Hardware Implementation of Phong Shading using Spherical Interpolation

Abbas Ali Mohamed, László Szirmay-Kalos, Tamás Horváth
Department of Control Engineering and Information Technology
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics, Hungary
email: abbas@seeger.iit.bme.hu

Abstract

Computer image generation systems often represent curved surfaces as a mesh of planar polygons that are shaded to restore a smooth appearance. In software rendering Phong shading has been one of the most successful algorithms, because it can realistically handle specular materials. Since it requires the rendering equation to be evaluated for each pixel, its hardware support poses problems. This paper presents a reformulation of the Phong shading algorithm, that is based on interpolating on the surface of spheres. The reformulation results in simpler formulae that can be directly implemented in hardware. The software simulations and the VHDL description of the shading hardware are also presented.

Keywords: Reflectance functions, BRDF representation, real-time graphics, Phong shading.

1 Introduction

Computer graphics aims at rendering complex virtual world models and presenting the image for the user. To obtain an image of a virtual world, surfaces visible in pixels should be determined, and the rendering equation or its simplified form is used to calculate the intensity of these surfaces, defining the color values of the pixels. The rendering equation [5] expresses the output radiance $I^{out}(\vec{x},\vec{V})$ of a surface point $\vec{x}$ at direction $\vec{V}$ as the function of the local surface properties and the incoming radiance I^{in} emitted by the light sources or reflected off other surfaces from direction $\vec{L}$:

$$I^{out}(\vec{x},\vec{V}) = \int_{\Omega'} I^{in}(\vec{x},\vec{L}) \cdot f_r(\vec{L},\vec{x},\vec{V}) \cdot \cos\theta^{in} d\omega_{\vec{L}} \tag{1}$$

where θ^{in} is the angle between the incoming direction $\vec{L}$ and the surface normal at the reflection point $\vec{x}$, i.e. $\cos\theta^{in} = \vec{L}\cdot\vec{N}$ if the vectors have unit length, f_r is BRDF (bi-directional reflected distributed function), and Ω' is the set of possible incoming directions forming a hemisphere.

If the indirect illumination coming from other surfaces is ignored and only directional and positional light sources are present, I^{in} is a Dirac-delta type function which simplifies the integral to a discrete sum:

$$I^{out}(\vec{x},\vec{V}) = \sum_l I_l^{in}(\vec{x},\vec{L}_l) \cdot f_r(\vec{L}_l,\vec{x},\vec{V}) \cdot \cos\theta_l^{in} \tag{2}$$

where I_l^{in} is the incoming radiance generated by light source l (figure 1). Note that this model does not account for the multiple reflections of the light, only the direct illumination of the light sources is taken into account.

The BRDF function f_r is responsible for the optical properties of the surface. In practice BRDF functions are mathematical formulae that have some free parameters that can be set to mimic a given material.

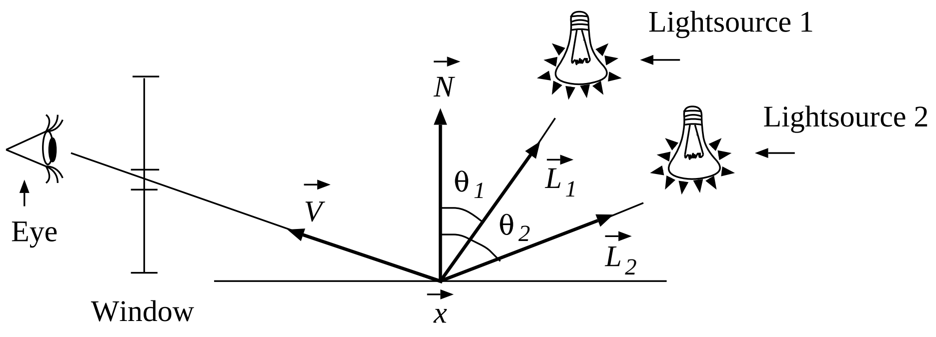


Figure 1: Radiance calculation in local illumination methods

2 Simple optical material models

Some materials are dull and reflect light dispersely and about equally in all directions (diffuse reflections); others are shiny and reflect light only in certain directions relative to the viewer and light source (specular reflections).

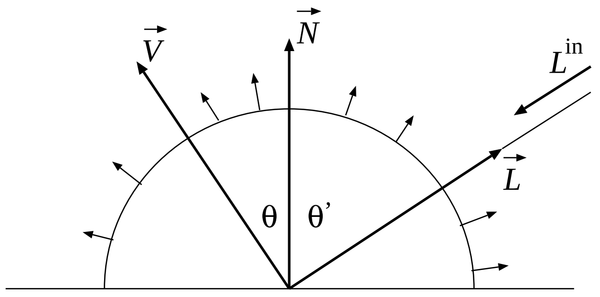


Figure 2: Diffuse reflection

First of all, consider diffuse — optically very rough — surfaces reflecting a portion of the incoming light with radiance uniformly distributed in all directions. Looking at the wall, sand, etc. the perception is the same regardless of the viewing direction (figure 2). If the BRDF is independent of the viewing direction, it must also be independent of the light direction because of the Helmholtz-symmetry [7], thus the BRDF of these **diffuse** surfaces is constant on a single wavelength:

$$f_{r,\text{diffuse}}(\vec{L},\vec{V}) = k_d. \tag{3}$$

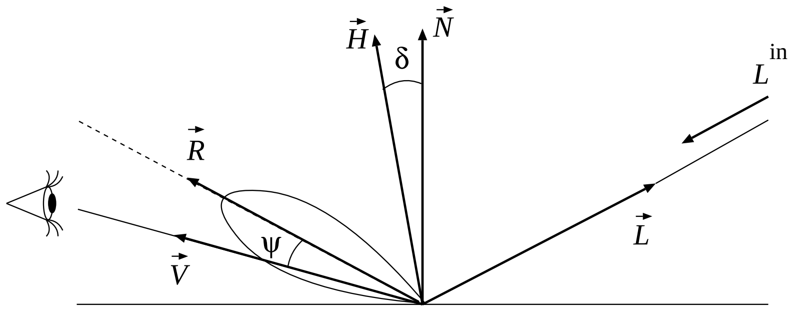


Figure 3: Specular reflection

Specular surfaces reflect most of the incoming light around the ideal reflection direction, thus the BRDF should be maximum at this direction and should decrease sharply (figure 3). The **Phong BRDF** [8] was the first model proposed for specular materials, which uses the following function for this purpose:

$$f_{r,\text{Phong}}(\vec{L},\vec{V}) = k_s \cdot \frac{\cos^n \psi}{\cos\theta'} = k_s \cdot \frac{(\vec{R}\cdot\vec{V})^n}{(\vec{N}\cdot\vec{L})}, \tag{4}$$

where $\vec{R}$ is the mirror direction of $\vec{L}$ onto the surface normal, and $\vec{R}$, $\vec{L}$, $\vec{N}$ and $\vec{V}$ are supposed to be unit vectors.

