The Visual Computer

Improved quadratic normal vector interpolation for realistic shading

Yuan-Chung Lee,
Chein-Wei Jen

Department of Electronics Engineering, National Chiao-Tung University, Hsinchu, Taiwan, ROC
E-mail: {yzlee, cwjen}@twins.ee.nctu.edu.tw

Published online: 19 July 2001
© Springer-Verlag 2001

Interpolation for normal vectors is an important stage of Phong shading. Linear interpolation cannot represent the geometry of surfaces adequately in some situations. Previous quadratic interpolation for normal vectors generates visual artifacts for arch-type curves and triangle scan conversion. To eliminate those artifacts, we propose an improved quadratic interpolation for normal vectors with little computational overhead. Perspective-correct version for obtaining perspective foreshortening is also presented.

Key words: Shading – Quadratic interpolation – Perspective correction

1 Introduction

With the rapid progress in VLSI technology and advanced hardware architecture, the processing capability applied in 3D graphics has increased quickly in recent years. Hence, visual realism in real-time 3D rendering can be improved significantly. One basic and useful approach that improves visual realism is advanced shading technology. A variety of shading methods have been used to shade surfaces in conventional polygon-based rendering. Gouraud shading (Gouraud 1971) suffers from the Mach band effect and fails to capture detailed lighting characteristics. Quadratic color interpolation (Lathrop et al. 1992) increases the continuity of intensity but cannot simulate accurate highlight spots or a multi-light environment. Moreover, in specific situations, quadratic color interpolation may produce problematic negative color, which should be replaced with unpleasant clamped values. Phong shading (Phong 1975) reduces the Mach band effect greatly, because per-pixel lighting yields more realistic illumination. However, the linear interpolation scheme for normal vectors still has many problems (see Foley et al. 1996), such as non-realistic polygonal silhouette, perspective distortion, orientation dependence, and unrepresentative vertex normals. Numerous improved algorithms have been proposed to solve these problems. For example, to reduce the computational cost, Duff (1979) combined normal interpolation and reflection equations. He also suggested a rotation-independent method to solve the orientation problem. Bishop and Weimer (1986) used a second-order Taylor series approximation to simplify the normalization operations. Kuijk and Blake (1989) interpolated polar angles instead of normal vectors to eliminate the normalization operations. To solve the non-realistic polygonal silhouette problem, geometrical-modification methods (Catmull and Clark 1978; van Overveld and Wyvill 1997a; Lee and Jen 2000) refine planar polygon silhouettes to enhance visual appearance.
Additionally, to solve the problem that normal vector distribution cannot be represented by linear interpolation, several methods have been proposed. Shantz and Lien (1987) presented techniques for shading bicubic patches while rendering curve surfaces, but their techniques need the derivatives of surface functions. Max (1989) ensured C^1 interpolation for normal vectors, but his input data must contain either the second derivatives of surface functions or the total connection lists for each vertex.

Nevertheless, those additional requirements are not available in conventional polygon-based rendering pipeline. Van Overveld and Wyvill (1997b) proposed a quadratic normal vector interpolation algorithm to replace the traditional linear interpolation. Their algorithm overcomes the inappropriateness of traditional linear interpolation when a surface has a serpentine profile, i.e., a surface with inflection points. However, we found that their algorithm produces visual defects in many situations. Furthermore, their algorithm is unsuited to standard triangle scan conversion. In this paper, we propose an improved quadratic normal vector interpolation to eliminate the defects. Our algorithm constructs a bi-quadratic normal function that is suitable for triangle scan conversion. This improved bi-quadratic normal vector interpolation can replace traditional linear interpolation of Phong shading to produce more realistic shading effects. We also discuss perspective distortion for shading operations and propose a correction method, especially as applies to our interpolation algorithm.

This paper is organized as follows. Section 2 describes the algorithm proposed by van Overveld and Wyvill (1997b) and analyzes the defects arising from their algorithm. In Sect. 3, an improved algorithm that can eliminate these defects is proposed and is compared to the original in terms of visual quality and computational cost. In Sect. 4, the perspective correction for shading operations is considered. Finally, Sect. 5 is the summary.

