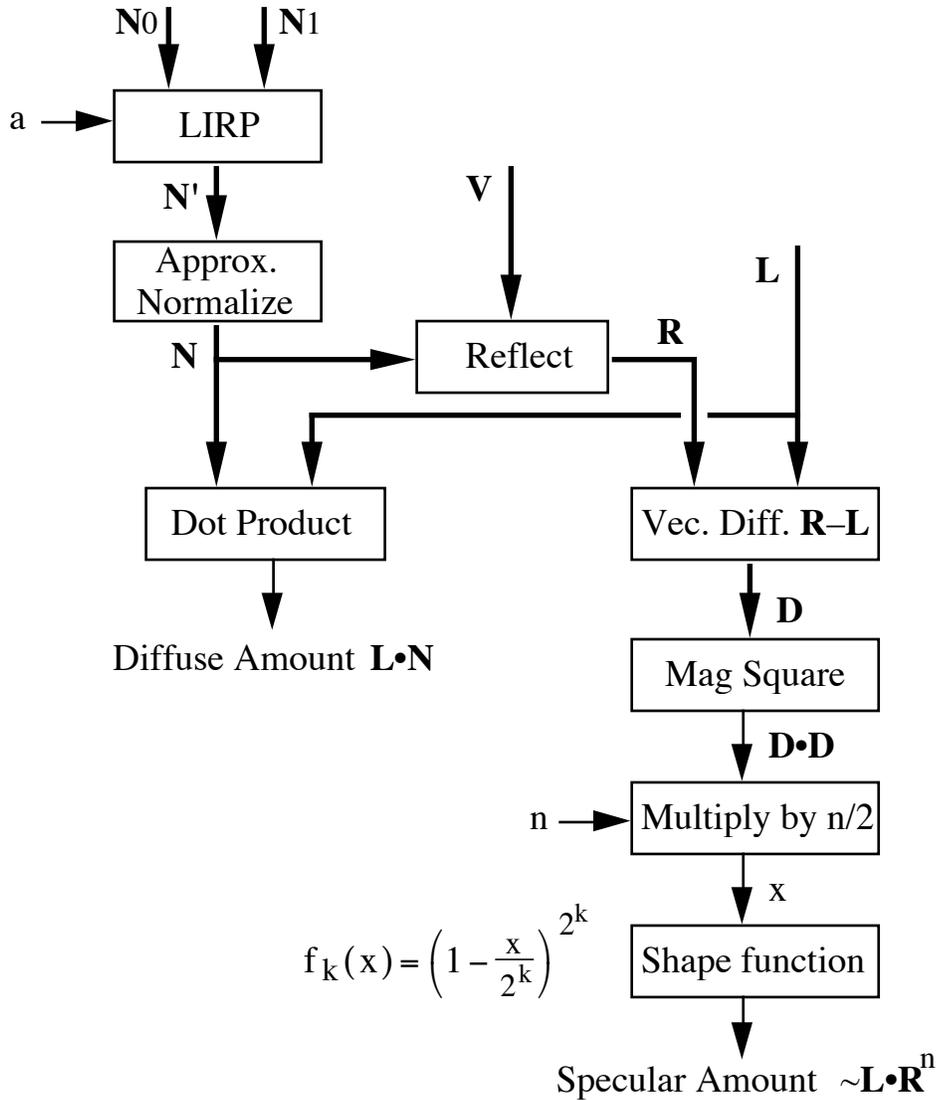


# Phong Shading Reformulation for Hardware Renderer Simplification

Apple Technical Report #43

Richard F. Lyon  
Advanced Technology Group  
Apple Computer, Inc.

August 2, 1993



# Phong Shading Reformulation for Hardware Renderer Simplification

Apple Technical Report #43

**Richard F. Lyon**

dicklyon@acm.org

Advanced Technology Group

Apple Computer, Inc.

August 2, 1993

©1993 Apple Computer, Inc.

## Abstract

The specular calculation of the Phong shading method is reformulated to use the squared length of a difference vector in place of a dot product to characterize the closeness of the viewpoint to the specularity. The new formulation is easier to compute, using only a modest numbers of multiplications and additions, and is much better behaved numerically. Due to the better numerical properties, the new method also allows much simpler approximations to vector normalization. The reformulation does not reproduce the standard  $\cos^n$  Phong specularity shape exactly, but allows a range of options in approximating that shape. This new method is especially well suited for fixed-point hardware implementation.

## Dedication

This report is dedicated to Bui Tuong Phong, whose work before his untimely death advanced the field enormously. Dr. Bui Tuong is known to me and many others only through his work and through his colleagues from the University of Utah graphics group, many of whom I've had the pleasure of knowing and learning from over the last two decades.

## Introduction

The Phong shading method [Bui Tuong73, Bui Tuong75] of generating images of 3D surfaces with realistic lighting hilites is a popular and standard approach in both software and hardware renderers. However, it is a computationally expensive process, and is often omitted in favor of a simpler method, such as Gouraud shading, when speed matters—especially in real-time hardware.

The computational cost comes from the large amount of per-pixel lighting calculation that is needed, including one or more square roots to normalize interpolated vectors, and an exponentiation (typically a log and an exponential) for the specular calculation.

We show how to formulate a close approximation to the Phong specular calculation, making it easy to compute using only a modest number of multiplications and additions, and also making it much less sensitive to errors in the length normalization of direction vectors and to quantization error. In addition, we present a simple approximate vector normalization technique, using only multiplications and additions, that is compatible with the reformulated shading calculation.

## Background

Bui Tuong Phong's shading technique introduced two important but expensive innovations: first, rather than interpolate colors across surface patches (as in the predecessor Gouraud shading), it interpolates surface normals and evaluates a lighting model at each pixel; second, a specular reflection component is added to the lighting model to produce hilites. The resulting image color at a pixel typically consists of three components; ambient, diffuse (Lambertian), and specular reflection, each of which is a product of a material color, a lighting color, and a geometric factor:

$$\mathbf{I}_{\text{total}} = \mathbf{C}_a \mathbf{I}_a + \sum_i \left( \mathbf{C}_d \mathbf{I}_{d_{s_i}} (\mathbf{N} \cdot \mathbf{L}_i) + \mathbf{C}_s \mathbf{I}_{d_{s_i}} (\mathbf{R} \cdot \mathbf{L}_i)^n \right)$$

in which the sum is over the set of light sources. Vectors are represented by bold capital letters.  $\mathbf{I}_{d_{s_i}}$  is the color of the  $i^{\text{th}}$  light source (for both diffuse and specular models), and  $\mathbf{I}_a$  is the color of ambient (non-directional) lighting.  $\mathbf{C}_a$ ,  $\mathbf{C}_d$ , and  $\mathbf{C}_s$  are color vectors describing the reflectivity of the material in ambient, diffuse, and specular lighting.  $\mathbf{N}$  is the surface normal direction vector,  $\mathbf{L}_i$  is a direction vector

pointing from the surface to the  $i^{\text{th}}$  light, and  $\mathbf{R}$  is the direction vector of the viewpoint  $\mathbf{V}$  reflected off the surface (viewpoint vector  $\mathbf{V}$  is also commonly referred to as eye vector  $\mathbf{E}$ ). The material property that controls the concentration of the specular reflection is the “shininess”  $n$ .

In the specularity calculation, sometimes  $\mathbf{R}_{L_i} \cdot \mathbf{V}$  is used instead of  $\mathbf{R} \cdot \mathbf{L}_i$ , where  $\mathbf{R}_{L_i}$  is the reflection of the  $i^{\text{th}}$  light vector off the surface; but that approach requires more reflection operations when there are multiple lights. We take  $\mathbf{R}$  as the reflection of  $\mathbf{V}$  and omit the subscript  $i$  from here on for convenience—much of the calculation will still need to be repeated in the case of multiple lights.

Figure 1 shows a generalized shader in block diagram form, for a single light and ignoring the ambient term (which is often not used, or is incorporated in simplified form as an addition to the diffuse amount). The vectors used in our discussion are illustrated in Figure 2. The “illumination model” of both of these figures, and much of what follows, can be applied also in ray tracing, as well to Gouraud shading and flat shading, though these simpler shading approaches are not generally very good at handling specularities.

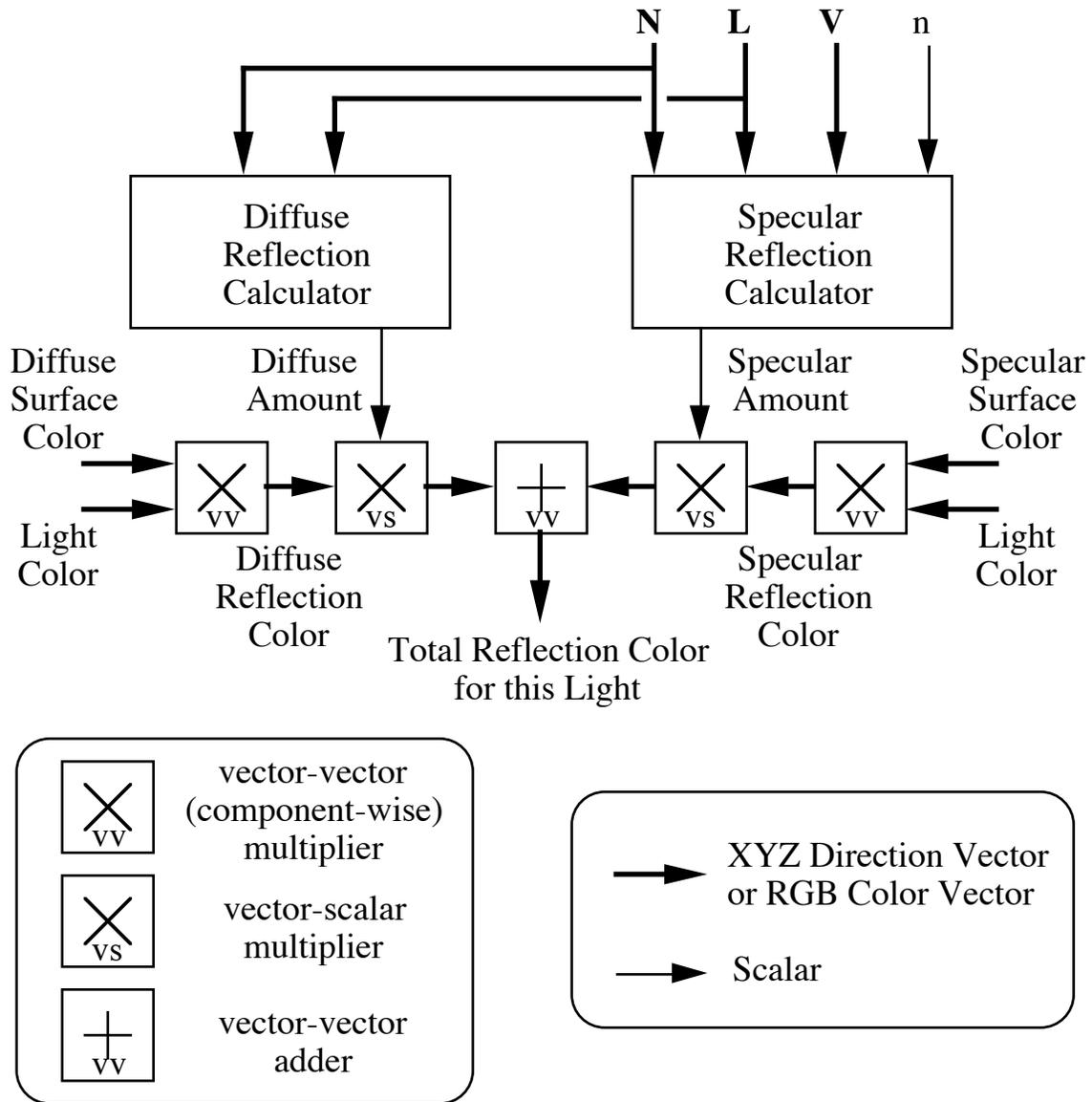


Figure 1: Block diagram of a generalized shader. The block “specular reflection calculator” is of particular interest in this report. The calculations are repeated and summed for each light source. In Phong shading, the input  $N$  and possibly also  $L$  and  $V$  are interpolated across spans of polygons to be shaded. In Phong shading and in ray tracing, the calculations are repeated for each pixel. In Gouraud shading, the calculations are done at each vertex, and in flat shading at each polygon. For some physical illumination models, the light wavelength interacts with the geometric calculations, or more material properties are required, so the shader can not be decomposed in quite this way.