Blinn [1] proposed an alternative to this BRDF, which has the following form:

$$f_{r,\text{Blinn}}(\vec{N},\vec{H}) = k_s \cdot \frac{\cos^n \delta}{\cos\theta'} = k_s \cdot \frac{(\vec{N}\cdot\vec{H})^n}{(\vec{N}\cdot\vec{L})}, \tag{5}$$

where $\vec{H}$ is the halfway unit vector between $\vec{L}$ and $\vec{V}$ defined as

$$\vec{H} = \frac{\vec{L}+\vec{V}}{|\vec{L}+\vec{V}|}. \tag{6}$$

Unlike the Phong and the Blinn models, which are only empirical constructions, the Cook-Torrance BRDF [3] is derived from physical laws and from the statistical analysis of the microfacet structure of the surface and results in the following formula:

$$f_{r,\text{Cook}}(\vec{N},\vec{H},\vec{L},\vec{V}) = \frac{P(\vec{H})\cdot F(\lambda,\vec{H}\cdot\vec{L})}{4(\vec{N}\cdot\vec{L})(\vec{N}\cdot\vec{V})} \cdot \min\{2\cdot\frac{(\vec{N}\cdot\vec{H})\cdot(\vec{N}\cdot\vec{V})}{(\vec{V}\cdot\vec{H})}, 2\cdot\frac{(\vec{N}\cdot\vec{H})\cdot(\vec{N}\cdot\vec{L})}{(\vec{L}\cdot\vec{H})}, 1\}, \tag{7}$$

where P is the probability density of the microfacet normals, and F is the wavelength (λ) dependent Fresnel function computed from the refraction index and the extinction coefficient of the material [10].

Examining these BRDF models, we can come to the conclusion that the reflected radiance formulae are relatively simple functions of dot products (i.e. cosine angles) of the pairs of unit vectors, including, for example, the light vector $\vec{L}$, halfway vector $\vec{H}$, normal vector $\vec{N}$, etc.

For example, using the Blinn BRDF for specular reflection and also allowing diffuse reflection, the reflected radiance is

$$I^{out}(\vec{x},\vec{V}) = \sum_l I_l^{in}(\vec{x},\vec{L}_l)\cdot\left[k_s(\vec{N}\cdot\vec{H}_l)^n + k_d(\vec{N}\cdot\vec{L}_l)\right]. \tag{8}$$

3 Phong shading

Since the radiance values are needed for each pixel, which, in turn, require the rendering equation to be solved for the visible surface. The rendering equation, even in its simplified form, contains a lot of complex operations, including the computation of the vectors, their normalization and the evaluation of the output radiance, which makes the process rather resource demanding.

The speed of rendering could be significantly increased if it were possible to carry out the expensive computations just for a few points or pixels, and the rest could be approximated from these representative points by much simpler expressions. One way of obtaining this is the tessellation of the original surfaces to polygon meshes and using the vertices of the polygons as representative points. These techniques are based on linear (or in the extreme case, constant) interpolation requiring a value of the function to be approximated at the representative points, which leads to the incremental concept. These methods are particularly efficient if the geometric properties can also be determined in a similar way, connecting incremental shading to the incremental visibility calculations of polygon mesh models.

In this paper only triangle mesh models are considered, thus the geometry should be approximated by a triangle mesh before the algorithms can be used. It is assumed that the geometry has been transformed to the screen coordinate system suitable for visibility calculations and projection. In the screen coordinate system the X, Y coordinates of a point are equal to the corresponding coordinates of that pixel in which this point can be seen, and the Z coordinate increases with the distance from the viewer, thus it is the basis of visibility calculations (figure 4). Note, on the other hand, that the vectors used by the rendering equation are not transformed, because the viewing transformation is not angle preserving thus it may distort the angles between them.

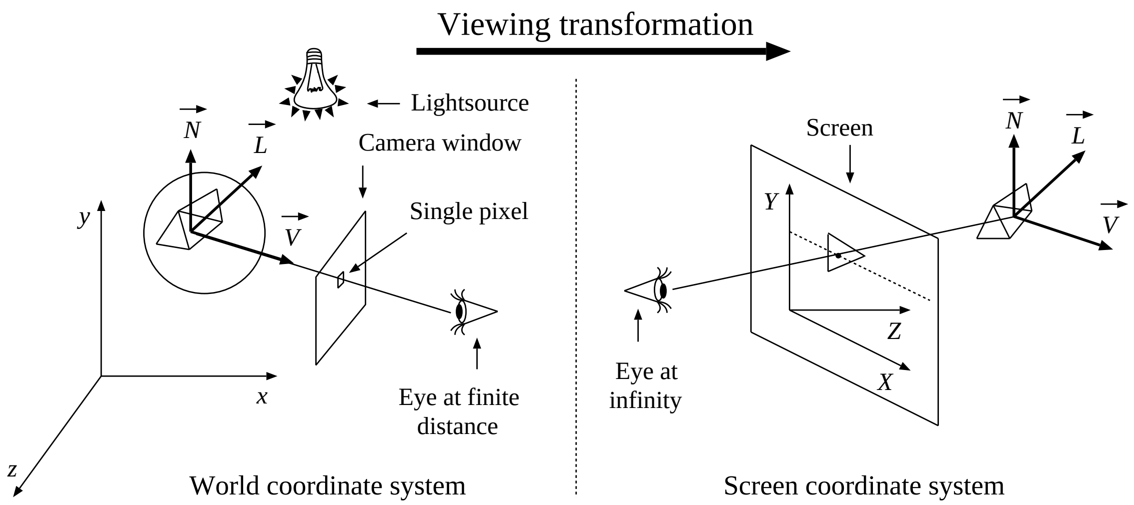


Figure 4: Transformation to the screen coordinate system

As mentioned, interpolation can be used to speed up the rendering of the triangle mesh, where the expensive computations take place just at the vertices and the data of the internal points are interpolated. A simple interpolation scheme would compute the color and linearly interpolate it inside the triangle (Gouraud shading [4]). However, specular reflections may introduce strong non-linearity, thus linear interpolation can introduce severe artifacts (left of figure 5).

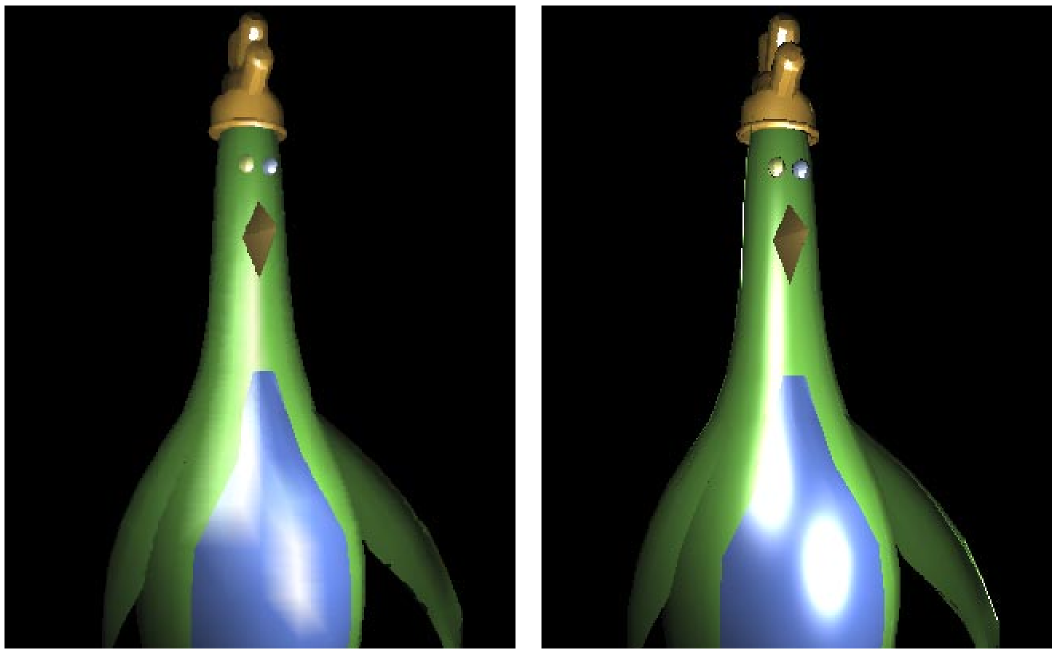


Figure 5: Comparison of linear interpolation i.e. Gouraud shading (left) and non-linear interpolation by Phong shading (right).

