Consistent Normal Interpolation

Article *in* ACM Transactions on Graphics · December 2010

DOI: 10.1145/1866158.1866168 · Source: DBLP

3 authors:

Alexander Reshetov
NVIDIA
23 PUBLICATIONS **517** CITATIONS
SEE PROFILE

Alexei Soupikov
Intel
17 PUBLICATIONS **475** CITATIONS
SEE PROFILE

William R. Mark
Startup - stealth
47 PUBLICATIONS **3,172** CITATIONS
SEE PROFILE

Consistent Normal Interpolation

Alexander Reshetov Alexei Soupikov William R. Mark

Intel

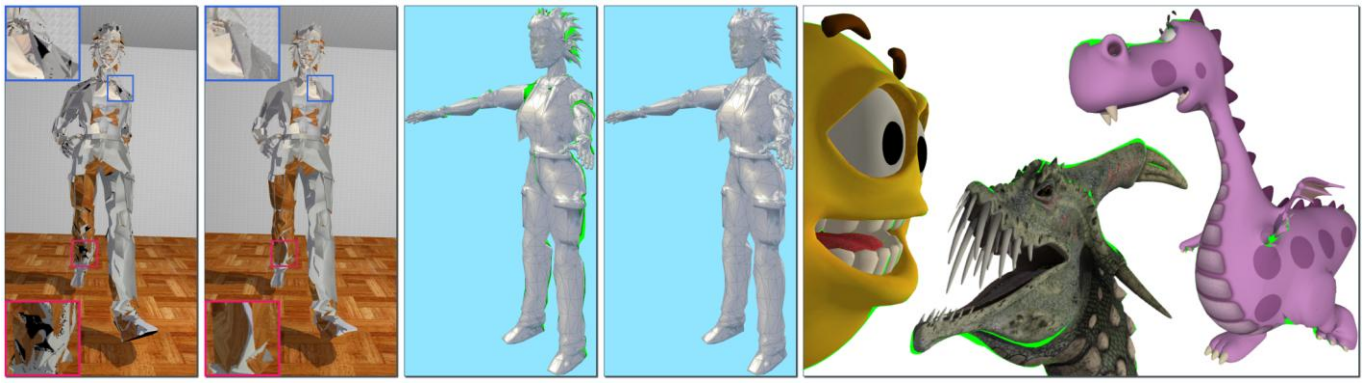

Figure 1. Comparison of consistent normal interpolation with Phong normal interpolation. Reflected rays computed using Phong normals can incorrectly reflect inside the model (black pixels in **a**), but when consistent normal are used (**b**), this artifact is eliminated. Similar problems occur with environment mapping (**c** and **d**), where green pixels show rays reflected inside the model. For highly tessellated models, reflection artifacts may occur at grazing angles or silhouettes (**e**, inward-reflected rays shown as green). Images **a**, **b**, and **e** are ray-traced. Environment lighting is used for images **c** and **d**.

Abstract

Rendering a polygonal surface with Phong normal interpolation allows shading to appear as it would for a true curved surface while maintaining the efficiency and simplicity of coarse polygonal geometry. However, this approximation fails in certain situations, especially for grazing viewing directions. Well-known problems include physically impossible reflections and implausible illumination. Some of these artifacts can be mitigated through special-case processing, although no universal or generally accepted approaches are available. In particular, all known solutions that guarantee that reflected rays will always point outward from the surface also create discontinuities in the reflection ray direction.

We present a simple modification of Phong normal interpolation that allows physically plausible reflections and creates an appearance of a smooth surface. We introduce an additional scalar parameter that characterizes the deviation between per-vertex normals and per face normals and use it to adjust linearly interpolated normals. The proposed technique eliminates perceptually objectionable artifacts caused by inconsistencies between the shading and geometric normals while retaining most of the practical advantages and simplicity of the original Phong formulation.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: Phong normal interpolation, reflection model, rendering

e-mail: {alexander.reshetov, alexei.soupikov, william.r.mark}@intel.com

1 Introduction

When a curved surface is approximated by a polygonal surface the geometric normals become discontinuous at each polygonal edge. If these geometric normals are used directly for shading, the result is a faceted appearance. Phong showed that this faceted appearance could be eliminated by computing a new "shading normal" vector and using it for shading [Phong 1975]. This Phong normal vector is different from the geometric normal and is continuously interpolated across the surface from values stored or computed at vertices. Phong normals can be used wherever surface normals are required: for computing reflected ray directions, for environment lighting, or directly for shading. However, the fact that the shading normal is different from the geometric normal can cause problems in certain situations [Woo et al. 1996].

In particular, if the shading normal is used to compute the direction of reflected rays, the reflected rays can point into the surface as shown in Figure 2. This problem was first described by Snyder & Barr [1987]. Figure 1a illustrates the artifacts that are produced when ray-tracing a completely reflective polygonal object using Phong normals. Figure 1c shows that the problem affects environment lighting as well; reflections into the model are artificially highlighted in green. The problems are not limited to coarsely tessellated models, as shown in Figure 1e.

Interpolated normals are also used in local shading computations [Walia & Singh 2006], typically by taking the dot product of the interpolated normal with some other vector. The dot product is then used as an argument of a power function or an exponent [Schwenk 2009]. For example, the Blinn-Phong reflection model [Blinn 1977] requires a computation of the dot product of the surface normal with the halfway vector (the halfway vector is defined as a normalized sum of the eye vector and the light vector). A negative dot product implies that the light source is behind the shaded polygon. With Phong normal interpolation, it is possible for this dot product to be negative across part of a polygon even if both the light source and the eye are on the front side of the polygon. This result is physically incorrect in a noticeable way,

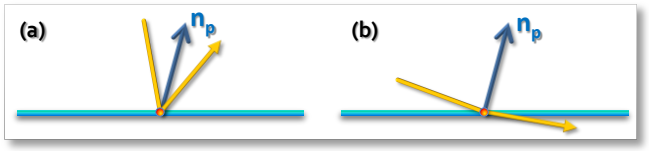

Figure 2: If Phong normals $\mathbf{n}_p$ are used to compute reflections by applying the laws of reflection, the reflected ray will point into the surface at certain grazing angles (b).

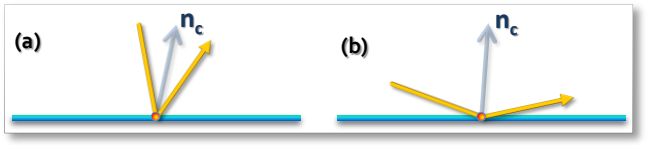

Figure 3: Our algorithm guarantees that all rays reflect outward from the surface, by using the laws of reflection with the new consistent normal $\mathbf{n}_c$. This guarantee is achieved by making $\mathbf{n}_c$ depend on the direction of the incoming ray.

since all visible points should receive some illumination from any light source that is not occluded. Figure 13 illustrates this situation for a light source placed at the camera position.

