to the complexity of the original Phong shading method. We will now show how to modify it in order to derive a number of quadratic curves for each scanline using expressions that need only be updated at scanline rate.

For the moment, assume that we are dealing with the diffuse situation. Let i denote an index ranging from 0, the upper scanline of the trapezium to k, the lower scanline of the trapezium. Because two planes through the origin must intersect, we know there is an intersection *int* of the planes through l_0, l_k and r_0, r_k respectively. The angle between *int* and l_i is called λ_i, the angle between int and r_i is called ρ_i. l_i, the vector marching along the left edge of the polygon can therefore be expressed using the aforementioned plane and its angle with *int*, λ_i. Similarly, ρ_i defines the location of r_i.

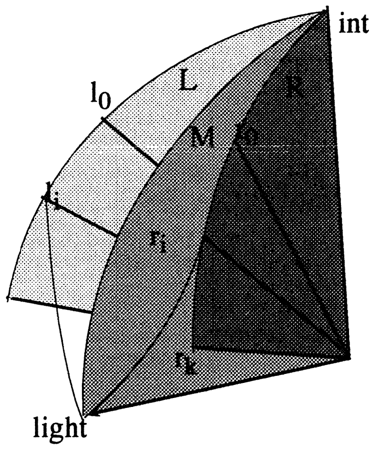


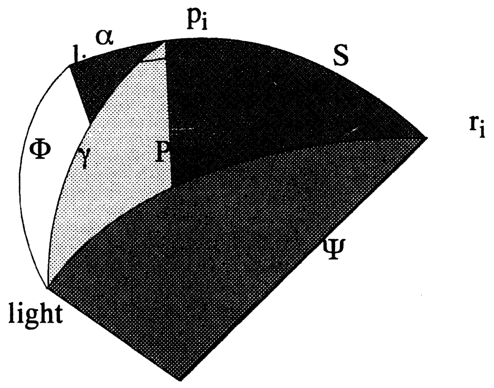
Assuming the light vector fixed, we define φ_i and ψ_i as the angles between the light vector and λ_i and ρ_i respectively.

We can now express the value of the diffuse component, $cos\delta$, as a function of i and the angle with respect to l_i. Let p_i denote the projection vector of the *light* vector on the plane through l_i and r_i.

1) A trapezium is a possibly degenerate quadrilateral of which two boundaries are parallel with the horizontal axis.

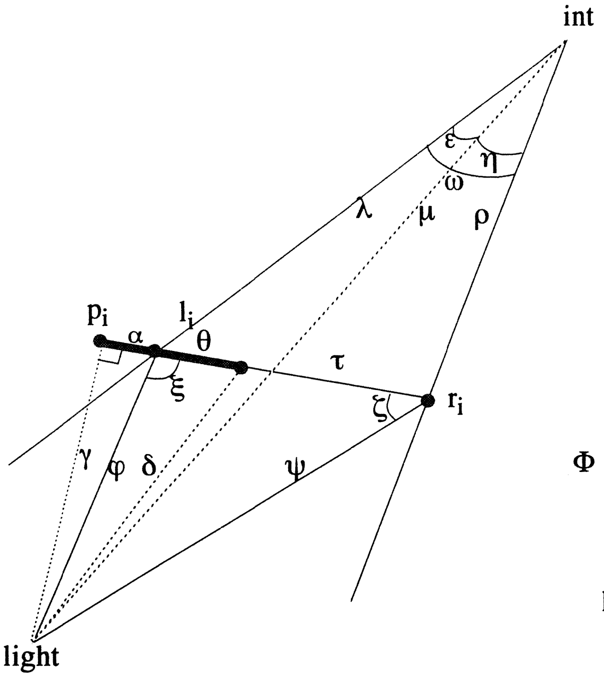
Observe that p_i is perpendicular to both $l_i X r_i$ and $light X(l_i X r_i)$, so that p_i must be a multiple of the vector$(l_i X r_i) X(light X(l_i X r_i))$.