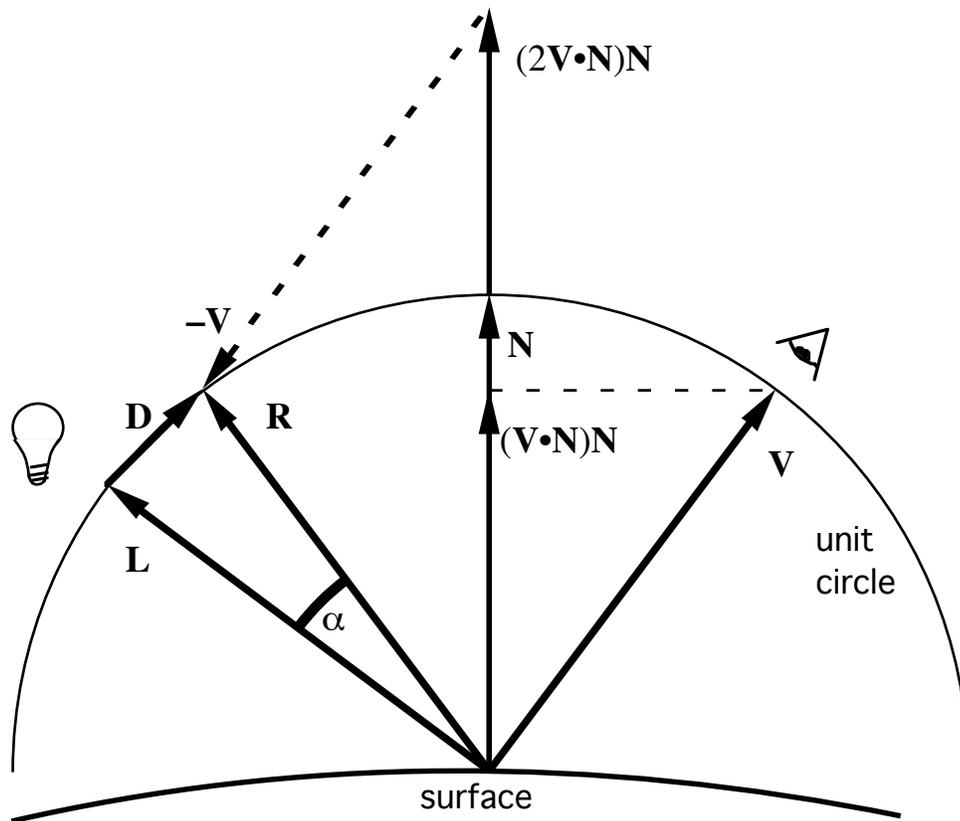


Figure 2: Illustration in two dimensions of a typical three-dimensional configuration of a surface being rendered, with surface normal  $N$ , light vector  $L$ , viewpoint vector  $V$ , reflected viewpoint vector  $R$ , angle  $\alpha$ , and difference vector  $D$ . The construction of the reflected vector  $R = 2(V \cdot N)N - V$  is illustrated in terms of the projection  $(V \cdot N)N$  of the viewpoint vector onto the surface normal vector.

When the lights and viewpoint are at an infinite distance from the object, their direction vectors do not vary across a surface patch or span, so only the surface normal needs to be interpolated and normalized. More generally, the lights and viewpoint may be “local,” or in the vicinity of the object being rendered, in which case their directions will need to be interpolated and normalized as well. In some systems, the viewpoint may be local but the lights at infinity.

The diffuse term, modeled by Lambert’s law  $N \cdot L$ , does not depend on the viewpoint, but requires surface normals and light vectors of unit length. A vector linearly interpolated between unit-length direction vectors has a length less than unity, and is corrected by dividing by the square root of the dot-product with itself. The divide and the square root are usually relatively expensive. The accuracy requirements on the normalization are not particularly severe—the final intensity is

simply proportional to each vector length, so up to a 1% maximum normalization error might be acceptable for the diffuse term.

The same normalization step is used for the vectors in the specular term, where normalization errors in  $\mathbf{N}$  and  $\mathbf{V}$  both affect  $\mathbf{R}$ . But in this term, the dot product  $\mathbf{R} \cdot \mathbf{L}$  is raised to the  $n^{\text{th}}$  power, where  $n$ , the material's "shininess coefficient" is sometimes as high as several thousand, resulting in tremendous error magnification. In systems that use this formulation directly, the error magnification puts severe accuracy requirements on the normalization operation—typically 16 bits of precision—which means that it is difficult to use a low-cost approximation. The exponentiation is similarly constrained to be an accurate operation on a 16-bit input.

## Phong Specularity Reformulation

The specular coefficient  $(\mathbf{R} \cdot \mathbf{L})^n$  may be thought of as the function  $\cos^n \alpha$  of the angle  $\alpha$  between the light direction  $\mathbf{L}$  and the reflected viewpoint  $\mathbf{R}$ . The key to our reformulation is to compute a similar function of  $\alpha$  from the difference  $\mathbf{R} - \mathbf{L}$  instead of from the dot product, and thereby to avoid the exponentiation and to a large extent the error magnification. This reformulation was developed as a good approximation for a fixed-point hardware renderer, but it also provides advantages in software, potentially even in floating-point implementations.

The key to the reformulation is the observation that the angle  $\alpha$  is approximately equal to the length of the difference vector  $\mathbf{D}$ :

$$\alpha \approx |\mathbf{D}| = |\mathbf{R} - \mathbf{L}|$$

Actually, the length of  $\mathbf{D}$  is  $2\sin(\alpha/2)$ , which is monotonic up to an angle of  $\alpha = \pi \approx 3.14$  ( $180^\circ$ ), where  $|\mathbf{D}|$  is 2. At  $\alpha = \pi/2 \approx 1.57$  ( $90^\circ$ ),  $|\mathbf{D}|$  is about 1.41, which is not too terrible; the approximation is excellent for smaller angles.

Using  $|\mathbf{D}|$  for  $\alpha$ , the function  $\cos^n \alpha$  can be approximated in various ways. Fortunately,  $\cos^n \alpha$  is an even function, so it can be computed from the squared length of  $\mathbf{D}$ , which is  $\mathbf{D} \cdot \mathbf{D}$ , avoiding the square root inherent in  $|\mathbf{D}|$ . Using  $\alpha^2 = \mathbf{D} \cdot \mathbf{D}$ , we can write the Taylor series expansion of  $\cos \alpha$  and find the leading term of its  $n^{\text{th}}$  power:

$$\cos^n \alpha = \left(1 - \frac{1}{2}\alpha^2 + \frac{1}{4!}\alpha^4 + \dots\right)^n \approx 1 - \frac{n}{2}\alpha^2$$

Therefore, a plausible first approximation for  $(\mathbf{R} \cdot \mathbf{L})^n$  is  $1 - (n/2)\mathbf{D} \cdot \mathbf{D}$ . For better approximations, it will be convenient to first define the intermediate value  $x$ :

$$x = (n/2)\mathbf{D} \cdot \mathbf{D}$$

We consider “shape functions”  $f(x)$  that define the shape of the specular reflection in terms of the squared length of the difference vector times half the surface shininess coefficient. Call this first approximation  $f_0(x)$ :

$$f_0(x) = 1 - x$$

When the difference vector  $\mathbf{D}$  is zero, the reflected view direction looks directly into the light. In this case,  $f(0)$  correctly matches  $\cos^n \alpha$  at 1.0 (assuming normalized inputs—we’ll discuss the error sensitivity later). As the light and viewpoint move away from their reflection relationship,  $x$  increases in proportion to the surface shininess and the squared distance. Soon,  $f_0(x)$  becomes negative, which is nonsensical. Therefore, we define  $f_0(x)$  to be zero for  $x > 1$ .

Notice that the specularity has an angular size that can be characterized by a nominal radius  $1/\sqrt{n}$ , where  $x$  is  $1/2$ . For large  $n$ ,  $\cos^n \alpha$  falls to about  $\exp(-1/2) = 0.6065$ , like a Gaussian at one standard deviation out; the approximation  $f_0(x)$  falls to  $1/2$  at this nominal radius, which is a significant but not gross error.

For a better approximation, we could take more terms of the Taylor series for  $\cos^n \alpha$ , but we don’t expect that that approach would give a very good result, knowing how ill-behaved Taylor series tend to be. Instead, we want to assure that  $f(x)$  will smoothly approach zero for large values of  $x$ , in the nature of the bell-shaped  $\cos^n \alpha$  function. Where  $1-x$  crosses zero linearly, we can get a smooth quadratic approach to zero by squaring  $1-x$ , but this also narrows the main peak of the shape function. We compensate by dividing  $x$  by two, to arrive at the following second approximation which still matches the Taylor series quadratic term:

$$f_1(x) = (1 - x/2)^2$$

Since squaring is cheap, and squaring the square is not much worse, we are able to make a general family of shape functions by repeating the above modification as follows:

$$f_2(x) = (1 - x/4)^4$$

$$f_k(x) = (1 - x/2^k)^{2^k}$$

The general form looks imposing, since it has the exponential  $2^k$  in the exponent; but it is computed simply as  $k$  successive squarings. And the factor  $2^k$  that multiplies  $x$  is just a shift (in a fixed-point system). In each case, the function  $f_k(x)$  should be taken as zero for  $x > 2^k$ . Functions with  $k$  of 0 or 1 may be useful, and  $k=2$  provides an approximation that is visually indistinguishable from true Phong shading.

These shape functions are plotted as a function of  $\alpha$  for  $n=100$  in Figure 3, along with  $\cos^n \alpha$  (for this large  $n$  value, we don't need to correct for the error between  $\alpha$  and the length of  $\mathbf{D}$ ). Some people think that the standard Phong specularity shape has too broad a tail; they might prefer the simpler approximations with  $k$  of 0 or 1.