Many variations of the original Phong interpolation approach have been proposed. Instead of using linear normal interpolation, Overveld & Wyvill [1997] proposed a quadratic normal vector interpolation, designed to compensate for certain limitations of linear interpolation, in particular undersampling (similar to the example shown in Figure 8). Lee & Jen [2001] observed that quadratic interpolation could cause significant rendering artifacts and proposed using biquadratic interpolation to mitigate these defects. Lee & Jen's approach requires pre-computation of six vector coefficients per triangle.

Neither of these algorithms specifically addresses the problem we are trying to solve: resolving inconsistences between face normals and shading normals. Whenever an interpolated normal is different from the corresponding face normal, a reflection vector computed directly from the interpolated normal can point into the surface (as shown in Figure 2b). This problem can occur even for higher-order surfaces if a separate normal channel is used. One example of such a dual-channel surface representation is the recently proposed approximate Catmull-Clark surface [Loop & Schaefer 2008].

There are some well-known simple fixes that can be used to eliminate inward-reflected rays, but they introduce undesirable artifacts. Snyder & Barr [1987] proposed to use the face normal rather than the shading normal to compute reflections in cases where the shading normal produces an invalid result. This fix helps avoid inward-reflected rays, but creates discontinuities at transition points. Similarly, if clamping to zero is used to avoid negative dot products with shading normals, it results in flat illumination for those points (Figure 13b).

The technique that we present eliminates these discontinuities, which are caused by direct dependence on face normals. To achieve this, we consider face normals only at triangle vertices. More specifically, at each vertex we compute a maximum angle between the vertex normal and all adjacent face normals and interpolate this angle. At all points inside the polygon, we use the interpolated angle to conservatively characterize the discrepancy between the face normal and the standard interpolated normal, and to make appropriate adjustments to the computation of the reflection vector. As we will show, this approach guarantees that the reflected ray will always point outward from the surface, as sketched in Figure 3, and guarantees that reflection directions will vary continuously across the mesh. Angle interpolation has been considered previously by Kuijk and Blake [1989] and Barrera et al. [2004], but only as a standalone technique designed to simplify shading computations.

In our approach, the reflection vector is computed directly from the incoming direction vector, without explicitly computing the normal vector. But if desired, the new perturbed shading normal that is implicitly defined by our technique can be expressed explicitly. We refer to this normal vector as a *consistent normal,* and to our overall technique as *consistent normal interpolation*. It is important to understand that the consistent normal depends on the direction of the vector that is being reflected, unlike a standard Phong normal. Because this dependence is smoothly varying and the normal is not typically visualized directly, this tradeoff seems worthwhile.

Our approach can also be used to guarantee a positive dot product between the consistent shading normal and any chosen front-sided vector. For example, it can guarantee a positive dot product between the Blinn-Phong halfway vector and the consistent normal. For this use, the consistent normal depends on the direction of the halfway vector.

In general, the consistent normal is different from the Phong normal, even in cases where the Phong normal would not cause any problems. This behavior is necessary to guarantee continuity. However, it means that the shape and size of highlights will be slightly different from those produced directly from Phong normals. If the original Phong normals are considered to represent the "true" surface, then one can also think of the highlights generated from consistent normals as representing a slightly different surface. This behavior is a consequence of our goal of providing valid reflection vectors for all front-sided incoming directions. The absolute value of the adjustment is largest for grazing angles and is smallest for almost orthogonal directions (as shown on Figure 2a and 3a), thus providing plausible continuously varying illumination.

2 Consistent Normals

Before explaining our technique in detail, we will define some notation and formally describe Phong normal interpolation. We will use the subscript 'v' for values defined at vertices and the subscript 'f' for linearly interpolated values (used in fragment shaders). Phong normal interpolation conceptually requires the following steps (see Figure 5):

1. For each vertex, compute the vertex normal $\mathbf{n}_v$. Often this normal is computed by averaging the adjacent face normals, but the algorithm does not have any dependency on how the normal is computed.
2. Linearly interpolate the vertex normal for any point inside the triangle to get the vector $\mathbf{n}_f$.
3. Normalize this vector to obtain the unit-length Phong shading normal $\mathbf{n}_p = \mathbf{n}_f/|\mathbf{n}_f|$.
4. Compute the dot product of a unit vector $\mathbf{i}$ with the shading normal $\mathbf{n}_p$. The vector $\mathbf{i}$ may be any desired vector. If required, the reflection vector can be computed as $\mathbf{r} = 2\,(\mathbf{i} \cdot \mathbf{n}_p)\,\mathbf{n}_p - \mathbf{i}$.

The vector $\mathbf{n}_p$ is an approximation of the curved surface's normals. However, there is no guarantee that any true surface exists with normals equal to the Phong normals everywhere [Overveld & Wyvill 1997; Lee & Jen 2001]. The Phong normal interpolation is just a convenient mechanism to compute a continuously varying dot product or a reflection vector.

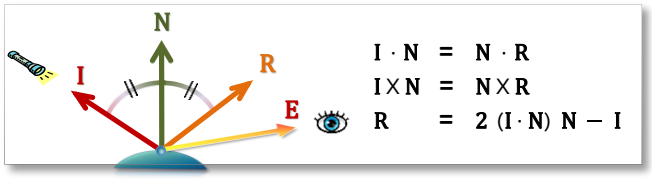

Figure 4: The laws of reflection define the relationship between the incident vector **I**, the true surface normal **N**, and the reflected ray **R**. The angle of incidence is equal to the angle of reflection; vectors **I**, **N**, and **R** are coplanar. The maximum radiance is received for the camera direction **E** = **R**. These same equations are also used to compute the reflection direction for a shading normal.

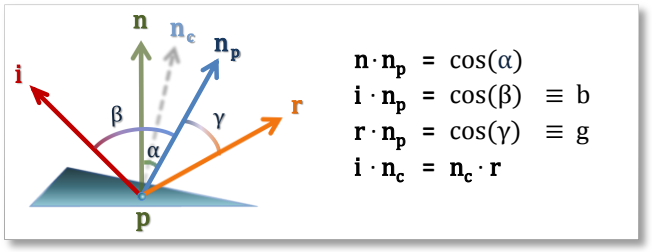

Figure 5: Definition of vectors and angles used in equations. $\mathbf{n}$ – the surface normal; $\mathbf{r}$ – the reflected ray; $\mathbf{i}$ – the given vector (same as incident vector, but with reversed direction) , $\mathbf{n}_p$ – the Phong shading normal, $\mathbf{n}_c$ – the new consistent normal. We want to find an expression for the reflected vector $\mathbf{r}$ that will depend only on the vectors $\mathbf{i}$ and $\mathbf{n}_p$ and that satisfies certain goals stated in this section.

Additional per-vertex values can be defined in step 1 and used to modify the normal $\mathbf{n}_p$. For example, displacement mapping and normal mapping define texture coordinates at the vertices, fetch a value at each fragment using the interpolated texture coordinates, and use this value to modify the normal.