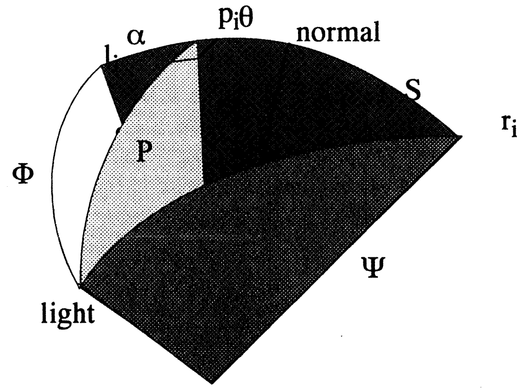
We define L,M and R as the planes through *int* and l_i, *light* and r_i respectively, S as the plane through l_i and r_i, Φ as the plane through l_i and *light*, and Ψ as the plane through r_i and *light*. P is the plane through *light* and p_i.

Let p_i be a unit vector. We can now use the fact that $S \perp P$. Using p_i and the cosine rule for spherical trigonometry, we can express $\cos\delta$ as $\cos\gamma_i \cos(\theta-\alpha_i)+\sin\gamma_i \sin(\theta-\alpha_i)\cos(\pi/2)$ where γ_i is the angle between p_i and *light* and which is constant per scanline. The offset α_i of l_i with respect to p_i can also be easily derived using the cosine rule: $\cos\alpha_i=\cos\varphi_i/\cos\gamma_i$.



$$\cos\gamma = \frac{p_i \bullet light}{\sqrt{p_i \bullet p_i}}$$

$$\cos\alpha = \frac{\cos\varphi}{\cos\gamma}$$



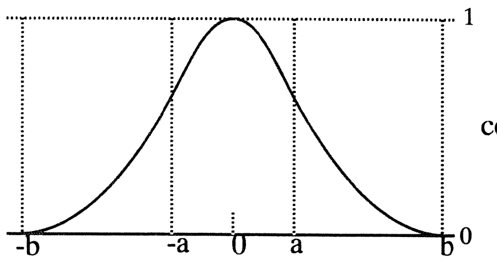
The $\cos\gamma_i$ can be expressed in terms of $\cos\varphi_i$, $\cos\tau_i$ and $\cos\psi_i$. Again, the cosine rule allows several ways to express $\cos\varphi_i$ and $\cos\psi_i$.

$$\cos\varphi = \cos\lambda\cos\mu + \sin\lambda\sin\mu\cos\varepsilon$$
$$\cos\psi = \cos\rho\cos\mu + \sin\rho\sin\mu\cos\eta$$
$$\cos\tau = \cos\lambda\cos\rho + \sin\lambda\sin\rho\cos\omega$$
$$\cos\delta = \cos\gamma\cos(\theta-\alpha)$$
$$\cos\gamma = \operatorname{sgn}(\cos\varphi)\sqrt{\frac{\cos^2\psi - 2\cos\varphi\cos\tau\cos\psi + \cos^2\varphi}{1-\cos^2\tau}} = \sin\angle(light, l_i \times r_i)$$

ε,η and ω are the angles between L and M, M and R and L and R respectively.These angles are constant during interpolation. If S is the plane through l_i and r_i, then the angles ξ_i and ζ_i are the angles between L and S, and S and R, respectively.These angles vary during interpolation. μ is the angle between *light* and *int*, and τ_i is the angle between l_i and r_i.

Updating the expressions for $cos\varphi_i$, $cos\tau_i$ and $cos\psi_i$ can be performed incrementally. This, however, poses severe restrictions on the accuracy of the incremental computations. Furthermore, there are many situations in which special cases must be handled and even more situations in which errors explode, e.g. when τ is almost *0*.