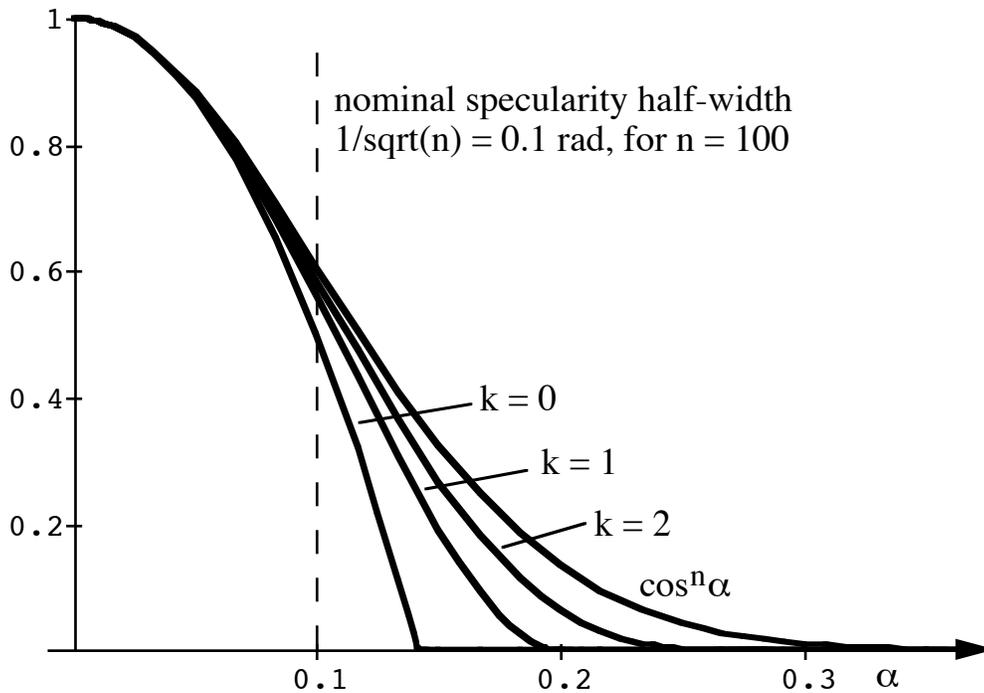


Figure 3: Specularity shapes as a function of angle  $\alpha$ , compared to the standard Phong shape  $\cos^n \alpha$ , illustrated for  $n=100$ .

The approximation to  $\cos^n \alpha$  breaks down for very small  $n$ , for two reasons. First, the length of  $\mathbf{D}$  is significantly less than  $\alpha$  for large angles; second, the function  $\cos^n \alpha$  depends on  $n$  and  $\alpha$  separately, not exactly on the product  $n\alpha^2$ . In particular,  $\cos^n \alpha$  goes to zero at  $\alpha = \pi/2$ , independent of  $n$ , rather than at a particular value of  $n\alpha^2$ . For  $n=1$ ,  $f_1(x)$  doesn't go to zero until  $\alpha = \pi$ , and  $f_2(x)$  never goes to zero. For  $n=4$ ,  $f_1(x)$  goes to zero at  $\alpha = \pi/3$ , and  $f_2(x)$  goes to zero at  $\alpha = \pi/2$ , which are much more reasonable. These very low  $n$  values probably don't matter in many cases, but if

they do, at least the reformulated specular reflection still falls off from the peak at the right quadratic rate.

## Length Error Sensitivity

Unfortunately, the functions  $(\mathbf{R} \cdot \mathbf{L})^n$  and  $f_k(x)$  depend not only on the angle  $\alpha$ , but also on the lengths of the vectors  $\mathbf{R}$  and  $\mathbf{L}$ . For example, suppose  $\mathbf{L}$  is unit length, but  $\mathbf{R}$  is of length  $1-\epsilon$ . Then when  $\mathbf{R}$  aligns with  $\mathbf{L}$ , such that  $\alpha$  is zero,  $(\mathbf{R} \cdot \mathbf{L})^n$  is  $(1-\epsilon)^n$ , while  $f_k(x)$  is about  $1-(n/2)\epsilon^2$ , which is much closer to unity. The relative errors, if small enough, are well approximated by  $n\epsilon$  and  $(n/2)\epsilon^2$ . Thus for a relative error of 5% at the specular peak, and  $n$  of 1000, the standard formulation requires  $e < 0.00005$  (1 part in 20,000 normalization accuracy—more than 14 bits), while the new formulation allows  $e < 0.01$  (only 1 part in 100—fewer than 7 bits). So we should be able to get away with a cheap approximate normalization with up to about a 1% residual length error (more or less, depending on the maximum  $n$  used and the maximum angle interpolated across).

## Approximate Normalization

A vector  $\mathbf{N}'$  is normalized to produce  $\mathbf{N}$  by multiplying by  $1/\sqrt{\mathbf{N}' \cdot \mathbf{N}'}$ , so for cost-effective hardware we need a good approximation to a reciprocal square root. A Taylor series expansion about 1 is a good start, using  $z$  to represent  $\mathbf{N}' \cdot \mathbf{N}'$ :

$$g(z) = \frac{1}{\sqrt{z}} = 1 - \frac{1}{2}(z-1) + \frac{3}{8}(z-1)^2 + \dots$$

If the vectors being interpolated are well normalized to unity length at the ends of each span, the linearly interpolated vectors will be shorter than unity length; in fact, they can be as short as  $\cos(\theta/2)$  if the vectors at the ends of the span are at an angle  $\theta$  from each other. The length can approach zero only if the vectors are nearly  $180^\circ$  apart, which can happen only for a local light or a local viewpoint very close to a surface. With little loss of generality, we can consider  $\theta$  to be restricted to  $90^\circ$ , in which case the squared length is not less than  $1/2$ .

Since  $x$  is less than 1 in the usual case, we negate  $z-1$  in the Taylor series to a positive error  $y = 1-z$ , and consider the following approximations of first and second order:

$$g_1(z) = 1 + \frac{1}{2}(1-z) = 1 + \frac{y}{2}$$

$$g_2(z) = 1 + \frac{1}{2}(1-z) + \frac{1}{2}(1-z)^2 = 1 + \frac{y+y^2}{2}$$

In the second-order case, we use an *ad hoc* modification of the second-order Taylor series coefficient, overcorrecting for a better overall fit as seen in Figure 4—the coefficients are also nicer, being in common and a power of two.

When  $y$  is  $1/2$ , the maximum error for interpolating across  $90^\circ$ , the correct answer is  $g(z) = 1.414$ . The approximations give  $g_1(z) = 1.25$  and  $g_2(z) = 1.375$ , for errors of just under 12% and 3%, respectively. The former is probably too much error, but the latter is probably acceptable—it implies a significant (factor of two) reduction in specular height near the middle of a span for very shiny ( $n = 1000$ ) surfaces when interpolating across large angles ( $90^\circ$ ).

Figure 4 shows the length of vectors after approximate normalization for the first- and second-order approximations, and shows that the second-order approximation is good to better than 0.5% for input vector lengths as low as 0.8, corresponding to interpolation across more than  $60^\circ$ , which is plenty for surface normals in reasonable object models. The first-order approximation is seen to be reasonable only over a more limited range. Note that the second-order version over-corrects vectors of length near 0.9 to a final length of about 1.002, which is a completely negligible error for our purposes.

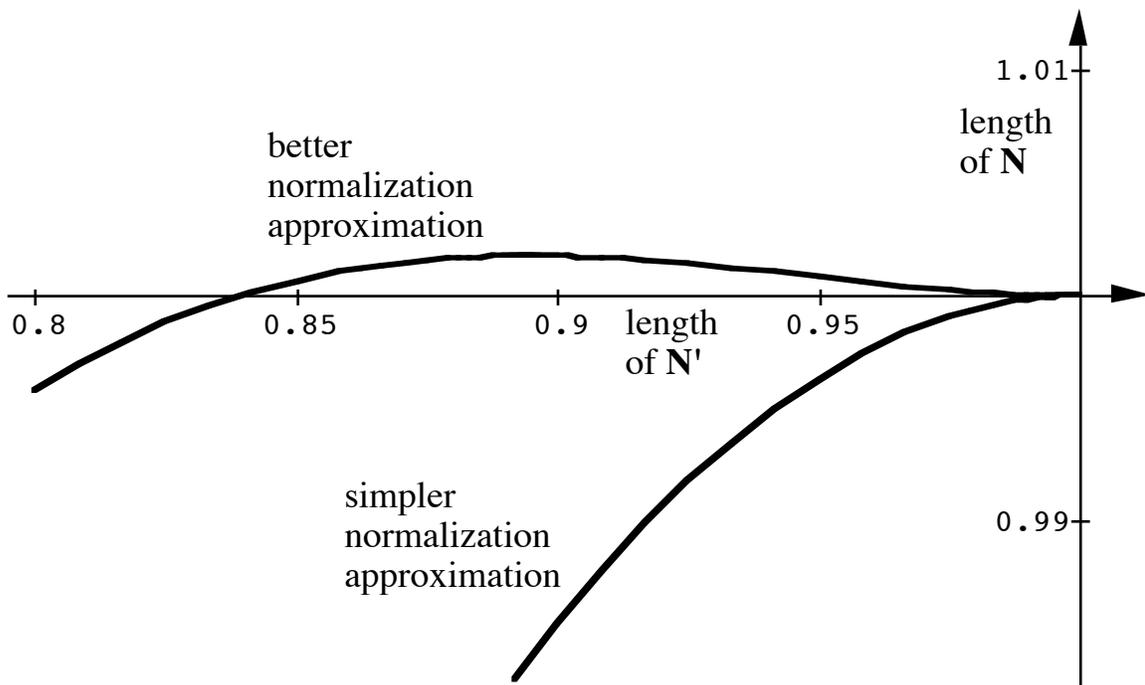


Figure 4: Graph of the length of an approximately normalized vector as a function of the length of the original vector, for first order (lower curve) and second-order (upper curve) approximations.

## Gaussian and Exponential Relationships

After reformulating the specular reflection model more as less as described above, we noticed a simpler approach based on approximately Gaussian specular reflection shapes and on an exponential approximation. We also observe that a number of physical models have been invoked in deriving various related specular reflection shapes, including Gaussian, which are mostly well approximated the same way [Blinn77].

The key exponential relationship we need is

$$(1 + x/K)^K \approx \exp(x) \text{ for } x/K \ll 1$$

In addition, we use the definition of a zero-mean Gaussian:

$$\exp(-(1/2)(x/\sigma)^2)$$

We also need  $\cos^n \alpha$  in terms of  $\alpha$ , either using the Taylor series as before, or this way using  $\sin \alpha \approx \alpha$ :

$$\cos^n \alpha = (\sqrt{1 - \sin^2 \alpha})^n = (1 - \sin^2 \alpha)^{(n/2)} \approx (1 - \alpha^2)^{(n/2)}$$

For large  $n$  and small  $\alpha$ , the above is recognized as a Gaussian of standard deviation  $\sigma = 1/\sqrt{n}$ , by applying the exponential approximation:

$$\cos^n \alpha \approx (1 - \alpha^2)^{(n/2)} \approx \exp(-(n/2)\alpha^2)$$

Working from the Gaussian back to the exponential form with a different specified exponent, one obtains the reformulated shape function, where now  $K$  is a more general form for the previous  $2^k$ :

$$\cos^n \alpha \approx \exp(-(n/2)\alpha^2) \approx (1 - (n/2K)\alpha^2)^K$$

To complete the derivation,  $\alpha$  must be approximated by the length of the chord or difference vector  $\mathbf{D}$ , as before. Of course, for ease of implementation using successive squarings,  $K$  should still be chosen to be a power of two.

## The Half-way Vector Approach

Figure 5 illustrates, using a vector diagram similar to that of Figure 2, three alternative ways to compute a difference vector  $\mathbf{D}$  using a half-way vector  $\mathbf{H}$  defined by  $\mathbf{H} = \mathbf{V} + \mathbf{L}$ , half way between the light and viewpoint, rather than by using a reflected vector  $\mathbf{R}$ .