Similarly, in our algorithm we define one additional scalar value at each vertex, which is the maximum angle between the vertex normal and any of the adjacent face normals. This value is then interpolated and used to compute the new "consistent normal" with a formula that we will define.

In the next section, we will describe the requirements that this formula will satisfy and derive a closed-form expression for the new consistent normal $\mathbf{n}_c$, which is a very simple perturbation of the Phong normal $\mathbf{n}_p$.

2.1 Goals and Notation

In describing our algorithm, we will use the vectors and angles defined in Figure 5. Initially, we will pursue the goal of defining a new consistent shading normal which guarantees outward facing reflection vectors. Then we will show how to use the consistent normal in the Blinn-Phong and other shading models.

According to the laws of reflection (Figure 4), a reflected ray lies in the plane defined by the incident ray and the true surface normal. In the spirit of Phong approach, we will apply the laws of reflection to shading normals as well, even though the shading normal does not correspond to a real surface. Since the incoming ray $\mathbf{i}$, the Phong normal $\mathbf{n}_p$, and the reflected ray $\mathbf{r}$ are coplanar, we can use scalar angles to characterize the mutual positions of the vectors, such as the angle β between $\mathbf{i}$ and $\mathbf{n}_p$, and the angle γ between $\mathbf{r}$ and $\mathbf{n}_p$. The angles β and γ are in the $[0, \pi]$ interval.

Informally, we would like to determine the angle γ as a function of the angle β, such that we guarantee that the reflection vector is outward facing as illustrated in Figure 3. Note that if Phong normals are used for reflections, then $\gamma \equiv \beta$. In contrast, we will determine γ such that it is always less than $\pi/2 - \alpha$. We will maintain the qualitative behavior of γ: its value will grow monotonically with growing values of β.

It is more convenient to work with cosines of angles rather than with the angles themselves, since dot products can be easily computed. Accordingly, our goal is to determine $g = \cos(\gamma)$ as a function of $b = \cos(\beta)$. More formally, we want to determine a function $f(b, \mathbf{p})$ such as $g = f(b, \mathbf{p})$ for every given point $\mathbf{p}$ inside a triangle. In a traditional case, when the Phong normal $\mathbf{n}_p$ is used for reflection, $f(b, \mathbf{p}) \equiv b$ for each pixel (yielding $\gamma \equiv \beta$).

If we want $\mathbf{r}$ to be continuous across edges, f should not depend on the geometric normal $\mathbf{n}$, either explicitly or implicitly (since the geometrical normal is discontinuous). Yet the requirement that the reflected ray point away from the surface is defined in terms of $\mathbf{n}$ (as $\mathbf{n} \cdot \mathbf{r} \geq 0$). There would seem to be a problem here, but it is easy to see that there are functions f that yield a continuous $\mathbf{r}$ and satisfy the condition $\mathbf{n} \cdot \mathbf{r} \geq 0$. For example, $f \equiv 1$ is one such function. Unfortunately, this function results in $\mathbf{r} \equiv \mathbf{n}_p$, which does not yield the expected behavior for the reflection vector.

To find more perceptually plausible expressions for f, we will first formulate physically-based requirements that all such functions should satisfy (thereby formalizing the behavior sketched in Figure 3). We start with several observations. The maximum value of the argument b is 1 (reached when the angle β is at its minimum value of 0). Its minimum value is $\cos(\alpha + \pi/2)$, since β cannot exceed $\alpha + \pi/2$, where α is an angle between $\mathbf{n}_p$ and the geometric normal $\mathbf{n}$. This statement is true for any incoming direction, not just one in the same plane as $\mathbf{n}_p$ and $\mathbf{n}$ as shown in Figure 5.

The following goals stipulate that f should have its minimum for $b = \cos(\alpha + \pi/2)$, have its maximum for $b = 1$, and vary continuously and monotonically in between:

GOAL 1. f should be continuous to allow continuously varying illumination.

GOAL 2. $f(b, \mathbf{p})$ should be a monotone function of b for any given $\mathbf{p}$. This requirement precludes extraneous highlights.

GOAL 3. The reflected ray should always point away from the surface – that is, $\mathbf{n} \cdot \mathbf{r} \geq 0$.

GOAL 4. We require that $0 \leq \gamma \leq \pi/2$, which corresponds to $0 \leq f \leq 1$. Rays reflected from true curved surfaces possess this property, and we want the function f to behave in the same way.

GOAL 5. For a triangle with all per-vertex normals equal to the face normal, we want $\gamma \equiv \beta$, i.e. $f(b, \mathbf{p}) \equiv b$. This is equivalent to the requirement that reflections be defined by the geometric normal for "flat" surfaces.

GOAL 6. For "orthogonal" incident directions (that is, directions for which $b = \mathbf{i} \cdot \mathbf{n}_p = 1$), the new normal should be identical to the Phong normal, since using Phong normals does not cause problems for such directions. Mathematically this requirement is $f(1, \mathbf{p}) \equiv 1$.

GOAL 7. The maximum value of γ (and hence the minimum of $f(b, \mathbf{p})$) should be reached when the angle β between the $\mathbf{i}$ and $\mathbf{n}_p$ vectors reaches its maximum value of $\alpha + \pi/2$. This behavior is what we expect for a reflection vector.

2.2 Solution for a Single Point

We first consider how to achieve goals 2 through 7 at a single point on the surface, ignoring issues of continuity across a triangle mesh. We will then expand this simplified solution to enforce continuity (which is formulated as goal 1).

We will look for a linear expression with respect to b for the function $f(\mathrm{b}, \mathbf{p})$, for a given $\mathbf{p}$. Recall that $f(\mathrm{b}, \mathbf{p})$ defines $\mathrm{g} \equiv \cos(\gamma)$ as a function of $\mathrm{b} \equiv \cos(\beta)$. Goal 6 specifies that the maximum of this function occurs at $\mathrm{b} = 1$, and goal 7 specifies that its minimum occurs at $\mathrm{b} = \cos(\alpha + \pi/2)$. The value at the minimum corresponds to a reflected vector which is orthogonal to the face normal. That is, $f(\cos(\alpha + \pi/2), \mathbf{p}) = \cos(\pi/2 - \alpha) = \sin(\alpha)$, allowing us to obtain

$$\begin{aligned} q(\alpha) &= \frac{1 - \sin(\alpha)}{1 + \sin(\alpha)} \\ f(\mathrm{b}, \alpha) &= 1 + q(\alpha)\,(\mathrm{b} - 1) \\ &\text{where } \mathrm{b} = \cos(\beta) = \mathbf{i} \cdot \mathbf{n}_\mathrm{p} \end{aligned} \tag{1}$$

We can verify that the expression (1) satisfies all goals 2 through 7. Goals 6 and 7 are satisfied by design. For goal 5, we can see that at $\alpha \equiv 0$, f is equal to b. Goal 4 is easily verified. Goal 3 is satisfied as follows: the $\mathbf{n} \cdot \mathbf{r} \geq 0$ condition is equivalent to the requirement that $\gamma \leq \pi/2 - \alpha$, or $\cos(\gamma) \geq \cos(\pi/2 - \alpha)$. The equations (1) allows us to compute $f = \cos(\gamma)$, and this inequality is simplified to

$$\frac{(\mathrm{b} + \sin(\alpha))(1 - \sin(\alpha))}{1 + \sin(\alpha)} \geq 0 \tag{2}$$

It is always true for $\mathrm{b} \geq -\sin(\alpha) = \cos(\alpha + \pi/2)$, i.e. for all front-sided vectors $\mathbf{i}$.