2 Quadratic interpolation algorithm

In the paper on quadratic interpolation for normal vectors by van Overveld and Wyvill (1997b), they claimed that no surface would yield the illumination distribution generated by traditional linearly interpolated Phong shading. Their claim is correct; however, their inference contains some flaws. We provide the revised proof in the Appendix.

They proposed a quadratic interpolation algorithm to replace traditional linear interpolation. Although this algorithm still cannot match quadratic curves (to be proven later), it can handle the serpentine profile of some cubic curves. They use a quadratic function:

$$\boldsymbol{n}(s) = \boldsymbol{c} + \boldsymbol{a}s + \boldsymbol{b}s^2 \tag{1}$$

with three boundary conditions:

$$\boldsymbol{n}(0) = \boldsymbol{n}_0 \tag{2}$$

$$\boldsymbol{n}(1) = \boldsymbol{n}_1 \tag{3}$$

$$\left(\int_0^1 \boldsymbol{n}(s)\,\mathrm{d}s,\ \boldsymbol{\Delta} \right) = 0 \text{ and minimize } \int_0^1 \left(\ddot{\boldsymbol{n}}(s)\right)^2 \mathrm{d}s \tag{4}$$

where s is a parameter from 0 to 1, $\boldsymbol{n}_0$ and $\boldsymbol{n}_1$ are normal vectors of the two end-points, $\boldsymbol{\Delta}$ is the edge vector over which normal interpolation takes place, $\ddot{\boldsymbol{n}}$ denotes second-order differentiation, and $(,)$ denotes an inner product.

Conditions (2) and (3) give $\boldsymbol{c} = \boldsymbol{n}_0$ and $\boldsymbol{a} = \boldsymbol{n}_1 - \boldsymbol{n}_0 - \boldsymbol{b}$. The first equation in condition (4) implies that $(\boldsymbol{b}, \boldsymbol{\Delta}) = 3(\boldsymbol{n}_0 + \boldsymbol{n}_1, \boldsymbol{\Delta})$. This model requires the second equation in condition (4) to impose one additional constraint. Using variational calculus, the result is:

$$b = 3\frac{(\boldsymbol{n}_0 + \boldsymbol{n}_1,\ \boldsymbol{\Delta})}{\|\boldsymbol{\Delta}\|^2}\boldsymbol{\Delta}. \tag{5}$$

These boundary conditions complete the derivation of the quadratic function for normal vectors. The quadratic interpolation replaces linear interpolation in polygon scan conversion. Initially, they apply quadratic interpolation to the boundaries of the polygon. Next, they establish different quadratic functions for each scan-line segment according to the coordinates and normal vectors of two end points. Their approach applies to all (not necessarily convex) kinds of polygons. Forward difference can be used to reduce the interpolation cost.

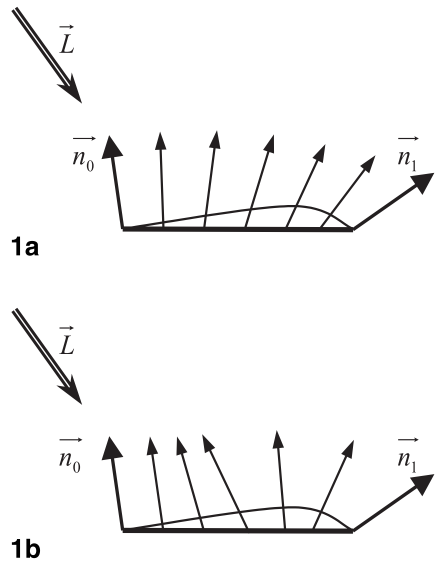
This algorithm has several drawbacks, however, which produce visual defects in many situations. We classify these drawbacks into four types, which are described in the Sects. 2.1–2.4.