Now that we have an expression for $cos\delta$, we must find a way to represent the cos^s function by quadratic curves. This problem has been solved in Kuijk and Blake [5], who propose the following function:



$$\cos^s\theta \approx \begin{cases} \dfrac{(\theta+b)^2}{b(b-a)} & -b \le \theta \le -a \\ 1 - \dfrac{\theta^2}{ba} & -a \le \theta \le a \\ \dfrac{(\theta-b)^2}{b(b-a)} & a \le \theta \le b \\ 0 & |\theta| \ge b \end{cases}$$

where

$$a = \frac{s+5.6}{s(0.09s+5.2)}$$
$$b = \frac{s+65}{5s+31.7}$$

With respect to the cosine, this function has several merits that come for free, namely that its value outside the range *-b..b* is *0* and that the approximation for cos^1 does not have the discontinuities in slope at $-\pi/2$ and $\pi/2$, thus avoiding Mach banding. So the above expressions can be used to update α_i and τ_i for each scanline, and these fully determine the shape of the approximating function, consisting of at most three quadratic curves. The shape of this shading function differs slightly from the original cosine power. The largest difference is due to the smoothing of the cosine function itself; it appears when using $s=1$, namely *0.05*. These differences are, however, no threat to the image quality, but merely the result of a slightly alternative model with respect to the original Phong function, which is a model based on empirical observations itself. The average error over the range of $\theta\in[-\pi/2..\pi/2]$ and $s\in[1.128]$ is *0.0006*.

The rotations are in practice performed by updating cosines of angles and computing their arccosine when needed. These trigonometrical operations must be evaluated accurately in order to assure correct results. Because of the substantial overhead involved, we integrated this method with the ones using vector coordinate interpolation.

6 Hybrids

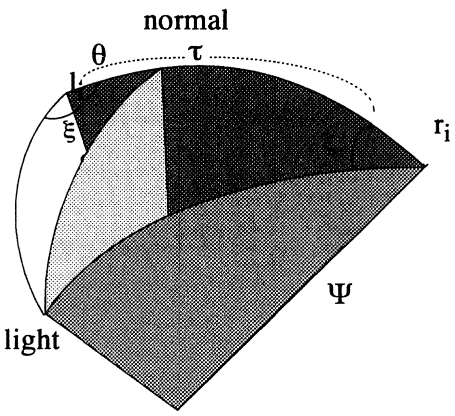
The main idea of integrating the Duff and Bishop & Weimer methods with the above method is to derive the parameters needed for the approximation function in a cheap way. These hybrids all suffer from the drawback that the scanline parameters are derived using vector coordinate interpolation while the scanline shading function uses vector rotation. This might result in highlight contortion.

We use a hybrid where the $cos\varphi_i$, $cos\tau_i$ and $cos\psi_i$. are assumed to be computed using a scalar product and a division by the vector lengths.

What we are interested in is the angle between the (constant) light vector and normal vector, δ.

Let θ denote the angle between l_i and the interpolated vector, which ranges from *0* to τ. Using the cosine rule, we find four relationships between the angles:

$$\cos\varphi = \cos\tau\cos\psi + \sin\tau\sin\psi\cos\zeta$$
$$\cos\psi = \cos\tau\cos\varphi + \sin\tau\sin\varphi\cos\xi$$
$$\cos\delta = \cos(\tau-\theta)\cos\psi + \sin(\tau-\theta)\sin\psi\cos\zeta$$
$$\cos\delta = \cos\theta\cos\varphi + \sin\theta\sin\varphi\cos\xi$$



For the definitions of these angles, we refer to section 5, "A vector rotation method", on page 6, and to the appendix.