Goal 2 is satisfied as follows: The function $q(\alpha)$ is monotonically decreasing on the interval $\alpha \in [0, \pi/2]$, since its derivative is equal to $-2\cos(\alpha)/(1+\sin(\alpha))^2$. The plot of $q(\alpha)$ is given in Figure 6. Bigger values of α result in smaller angles γ. For the extreme case of $\alpha = \pi/2$, the reflected ray coincides with $\mathbf{n}_\mathrm{p}$.

We can provide an intuitive description of our solution. The equations (1) for $f(\mathrm{b}, \alpha)$ is a linear blend of $f \equiv \mathrm{b}$, which corresponds to a Phong reflection, and $f \equiv 1$, which corresponds to a reflected ray which is equal to the Phong normal. The blending weight is given by $q(\alpha)$. Therefore, the reflected ray computed using the new approach always lies between the Phong reflection direction and the Phong normal direction.

2.3 Solution for a Mesh

Now that we understand how to compute a consistent normal for a single point, we will extend the algorithm to a mesh. The additional constraint we must meet is to guarantee that continuity is maintained across edges in the mesh (goal 1).

The basic approach will be to use the solution just derived in equations (1), but where the angle α is approximated in a manner

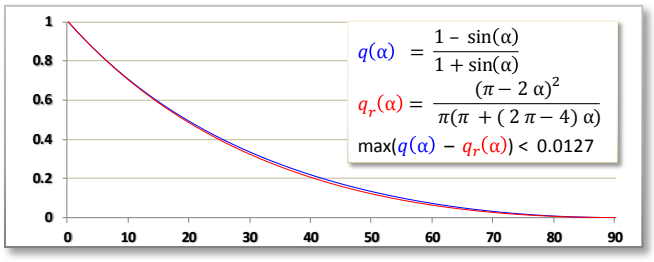

Figure 6. The function $q(\alpha)$ in the equations (1) can be approximated by the rational expression $q_r(\alpha)$ with about 1% accuracy.

that is continuous across the mesh. Additionally, we will want an approximation that is always greater than the true value of α. Such an approximation will guarantee that reflected rays will always face outward from the surface, because $q(\alpha)$ is a monotonically decreasing function. We will refer to this kind of approximation as a *conservative approximation*. To simplify the discussion, we will first explain how to compute α in a manner that is continuous, and then later explain how to adjust the computation to provide the conservative approximation property.

To maintain continuity across the mesh, we follow the standard strategy of computing a value at each vertex and then linearly interpolating this value across each mesh polygon. Thus, we introduce the following variables:

α_{v0} - a value computed at each vertex in the mesh, representing the maximum angle between the vertex normal and all adjacent face normals.

α_v - an adjusted value of α_{v0} computed at each vertex. For now, let $\alpha_v = \alpha_{v0}$, although we will change this computation later. (3)

α_f - the linearly interpolated value of α_v; it is defined for any point inside the triangle.

Let us now consider how to guarantee that α_f is a conservative approximation of α. Unfortunately the interpolated angle α_f is not always greater than α, but fortunately the difference is rather small. Thus, our goal will be to scale α_f, such that the scaled angle will be greater than α but not deviate too much from α_f. Appendix A proves that the global minimum of the difference $\alpha_f - \alpha$ for all possible triangles and per-vertex normals is equal to the global minimum of the following function on the $\mathrm{u} \in [0, 1]$ interval:

$$\tau(\mathrm{u}) = \alpha_f - \alpha = \frac{\pi\,(1-\mathrm{u})}{2} - \tan^{-1}\left(\frac{1-\mathrm{u}}{\mathrm{u}}\right) \tag{4}$$

This function has only one minimum on the $[0, 1]$ interval, which can be easily found analytically to be at $\mathrm{u} = (1 - \sqrt{4/\pi - 1})/2$ (this minimum is negative). Therefore, we can guarantee that for any given triangle

$$\alpha_f \geq \alpha + \tau((1 - \sqrt{(4/\pi - 1)})/2) \geq \alpha - 0.071115 \tag{5}$$

The numerical constant in this inequality is less than 4.075°. We could just increase all interpolated α_f values by this small angle to guarantee that $\mathbf{n} \cdot \mathbf{r} \geq 0$. However, this approach would introduce a constant bias, violate goal 5, and require additional per-pixel computations.

Instead, we will introduce a variable bias at vertices, which will be added to the value of α_{v0}. We want the bias to be relatively small for small values of α_{v0} (characteristic for finely tessellated surfaces), and then monotonically increase as α_{v0} becomes larger, since coarsely tessellated surfaces are more likely to cause problems. The simplest way to achieve this behavior is to scale each per-vertex angle α_{v0}, as follows:

$$\alpha_v = (1 + w\,(1 - \cos(\alpha_{v0}))^2)\,\alpha_{v0} \tag{6}$$

The cosine here is just a dot product of the applicable face normal and vertex normal. Thus, if the vertex normal is equal to the face normal, there is no bias. As the vertex normal and the face normal diverge, the bias increases. The constant w can be found by solving a related 2-dimensional optimization problem (see Appendix A) and is equal to 0.06096. For all vertex normals that have $\alpha_{v0} \leq 45°$, this scaling results in an adjustment of less than 0.3°.

We now have a method to compute consistent normals that satisfy all of the stated goals for a mesh. To recap, at each vertex we compute α_{v0} as specified in declaration (3) and scale it as defined in equation (6) to compute α_v. We interpolate this value across the triangle to obtain α_f, then apply equations (1) at each point on the triangle to compute the cosine of the angle γ between the reflection vector and the interpolated normal. To compute the reflection vector $\mathbf{r}$, we apply the law of sines to the triangles formed by the relevant vectors, resulting in

$$\mathbf{r} = (\sin(\beta + \gamma)\,\mathbf{n}_p - \sin(\gamma)\,\mathbf{i}) / \sin(\beta) \qquad (7)$$

The calculated vector $\mathbf{r}$ has a unit length and does not require normalization. When $\gamma = \beta$, the formula for $\mathbf{r}$ simplifies to the well-known expression $\mathbf{r} = 2\,(\mathbf{i} \cdot \mathbf{n}_p)\mathbf{n}_p - \mathbf{i}$.