The artifacts of Gouraud shading can be eliminated by a non-linear interpolation called Phong shading [8] (right of figure 5). In Phong shading, vectors used by the BRDFs and the rendering equation are interpolated from the real vectors at the vertices of the approximating polygon; the interpolated vectors are normalized and the rendering equation is evaluated for each pixel. Originally, the interpolating function is linear. For example, the normal vector of a pixel (X, Y) is

$$\vec{N}(X,Y) = a_1(X,Y) \cdot \vec{N}_1 + a_2(X,Y) \cdot \vec{N}_2 + a_3(X,Y) \cdot \vec{N}_3 \tag{9}$$

where $a_i(X,Y) = a_{ix}X + a_{iy}Y + a_{i0}$ $(i = 1, 2, 3)$ is a linear weighting function and $\vec{N}_i$ is the normal vector at vertex i. The interpolation criterion requires that $a_i(X,Y) = 1$ at vertex i and 0 in the other two vertices. From this criterion, the parameters (a_{ix}, a_{iy}, a_{i0}) of each weighting function can be determined.

It is usually simpler to replace the two-variate interpolation scheme by two one-variate schemes, one running on the edges of the triangle and the other running inside horizontal spans called scan-lines (figure 6).

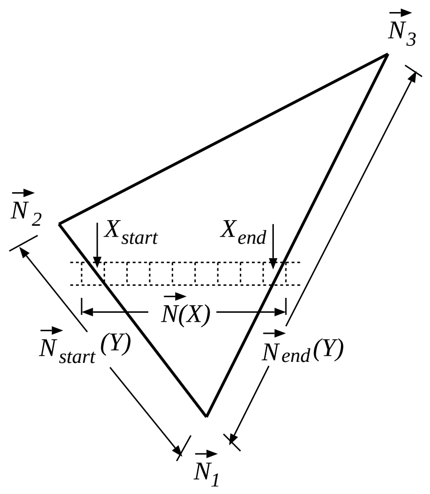


Figure 6: Decomposition of the two-variate interpolation inside the triangle to one-variate interpolations

Thus it is enough to consider a one-variate interpolation either on the edge of the triangle or inside the scan-line. Let us introduce a generic interpolation parameter t, which can be obtained from the pixel coordinates. If the interpolation in a scan-line is considered, then

$$t = \frac{X - X_{start}}{X_{end} - X_{start}}$$

where t is a running variable along the pixels in each scan-line, $t = 0$ at the start of the scan-line and $t = 1$ at the end of it.

Phong shading implies that at every pixel the vectors being involved in the BRDF are interpolated, normalized and their dot product is computed, then substituted into the simplified rendering equation. To be general, let us consider the interpolation of two vectors $\vec{u}$ and $\vec{v}$ that can be any from the light vector $\vec{L}$, viewing vector $\vec{V}$, normal vector $\vec{N}$, etc.

The generic formulae of the computation of the cosine of the angle between $\vec{u}(t)$ and $\vec{v}(t)$ are then:

$$\begin{aligned}
\vec{u}(t) &= (1-t)\vec{u}_{start} + t\vec{u}_{end}, \\
\vec{u}^0 &= \frac{\vec{u}(t)}{|\vec{u}(t)|}, \\
\vec{v}(t) &= (1-t)\vec{v}_{start} + t\vec{v}_{end}, \\
\vec{v}^0 &= \frac{\vec{v}(t)}{|\vec{v}(t)|}, \\
\cos\theta &= \vec{u}^0 \cdot \vec{v}^0
\end{aligned} \tag{10}$$

Note that in these operations the interpolation is always followed by a normalization, since dot products provide the cosine angle only if the vectors are unit vectors. The normalization, on the other hand, involves 3 multiplications, 2 additions, a square root and 3 divisions, which is rather expensive computationally.

Thus in this paper we propose a new interpolation scheme that can eliminate the expensive normalization operations and provide the cosine angle by a simple formula, which can even be implemented in hardware. The new interpolation works on the surface of unit spheres, that is why we call it spherical interpolation.

3.1 Spherical interpolation of a single vector

Suppose that we intend to interpolate between two unit vectors $\vec{u}_1$ and $\vec{u}_2$ in a way that the interpolant $\vec{u}(t)$ is moving uniformly between the two vectors and its length is always one. An appropriate interpolation method must generate the great arc between $\vec{u}_1$ and $\vec{u}_2$, and as can easily be shown, this great arc has the following form:

$$\vec{u}(t) = \frac{\sin(1-t)\gamma}{\sin\gamma} \cdot \vec{u}_1 + \frac{\sin t\gamma}{\sin\gamma} \cdot \vec{u}_2, \tag{11}$$

where $\cos\gamma = \vec{u}_1 \cdot \vec{u}_2$ (figure 7).

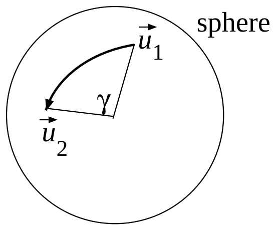


Figure 7: Interpolation of vectors on a unit sphere

In order to demonstrate that this really results in a uniform interpolation, the following equations must be proven for $\vec{u}(t)$:

$$|\vec{u}(t)| = 1, \quad \vec{u}_1 \cdot \vec{u}(t) = \cos t\gamma, \qquad \vec{u}_2 \cdot \vec{u}(t) = \cos(1-t)\gamma. \tag{12}$$

That is, the interpolated vector is really on the surface of the sphere, and the angle of rotation is a linear function of parameter t.

Let us first prove the second assertion (the third can be proven similarly):

$$\vec{u}_1 \cdot \vec{u}(t) = \frac{\sin(1-t)\gamma}{\sin\gamma} + \frac{\sin t\gamma}{\sin\gamma} \cdot \cos\gamma = \frac{\sin\gamma \cdot \cos t\gamma}{\sin\gamma} - \frac{\sin t\gamma \cdot \cos\gamma}{\sin\gamma} + \frac{\sin t\gamma}{\sin\gamma} \cdot \cos\gamma = \cos t\gamma. \tag{13}$$

Concerning the first assertion, i.e. the norm of the interpolated vector, we can use the definition of the norm and the previous results, thus we obtain:

$$|\vec{u}(t)|^2 = \vec{u}(t) \cdot \vec{u}(t) = \left(\frac{\sin(1-t)\gamma}{\sin\gamma} \cdot \vec{u}_1 + \frac{\sin t\gamma}{\sin\gamma} \cdot \vec{u}_2\right) \cdot \vec{u}(t) =$$

$$\frac{\sin(1-t)\gamma}{\sin\gamma} \cdot \cos t\gamma + \frac{\sin t\gamma}{\sin\gamma} \cdot \cos(1-t)\gamma = \frac{\sin((1-t)\gamma + t\gamma)}{\sin\gamma} = 1. \tag{14}$$

3.2 Spherical interpolation of a pair of vectors

Having discussed how vectors can be interpolated without modifying their length, we can start examining how the angle between two interpolated vectors can be determined. Let us assume that $\vec{u}(t)$ is interpolated from $\vec{u}_1$ to $\vec{u}_2$ while $\vec{v}(t)$ is interpolated from $\vec{v}_1$ to $\vec{v}_2$, and we are interested in $\cos\theta(t) = \vec{u}(t)\cdot\vec{v}(t)$ (left of figure 8).