In the method of Figure 2, vector  $\mathbf{D}$  was calculated as  $\mathbf{D} = \mathbf{R} - \mathbf{L}$ , or  $\mathbf{D} = (2\mathbf{V} \cdot \mathbf{N})\mathbf{N} - \mathbf{V} - \mathbf{L}$ . As a first method in Figure 6, we show a vector  $\mathbf{D}_1$ , which is identical to vector  $\mathbf{D}$ , which may be seen by substituting  $-\mathbf{H}$  for  $-\mathbf{V} - \mathbf{L}$  in the definition of  $\mathbf{D}$  to obtain difference vector  $\mathbf{D}_1$  as  $\mathbf{D}_1 = (2\mathbf{V} \cdot \mathbf{N})\mathbf{N} - \mathbf{H}$ . But now the difference is expressed relative to the half-way vector  $\mathbf{H}$ .

In Figure 5 we have found it convenient to illustrate the calculation of  $(2\mathbf{V} \cdot \mathbf{N})\mathbf{N}$  by first doubling vector  $\mathbf{V}$  to obtain vector  $2\mathbf{V}$ , and then projecting it onto  $\mathbf{N}$ , rather than by first projecting and then doubling as in Figure 2; this difference is not relevant except to reduce clutter in the center of the vector diagram.

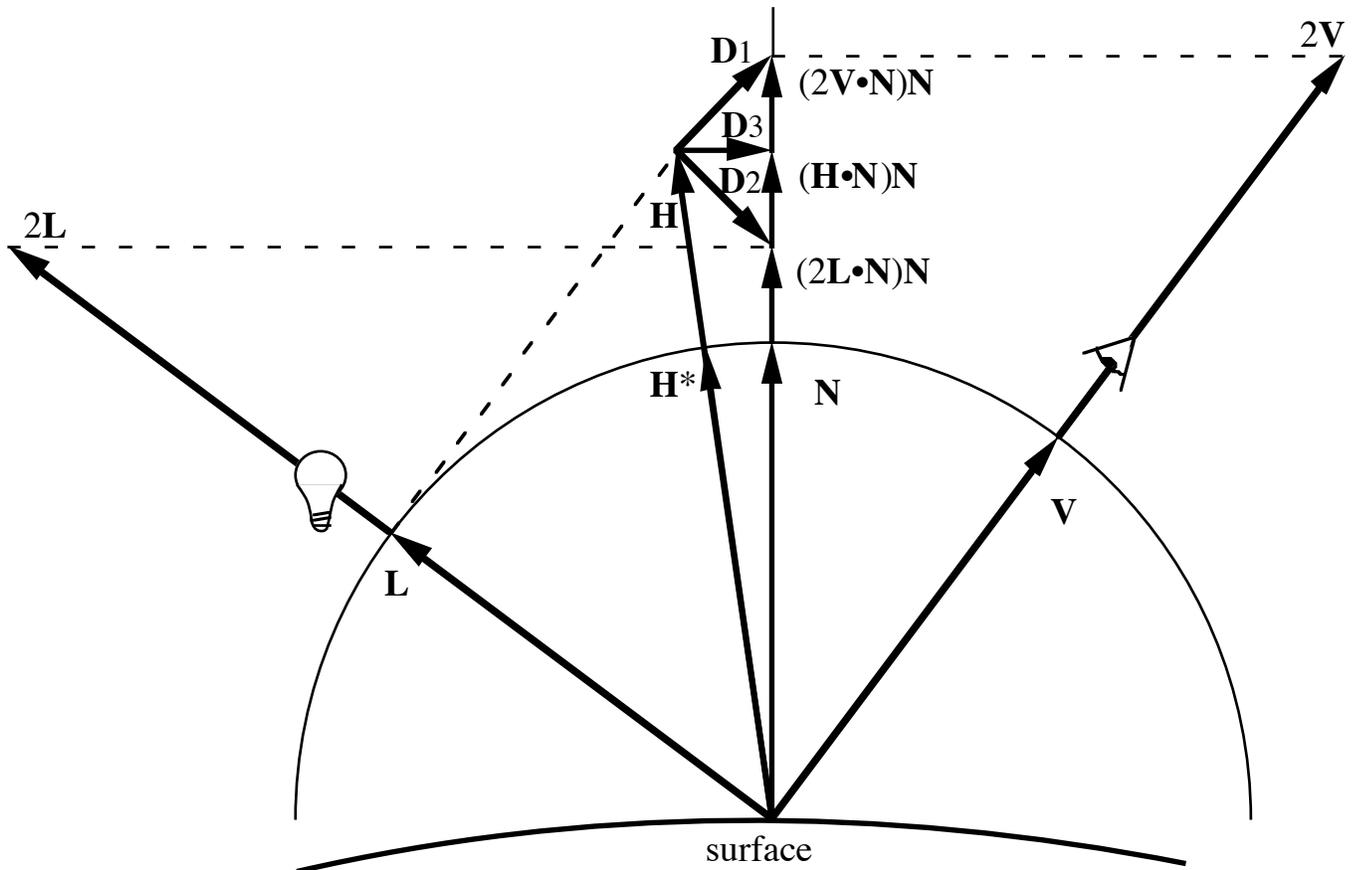


Figure 5: Vector diagram showing three ways to use an un-normalized half-way vector  $H = V+L$  to compute a difference vector.

As mentioned above, instead of comparing the light vector with the reflected viewpoint vector, the same result (in terms of dot product) may be obtained by comparing the viewpoint vector with the reflected light vector. Correspondingly, in our reformulation, a difference vector may be generated by subtracting a viewpoint vector from a reflected light vector. This method is not illustrated explicitly, but produces the identical vector as the second approach illustrated in Figure 5, which computes  $D_2$  as  $D_2 = (2L \cdot N)N - H$ , using the half-way vector  $H$  and the projection of the doubled light vector  $2L$  onto  $N$ .

Vector  $D_2$  is not equal to vector  $D_1$ , but their lengths are equal, so they may be used interchangeably—ignoring normalization imperfections. A third method of computing a difference vector  $D_3$  that is not equivalent is motivated by the observation that  $2V$  and  $2L$  appear in equivalent places in the definitions of equivalent vectors, so perhaps  $V+L$  would work as well or better. This reasoning leads to the computation of vector  $D_3$  as  $D_3 = (H \cdot N)N - H$ , the difference between the half-way vector  $H$  and its projection  $(H \cdot N)N$  onto  $N$ . We have not yet analyzed the extent to which this variation might impact the appearance of a rendered image,

as might be expected particularly in the case of grazing angles of reflection, where a short  $\mathbf{H}$  would tend to produce a short  $\mathbf{D}$ , for an increased reflection; this effect could be a feature or a bug.

In the half-way vector approach of Blinn [Blinn 1977], the normalized half-way vector  $\mathbf{H}^*$  is used, and  $\mathbf{H} \cdot \mathbf{N}$  is used as the basis of the specular reflection computation. Our  $\mathbf{D}_3$  method may be viewed as an improvement of Blinn's method, since it entirely avoids the need for a normalization of the half-way vector—we need the projection operation instead, but no square root or divide. The half-way vector is in general difficult to normalize accurately, since the length of  $\mathbf{E} + \mathbf{L}$  may be near zero, even at a maximal specularity. If it is desirable to better approximate Blinn's method, it is possible to normalize or approximately normalize  $\mathbf{H}$  to obtain  $\mathbf{H}^*$ , and to then compute  $\mathbf{D}$  as a difference either between  $\mathbf{H}^*$  and  $\mathbf{N}$  or between  $\mathbf{H}^*$  and its projection  $(\mathbf{H}^* \cdot \mathbf{N})\mathbf{N}$ . These additional methods are not illustrated. There may be a variety of other ways to compute a difference vector whose length is useful as an indicator of the extent to which the configuration being rendered deviates from the maximal specular reflection.

## Pixel Shading Hardware Datapaths

Figure 6 shows a block diagram of a portion of a hypothetical hardware arrangement that uses the first version of our reformulation—i.e., the version with the reflected viewpoint vector. Linear interpolation (LIRP) and approximate normalization are shown only for the surface normal, but could also be used for the light and viewpoint vectors. We show only basic data paths, and not the logic that controls iterations over multiple lights—it's clear from the diagram which parts depend on  $\mathbf{L}$  and would need to be repeated.

The dot product and exponentiation calculations of the standard Phong renderer have been replaced by the more easily implemented reformulation as shown. The squared magnitude of the difference vector  $\mathbf{D}$ , the dot product of  $\mathbf{D}$  with itself, is multiplied by shininess  $n$  to generate the intermediate scalar  $x$ . We show a divide by two here, based on our chosen definition of  $x$ , but since the next block divides  $x$  further, the shifts would probably be combined into one place—so don't take the hardware block diagram too literally. Finally, a shape function  $f_k(x)$  is performed on  $x$  to simulate the drop-off of specular reflection with distance from the specularity. Internally, the shape function is just a shift, a subtraction, and zero or more cascaded squaring operations. Unlike typical implementations using exponentiation and exact normalization, this one uses no tables and no iterations.

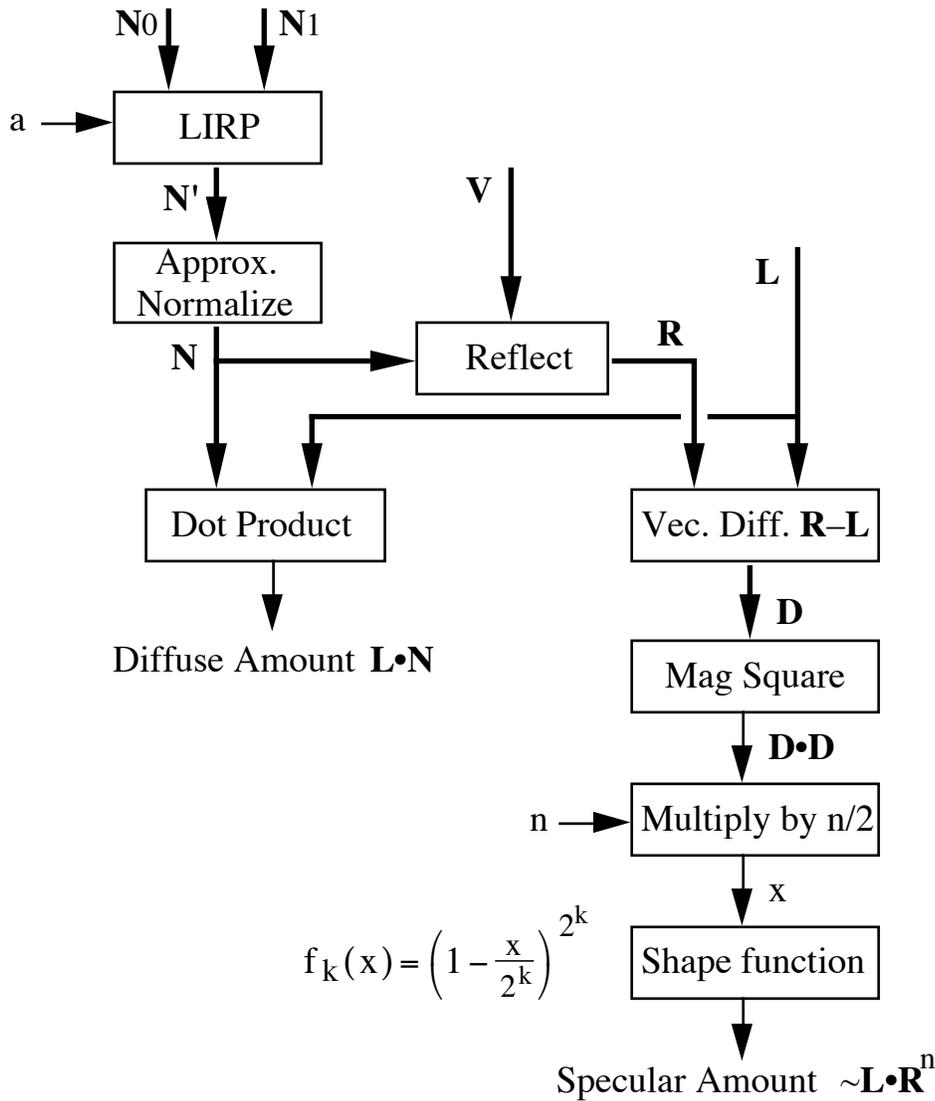


Figure 6: Block diagram of Phong shading diffuse reflection and specular reflection calculators, showing surface normal interpolation and approximate normalization.