If explicitly required, the new consistent normal $\mathbf{n}_c$ can be computed as $\mathsf{normalize}(\mathbf{i}+\mathbf{r})$. We can also use the expression for the cosine of a half-angle to directly compute $\mathbf{i} \cdot \mathbf{n}_c$, given $\mathbf{i} \cdot \mathbf{r}$.

This solution has the disadvantage that it computes a transcendental function $q(\alpha)$ at each pixel. This function can be costly to evaluate on some hardware architectures. To make the algorithm less computationally expensive, $q(\alpha)$ can be approximated by a rational function $q_r(\alpha)$. If we choose an approximation for which $q_r(\alpha) \le q(\alpha)$ for any α, it will have the same effect as increasing the angles α_v. This approximation allows us to decrease the constant w and still guarantee that $\mathbf{n} \cdot \mathbf{r} \ge 0$. The function $q_r(\alpha)$ shown in Figure 6 provides a good compromise between accuracy of approximation and simplicity of computation. For this function, the constant w is 0.03632 (obtained by solving a 2-dimensional optimization problem described at the end of Appendix A).

Altogether, the algorithm for computing the reflection vector for any front-sided vector vector $\mathbf{i}$, as well as the corresponding consistent normal $\mathbf{n}_c$ and the dot product $\mathbf{i} \cdot \mathbf{n}_c$ is:

At each vertex, consider the dot products $\mathbf{n}\cdot\mathbf{n}_v$ of the per-vertex normal $\mathbf{n}_v$ and all adjacent face normals $\mathbf{n}$ and choose the one with the minimum value. Then compute

$$w = 0.03632$$

$$\alpha_v = \cos^{-1}(\mathbf{n}\cdot\mathbf{n}_v)\,(1 + w\,(1 - \mathbf{n}\cdot\mathbf{n}_v)^2)$$

For a given point in a triangle, compute

- interpolated α_f using α_v values at three vertices
- coefficient

$$q = \frac{(1 - (2/\pi)\,\alpha_f)^2}{1 + 2\,(1 - 2/\pi)\,\alpha_f}$$

Then, for any given front-sided vector $\mathbf{i}$, the cosines of angles β and γ (see Figure 5) are (8)

$$b = \mathbf{i} \cdot \mathbf{n}_p$$

$$g = 1 + q\,(b - 1)$$

The ratio $\sin(\gamma)/\sin(\beta)$ is

$$\rho = \sqrt{q\,(1 + g)/(1 + b)}$$

Allowing us to compute

$$\mathbf{r} = (g + \rho\,b)\,\mathbf{n}_p - \rho\,\mathbf{i}$$

$$\mathbf{n}_c = \mathsf{normalize}(\mathbf{i} + \mathbf{r})$$

$$\mathbf{i} \cdot \mathbf{n}_c = \sqrt{(1 + (g + \rho\,b)\,b - \rho)/2}$$

The per-vertex angles α_v can be evaluated and stored when per-vertex normals are computed or updated. The computation at each fragment (or hit point) uses the linearly interpolated scalar value α_f as well as values that are typically available in fragment shaders ($\mathbf{n}_p$ and $\mathbf{i}$), allowing straightforward implementation of the algorithm (8) in any high-level shading language.

2.4 Application to Shading

So far we have shown how to compute a reflection vector that is guaranteed to be front-facing. In this section, we show that the same algorithm can be used to guarantee that a dot product between an arbitrary front-facing vector and the shading normal is positive. For example, we can guarantee that the Blinn-Phong shading model's dot product is always positive.

Consider a front-facing vector $\mathbf{i}$. We will now prove that the dot product $\mathbf{i} \cdot \mathbf{n}_c$ is non-negative. First, observe that even if we do not use the reflection vector $\mathbf{r}$, it is still defined by algorithm (8), and we know that it is outward facing, i.e. the angle between $\mathbf{n}$ and $\mathbf{r}$ is less than $\pi/2$. Second, since $\mathbf{i}$ is also front-facing, we know that the angle between vectors $\mathbf{i}$ and $\mathbf{r}$ is less than π. Third, observe that $\mathbf{n}_c$ bisects the angle between vectors $\mathbf{i}$ and $\mathbf{r}$ (Figure 5), and that all three vectors are co-planar by the original definition of $\mathbf{r}$ and $\mathbf{n}_c$. Thus, the angle between $\mathbf{i}$ and $\mathbf{n}_c$ is equal to one-half the angle between $\mathbf{i}$ and $\mathbf{r}$, and is therefore less than or equal to $\pi/2$. Thus, $\mathbf{i} \cdot \mathbf{n}_c \ge 0$.

The Blinn-Phong reflection model [Blinn 1977] requires computation of the dot product of the shading normal with the halfway vector. Since the halfway vector is collinear with the sum of the eye vector and the light vector, it will be front-sided if both the eye vector and the light vector are front-sided. This allows us to directly use expressions (8) (treating $\mathbf{i}$ as the given half-vector) to compute the dot product $d_c = \mathbf{i} \cdot \mathbf{n}_c$, which is guaranteed to be non-negative. Figure 13 shows that the visual result produced by this technique is superior to that produced by the standard technique of computing $\mathbf{i} \cdot \mathbf{n}_p$ and clamping negative values to zero.

Note that the expression (8) for the dot product $\mathbf{i} \cdot \mathbf{n}_c$ is applicable to both local lights (for which the direction $\mathbf{i}$ is different at each fragment) and to lights located at infinity (for which $\mathbf{i}$ is always the same).

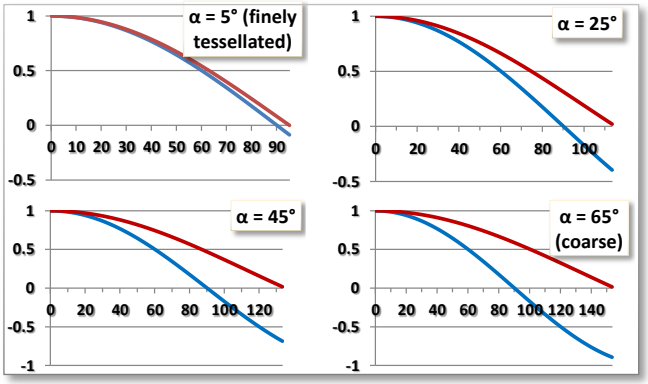

Figure 7. Consistent normals guarantee a positive dot product with any given vector $\mathbf{i}$ (red line), but behave almost the same as Phong normals except for grazing incident angles on coarse meshes. In contrast, the dot product of vector $\mathbf{i}$ with Phong normals (blue line) can become negative (requiring clamping). Four plots for different values of the angle α between the face normal and the interpolated normal are shown. The horizontal axis is the angle β between $\mathbf{i}$ and the Phong normal (varying from 0 to $\pi/2 + \alpha$).

3 Discussion

Consistent normals can be thought of as a fine-tuning of Phong normals to provide additional consistency guarantees. To better understand the differences between consistent normals and Phong normals, we plot values of shading dot products computed using the two methods. Specifically, we plot $d_p = \mathbf{i} \cdot \mathbf{n}_p$ and $d_c = \mathbf{i} \cdot \mathbf{n}_c$ for different incident directions $\mathbf{i}$ specified by the angle β (Figure 7).