2.1 Inappropriateness to quadratic curves and some arch-type cubic curves

The algorithm from van Overveld and Wyvill (1997b) sets the integration of the inner product to zero in (4). This is appropriate to the serpentine cases of cubic curves. However, for some oblique arch cases (i.e., without inflection points) of cubic curves or quadratic curves, the integration of the inner product should not be zero. Their quadratic interpolation

1a

1b



2

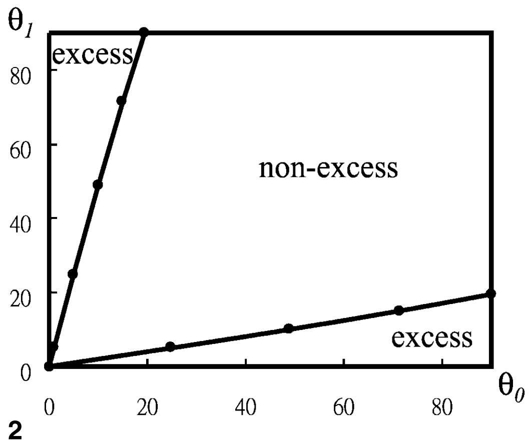


Fig. 1a,b. Inappropriateness to arch-type curves using the quadratic normal vector interpolation of van Overveld and Wyvill (1997b). **a** Linear interpolation matches with the profile of the curve, and a highlight effect does not appear in the middle of the edge. **b** Quadratic interpolation does not match with the profile of the curve, and an unwanted highlight effect appears in the middle of the edge

Fig. 2. Regions of normal vectors that produce excess using the algorithm of van Overveld and Wyvill (1997b). θ_0 and θ_1 are the corresponding angles between the perpendicular of the edge and $\boldsymbol{n}_0, \boldsymbol{n}_1$

produces defects in these situations, as shown in the following example.

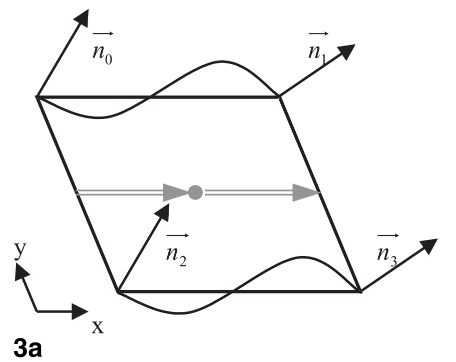
Consider an edge with two normal vectors at the two end points as shown in Fig. 1. The normal vectors are sampled from the oblique arch-type curve and are coplanar with the curve. $\boldsymbol{L}$ is the incident light vector. When their quadratic interpolation is applied, the normal vectors in the middle of the edge will exceed the convex combinations of two normal vectors at the two end points, and may produce an unwanted highlight effect. Linear interpolation is more appropriate to this situation. However, according to (5), the second-order coefficient $\boldsymbol{b}$ differs from zero, except for the case of mirror-symmetric normal vectors in the two end points.

Let us examine the kinds of normal vector configurations that result in the excess. For simplification, we only check the interpolated normal vector at the center of the edge. θ_0 is the angle between $\boldsymbol{n}_0$ and the perpendicular plane of the edge. θ_1 is the angle between $\boldsymbol{n}_1$ and the perpendicular plane of the edge. Figure 2 shows the results of the examination. There are many kinds of normal vector configurations that generate an unwanted highlight effect. Moreover, even in the range without the excess, the quadratic interpolation of their algorithm sometimes results in a bent normal vector profile, which does not appear in arch-type curves.

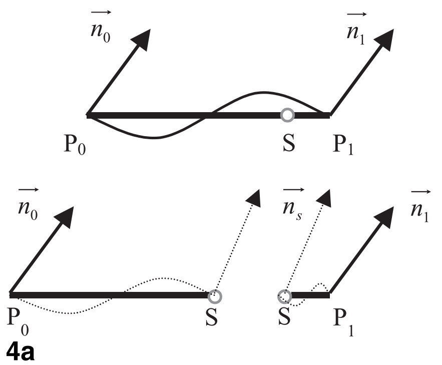
As we observe for general approximation, linear interpolation is appropriate to arch-type curves, while their quadratic interpolation is appropriate to serpentine-type curves. Although these interpolation methods may not perfectly produce ideal normal vectors of the underlying curves, they can simulate the highlight effect plausibly.