Adding the last two expressions for cosδ and dividing by two, and substituting the expressions for $cos\xi_i$ and $cos\zeta_i$ yields the following expression for cosδ:

$$\cos\delta = \frac{\cos\psi - \cos\varphi\cos\tau}{\sin\tau}\sin\theta + \cos\varphi\cos\theta$$

$$\cos\delta = \cos\varphi\left(\frac{\cos\psi - \cos\varphi\cos\tau}{\cos\varphi\sin\tau}\sin\theta + \cos\theta\right)$$

$$\cos\delta = \cos\varphi(\tan\alpha\sin\theta + \cos\theta) \qquad \text{where} \qquad \alpha = \operatorname{atan}\frac{\cos\psi - \cos\varphi\cos\tau}{\cos\varphi\sin\tau}$$

$$\cos\delta = \frac{\cos\varphi}{\cos\alpha}(\cos\theta\cos\alpha + \sin\theta\sin\alpha)$$

$$\cos\delta = \frac{\cos\varphi}{\cos\alpha}\cos(\theta - \alpha)$$

Using the fact that

$$\frac{p}{\cos \operatorname{atan} \frac{q}{p}} = \operatorname{sgn} p \sqrt{p^2 + q^2}$$

this can be rewritten as

$$\cos\delta = \operatorname{sgn}\cos\varphi \sqrt{\cos^2\varphi + \frac{\cos^2\psi - 2\cos\varphi\cos\tau\cos\psi + \cos^2\varphi\cos^2\tau}{\sin^2\tau}} \cos(\theta - \alpha)$$

or, as was mentioned before, as

$$\cos\delta = \Gamma\cos(\theta - \alpha) ; \Gamma = \operatorname{sgn}\cos\varphi \sqrt{\frac{\cos^2\psi - 2\cos\varphi\cos\tau\cos\psi + \cos^2\varphi}{\sin^2\tau}}$$

Substituting

$$\cos\varphi = \frac{l_i \bullet v}{|l_i|} \qquad \cos\tau = \frac{l_i \bullet r_i}{|l_i||r_i|} \qquad \cos\psi = \frac{r_i \bullet v}{|r_i|}$$

We derive

$$\cos\delta = \Gamma\cos(\theta - \alpha - \omega)$$

$$\Gamma = \sqrt{\frac{(r_i \bullet r_i)(l_i \bullet v)^2 - 2(l_i \bullet v)(l_i \bullet r_i)(r_i \bullet v) + (r_i \bullet v)^2 (l_i \bullet l_i)}{(l_i \bullet l_i)(r_i \bullet r_i) - (l_i \bullet r_i)^2}}$$

$$\alpha = \operatorname{atan}\frac{(l_i \bullet l_i)\frac{r_i \bullet v}{l_i \bullet v} - (r_i \bullet r_i)}{\sqrt{(l_i \bullet l_i)(r_i \bullet r_i) - (l_i \bullet r_i)^2}}$$

$$\omega = \begin{matrix} (l_i \bullet v) < 0 \rightarrow \pi \\ (l_i \bullet v) > 0 \rightarrow 0 \end{matrix}$$

This is easily rewritten to allow second order forward differencing by observing that the various scalar products *ll, lr, rr, lv* and *rv* are quadratic or linear in *y,* where *y* is the scanline offset with respect to the middle of the object.

$$\Gamma = \sqrt{\frac{(rr)\,(lv)^2 - 2\,(lv)\,(lr)\,(rv) + (rv)^2\,(ll)}{(ll)\,(rr) - (lr)^2}}$$

$$\cos\delta = \Gamma\cos(\theta - \alpha - \omega) \qquad \alpha = \operatorname{atan}\frac{(ll)\,(rv) - (rr)\,(lv)}{(lv)\sqrt{(ll)\,(rr) - (lr)^2}}$$

$$\omega = \begin{array}{l}(lv) < 0 \rightarrow \pi \\ (lv) > 0 \rightarrow 0\end{array}$$

For their definitions, we refer to section 10 on page 13. These are derived using the fact that for any scanline, l_i and r_i are defined using $l_i{=}Ax_l{+}By{+}C$ and $r_i{=}Ax_r{+}By{+}C$.

$$x_l = x_{l_0} + dx_l y \qquad dx_l = \frac{x_{l_k} - x_{l_0}}{y_k - y_0} \qquad dx_r = \frac{x_{r_k} - x_{r_0}}{y_k - y_0}$$
$$x_r = x_{r_0} + dx_r y$$

It is possible to use fourth or third order differencing for the entire expressions in the denominators and numerators of the expressions for Γ and α, but this is more complex, less accurate and exactly as costly in terms of elementary operations. If a multiplication is by far more expensive than an addition, this is an alternative.