One obvious possibility is to use the previous results separately for $\vec{u}(t)$ and $\vec{v}(t)$ and to compute the dot product for each t. However, we can realize that a similar interpolation can be obtained keeping one vector — say $\vec{v}_1$ — fixed and the other is rotated by the composition of its own transformation and the inverse of the transformation of the other vector (right of figure 8). It means that while $\vec{v}'(t) = \vec{v}_1$ is fixed, $\vec{u}'(t)$ is interpolated between $\vec{u}_1$ and $\vec{u}'_2$ which is obtained by rotating $\vec{u}_2$ by the inverse of the rotation from $\vec{v}_1$ to $\vec{v}_2$.

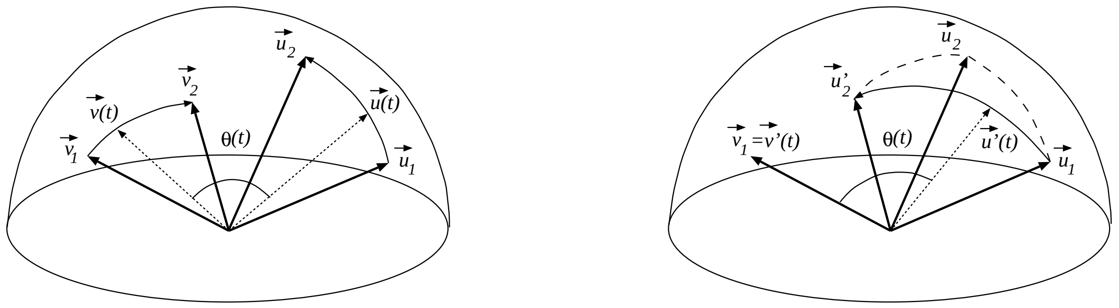


Figure 8: Interpolation of two vectors

The new end point $\vec{u}'_2$ can, for instance, be expressed by quaternion multiplications [10]. The unit quaternion that rotates $\vec{v}_1$ to $\vec{v}_2$ is

$$q = \left[\cos\frac{v}{2}, \sin\frac{v}{2}\cdot\frac{\vec{v}_1\times\vec{v}_2}{|\vec{v}_1\times\vec{v}_2|}\right] = \left[\cos\frac{v}{2}, \sin\frac{v}{2}\cdot\frac{\vec{v}_1\times\vec{v}_2}{\sin v}\right],$$

where v is the angle between $\vec{v}_1$ and $\vec{v}_2$. Applying the inverse of this quaternion to $\vec{u}_2$ we get:

$$[0,\vec{u}'_2] = q^{-1}\cdot[0,\vec{u}_2]\cdot q = \left[0, \vec{u}_2 - (\vec{v}_1\times\vec{v}_2)\times\vec{u}_2 + \frac{(\vec{v}_1\times\vec{v}_2)\times((\vec{v}_1\times\vec{v}_2)\times\vec{u}_2)}{1+\cos v}\right].$$

Vector $\vec{u}'(t)$ is obtained by spherical interpolation from $\vec{u}_1$ to $\vec{u}'_2$, thus the angle between this vector and the fixed $\vec{v}_1$ is:

$$\cos\theta(t) = \vec{u}'(t)\cdot\vec{v}_1 = \frac{\sin(1-t)\gamma}{\sin\gamma}\cdot(\vec{u}_1\cdot\vec{v}_1) + \frac{\sin t\gamma}{\sin\gamma}\cdot(\vec{u}'_2\cdot\vec{v}_1) = \frac{\sin(1-t)\gamma}{\sin\gamma}\cdot\cos\theta_1 + \frac{\sin t\gamma}{\sin\gamma}\cdot\cos\theta_2 =$$

$$\cos t\gamma\cdot\cos\theta_1 + \sin t\gamma\cdot\frac{\cos\theta_1\cos\gamma - \cos\theta_2}{\sin\gamma}, \tag{15}$$

where $\cos\gamma = \vec{u}_1\cdot\vec{u}'_2$. Note that this interpolation does not give exactly the same values as interpolating the two vectors separately. Since the interpolation is only used for approximating the vectors, this is as acceptable as the separate spherical interpolation.

Let us express $\cos\theta_1$ and $(\cos\theta_1\cos\gamma - \cos\theta_2)/\sin\gamma$ by A and α in the following way:

$$A\cdot\cos\alpha = \cos\theta_1, \qquad A\cdot\sin\alpha = \frac{\cos\theta_2 - \cos\theta_1\cos\gamma}{\sin\gamma}. \tag{16}$$

Substituting these into equation 15, we obtain:

$$\cos\theta(t) = A\cdot(\cos t\gamma\cdot\cos\alpha + \sin t\gamma\cdot\sin\alpha) = A\cdot\cos(t\gamma - \alpha). \tag{17}$$

Let us realize that the complex sequence of operations including the spherical interpolations of two vectors and the computation of their dot product have been traced back to the calculation of a single cosine value. Based on this simplification, even the hardware realization of Phong shading becomes possible, as detailed in the next section for the Blinn illumination model. Similar hardware architectures can be developed for other BRDFs as well.

4 Interpolation and Blinn BRDF calculation by hardware

Subsituting equation 17 into the reflected radiance formula (equation 2), and assuming Blinn type BRDF and a single lightsource, we get:

$$I^{out}(\vec{x}, \vec{V}) = I^{in}(\vec{x}, \vec{L}) \cdot k_s \cdot \cos^n \delta = I^{in}(\vec{x}, \vec{L}) \cdot k_s \cdot (\vec{N} \cdot \vec{H})^n = I^{in}(\vec{x}, \vec{L}) \cdot k_s \cdot A^n \cdot \cos^n(t\gamma - \alpha). \quad (18)$$

Factor $I^{in}(\vec{x}, \vec{L}) \cdot k_s \cdot A^n = C$ is constant in the scan-line, thus only $\cos^n(t\gamma - \alpha)$ should be computed pixel by pixel and the result should be multiplied by this constant C.

The computation of $\cos^n(t\gamma - \alpha)$ consists of three elementary operations: the calculation of $t(X) \cdot \gamma - \alpha$ from the actual pixel coordinate X, the application of the cosine function, and finally exponentiation with n. These operations are too complex to allow direct hardware implementation thus further simplifications are needed.

The $t(X) \cdot \gamma - \alpha$ term is a linear function, thus it is a primary candidate for the application of the incremental concept [9]. The cosine and the exponentiation are a little bit more difficult. In fact, we could use two tables of tabulated function values for this purpose. This would work well for the cosine function since it is relatively flat, but the accurate representation of the exponentiation would require a large table, which should be reinitialized each time when n changes (note that the practical values of n can range from 2 to a few hundred). Thus a different approximation strategy is used.

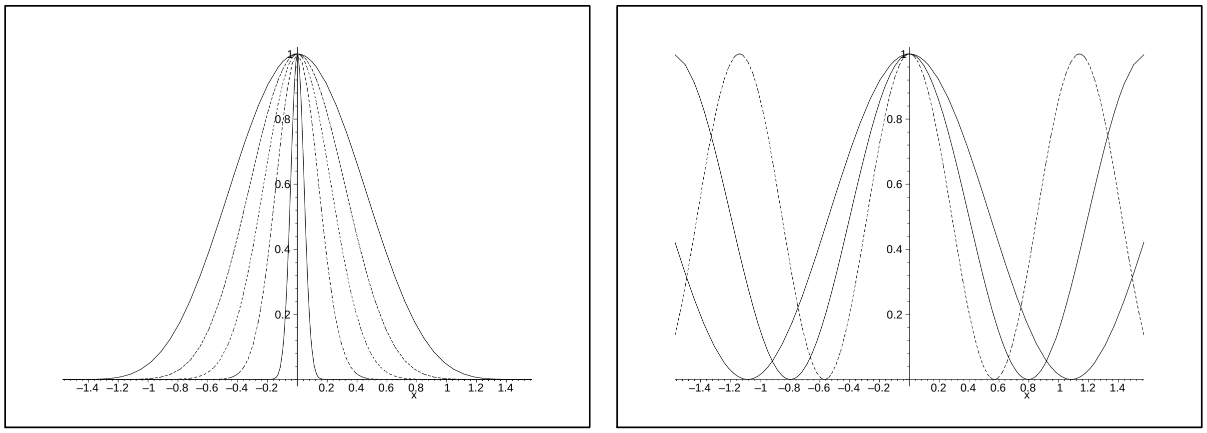


Figure 9: The bell shapes of $\cos^n x$ $n = 5, 10, 20, 50, 500$ (left) and of $\cos^2 ax$ (right) for $a = 1.45, 1.98, 2.76$