2.2 Defects when normal vectors and the edge vector are not coplanar

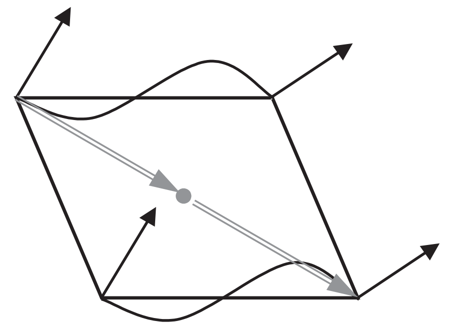
Van Overveld and Wyvill assumed the interpolated curves to be sufficiently planar, so that the normal vectors are nearly coplanar with the edge vector along the interpolation direction. However, this as-



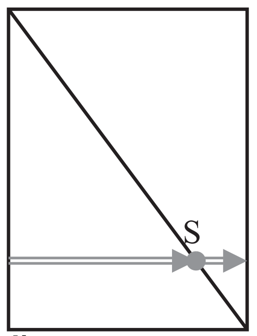
3a



4a



3b



4b

Fig. 3a,b. An example showing the defect when normal vectors and the edge vector are not coplanar. The quadratic interpolation takes place **a** along the x direction and **b** along the diagonal direction. The interpolated normal vectors at the center of the polygon are different in **a** and **b**

Fig. 4a,b. Subdivision variance of the original quadratic interpolation. **a** An edge representing a serpentine curve is subdivided into two segments at the point S near the end point P_1. The original quadratic interpolation causes inconsistency with the curve after subdivision. **b** Triangle scan conversion involves subdivision

sumption does not usually hold. Consider the serpentine surface approximated by quadrilateral polygon as shown in Fig. 3. $\boldsymbol{n}_0$ is the same as $\boldsymbol{n}_2$, and $\boldsymbol{n}_1$ is the same as $\boldsymbol{n}_3$. The normal vectors of the serpentine surface have y coordinate of zero. They only change in the x direction. If the normal vectors are interpolated along the x direction, the results will match with the surface profile. However, polygons are probably decomposed into triangles before the rendering process, especially in most hardware renderers. Therefore, normal vectors along the diagonal direction should first be interpolated to establish the boundary information. The interpolated normal vectors along the diagonal direction do not have y coordinate of zero because the coefficient vector $\boldsymbol{b}$ in (5) now parallels the diagonal vector. Hence, defects occur when normal vectors and the edge vector are not coplanar. In Fig. 3a and b, the normal vectors at the center of the polygon are different when interpolation takes place along different directions. Notably, this defect is not similar to the rotation-dependent problem indicated by Duff (1979). It occurs even within a triangle, while linear interpolation does not have this defect.

2.3 Subdivision variance

Figure 4a illustrates a serpentine curve approximated by an edge P_0P_1 and two normal vectors. Each subdivided segment presents a new curve, if the normal vector at the point S near the end-point P_1 is determined by quadratic interpolation and the edge is subdivided into two segments at S. After applying quadratic interpolation to these two segments, the results differ from the original curve, especially in the shorter segment SP_1. The normal vectors in the mid-

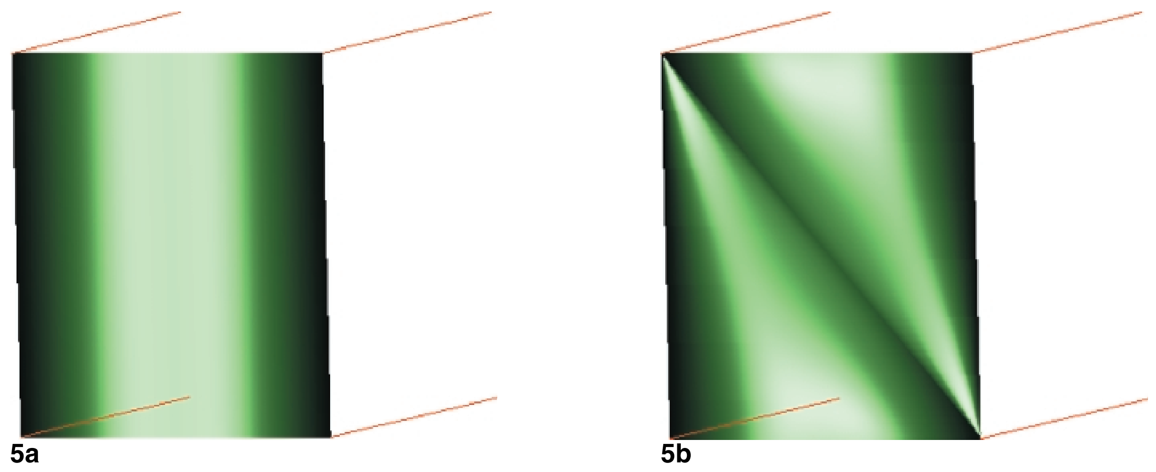


Fig. 5. A shading example showing the defects arising from subdivision variance and non-coplanar error. The *red lines* are the normal vectors at the vertices. **a** ideal surface; **b** original quadratic interpolation

dle of the shorter segment are bent incorrectly toward the opposite direction.