Now, this expression must be rewritten in terms of the *x*-coordinate instead of θ. This is done using

$$\theta = \frac{\Delta\theta}{\Delta x}x = \frac{\tau}{width}x = \frac{\operatorname{acos}\left(\frac{(l_i \bullet r_i)}{\sqrt{(l_i \bullet l_i)\,(r_i \bullet r_i)}}\right)}{width}x = \frac{\operatorname{acos}\left(\frac{lr}{\sqrt{(ll)\,(rr)}}\right)}{width}x$$

The implementation of this method is straightforward. Below, a stripped version in pseudo code will be presented where we assume the availability of a processor capable of performing an arctangent, an arccosine, two reciprocal square roots, two divisions and several additions and multiplications in scanline time. The more complex functions could be computed by table lookup, if desired.

The computation of the specular term is performed by raising Γ to the power *s* and supplying the parameters corresponding to *-b,-a,a* and *b* to the shading function. When *s* is an integer, this can easily be performed using either a number of multiplications logarithmic in the maximum value of *s* or table lookup.

Observe that we do not need this expensive method for the diffuse term, but can simply use Bishop and Weimers expression. Therefore, the total overhead for a scanline is practically equal to the complexity of the aforementioned floating point operations or table lookups.

7 Complexity

Assuming a RISC-like architecture, we can perform a reciprocal with the desired accuracy using 7 floating point operations, while a reciprocal square root amounts to 17 operations. An arctangent costs 19 operations and an arccosine 33.

Assuming the situation where the *light* and *H* vector are given and constant, and the normal vectors are specified at the vertices of a trapezium, precomputing the *A*,*B* and *C* vectors, the Duff variables, the T_0 through T_5 for the diffuse term as defined by Bishop and Weimer and the above parameters, implies a frame complexity of 287 floating point operations. The scanline update costs are 182 operations while the pixel update cost is 5 integer additions. For original Phong shading, these figures are 24, 21 and 46 floating point operations, respectively.

Assuming that the integer additions are as costly as the floating point operations, this implies that our method has a break-even point with respect to the original Phong shading method at 23 pixels.

As most polygons will be larger than 5X5 pixels, this clearly justifies the increased scanline and frame complexity, even if no additional hardware is used for forward differencing.

8 Conclusions

Our research shows that Phong shading can be approximated by using quadratic curves at scanline level. This enables the use of extremely simple and fast evaluation hardware pipelines at pixel generation level. The difference with the original Phong shading model concerns mainly the interpolation technique used at scanline level.

We presented two methods for efficient computation of the specular term. The first is based on vector rotation horizontally as well as vertically, while the second one applies vector coordinate interpolation in the vertical direction. One is free to combine these specular terms with a diffuse term that is derived analogously or computed using the technique of Bishop and Weimer.

Although in theory, the hybrid methods might suffer from inconsistencies in highlight location due to the different interpolation methods, we did not find any evidence that this would affect image quality in practice.

Our method drastically reduces the pixel rate complexity at the expense of an increase of scanline complexity. However, the break-even point of 23 pixels with respect to the original Phong shading method immediately justifies this.

9 Extensions

Simple modifications allow lightsources located in the vicinity of the object. A method that changes the normal vectors in order to approximate this effect has been described in the article by Kuijk and Blake [5]. The idea is to choose a representative from the *light* or *H* vectors defined at the vertices. The same rotation that would map the *light* or *H* vector at a vertex to the representative is applied to the normal, thereby assuring that the intensities at the edges are continuous.