Looking at the bell shapes of the $\cos^n x$ functions for different n values (figure 9), we can realize that these functions are approximately similar and can be transformed to each other by properly scaling the abscissa. For example, we can use the horizontally scaled versions of $\cos^2 x$, i.e. $\cos^2 ax$ to approximate $\cos^n x$ for arbitrary n. The reason of using the square of the cosine function is that n is greater than 2 in practical cases and the square cosine already has the bell shape caused by the inflection point. Thus our formal approximation is

$$\cos^n x \approx \cos^2 ax \qquad \text{if } -\frac{\pi}{2a} \leq x \leq \frac{\pi}{2a}, \quad (19)$$

and zero otherwise. Parameter a should be found to maximize the accuracy for all possible x values. We can, for example, require the weighted integrals of the two functions to be equal in order to obtain the parameter a. Note that the approximation is exact for $x = 0$ regardless of the parameter a, that is the zero point of all cosines are fixed. This consideration makes it worth emphasizing the accuracy of larger x values when a is determined. Let us use the $\sin x$ weighting function, thus the criterion for determining a is

$$\int_0^{\frac{\pi}{2}} \cos^n x \cdot \sin x \, dx = \int_0^{\frac{\pi}{2a}} \cos^2 ax \cdot \sin x \, dx. \tag{20}$$

Expressing these integrals in closed form, we get:

$$\frac{1}{n+1} = \frac{2a^2 \cdot (1 - \cos\frac{\pi}{2a}) - 1}{4a^2 - 1}.$$

This equation needs to be solved once for a set of values and the results can be stored in a table. A few representative results are shown in table 1. The quality of the approximation is quite good as demonstrated by figure 10.

n	2	5	10	20	50	100	500
a	1.0000	1.4502	1.9845	2.7582	4.3143	6.0791	13.5535

Table 1: Correspondence between n and a

Let us return to the computation of the reflected radiance. The $C \cdot \cos^n(t(X) \cdot \gamma - \alpha)$ has been simplified to the evaluation of

$$C \cdot \cos^2(a(t(X) \cdot \gamma - \alpha)) = C \cdot \cos^2 \xi(X),$$

where

$$\xi(X) = a \cdot \gamma \cdot \frac{X - X_{start}}{X_{end} - X_{start}} - a \cdot \alpha = s \cdot X + b.$$

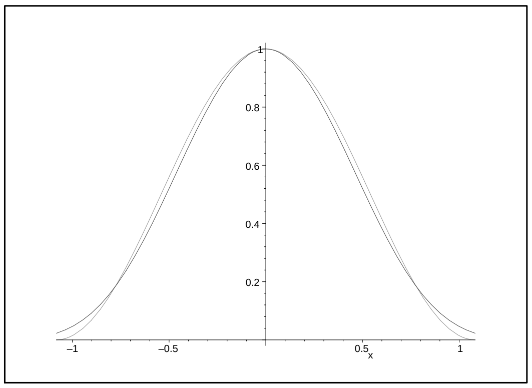
Since $\xi(X)$ is a linear function, it can conveniently be generated by the incremental concept. Its basic idea is that instead of computing ξ from X, it can be computed from the previous value, i.e. from $\xi(X-1)$. Recall that a complete scan-line is filled, that is when pixel X is shaded, the results of pixel $X - 1$ are already available. In our case:

$$\xi(X) = \xi(X-1) + s,$$

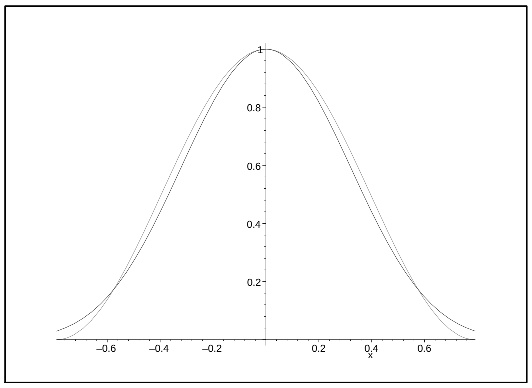
thus the new value of ξ requires just a single addition.

Having the ξ value generated, it should be input to the $\cos^2$ function that can be implemented as a read only memory. The number of address and data bits of this memory, i.e. the number and the length of the words are determined from the requirement of accurate representation. Figure 11 shows the original $\cos^2$ function together with its table representations for different address and data bit numbers. Note that using six bit address and data, which means that our memory stores $2^6 \times 6 = 384$ bits, provides sufficient accuracy.

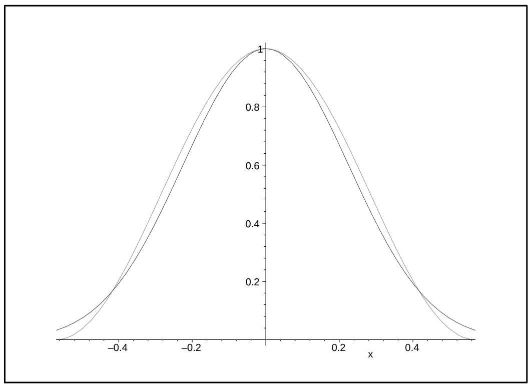
The complete hardware is shown in figure 12. The hardware has two parts, one for the diffuse and one for the specular components, and each part has two stages. In the specular part, the first stage is a linear interpolator, which provides the $\cos^2$ table with angle ξ, according to $\xi(x+1) = \xi(x) + b$. Since it has a register at its output, this stage can operate in parallel with the multiplier unit. Assuming white light sources and wavelength independent specular factor k_s, a single linear interpolator can be used for all color channels. However, the diffuse part, which is responsible for coloring, requires 3 channels. The cosine, and square cosine functions can be implemented by ROMs. At the initial phase, for each scan line, the constant parameters must be loaded into hardware. Then, for each step, the hardware will generate R, G, B values.