This kind of subdivision always occurs in triangle scan conversion. In Fig. 4b, a quadrilateral polygon is decomposed into two triangles, and each triangle is scan-converted for rendering. The algorithm first determines the boundary normal vectors of the triangle, and then performs interpolation along each scan line. The interpolated normal vectors differ from the normal vectors in quadrilateral scan conversion because of subdivision variance. Figure 5 shows the visual defects after shading. The quadrilateral polygon with the same profile as Fig. 3 is illuminated by a light incident from the left side.

2.4 2-level setup

Definitions:

1-level setup: Scan-conversion process only requires a single setup stage to establish interpolation coefficients for all polygon boundaries and scan-line segments. All current triangle-based graphics systems belong to this setup.

2-level setup: Scan-conversion process requires the interpolation coefficients for polygon boundaries to be established initially. Subsequently, while walking through boundaries, setup operations for each scan-line segment (span) are performed. It is usually used in multi-side polygon rendering, such as in Akeley and Jermoluk (1988).

2-level setup systems have more computational consumption than 1-level setup systems. Furthermore, 2-level setup systems require higher precision interpolators to prevent inconsistencies between adjacent scan-line segments. Figure 6 shows the precision problem illustrated by Phong shading. 1-level and 2-level setup schemes result in Fig. 6a and b, respectively. Both schemes utilize 11-bit fixed-point interpolators for each dimension of the normal vector. The 2-level setup scheme causes shading breaks between scan-line segments until the precision is upgraded to 15 bits.

The quadratic interpolation of van Overveld and Wyvill belongs to the 2-level setup scheme. The coefficients of the quadratic function are recomputed for each scan-line segment. The setup overhead and precision requirement are not efficient.

3 Improved quadratic normal vector interpolation

After analyzing the drawbacks of the quadratic interpolation of van Overveld and Wyvill, we come up with an improved interpolation scheme to eliminate

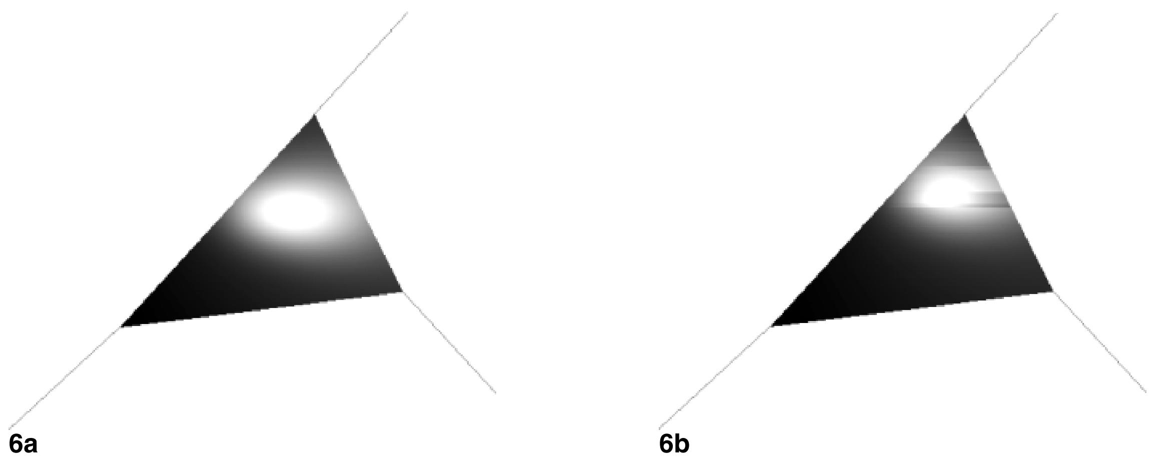


Fig. 6a,b. Precision simulation with 11-bit interpolators for 1-level and 2-level setup. **a** 1-level setup scheme generates ideal result. **b** 2-level setup scheme results in breaks

these drawbacks. First, an adaptive normal vector interpolation is introduced to cure inappropriateness to arch-type curves in their algorithm. Next, the interpolation is restricted to the plane defined by two normal vectors of two end points. Finally, bi-quadratic normal vector interpolation, combined with the previous techniques, is proposed to achieve better quality and more efficient computation.