By design, when the incident vector is nearly orthogonal to the surface ($\beta \approx 0$), the Phong normal and the consistent normal are almost the same (see also Figure 2a and 3a). For incident vectors at more grazing angles (larger β), Phong normals and consistent normals diverge: d_c is always non-negative, while d_p can be less than 0. However, if the linearly interpolated normal is close to the face normal ($\alpha = 5°$), the difference between these two functions is still small. This is the situation for a highly tessellated smoothly varying mesh (see right column in Figure 9).

Our goal was *not* to approximate α with good accuracy, because this would just result in discontinuous reflections. Instead, we derived the algorithm (8), which guarantees that $\alpha_f \geq \alpha$ and satisfies goals 1-7 described in section 2.1. The difference $\alpha_f - \alpha$ can reach $\pi/2$ for some pathological triangles in which all per-vertex normals reside in the plane of the triangle. In such cases, the reflection vector will always be set equal to the Phong normal, regardless of the incident direction. Even for such cases, the use of consistent normals results in believable pictures for close-to-orthogonal viewing directions. However, for grazing viewing angles, no known method will be able to hide the fact that the underlying mesh is very coarse.

3.1 Performance Results

Table 1 shows that the performance impact of using consistent normals for Blinn-Phong shading is minimal (~1%) on modern graphics architectures. The table compares frame rates using standard Phong normal interpolation to frame rates using two variants of consistent normal interpolation – the first variant uses a trigonometric expression and the second variant uses a rational expression (see $q(\alpha)$ and $q_r(\alpha)$ respectively in Figure 6). Performance is measured on three different graphics architectures: the Intel® Core™ i5-450M processor with mobile Intel® HD Graphics, the mid-range discrete NVIDIA® GeForce® 9800 GT, and the top of the line discrete NVIDIA® GeForce® GTX 480. The performance measurements are made using the "Tiny" model shown in Figure 13. The performance differences for the GTX 480 are statistically irrelevant, since the standard deviation for these measurements was about 4%. For the two other platforms, the standard deviation was less than 0.2% and the performance cost of consistent normal interpolation is about 1%.

shader \ card	i5-450M (laptop)	GeForce® 9800 GT	GeForce® GTX 480
Phong normals	291.33	1111.86	2057.72
trigonometric CNI	288.20	1100.84	2058.26
rational CNI	288.16	1101.30	2055.98

Table 1. Performance differences between Phong normals and consistent normals are minimal across different platforms. The data (frames per second) is for the "Tiny" model shown in Figure 13.

We also measured the performance impact of using consistent normals to compute reflection directions in our research group's CPU ray tracer. In this case, the slowdown for using consistent normals is about 5%. However, some of this additional cost is due to the tracing of extra reflection rays, since we discard all inward facing reflections when Phong normals are used.

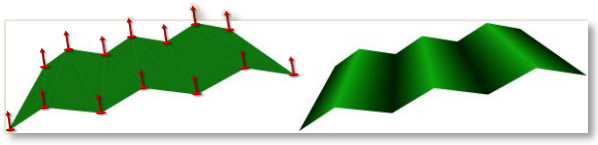

Figure 8. Zigzag-shaped object shaded with a traditional Phong normal interpolation (left) and with modulated consistent normals (right). All per-vertex shading normals are equal. For consistent normal interpolation, the per-vertex angle is additionally modulated by the curvature of the surface to achieve the desired effect.

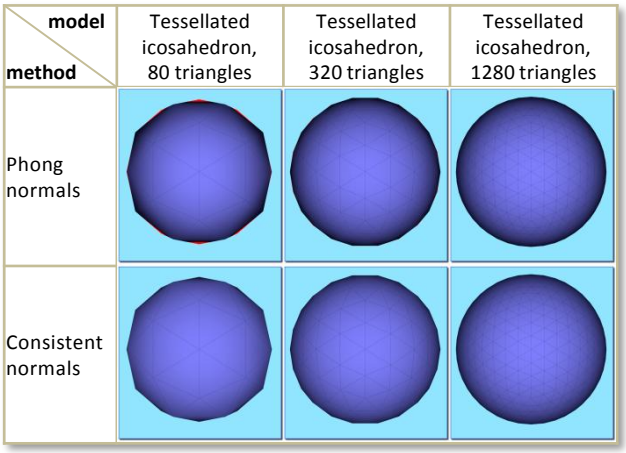

Figure 9. Increasing surface smoothness results in almost identical shading for Phong and consistent normals. Red color indicates pixels with negative dot products.

3.2 Practical Considerations

All variants of our algorithm require knowledge of the scalar value α_v at each vertex. This value depends on the shading normal at the vertex, as well as the face normals of all faces adjacent to the vertex. This value can be either provided by the application or computed on the fly in a geometry shader. The natural time to compute α_v is whenever per-vertex shading normals are created or updated. The value of α_v can be stored as a fourth component of the vertex normal.

It is also possible to modulate the value of α_v to provide additional functionality. In particular, we can modulate it by the surface curvature to overcome certain limitations of linear interpolation [Walia & Singh 2006, Lee & Jen 2001]. Figure 8 shows how this approach can be used to fix problems of zigzag-shaped geometry, in which all per-vertex normals are equal.

Our approach requires that the angle between a vertex's shading normal and an adjacent face normal always be $\leq \pi/2$, since the approach was derived under that assumption. This restriction is reasonable since there is no plausible physical interpretation for an angle greater than $\pi/2$. But for some meshes we do encounter such angles, especially if the per-vertex normals were automatically created. Fortunately, in most cases (and always for vertices with valence three or four) this situation can be remedied. For vertices with valence greater than four, we sort all angles and then, for all adjacent faces for which $\mathbf{n} \cdot \mathbf{n}_v < 0$, modify the vertex normal as

$$\mathbf{n}_v := \mathbf{n}_v - (\mathbf{n} \cdot \mathbf{n}_v)\,\mathbf{n}\ (1 + \text{epsilon})$$

This adjustment creates a new vertex normal for which the dot product with the given face normal $\mathbf{n}$ is positive. It might also create problems for other adjacent faces at which the dot product was positive before this adjustment. Accordingly, a few iterations may be necessary. Though not guaranteed, this approach mostly works for practical models. Figure 10 shows that it works even for the coarse hairline polygons in the "Tiny" model.