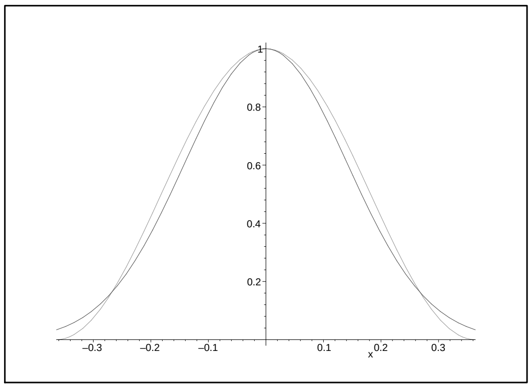
$n = 5, a = 1.4502$



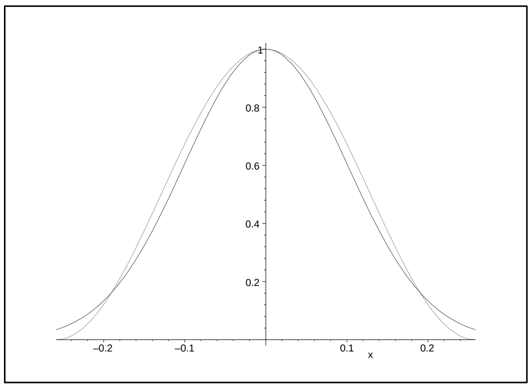
$n = 10, a = 1.9745$



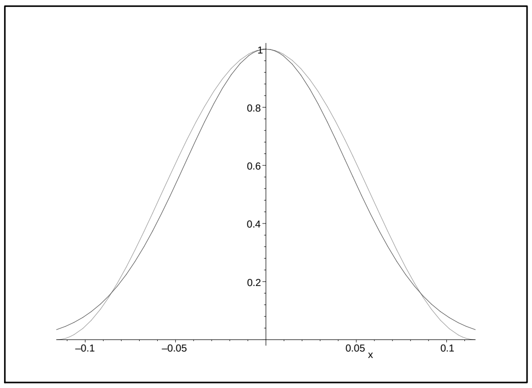
$n = 20, a = 2.7582$



$n = 50, a = 4.3143$

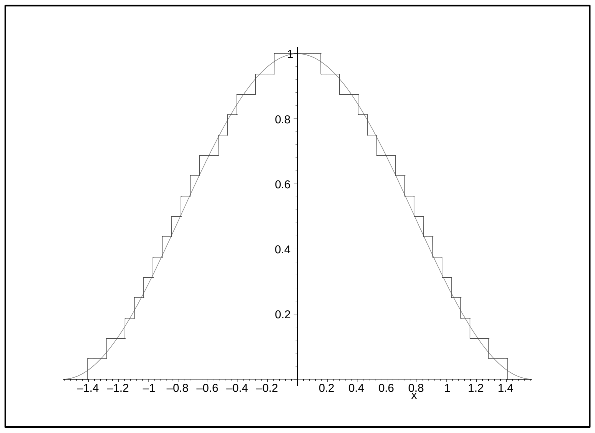


$n = 100, a = 6.0791$



$n = 500, a = 13.5535$

Figure 10: Approximation of $\cos^n x$ by $\cos^2 ax$



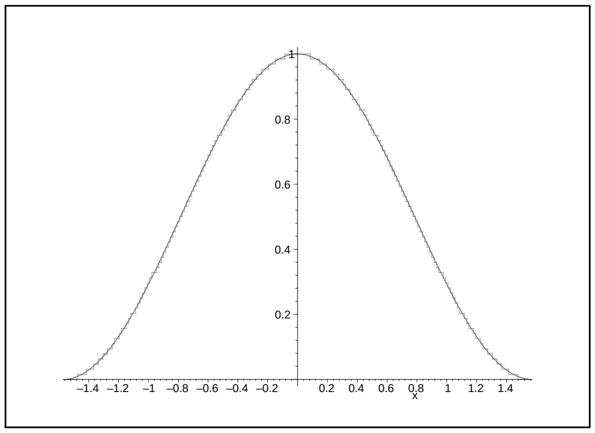


Figure 11: Quantization errors of the $\cos^2 x$ function for 4 and 6 address/data bits

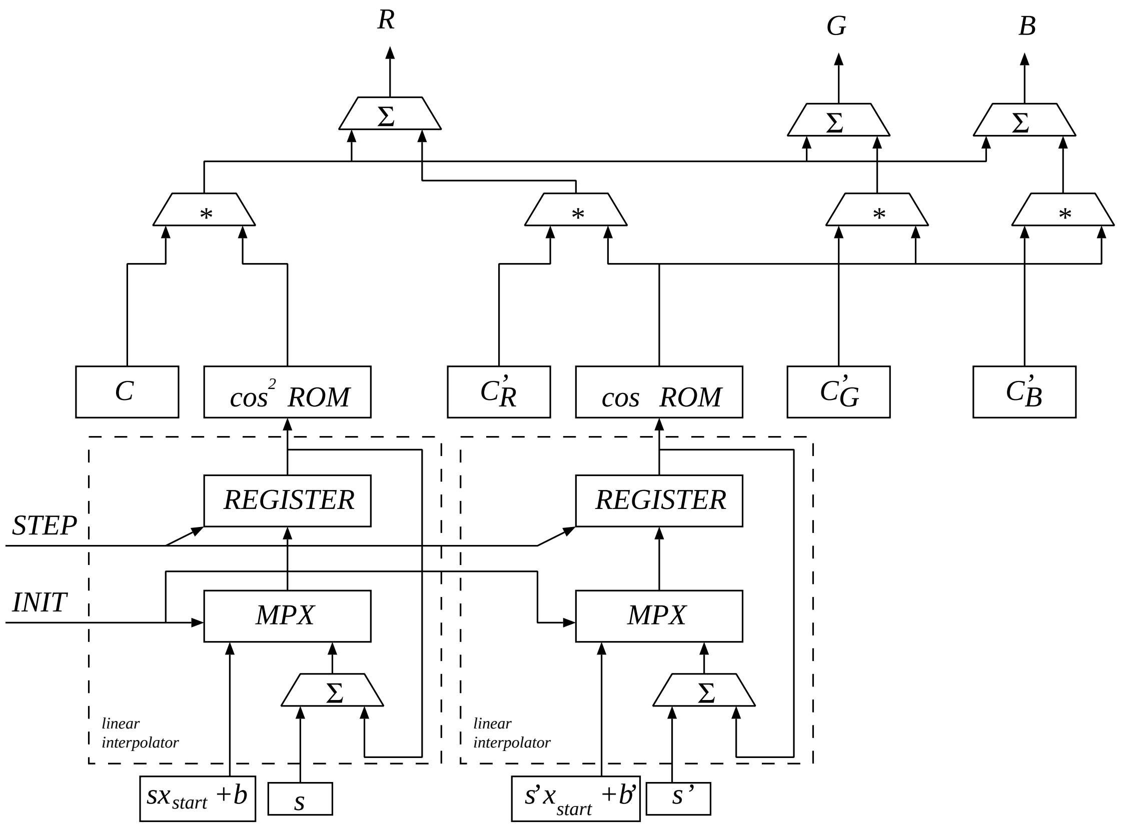


Figure 12: Phong shading hardware

The VHDL specification is straightforward for the multiplicators and for the ROM. Here, as an example, the behavioral model of linear interpolator is given:

```
USE work.phong_pack.all;

ENTITY line_interpolator IS
  GENERIC (t_mpx : time := 5 ns; t_add : time := 10 ns; t_reg : time := 5 ns );
  PORT ( sxsb,s : IN bit_v_12; ra : OUT bit_v_12; init, step : IN bit;);
END line_interpolator;

ARCHITECTURE behaviour OF line_interpolator IS
SIGNAL add_out, mpx_out, reg_out: bit_v_12;
BEGIN
    add_out <= s + reg_out AFTER t_add;
    reg_out <= mpx_out after t_reg WHEN step’EVENT AND step = ’1’;
    ra <= reg_out(11 DOWNTO 6);
mux_proc: PROCESS(sxsb,add_out,init)
    BEGIN
           IF init = ’1’ THEN mpx_out <= sxsb AFTER t_mpx ;
                          ELSE mpx_out <= add_out AFTER t_mpx;
           END IF;
    END PROCESS;
END behaviour;
```

5 Interpolation on the triangle

So far we have discussed the interpolation in a scan-line. A complete triangle is rendered by generating those scan-lines which cover this triangle one after the other. For each scan-line, the start and end points should be identified and the interpolation parameters need to be initialized, then the scan-line interpolation can be initiated.