3.1 Adaptive normal vector interpolation

In Sect. 2.1, we indicate that linear interpolation is generally appropriate to arch-type curves while their quadratic interpolation is appropriate to serpentine-type curves. Accordingly, zigzag detection is proposed to determine whether the curve is serpentine or not.

$$((\boldsymbol{n}_0, \boldsymbol{\Delta}) \geq 0) \oplus ((\boldsymbol{n}_1, -\boldsymbol{\Delta}) \geq 0) \qquad (6)$$

where $\boldsymbol{n}_0$ and $\boldsymbol{n}_1$ are normal vectors of the two end points, $\boldsymbol{\Delta}$ is the edge vector, $\oplus$ denotes logical exclusive OR, and (,) denotes the inner product.

Each term detects whether the normal vector points inward or outward from the perpendicular plane of the edge. If the two terms have opposing signs, the original curve must be serpentine-type and has an inflection point. The adaptive normal vector interpolation is expressed as:

```
if
$\{[((\boldsymbol{n}_0, \boldsymbol{\Delta}) \geq 0)\&\&(\boldsymbol{n}_1, -\boldsymbol{\Delta}) \geq 0)]\|[((\boldsymbol{n}_0, \boldsymbol{\Delta}) \leq 0)$
        $\&\&(\boldsymbol{n}_1, -\boldsymbol{\Delta}) \leq 0)]\}$
    apply linear interpolation for normal vectors
else
    apply quadratic interpolation for normal vectors
```

Although the zigzag detection can determine the type of curves up to the third degree, it may fail in quartic curves or higher degrees with even inflection points. However, curves with degrees higher than cubic are seldom used or approximated by a single polygon. Thus, the zigzag detection remains an efficient method for this adaptive selection.

3.2 Restricting the normal vector interpolation to be coplanar

To eliminate the defect described in Sect. 2.2, the normal vector interpolation should be independent of the direction where interpolation takes place. Let P be the plane defined by normal vectors of two end points. $\boldsymbol{\Gamma}$ is the projection vector of the edge vector $\boldsymbol{\Delta}$ on the plane P. Substituting $\boldsymbol{\Gamma}$ for $\boldsymbol{\Delta}$ in the boundary condition of (4), the following is obtained:

$$\left(\int_0^1 \boldsymbol{n}(s)\,\mathrm{d}s, \boldsymbol{\Gamma} \right) = 0 \qquad (7)$$

With the same derivation, the coefficient becomes

$$\boldsymbol{b} = 3\frac{(\boldsymbol{n}_0 + \boldsymbol{n}_1, \boldsymbol{\Gamma})}{\|\boldsymbol{\Gamma}\|^2}\boldsymbol{\Gamma}. \tag{8}$$

All coefficients $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are in the plane P. Hence, the interpolated normal vectors are all restricted in this plane. This substitution makes the results independent of the interpolation directions in scan conversion.

If two normal vectors point toward the same direction, the plane P will not be determined. The triangle plane normal vector $\boldsymbol{N}_T$ can be further used in this situation. The plane P is now defined by $\boldsymbol{N}_T$ and the normal vector on one of the end points. Another issue is when to use the triangle plane normal vector, $\boldsymbol{N}_T$, instead of the vertex normals. This should be the case when the edge vector is nearly perpendicular to the plane defined by two end point normal vectors. Moreover, if $\boldsymbol{N}_T$ is also the same as the normal vectors of the two end points, the original curve must be a straight line. This condition is already detected by the zigzag detection. Zero-order or linear interpolation is applicable to this condition.

3.3 Bi-quadratic normal vector interpolation

Although the previous two techniques improve the original quadratic interpolation, they still do not apply to standard triangle scan conversion. Hence, bi-quadratic normal vector interpolation is employed to eliminate the defects described in Sects. 2.3 and 2.4 occurring in triangle scan conversion.