Anti-aliasing using exact pixel area integration is trivial because the weighting function that must be applied over a scanline consists of at most three linear parts in x when shading polygons. Applying these linear weighting functions to the quadratic shading functions clearly produces cubic curves, which can be evaluated by third order forward differencing.

An architecture is being developed where instances of the aforementioned algorithm are executed in parallel for each trapezium visible on a scanline. The resulting polynomial shading functions are then sent to an evaluation pipeline performing second or third order forward differencing and accumulation of the various resulting shading values due to individual lightsources and their both components. For a description of this architecture, we refer to [8].

10 List of symbols

α	$\angle (p_i, l_i)$	l_0	initial left vector	ll_0	$aa\ xl_0^2+cc+2\ ac\ xl_0$		
β	$\angle (p_i, r_i)$	l_i	left interpolation vector	ll_1	$2(aa\ dxl\ xl_0+bc+ab\ xl_0+ac\ dxl)$		
δ	$\angle$ (*normal,light*)	l_k	eventual left vector	ll_2	$aa\ dxl^2+2\ ab\ dxl+bb$		
ε	$\angle (L,S)$	r_0	initial right vector	lr_0	$aa\ xl_0\ xr_0+cc+ac(xl_0+xr_0)$		
γ	$\angle (p_i$, *light*)	r_i	right interpolation vector	lr_1	$aa(dxl\ xl_0+dxl\ xr_0)+2bc$		
η	$\angle (R,S)$	r_k	eventual right vector		$+ab(xl_0+xr_0)+ac(dxl+dxr)$		
φ	$\angle (l_i$, *light*)	L	plane through l_0 and l_k	lr_2	$aa\ dxl\ dxr+ab(dxl+dxr)+bb$		
λ	$\angle$ (*int*, l_i)	R	plane through r_0 and r_k	rr_0	$aa\ xr_0^2+cc+2\ ac\ xr_0$		
μ	$\angle$ (*int, light*)	*int*	intersection of L and R	rr_1	$2(aa\ dxr\ xr_0+bc+ab\ xr_0+ac\ dxr)$		
θ	$\angle (l_i$, *normal*)	*light*	light vector	rr_2	$aa\ dxr^2+2\ ab\ dxr+bb$		
ρ	$\angle$ (*int*, r_i)	M	plane through *light* and *int*	lv_0	$av\ xl0+cv$		
σ	$\angle$ (*normal*, H)	S	plane through l_i and r_i	lv_1	$av\ dxl+bv$	aa	$A\bullet A$
τ	$\angle (l_i, r_i)$	*normal*	interpolated normal	rv_0	$av\ xr0+cv$	ab	$A\bullet B$
ω	$\angle (S, R)$	p_i	projection of *light* on S	rv_1	$av\ dxr+bv$	ac	$A\bullet C$
ξ	$\angle (\Phi, S)$	Γ	$\cos\gamma$	V	*light* or H	av	$A\bullet V$
ψ	$\angle$ (*light*, r_i)	P	plane through p_i and *light*	ll	$ll_0+y(ll_1+y\ ll_2)$	bb	$B\bullet B$
ζ	$\angle (\Psi, S)$	a	$(s+5.6)/(s(0.09s+5.2))$	lr	$lr_0+y(lr_1+y\ lr_2)$	bc	$B\bullet C$
s	shininess	b	$(s+65)/(5s+31.7)$	rr	$rr_0+y(rr_1+y\ rr_2)$	bv	$B\bullet V$
i	scanline index	Φ	plane through l_i and *light*	lv	$lv_0+y\ lv_1$	cc	$C\bullet C$
y	scanline$-y_{middle}$	Ψ	plane through r_i and *light*	rv	$rv_0+y\ rv_1$	cv	$C\bullet V$
k	maximum index	H	H vector,	H replaces *light* in the specular case.			