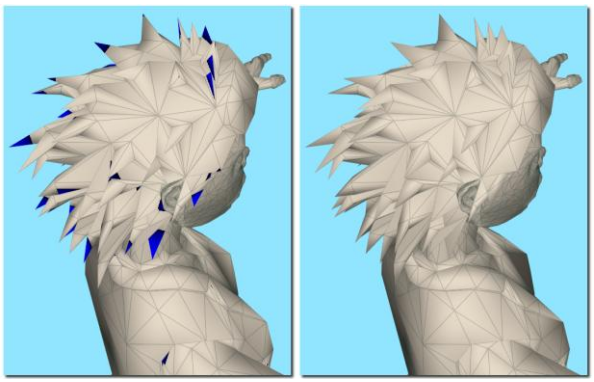

Figure 10. Greedy optimization can be used to insure that the angle between a vertex normal and adjacent face normal does not exceed 90 degrees. Blue pixels indicate an inward-directed interpolated normal. Left: original model; right: modified model, after applying a greedy optimization similar to Gram-Schmidt orthogonalization.

3.3 Limitations

Although our algorithm guarantees a positive dot product for a front-facing incoming vector, it does *not* guarantee a negative dot product for a back-facing incoming vector. Phong normal interpolation has the same limitation. Thus, if the sign of the dot product is used to cull lights that lie behind the surface, some lights may incorrectly contribute to the illumination computation. In practice, this error is usually not noticeable for either algorithm since the contribution of the erroneously included lights is weighted by the dot product (raised to a power), and the dot product is typically very small in these cases.

The shape and size of highlights produced by taking dot products with consistent normals will be slightly different than those produced by taking dot products with Phong normals. This difference can be interpreted either as a change in highlight behavior, or as a change in the surface as defined by the shading normal.

The perturbation introduced by our technique depends on the direction of the incoming vector. When our technique is used to compute the reflection vector this is not an issue, since reflections depend on the incoming directions anyway. However, if our technique is used for shading, a different shading normal will be used for each light direction. Thus, per-fragment computations that otherwise could be shared between lights must now be repeated for each light.

Another undesirable effect of view-dependency is that light paths become different for reverse directions. For example, in Figure 3 light coming from the direction of the "reflected ray" will not exit along the path of the "incoming ray".

In its current form, the technique does not support displacement mapping or normal mapping, although it could probably be extended to cover these techniques if per-mesh bounds on displacement or normal perturbation are available.

4 Conclusion and Future Work

Consistent normal interpolation is an easy-to-use technique for eliminating some of the rendering artifacts encountered when using Phong normal interpolation. The key insight behind the technique is to guarantee desirable consistency properties between face normals and interpolated normals. We have described applications of the technique to the computation of reflection vectors and shading dot products. It may also be possible to extend the technique to other uses such as the computation of Fernel terms.

Acknowledgements

We would like to thank the anonymous reviewers for their detailed and valuable comments and suggestions. The "Tiny" model is part of the media content of Microsoft DirectX SDK. The "Emotiguy", "Dinokonda", and "Toon Dragon" models are part of the digital content of the DAZ 3D editor.

References

BARRERA, T., ANDERS, H.A., AND BENGTSSON, E. 2004. Faster Shading by Equal Angle Interpolation of Vectors. In *IEEE Transactions on Visualization and Computer Graphics*, 10(2), 217–223.

BLINN, J. F. 1977. Models of Light Reflection for Computer Synthesized Pictures. In *Computer Graphics (Proceedings of SIGGRAPH 77)*, vol. 11, ACM, 192–198.

KUIJK, A., AND BLAKE, E. H. 1989. Fast Phong Shading via Angular Interpolation. In *Computer Graphics Forum,* vol. 8, 315–324.

LEE, Y. C., AND JEN, C. W. 2001. Improved Quadratic Normal Vector Interpolation for Realistic Shading. In *The Visual Computer*, vol. 17, 337–352.

LOOP, C., AND SCHAEFER, S. 2008. Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches. In *ACM Transactions on Graphics*, vol. 27(1), 1-11.

VAN OVERVELD, C. W. A. M., AND WYVILL, B. 1997. Phong Normal Interpolation Revisited. In *ACM Transactions on Graphics* vol. 16(4), 397–419.

PHONG, B.-T. 1975. Illumination for Computer Generated Pictures. In *Communications of the ACM,* vol. 18 (6), 311–317.

SCHWENK, K. A. 2009. A Survey of Shading Models for Real-time Rendering. URL: http://www.karsten-schwenk.de/downloads/a_survey_of_shading_models.pdf

SNYDER, J., AND BARR, A. 1987. A. Ray Tracing Complex Models Containing Surface Tessellations. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, vol. 21, ACM, 119-128

WALIA, E., AND SINGH, C. 2006. An Analysis of Linear and Non-Linear Interpolation Techniques for Three-Dimensional Rendering. In *Proceedings of the Conference on Geometric Modeling and Imaging: New Trends*, 69–76.

WOO, A., PEARCE, A., AND OUELLETTE, M. 1996. It's Really Not a Rendering Bug, You See... In *IEEE Computer Graphics and Applications*, vol. 16(5), 21–25.

Appendix A. We want to find the minimum possible difference between the interpolated angle α_f and the angle $\alpha = \cos^{-1}(\mathbf{n} \cdot \mathbf{n}_p)$. This difference is a function of 17 variables $\mathbf{v_0}$, $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{n_0}$, $\mathbf{n_1}$, $\mathbf{n_2}$, and two independent barycentric coordinates given by $\mathbf{b} = [u, v, 1-u-v]$. These variables describe all possible triangles, per-vertex normals, and sample points (taking into consideration that each normal could be represented by 2 angles). However, using symmetric properties of this function, the problem can be reformulated as a minimization of a function of only 1 argument. We will do it in several steps as follows:

17 → 8. By noticing that only the face normal $\mathbf{n}$ depends on vertices and the problem is rotationally invariant, we can use $\mathbf{n} = [0,0,1]$ and exclude vertices from further processing.

8 → 5. Each normal has 2 degrees of freedom and can be represented as $[\sin(\varphi)\cos(\omega), \sin(\varphi)\sin(\omega), \cos(\varphi)]$. The angle α_f, by definition, is computed by linearly interpolating angles φ, and does not depend on angles ω. Thus, for given values of angles φ, the minimum of $\alpha_f - \alpha$ is attained when α has a maximum value. Looking at the triangle formed by the Phong normal $\mathbf{n}_p$ and the face normal $\mathbf{n}$ (Figure 5), the angle α can be

computed as $\tan^{-1}(n_{xy}/n_z)$, where n_z is z coordinate of vector $\mathbf{n}_p$ and n_{xy} is its projection to xy plane. Since only n_{xy} depends on the angles ω, α reaches maximum when the xy projections of all per-vertex normals are parallel — that is, when all angles ω are the same. For simplicity, we will use $\omega \equiv 0$, eliminating 3 variables.

5 → 4. Without loss of generality, $\varphi_0 \le \varphi_1 \le \varphi_2$. We will vary $x = \varphi_1$ on the $[\varphi_0, \varphi_2]$ interval and consider a function $\delta(x) = \alpha_f - \alpha$. By analyzing its derivatives, we can conclude that $\delta(x)$ is either monotone or concave (dome-shaped; see example in Figure 11). In both cases, its minimum is achieved at either φ_0 or φ_2, allowing us to consider only 4 independent variables: φ_0, φ_2, and two barycentric coordinates.