Expanding on Figure 6, a more detailed hardware data path diagram is shown in Figure 7, annotated with plausible numbers of bits of fixed-point data representation at each point, specified as n.m for n+m bits with n bits of left of the point and m to the right, or Sn.m for signed values totaling 1+n+m bits. The proposed bit counts have not been simulated, and may have been too aggressively reduced in some cases. An extra high bit may be needed at some locations for edge-condition values (e.g. exactly 1.0). Additional tests may also be needed; for example, the “backside”

test  $\mathbf{N} \cdot \mathbf{L} < 0$  is usually applied to zero the diffuse component, but would introduce a small discontinuity if used to zero the specular component.

Figure 7 (parts a–d) is a detailed diagram of the specular reflection calculator in terms of elementary arithmetic operations, using the method of Figure 2 and the structure of Figure 6. Figure 7a shows the calculation of the interpolated and approximately normalized normal vector  $\mathbf{N}$ . Figure 7b shows the Reflect operation that calculates the reflected viewpoint vector  $\mathbf{R}$  from  $\mathbf{N}$  and  $\mathbf{V}$ . Figure 7c shows the calculation of the difference vector  $\mathbf{D}$ , its squared magnitude, and the intermediate scalar value  $x$  (actually  $2x$  in this case). Figure 7d shows the calculation of the final specular amount as a shape function of  $x$ . The calculation of diffuse amount  $\mathbf{N} \cdot \mathbf{L}$  is not shown. Triple lines represent X, Y, Z datapaths, and are not meant to imply a particular choice of parallelism versus serialization of the data and calculations.

In Figure 7a, surface normal vectors  $\mathbf{N}_0$  and  $\mathbf{N}_1$  representing two corners or edges of a surface patch to be rendered are provided as input to a linear interpolator, which produces the linear combination  $\mathbf{N}'$  as determined by the interpolation factor “ $a$ ”, which is applied as a weighting factor on  $\mathbf{N}_1$ . A “ $1-$ ” block subtracts its input from unity to produce “ $1-a$ ” as a weight for  $\mathbf{N}_0$ . An equivalent linear interpolation hardware structure might merge the subtraction and two multipliers into a single structure. An approximate normalization operation on  $\mathbf{N}'$  follows. The dot product  $\mathbf{N}' \cdot \mathbf{N}'$ , representing the squared length of  $\mathbf{N}'$ , is provided as the input “ $z$ ” to the  $g_2(z)$  approximation of the reciprocal square root, as discussed above. The result is used to scale  $\mathbf{N}'$ , to produce an approximately normalized normal vector  $\mathbf{N}$ .

With 10-bit fractional parts on all the direction vectors, length errors and angle errors will correspond to less than 0.001 radian, except when interpolating across large angles, in which case the approximation will introduce more error. The error of 0.001 radian is small compared to the smallest specularity angular radius, unless  $n$  is allowed to approach a million, so it should not be a big contributor to the final error.

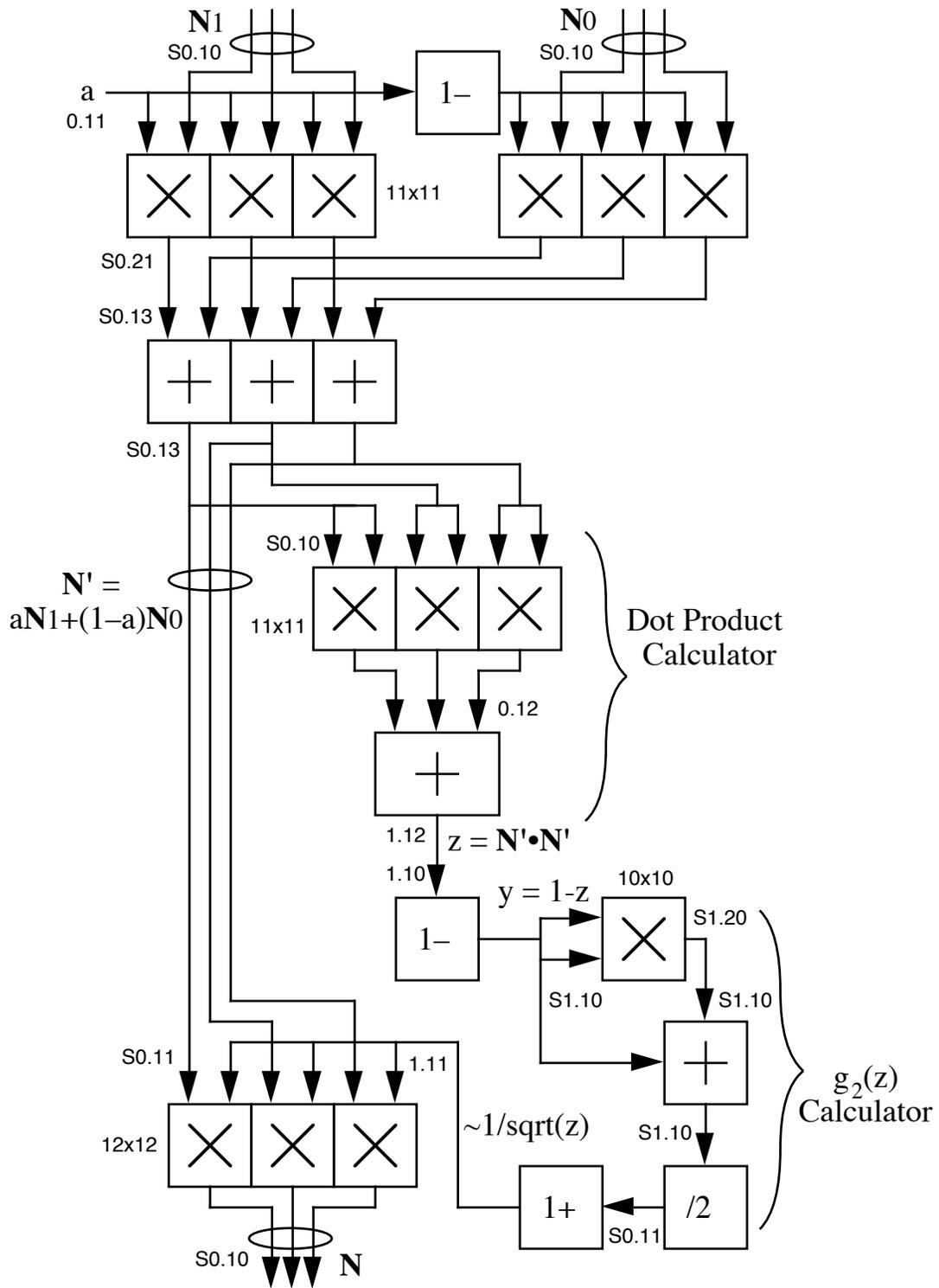


Figure 7a: Interpolation and Normalization

Figure 7b shows the Reflect operation. Normal vector  $N$  and viewpoint vector  $V$  are dotted to produce  $N \cdot V$ , which is then doubled and multiplied by  $N$  to produce the projection  $(2N \cdot V)N$ . Finally, subtractors compute the reflected viewpoint vector  $R$ .

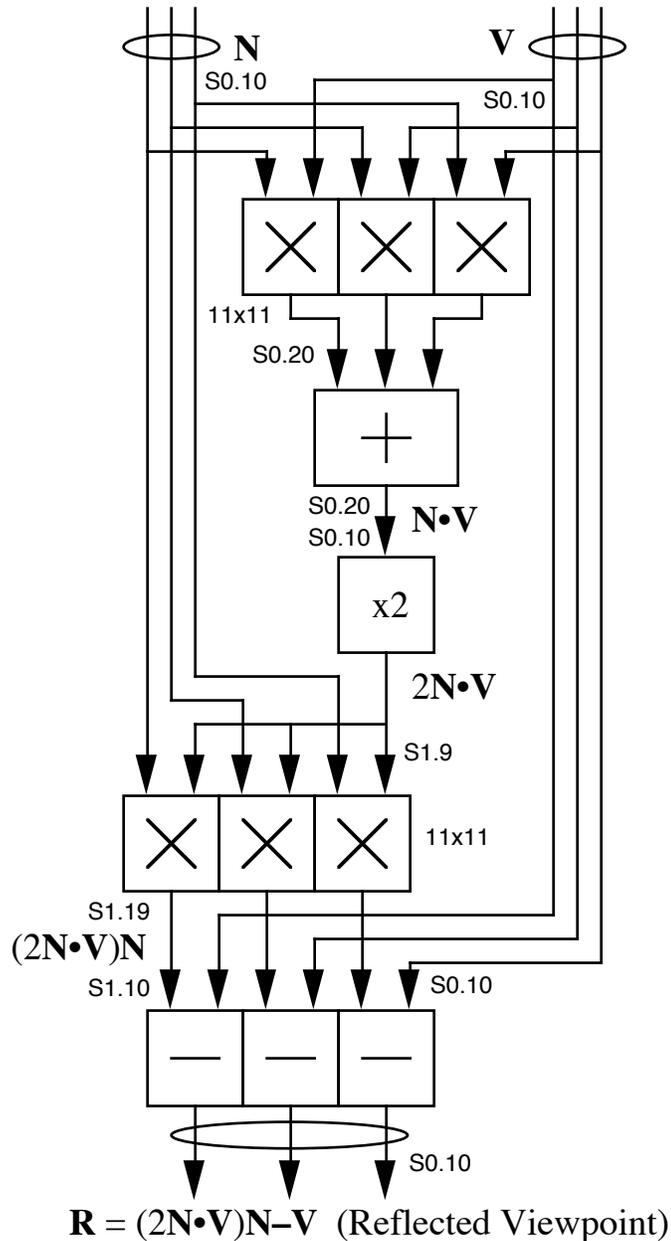


Figure 7b: Reflection of Viewpoint about Normal

Figure 7c shows the computation of  $\mathbf{D}$  as the vector difference  $\mathbf{R}-\mathbf{L}$ , followed by the dot product of  $\mathbf{D}$  with itself,  $\mathbf{D}\cdot\mathbf{D}$ , which is the squared length of  $\mathbf{D}$ . Finally, intermediate value  $2x$  is calculated by multiplying by the material shininess parameter  $n$ . A rather large multiplier is needed for this operation, since  $n$  can be large or small (1 to 8000 in the example) and it scales the wide dynamic range of  $\mathbf{D}\cdot\mathbf{D}$  into an even wider range number  $2x$ , of which the fractional part represents the smaller range over which the shape function is nonzero.