Let us consider a horizontal sided triangle. If the triangle were not horizontal sided, then it could be divided to two horizontal sided parts. In this section we will consider only the lower horizontal sided triangle, the upper part can be handled similarly. Image space triangle and horizontal sided triangle are shown in Figure 13.

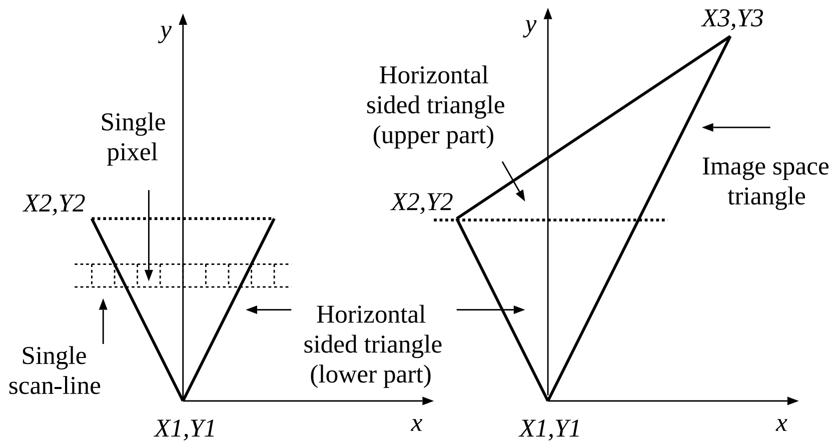


Figure 13: Image space triangle and horizontal sided triangle

In order to initialize the scan-line interpolation, the vectors used by the shading formula are needed for the start and end points of the scan-line. These vectors can be provided by spherical interpolations running simultaneously at the left and right edges of the triangle.

6 Simulation results

The proposed algorithm has been implemented in C++ and tested as a software. First the difference between the simultaneous vector interpolation and the method of keeping one vector fixed while rotating the other vector by the composition of the two rotations was investigated, and we concluded that the results are visually indistinguishable. Then the quality of the $\cos^n x = \cos^2 ax$ approximation has been studied.

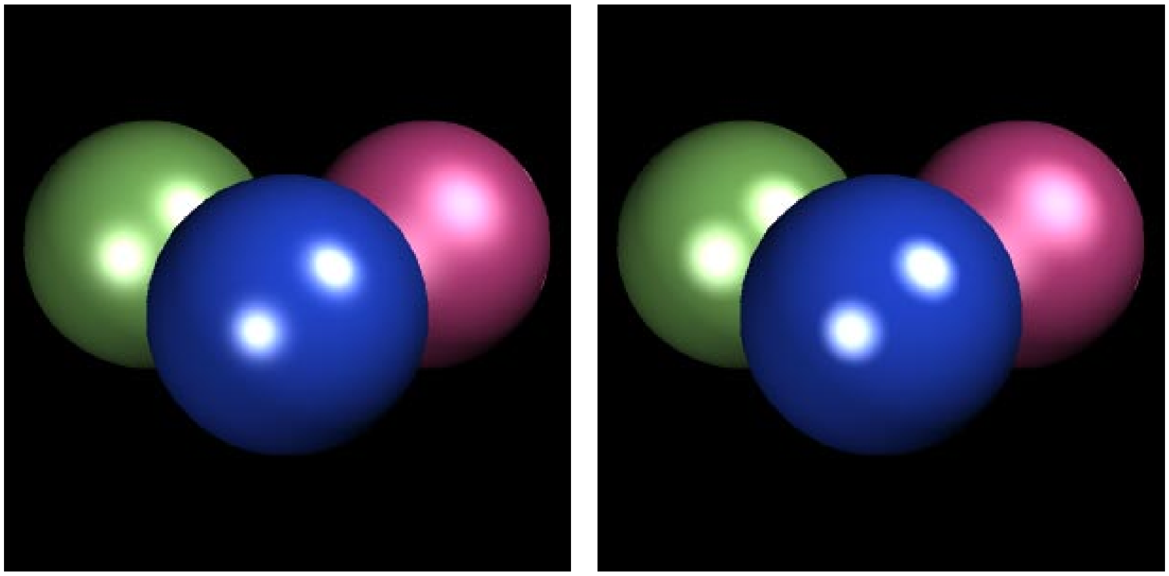


Figure 14: Evaluation of the visual accuracy $\cos^n x = \cos^2 ax$ approximation. Rendering the left and right images, we used the $\cos^n x$ and $\cos^2 ax$ functions, respectively. The shine (n) parameters of the spheres are 5, 10 and 20.

Note that the halos in the left image of figure 14 obtained with the $\cos^n$ function are slightly bigger but the centers are smaller. This is also obvious looking at the bell shapes of figure 10 since the $\cos^2 ax$ is zero if $x = \pi/2a$ while $\cos^n x$ only converges to zero, while having the same integral.

Finally, the necessary precision, i.e. the number of bits, was determined. Since the $\cos^2$ function is implemented as a memory, it is the most sensitive to the word length. Figure 15 shows the results assuming 4 and 6 bit precision, respectively, where a rather coarse surface tessellation was used to emphasize the possible interpolation errors. Note that with 4 bits the quantization errors are visible in the form of concentric halo circles around the highlight spots. However, these circles disappear when 6 bit precision is used. This also conforms with the quantization error functions of figure 11.

Finally, the hardware was specified in VHDL and simulated in ModelTech environment. The delay times are according to XILINX XCV300-6 FPGA. The timing diagram of the operation is shown by figure 17. In this figure we can follow the overlapped operation of the two stages while the cycle time of the “step” signal is 60 nanoseconds.

7 Conclusions

This paper proposed a different interpolation strategy for the cosine angles in Phong shading. This strategy simplifies the linear interpolation, normalization and dot product of a pair of 3D vectors to an addition, a cosine evaluation and a multiplication. Furthermore, when the reflected radiance is computed for specular surfaces the exponentiation of this cosine is replaced by the horizontal scaling of the square

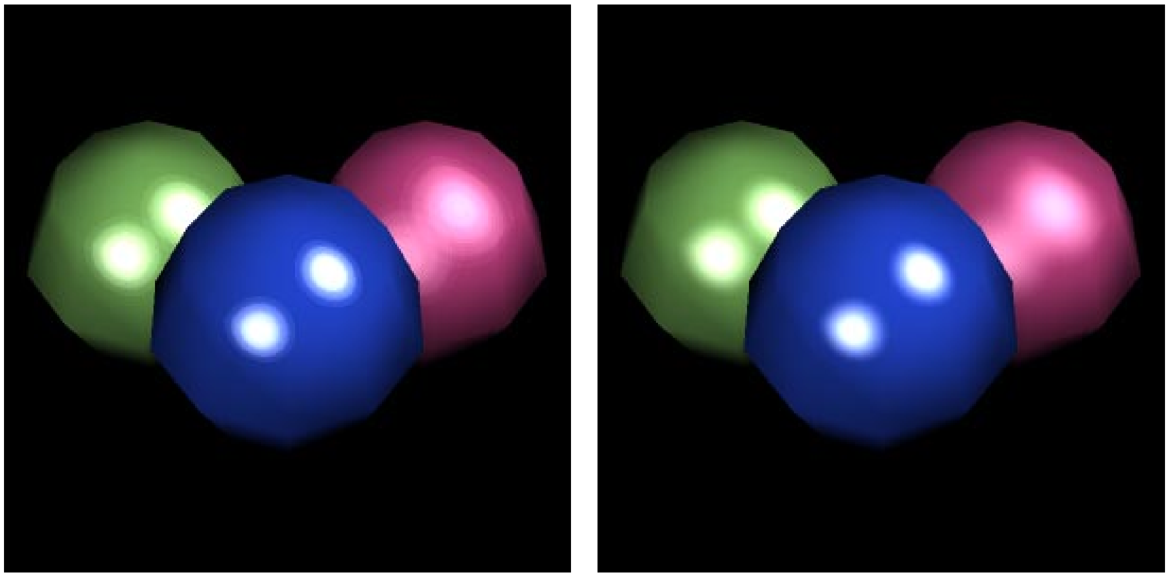


Figure 15: Rendering of coarsely tesselated spheres with the proposed method with 4 bit precision (left) and 6 bit precision (right)