11 References

[1] Gouraud, H., June 1971. *"Continuous Shading of Curved Surfaces,"* IEEE Transactions on Computers, vol. 20, no.6, pp. 623-628.

[2] Bui-Tuong, Phong, June 1975. *"Illumination for Computer-Generated Pictures,"* Communications of the ACM, vol. 18, no.6, pp. 311-317.

[3] Duff, T., 1979. *"Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays,"* ACM Computer Graphics, vol. 13, no.6, pp. 270-275.

[4] Bishop, G., and D.M. Weimer, August 1986. *"Fast Phong Shading,"* ACM Computer Graphics, vol. 20, no.4, pp. 103-106.(1)

[5] Kuijk, A.A.M., and E.H. Blake, December 1989. *"Faster Phong Shading via Angular Interpolation,"* Computer Graphics Forum, vol. 8, no.4, pp. 315-324.(2)

[6] Hall, R., 1988. *"Illumination and Color in Computer Generated Imagery,"* Springer Verlag New York.

[7] Blinn, J.F., 1977. *"Models of Light Reflection for Computer Synthesized Pictures,"* ACM Computer Graphics (SIGGRAPH 77), vol. 11, no.2, pp. 192-198.

[8] Jayasinghe, J.A.K.S., A.A.M. Kuijk and L. Spaanenburg, 1991. *"A Display Controller for an Object-Level Frame Storage System,"* Advances in Computer Graphics Hardware III (A.A.M. Kuijk, ed.), Springer Verlag Berlin, pp. 140-170.

---

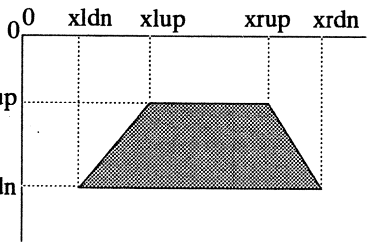
1) There are several errors in this paper. Some have been rectified, but for completeness, we will repeat these here:
Page 104, first column, first and second equation: Remove the three L vectors from the denominator. Page 104, second column, second equation: the first term in the denominator of T_5 is $3cg^2$ and not $3ig^2$. Page 105, second column, second equation: The sign of the term *fjlqr* should be + instead of - and the term $-2f^2nr$ should be replaced by $-2fl^2nr$.

2) This paper also contains an error: the definitions of a and b on page 321 have been interchanged. The definitions as shown in this paper are correct: $a=(n+5.6)/(n*(0.09n+5.2))$ and $b=(n+65)/(5n+31.7)$.