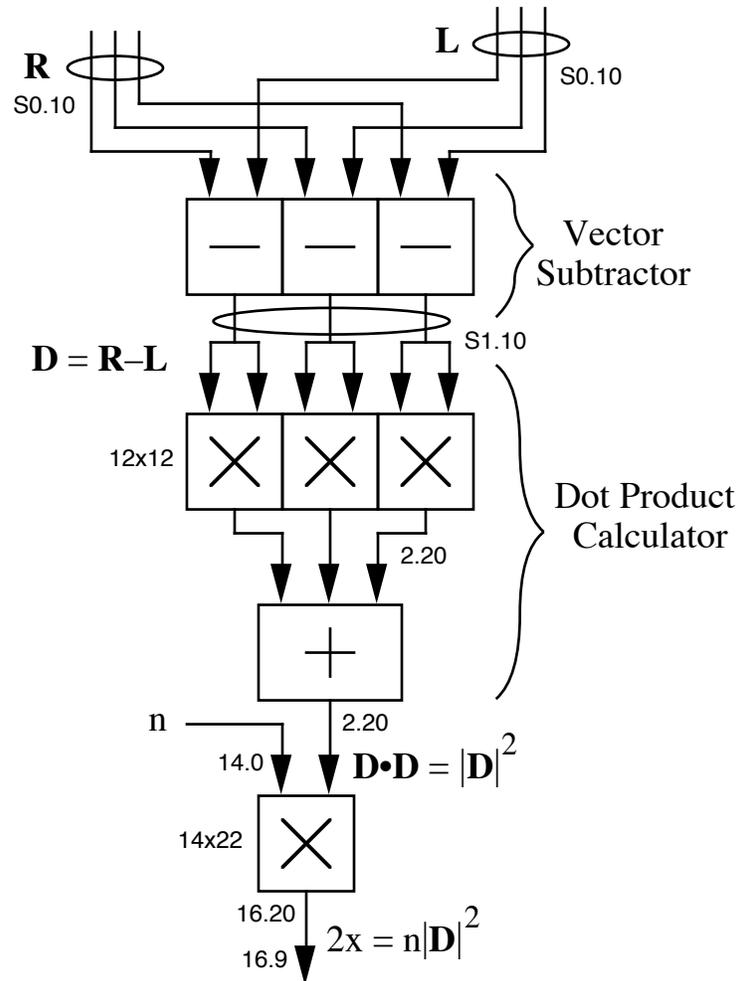


Figure 7c: Difference Vector and x calculation.

In Figure 7d, the intermediate value  $2x$  is arithmetically to the right by 3 places, producing  $x/4$ . The "1-" block computes  $1-x/4$ . Two multipliers are connected as successive squaring circuits to compute the fourth power  $(1-x/4)^4$ . A comparison of  $x/4$  to unity controls the output multiplexor to select either  $(1-x/4)^4$  if  $x/4$  is less than unity, or zero otherwise, to produce the specular amount.

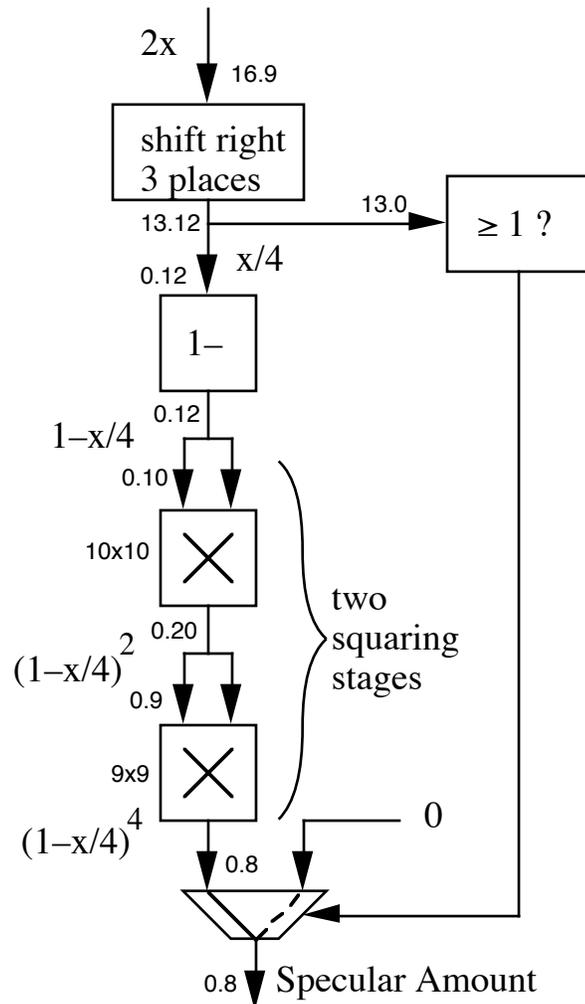


Figure 7d: Shape function calculation for  $k=2$ .

The particular structure described in Figure 7 is based on the choice of shape function characterized by  $k=2$ . A shape function characterized by a different value of  $k$  can be implemented simply by changing the shifter to shift by  $k+1$  places and changing the number of successive squaring multipliers to be  $k$ , with no further changes.

## Hardware Rearrangements for Bit Reductions

The wide dynamic range of the shininess coefficient  $n$  is itself a problem when designing an efficient hardware renderer, adding ten or more bits to the accuracy requirements on the  $\mathbf{D} \bullet \mathbf{D}$  calculation. Since  $n$  represents a specularity with angular radius of  $\sqrt{1/n}$  radians, we propose to substitute a modified shininess parameter  $m = \sqrt{n}$ , which specifies a specularity of angular radius  $1/m$ . If  $m$  is an integer from 1 to 63, for example, it covers the same range of shininess as  $n$  from 1 to 4000, though in coarser steps. The calculation  $x = (n/2)\mathbf{D} \bullet \mathbf{D}$  is still easy, multiplying the three components of  $\mathbf{D}$  by  $m$ , rather than the dot product by  $n$ :  $x = (1/2)m\mathbf{D} \bullet m\mathbf{D}$ .

While this approach appears to be more complex, using three multiplications in place of one, it can have significant beneficial impact on fixed-point hardware due to quantization issues.

Figure 8 shows a detailed diagram of an alternative implementation of the portion of a specular reflection calculator following the computation of the difference vector  $\mathbf{D}$ , but using  $m$  instead of  $n$ . The computation proceeds as follows: difference vector  $\mathbf{D}$  is rectified (absolute value of each component) for reasons that will become apparent shortly; the rectified vector  $\mathbf{D}^*$  is multiplied by the parameter  $m$  to produce vector  $m\mathbf{D}^*$ ; the dot product operator computes  $(m|\mathbf{D}^*|)^2$ , corresponding to  $x/4$ . The shape function is as in Figure 7d, except that we use  $k=1$  in this example and the comparator is augmented with more inputs to be compared to unity.

As described above, the implementation of Figure 8 will produce approximately the same result as the one of Figure 7, with the absolute value operators and the augmentations to the comparator having no effect. The scheme of Figure 8 is apparently more costly than that of Figure 7, since it uses two more multipliers to multiply three vector components by  $m$  than to multiply a scalar by  $n$ . But this scheme may be preferred in a hardware implementation because much smaller multipliers can be used, due to numerical advantages of this rearrangement. In particular, the dot product operator needs to accept as input only the bits representing the fractional parts of the vector components, if the components are positive as assured by the absolute value operators, and if the integer parts are detected by the augmented comparator. Many multiplier bits are saved; for example, the six  $7 \times 11$  and  $9 \times 9$  multipliers are likely to be smaller and faster than the three  $12 \times 12$  and one  $14 \times 22$  required for similar accuracy in the arrangement of Figure 7.

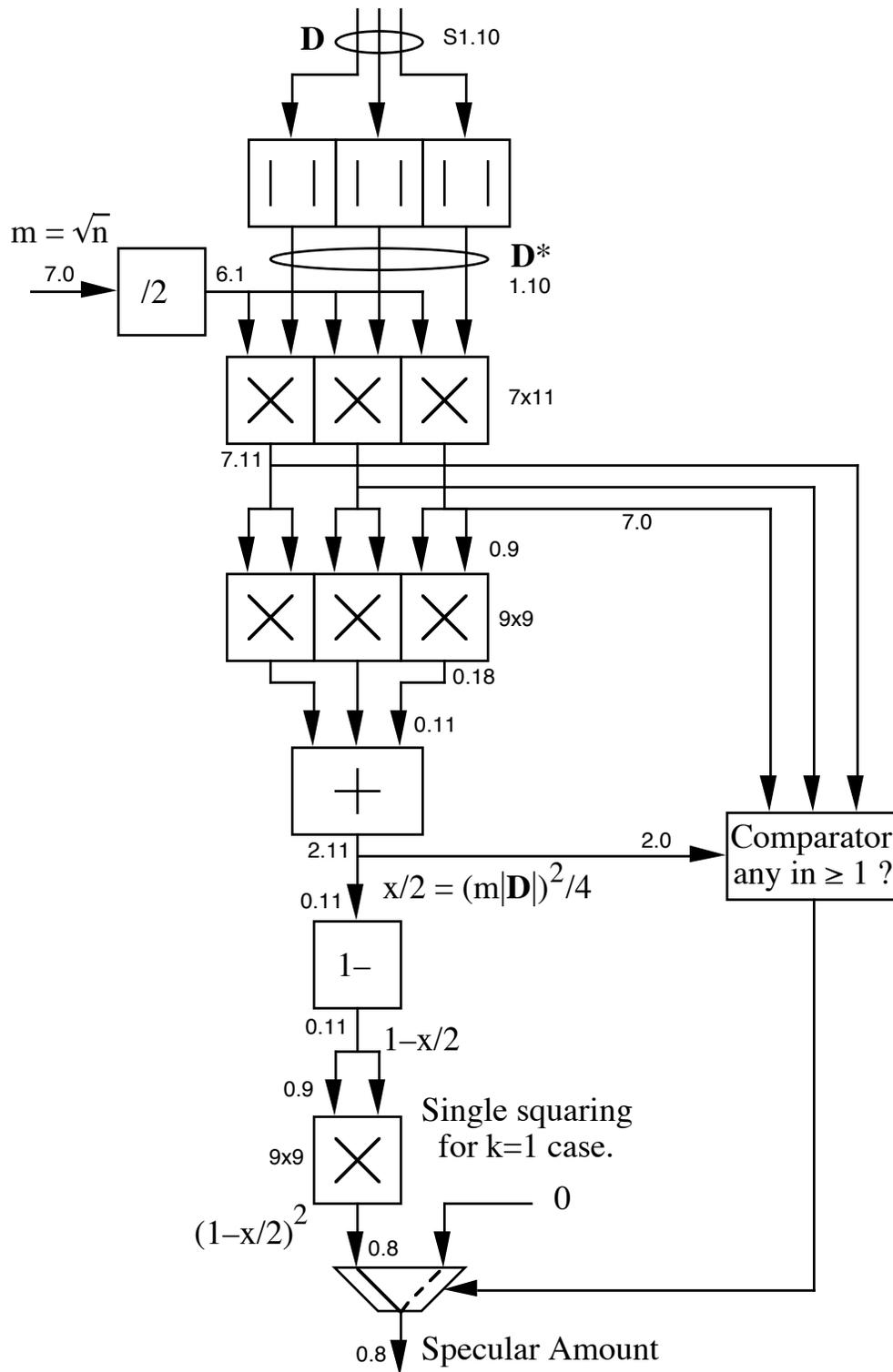


Figure 8: Hardware portion modified to use the proposed "root-shininess" parameter  $m$  and generally fewer bits.

## Simulation Results

In order to test the appearance of shiny objects rendered using the proposed new specular formulations, we have hacked together a simple test program in MATLAB (a “matrix laboratory” interactive environment available on most computers, including Macintosh). The few pages of code involved are shown in the appendix so that details may be verified and duplicated.

The simulation shades only unit-radius spheres, which have the nice property that the surface normal at any point is equal to the location of the point, with the sphere center as origin. The form of the figure is modeled after figures comparing Phong shading with the Blinn  $\mathbf{H} \cdot \mathbf{N}$  variant in *The Renderman Companion* [Upstill90].