4 → 3. Taking into consideration all of the simplifications described so far, the expressions for angles α_f and α are:

$$\alpha_f = \mathbf{b} \cdot [\varphi_0, \varphi_1, \varphi_2]$$

$$\alpha = \tan^{-1}\left(\frac{\mathbf{b} \cdot [\sin(\varphi_0), \sin(\varphi_1), \sin(\varphi_2)]}{\mathbf{b} \cdot [\cos(\varphi_0), \cos(\varphi_1), \cos(\varphi_2)]}\right) \quad (9)$$

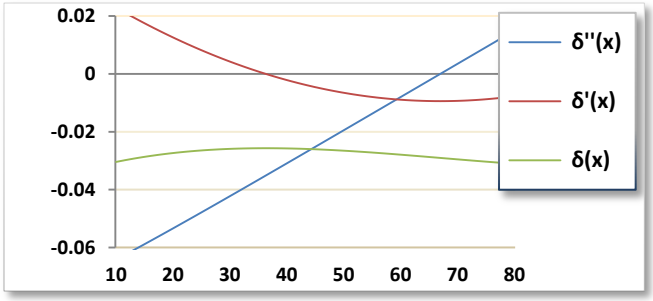

Figure 11. Derivatives of $\delta(x) = \alpha_f - \alpha$ for $\varphi_0 = 10°$ and $\varphi_2 = 80°$. $\delta(x)$ is either monotone or concave (as shown), allowing the reduction of the number of independent variables from 5 to 4.

We can eliminate φ_1, e.g. by using $\varphi_1 = \varphi_2$. For each such 4-dimensional problem, there is an equivalent 3-dimensional problem with a modified $\mathbf{b} = [b_0, 0, b_1 + b_2] = [u, 0, 1-u]$ and angles $[\varphi_0, 0, \varphi_2]$, yielding the same values for angles (9).

3 → 2. For given φ_0 and φ_2, the difference $\alpha_f - \alpha$ will be a function of u. Figure 12 plots this function on a [0, 1] interval for different values of angles φ_0 and φ_2. This function is completely defined by its value at $u = 0$ and its derivative on the [0, 1] interval. It is easy to see that the function value at $u = 0$ is always 0 (regardless of the values of φ_0 and φ_2) and that the derivative depends only on u and the difference $\varphi_2 - \varphi_0$.

2 → 1. We have 2 variables left: u and $\varphi = \varphi_2 - \varphi_0$, so the target function is reduced to

$$\alpha_f - \alpha = (1-u)\,\varphi - \tan^{-1}\left(\frac{u-1}{(u-1)\cot(\varphi) - u\csc(\varphi)}\right) \quad (10)$$

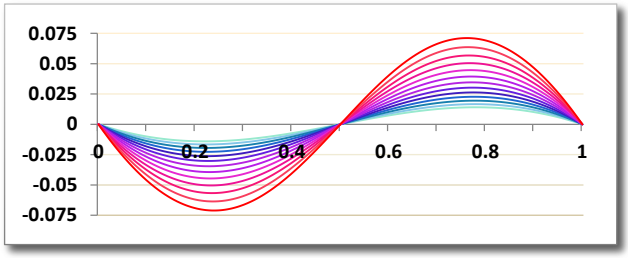

Figure 12. Parametric family of functions we want to minimize.

Its φ-derivative depends only on u and $\cos(\varphi)$, allowing us to find the value of φ, for which the difference has a minimum for any given u. This value does not depend on u and is equal to $\pi/2$.

1 → 0. We have reduced the problem to a minimization of a one-dimensional function given by (4). By computing the u-derivative of $\tau(u)$, we will get a quadratic equation for the two extremal points of this function (see Figure 12), yielding inequality (5).

The minimum value of the difference $\alpha_f - \alpha$ is achieved when one normal lies in the plane of the triangle and another is orthogonal to it (the third normal does not contribute to the difference, as the corresponding barycentric coordinate is 0). This minimum is negative. However, if, instead of α_f, we use the scaled value defined by (6), the minimum difference for all possible triangles could be made equal to 0 (which is our goal). We want to find a minimum value of w for which this is possible. This is achieved by solving a 2-dimensional optimization problem, dependent on two variables — the barycentric coordinate u and the scaling factor w. The solution gives us a minimum value of w which guarantees that the difference of the modified α_f and the angle α is non-negative.

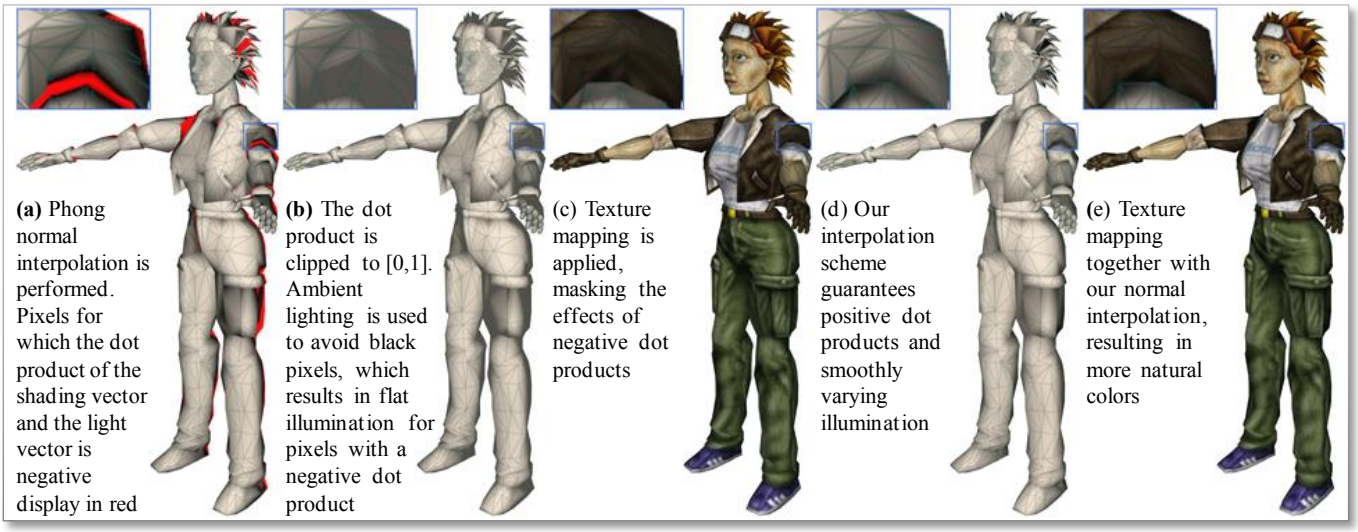

Figure 13. Comparison of the new consistent normal interpolation technique (**d** and **e**) with a classic Phong interpolation (**a** – **c**). The light source is placed at the camera position. The "Tiny" model from the Microsoft DirectX SDK is shown.