12 Pseudo-code

The following code has not been optimized. It is assumed that the *ABC* triplet and *light* vector serve as input, and that a trapezium is being shaded. No color- or anti-aliasing code or variable declarations are shown.Triangles, being degenerate trapeziums, can be processed without modifications.

```
void shader(A,B,C, V, xlup,xrup,xldn,xrdn,yup,ydn, s)
  fetch a(s) and b(s);
  height=ydn-yup; xl=xlup; xr=xrup;
  heightinv=1/height; dxl=(xldn-xlup)*heightinv; dxr=(xrdn-xrup)*heightinv;
  if frac(yup)!=0
    xl+=(1-frac(yup))*dxl; xr+=(1-frac(yup))*dxr;
  aa=A.A; ab=A.B; ac=A.C; av=A.V; bb=B.B; bc=B.C; bv=B.V; cc=C.C; cv=C.V;
  $ll_0$=...; $ll_1$=...; $ll_2$=...;     $ll_i$=$ll_0$; $dll_i$=$ll_1$+$ll_2$; $ddll_i$=2$ll_2$;
  $lv_0$=...; $lv_1$=...;                 $lv_i$=$lv_0$; $dlv_i$=$lv_1$;
  $lr_0$=...; $lr_1$=...; $lr_2$=...;     $lr_i$=$lr_0$; $dlr_i$=$lr_1$+$lr_2$; $ddlr_i$=2$lr_2$;
  $rv_0$=...; $rv_1$=...;                 $rv_i$=$rv_0$; $drv_i$=$drv_1$;
  $rr_0$=...; $rr_1$=...; $rr_2$=...;     $rr_i$=$rr_0$; $drr_i$=$rr_1$+$rr_2$; $ddrr_i$=2$rr_2$;
  for(scanline=ceil(yup); scanline<ceil(ydn); scanline++)
    $\cos\tau_i$=$lr_i$/sqrt($ll_i$*$rr_i$);
    if $\cos\tau_i$>1-verysmall
      $\cos\varphi_i$=$lv_i$/sqrt($ll_i$); $\cos\psi_i$=$rv_i$/sqrt($rr_i$);
      linearshade(xl,xr, sgn($\cos\varphi_i$)*pow(|$\cos\varphi_i$|,s),sgn($\cos\psi_i$)*pow(|$\cos\psi_i$|,s));
    else
      $\Gamma_i$=pow( ($rr_i$*$lv_i$*$lv_i$-2*$lv_i$*$lr_i$*$rv_i$+$ll_i$*$rv_i$*$rv_i$) / ($ll_i$*$rr_i$-$lr_i$*$lr_i$) , s/2);
      $\alpha_i$=sgn($lv_i$)*atan( ($ll_i$*$rv_i$-$lv_i$*$lr_i$) / sqrt( $lv_i$*$lv_i$*($ll_i$*$rr_i$-$lr_i$*$lr_i$) ) );
      $\tau_i$=acos($\cos\tau_i$);
      if $lv_i$<0 $\alpha_i$+=$\pi$;
      if max($lv_i$,$rv_i$)>0 and $\Gamma_i$>verysmall
        generatequadratics(xl,xr, $\Gamma_i$,$\alpha_i$,$\tau_i$, a,b,s);
  xl+=dxl; xr+=dxr;
  $ll_i$+=$dll_i$; $dll_i$+=$ddll_i$; $lv_i$+=$dlv_i$; $lr_i$+=$dlr_i$; $dlr_i$+=$ddlr_i$; $rv_i$+=$drv_i$; $rr_i$+=$drr_i$; $drr_i$+=$ddrr_i$;
```



The procedure that generates the actual curves does nothing more than efficiently combining the data and sending the quadratic curves to the evaluating soft- or hardware:

```
void generatequadratics(xl,xr, $\Gamma_i$,$\alpha_i$,$\tau_i$, a,b,s);
  width=xr-xl;
  if floor(xl)=floor(xr)
    exit;
  if $t_i$<verysmall
    constantshade(xl,xr,$\Gamma_i$);
    exit;
  dtdx=$\tau_i$/width; dxdt=1/dtdx; dtdx*=dtdx*$\Gamma_i$;
  middle=xl+$\alpha_i$*dxdt; adxdt=a*dxdt; bdxdt=b*dxdt;
  xbl=ceil(middle-bdxdt); xal=ceil(middle-adxdt);
  xar=ceil(middle+adxdt); xbr=ceil(middle+bdxdt);
  qshade(max(xl,xbl), min(xr,xal), dtdx/(b*(b-a)), 0, max(xl,xbl)-middle-bdxdt);
  qshade(max(xl,xal), min(xr,xar), dtdx/(-a*b), $\Gamma_i$, max(xl,xbl)-middle);
  qshade(max(xl,xar), min(xr,xbr), dtdx/(b*(b-a)), 0, max(xl,xbl)-middle+bdxdt);

void qshade(x, xn, second, def, angle)
  if x!=xn
    first=s/2; temp=second*angle; intensity=def+temp*angle/2; first+=temp;
    quadraticshade(x,xn-x, intensity, first, second);
```