To further simplify the test, light and viewpoint vectors are constant (at infinity, as opposed to “local”), and colors are monochrome. Since polygons and spans are not used, no normal vector interpolation is done, and the approximate normalization performance is therefore not tested—only the specular shape is tested. By fixing the viewpoint at (0,0,1), we implicitly project object points  $[x,y,z]$  to image points  $[x,y]$ , so that it is easy to go from the image point to the object point and normal vector as  $[x,y,\sqrt{1-x^2-y^2}]$ . Multiple spheres are rendered as if each is at the origin, and their images are overlapped by offsetting image plane indices. Thus, in this simple test program we have no explicit representation of objects, spans, colors, or depths, and no explicit computation of projection, hidden surface removal, etc. Two light source locations and some other parameters are specified as constants in the MATLAB code script. The function file `specfuncs.m` returns of a vector of specular amounts determined by each of the methods being compared, saving lots of duplicated overhead that would result by rendering separately by each method. Within `specfuncs.m`, we attempt to give the shininess coefficient  $n$  a consistent interpretation in terms of the quadratic rate of falloff of the specularity from its peak (i.e., the first nontrivial Taylor coefficient as a function of angle  $\alpha$  or approximate equivalent); this interpretation of specularity “width” is much simpler than Blinn's  $\beta$  definition [Blinn77], but may be less meaningful for very small  $n$ .

The code in the appendix produces the result presented in Figure 9, which shows dark (low diffuse reflectivity) spheres lit by two point light sources, one to the right, high, and in back of the spheres, and the other to the right, low, and in front (i.e. light direction vectors parallel to  $[1,1,-1]$  and  $[1,-1,1]$ ). Five different shininess values are tested, and twelve different specular formulations, as noted in the figure. The leftmost column, with  $n=1$ , is not a particularly useful case, but shows up some of the bad edge effects of some of the formulations in this limiting case. For example,

the original Phong method specularly shape goes abruptly to zero for  $n=1$  but not for  $n>1$ ; the Blinn  $(\mathbf{H}\cdot\mathbf{N})^{4n}$  method avoids this behavior at approximately the same specularly width, since it uses a four times greater exponent for a given width.

The first thing to notice is that for  $n>4$  or so, all the methods work OK, with the exception of row 10, which loses the glancing hilite. This defect is due to the vector  $\mathbf{D} = \mathbf{H}g_2(z)-\mathbf{N}$  never getting very short since  $\mathbf{H}g_2(z)$  is not well normalized (even in this case, where approximate normalization  $g_2(z)$  is used, since  $\mathbf{H}$  can start out way too short); that's why we suggest the projection of  $\mathbf{H}$  onto  $\mathbf{N}$ —so that the distance goes to zero when the angle goes to zero, even if the length is off. Note that in the code we define  $\mathbf{H} = (\mathbf{L}+\mathbf{V})/2$ , so that's its length is always unity or less.

Next, getting to the detail of specularly shape relative to our  $k$  parameter, notice that the  $k=2$  shape (row 4) is essentially indistinguishable from the  $\cos^n\alpha$  shape, which is quite "fuzzy," for large enough  $n$ . The  $k=1$  shape (row 3), which is a little easier to compute and behaves a little better for small  $n$ , actually looks "better" (according to the author and certain other individuals who shall remain anonymous for their own protection). This result holds for both the Phong-like and Blinn-like versions (rows 8 and 9). The  $k=0$  shape (row 2), on the other hand, drops too abruptly to look good; in fact, at  $n=1$ , it looks just as bad as the original Phong version, which it approximates excellently at that point.

The next thing to notice is the different shape of the glancing hilite between the Phong-like methods (1–4) and the Blinn-like methods (5–12), due to the different way that the angle between  $\mathbf{L}$  and  $\mathbf{R}$  evolves compared to that between  $\mathbf{H}$  and  $\mathbf{N}$ , in three dimensions. This difference is explicitly compared in the figure on page 318 of *The Renderman Companion* [Upstill90], but is difficult to appreciate there due to the non-comparable parameter values and different amounts of saturation in that figure. Row 6 shows an excellent approximation to Blinn's method, but requiring a normalization of  $\mathbf{H}$ .

The  $\mathbf{H}\cdot\mathbf{N}$  specularly function is not really what Blinn proposed, but is only the "D1" (distribution of surface normals) factor from his more complex model. This function is widely used by itself, however, and in many cases is actually referred to as the Phong specularly model, as explained by Hall [Hall89, p. 77]. Blinn also proposes other shapes "D2" and "D3" which are functions of the same angle, lead to similar appearance, and would lead to identical approximations under our proposed reformulation. The "D2" or "Torrance and Sparrow" function (row 11), being a Gaussian, would be identical to "D1" for large  $n$  and more like our reformulation for

small  $n$ . The "D3" or "Trowbridge and Reitz" function (row 12) is even more "fuzzy" than the  $\cos^n\alpha$  and Gaussian shapes, and never drops to zero.

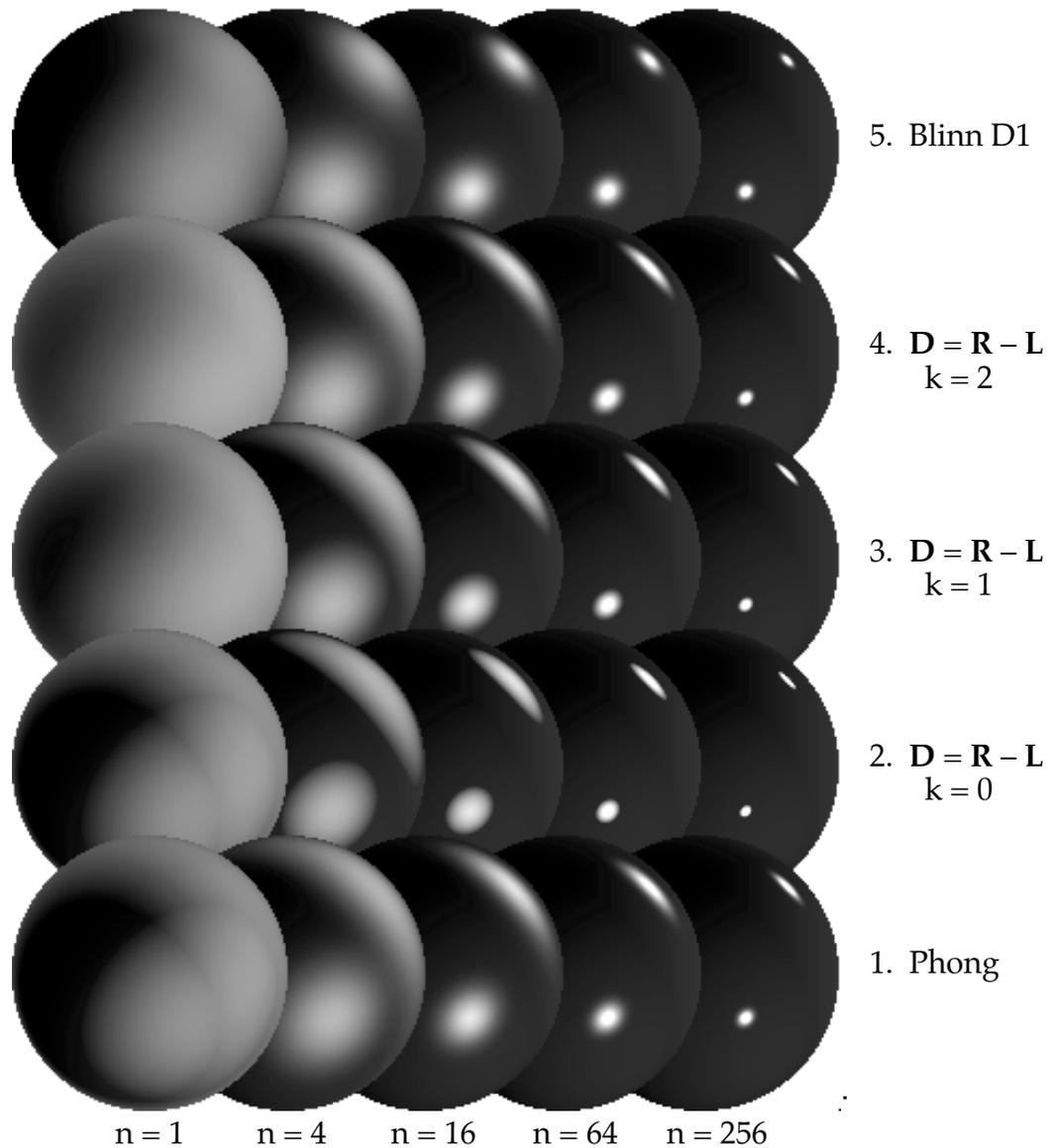
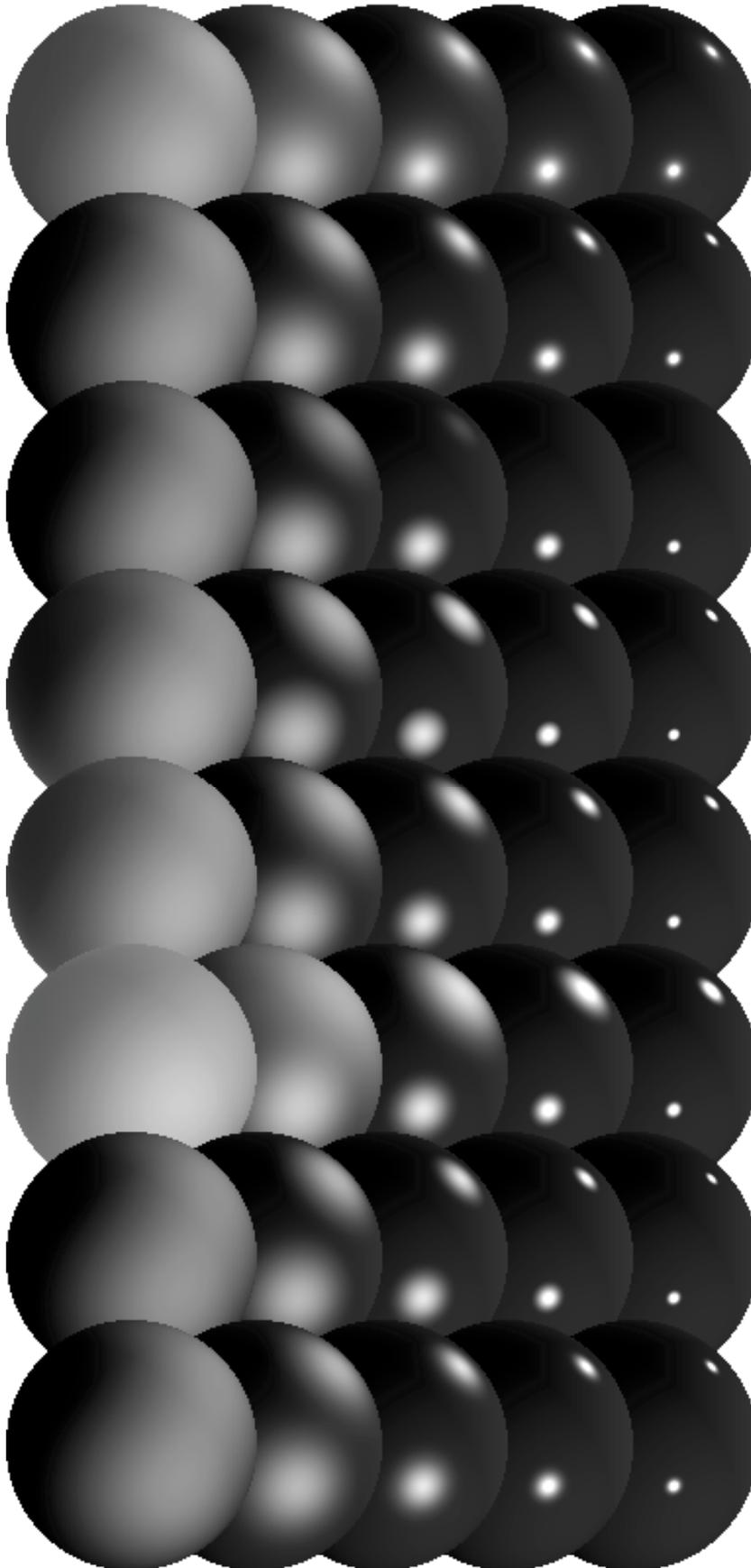


Figure 9: Shiny black balls as rendered by the MATLAB test program, with five shininess values  $n$  in each row, and a row for each of twelve techniques (continued on next page with one row duplicated).