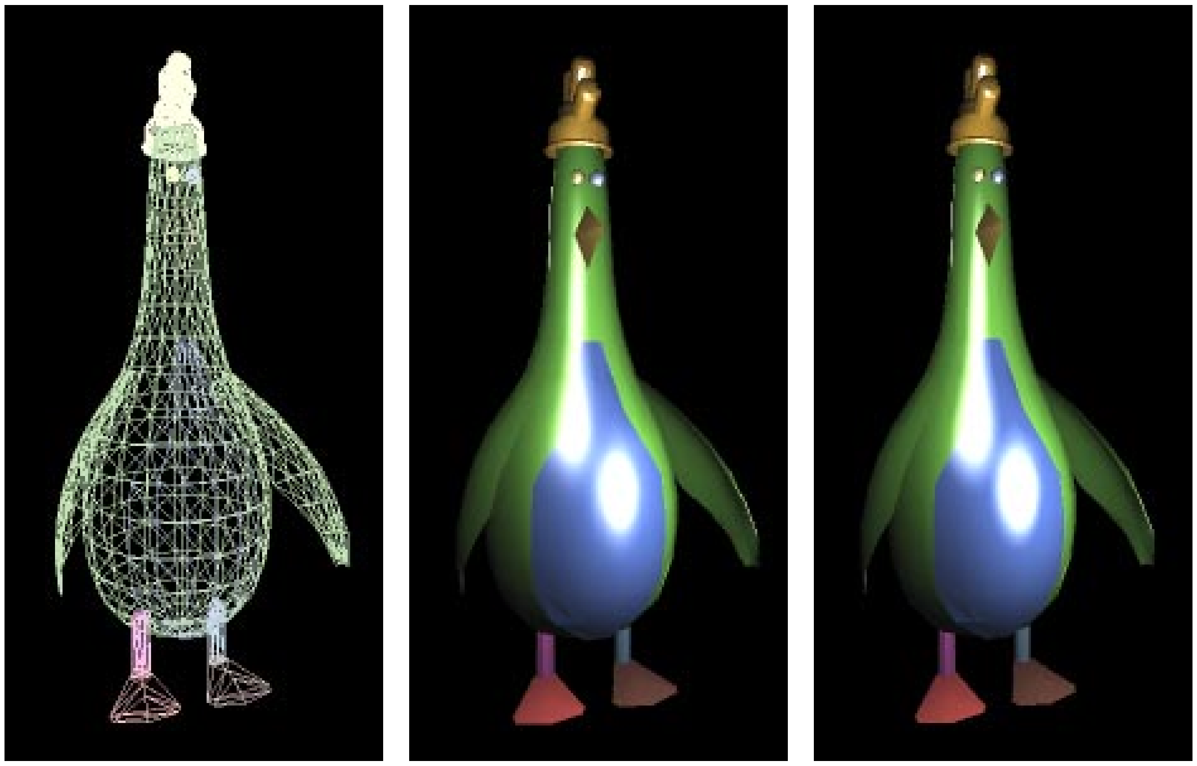


Figure 16: The mesh of a chicken (left) and its image rendered by classical Phong shading (middle) and by the proposed method (right)

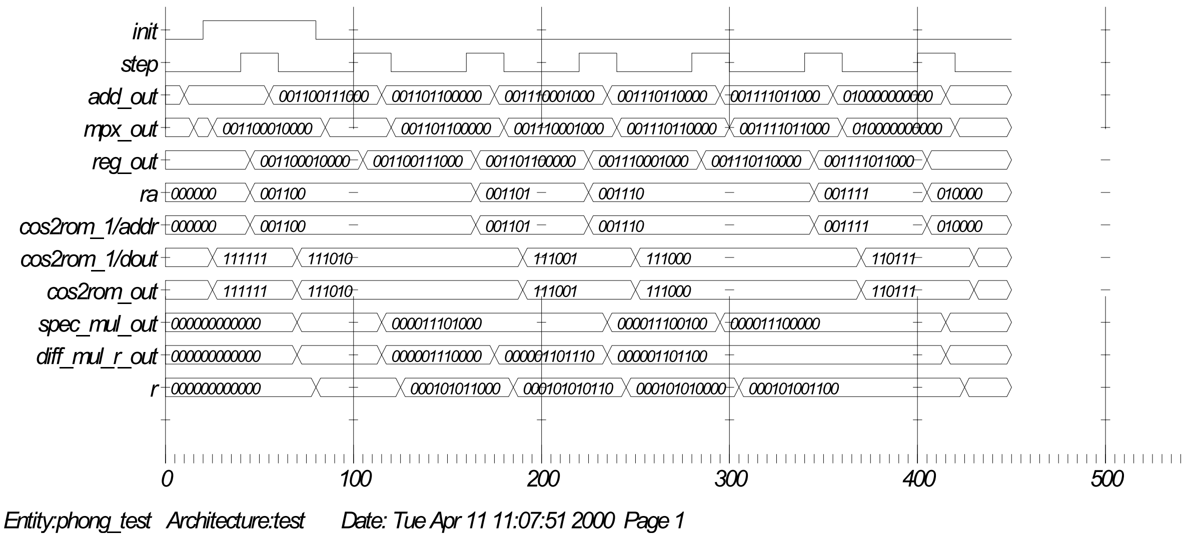


Figure 17: Timing diagram of the hardware

cosine function. This replacement can significantly reduce the size of the hardware lookup tables and a single small table (of a few hundred bits) can be used for more or less shiny surfaces. The algorithm has also been transformed to a hardware design that has been simulated in VHDL. Considering the components that are easily available on the market, the realization of the proposed hardware could generate a shaded pixel in every 60 nanoseconds. Even if the screen has about 1000×1000 resolution, the complete image can be redrawn 16 times per second which provides the illusion of continuous motion.

References

[1] J. F. Blinn. Models of light reflections for computer synthetized pictures. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, pages 192–198, 1977.

[2] U. Claussen. On reducing the Phong shading method. *Computer & Graphics*, pages 73–81, 1990.

[3] R. Cook and K. Torrance. A reflectance model for computer graphics. *Computer Graphics*, 15(3), 1981.

[4] H. Gouraud. Computer display of curved surfaces. *ACM Transactions on Computers*, C-20(6):623–629, 1971.

[5] J. T. Kajiya. The rendering equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, pages 143–150, 1986.

[6] A. M. Kuijk and E. H. Blake. Faster Phong shading via angular interpolation. *Computer Graphics Forum*, pages 315–324, 1989.

[7] M. Minnaert. The reciprocity principle in lunar photometry. *Astrophysical Journal*, 93:403–410, 1941.

[8] B. T. Phong. Illumination for computer generated images. *Communications of the ACM*, 18:311–317, 1975.

[9] L. Szirmay-Kalos and G. Márton. On hardware implementation of scan-conversion algorithms. In *8th Symp. on Microcomputer Appl.*, Budapest, Hungary, 1994.

[10] L. Szirmay-Kalos (editor). *Theory of Three Dimensional Computer Graphics.* Akadémia Kiadó, Budapest, 1995. http://www.iit.bme.hu/˜szirmay.