12. Trowbridge & Reitz  
(Blinn D3)

11. Torrance & Sparrow  
(Blinn D2, Gaussian)

10.  $D = Hg_2(z) - N$   
 $k = 2$

9.  $D = Hg_2(z) - \text{projection}$   
 $k = 1$

8.  $D = Hg_2(z) - \text{projection}$   
 $k = 2$

7.  $D = H - \text{projection}$   
 $k = 2$

6.  $D = H^* - N$   
 $k = 2$

5. Blinn D1

## Conclusions

The reformulation of Phong shading in terms of a distance instead of a dot product leads to a simpler hardware implementation with less numerical difficulty, because the length of a chord between two vectors is a simpler and more sensitive way to measure their alignment than a dot product is. Besides leading to simpler hardware, improved or faster software implementations may also be enabled by this reformulation.

## Acknowledgments

The author thanks Stephanie Winner, Mike Kelley, Kirk Gould, Frank Crow, and Ken Turkowski for many useful discussions and lots of good background information on rendering in hardware and software. Special thanks go to Stephen Johnson for implementing and testing a software version of this work, which will be described in a subsequent report.

## Appendix — MATLAB™ Simulation Code

-----file specular.m-----

```
% test of specular amount calculation

m = 64; % 64 steps from 0 to 1 radius of sphere.
ns = [1 4 16 64 256]; % shininess values to show
specfs = ns.^0.25; % specular amounts 1 to 4
specfs = 400/max(specfs)*specfs % max specular amount to 400
diff = 35; %diffuse factor
amb = 0; %ambient amount
bg = 255; %bright background

lv1 = [1 1 -1]; % light high in back right
lv2 = [1 -1 1]; % light low in front right
lv1 = lv1*1/sqrt(sum(lv1.*lv1)); % first light vector
lv2 = lv2*1/sqrt(sum(lv2.*lv2)); % second light vector
vv = [0 0 1]; % viewpoint vector
mmax = m*length(ns)

ny = length(specfuns([0 0 1],lv1,vv,1,0)); % number of functions to test
dj = (0:ny-1)*(2*m-floor(m/2)); % y displacements by 1.5 radii

clear pic;
pic = bg*ones((1+length(ns))*m+1,2*m+1+max(dj)); % image buffer, background
```

```

for i = -m:1:mmax; % x iteration
    fprintf('%g ',i);
    ii = m+1+i;
    xx = (0.5+i)/m;
    for j = -m:1:m % y iteration
        jj = m+1+j;
        y = (0.5+j)/m;
        for ni = 1:length(ns); % search through spheres
            n = ns(ni); % material shininess n for this sphere
            specf = specfs(ni); % specularity factor for this sphere
            x = xx - (ni-1); % unit radius offset spheres from left to right
            z = (1 - x*x - y*y);
            if z >= 0;
                z = sqrt(z);
                break; % jump out when first sphere is found
            end;
        end
        totals = amb*ones(1,ny);
        if z >= 0; % case where sphere was found
            nv = [x y z]; % x,y,z are now coordinates of normal to (hemi)sphere
            dif1 = lv1*nv'; if dif1 > 0; totals = totals + diff*dif1; end;
            dif2 = lv2*nv'; if dif2 > 0; totals = totals + diff*dif2; end;
            totals=totals+specf*(specfuns(nv,lv1,vv,n,ny)+specfuns(nv,lv2,vv,n,ny));
            pic(ii,jj+dj) = totals;
        end;
    end;
end;
writeimagefile(pic,'shinyballs');

```

-----file specfuns.m-----

```

function amounts = specfuns(nv,lv,vv,n,len)
%function amounts = specfuns(nv,lv,vv,n,len) -- multiple versions

amounts = zeros(1,len); % (maybe) allocate place for results
count = 0; % number of specularity models done so far

rv = 2*nv*(vv*nv')-vv; % reflected view vector
hv = (lv+vv)/2;
hl = hv*hv'; % hv length^2
dl = 1-hl; % length error z
hv1 = (1+0.5*(dl+dl*dl))*hv; % approx. normalized halfway vector
hv2 = 1/sqrt(hl)*hv; % normalized halfway vector
n4 = 4*n; % "n" for halfway vector method

% 1. Phong method
dot = rv*lv';
if dot > 0;
    amount = dot^n;
else;
    amount = 0;
end;
count = count+1; amounts(count) = amount;

% Lyon main method, several k values for this D vector and x value
dv = lv - rv; % difference vector

```

```

xs = (dv*dv')*n/2;

% 2. Lyon k=0
if xs > 1;
    amount = 0;
else;
    amount = (1 - xs);
end;
count = count+1; amounts(count) = amount;

% 3. Lyon k=1
if xs > 2;
    amount = 0;
else;
    amount = (1 - xs/2);
    amount = amount*amount; % -- square k=1 times
end;
count = count+1; amounts(count) = amount;

% 4. Lyon k=2
if xs > 4;
    amount = 0;
else;
    amount = (1 - xs/4);
    amount = amount*amount;
    amount = amount*amount; % -- square k=2 times
end;
count = count+1; amounts(count) = amount;

% use n4 (4*n) in all the h-vector methods

% 5. Blinn method
dot = hv2*nv';
if dot > 0;
    amount = dot^n4;
else;
    amount = 0;
end;
count = count+1; amounts(count) = amount;

% 6. Lyon/Blinn halfway method, normalized hv2, k=2
dv = hv2 - nv; % difference vector
xs = (dv*dv')*n4/2;
if xs > 4;
    amount = 0;
else;
    amount = (1 - xs/4);
    amount = amount*amount;
    amount = amount*amount; % -- square k=2 times
end;
count = count+1; amounts(count) = amount;

% 7. Lyon halfway method, no norm of h, k=2
dv = hv - nv*(hv*nv'); % difference vector to projection
xs = (dv*dv')*n4/2;
if xs > 4;
    amount = 0;
else;

```

```

    amount = (1 - xs/4);
    amount = amount*amount;
    amount = amount*amount; % -- square k=2 times
end;
count = count+1; amounts(count) = amount;

% 8. Lyon halfway method 2, projecting hv1, k=2
dv = hv1 - nv*(hv1*nv'); % difference vector
xs = (dv*dv')*n4/2;
if xs > 4;
    amount = 0;
else;
    amount = (1 - xs/4);
    amount = amount*amount;
    amount = amount*amount; % -- square k=2 times
end;
count = count+1; amounts(count) = amount;

% 9. Lyon halfway method 2, projecting hv1, k=1
% same xs from above
if xs > 2;
    amount = 0;
else;
    amount = (1 - xs/2);
    amount = amount*amount; % -- square k=1 times
end;
count = count+1; amounts(count) = amount;

% 10. Lyon halfway method 3, hv1 no projection, k=2
dv = hv1 - nv; % difference vector --
xs = (dv*dv')*n4/2;
if xs > 4;
    amount = 0;
else;
    amount = (1 - xs/4);
    amount = amount*amount;
    amount = amount*amount; % -- square k=2 times
end;
count = count+1; amounts(count) = amount;

% 11. Blinn method D2 (Torrance and Sparrow)
% using some algebra and angular standard dev. of 1/sqrt(n4)
%dot = sum(hv2.*nv);
alpha = acos(dot);
amount = exp(-(n4/2)*alpha*alpha);
count = count+1; amounts(count) = amount;

% 12. Blinn method D3 (Trowbridge and Reitz)
% using some algebra and n in place of (1/c2^2 - 1)
%dot = sum(hv2.*nv);
if dot > 0;
    amount = (1/(1+(1-dot*dot)*n))^2;
else;
    amount = (1/(1+n))^2; % as low as it gets!
end;
end;
count = count+1; amounts(count) = amount;

```



## References

The original Phong shading papers:

[Bui Tuong73] Bui Tuong Phong. "Illumination for Computer-Generated Images," Ph. D. Dissertation, Department of Computer Science, University of Utah, Gov. ordering no. AD-A008 786, 1973.

[Bui Tuong75] Bui Tuong Phong. "Illumination for Computer-Generated Pictures," *Comm. ACM* 18(6), pp. 311–317, June 1975. (also reprinted in: Beatty, J. C. and Booth, K. S. (eds.). *Tutorial: Computer Graphics*, Second Edition, IEEE Comp. Soc. Press, Silver Spring, MD, 1982.)

This paper shows the half-way vector method and proposes a different simplification of the shape function, still in terms of the dot product of accurately normalized vectors, and requiring a divide instead of an exponentiation; also shows a Gaussian specular shape:

[Blinn77] Blinn, James F. "Models of Light Reflection for Computer Synthesized Pictures," *Computer Graphics, ACM SIGGRAPH '77 Proceedings*, pp. 192–198.

This book compares Phong's and Blinn's specular formulations:

[Upstill90] Upstill, Steve. *The Renderman Companion*, Addison-Wesley, 1990 (second printing with corrections, 1992).

Hall's book is the best reference on illumination models; it analyzes and compares Phong, Blinn, Torrance and Sparrow, Trowbridge and Reitz, etc.

[Hall89] Hall, Roy. *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, 1989.

## Additional Reading

Here's the hardware project that got me interested in this topic, though the version reported does not include Phong shading:

Kelly, M., Winner, S., and Gould, K. "A Scalable Hardware Render Accelerator using a Modified Scanline Algorithm," *Computer Graphics, ACM SIGGRAPH '92 Proceedings*, 26(2), pp. 241–248, 1992.

These are previous attempts to speed up Phong shading, but are not referenced in the text:

This one uses a 2D Taylor series centered in each polygon to approximate the dot product, and then uses a table to exponentiate:

Bishop, Gary, and Weimer, David M. "Fast Phong Shading," *Computer Graphics, ACM SIGGRAPH '86 Proceedings*, 20(4), pp. 103–106, 1986.

This paper proposes a method of angle interpolation via quaternions, to avoid the need for normalization, and provides another simplification of the Phong shape function, based on fitting three quadratic segments. They show how to reduce the whole process to a quadratic Gouraud-like interpolation, with possible additional break points in a span. It seems that they still need to do some cosine evaluations along the edges:

Kujik, A. A. M., and Blade, E. H. "Faster Phong Shading via Angular Interpolation," *Computer Graphics Forum*, Vol. 8, pp. 315–324, 1989.

This paper is a study of several efforts at speeding up Phong shading, concentrating on different interpolation and normalization schemes, rather than changes to the functional shape of the specularly:

Claussen, Ute. "On Reducing the Phong Shading Method," *Eurographics '89*, W. Hansmann, F. R. A. Hopgood, and W. Strasser (eds.), pp. 333–344, Elsevier Science Publishers, 1989.