The steps of bi-quadratic normal vector interpolation are as follows.

1. Determine the normal vector at the three midpoints of the triangle edges. These three normal vectors are obtained from the improved quadratic normal vector interpolation with the techniques described in Sects. 3.1 and 3.2.
2. Define a bi-quadratic function (9) that evaluates the specified normal vectors at the six sample points. The six sample points are three vertices and three midpoints of the triangle edges. The derivation of coefficients will be described later.

$$\boldsymbol{n}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{a}x^2 + \boldsymbol{b}xy + \boldsymbol{c}y^2 + \boldsymbol{d}x + \boldsymbol{e}y + \boldsymbol{f} \tag{9}$$

3. Derive the coefficients for forward difference along the x axis and the triangle edges.

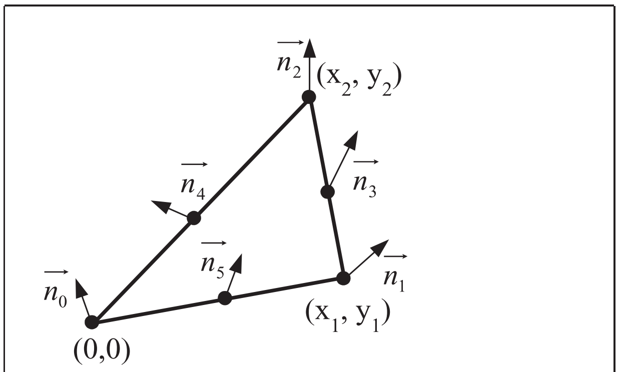


Fig. 7. A triangle with six sample points and their normal vectors to compose a bi-quadratic function

4. Interpolate normal vectors for each pixel in the triangle using forward difference.
5. Normalize each normal vector and perform lighting operations.

Notably, steps 1 to 3 are carried out in the setup stage for one triangle, and steps 4 and 5 are per-pixel operations.

The bi-quadratic function has six vector coefficients uniquely determined by six sample points. For simplification, one vertex is assigned to the origin of the bi-quadratic function. Then, all screen coordinates of other sample points are transformed according to the assignment. In (9), x and y are screen-coordinate parameters related to the new origin. The illustration is shown in Fig. 7.

To evaluate the coefficients, the share terms expression of quadratic color interpolation (Seiler 1998) are adopted. However, to reduce the number of division calculations, they are modified slightly. The share terms are as follows:

$$w_1 = x_1 y_2$$
$$w_2 = x_2 y_1$$
$$w_3 = w_1 - w_2$$
$$\boldsymbol{T} = \boldsymbol{n}_0 + \boldsymbol{n}_1 - 2\boldsymbol{n}_5$$
$$\boldsymbol{U} = \boldsymbol{n}_0 + \boldsymbol{n}_2 - 2\boldsymbol{n}_4$$
$$\boldsymbol{V} = \boldsymbol{n}_0 + \boldsymbol{n}_3 - \boldsymbol{n}_4 - \boldsymbol{n}_5$$
$$\boldsymbol{G} = 2\boldsymbol{n}_5 - 2\boldsymbol{n}_0 - \boldsymbol{T}$$
$$\boldsymbol{H} = 2\boldsymbol{n}_4 - 2\boldsymbol{n}_0 - \boldsymbol{U}$$

Using these share terms for simplification, the coefficients can be solved as

$$\begin{cases} \boldsymbol{a} = \dfrac{2(\boldsymbol{T}y_2^2+\boldsymbol{U}y_1^2-2\boldsymbol{V}y_1y_2)}{w_3^2} \\ \boldsymbol{b} = \dfrac{4[\boldsymbol{V}(w_1+w_2)-\boldsymbol{U}x_1y_1-\boldsymbol{T}x_2y_2]}{w_3^2} \\ \boldsymbol{c} = \dfrac{2(\boldsymbol{U}x_1^2+\boldsymbol{T}x_2^2-2\boldsymbol{V}x_1x_2)}{w_3^2} \\ \boldsymbol{d} = \dfrac{\boldsymbol{G}y_2-\boldsymbol{H}y_1}{w_3} \\ \boldsymbol{e} = \dfrac{\boldsymbol{H}x_1-\boldsymbol{G}x_2}{w_3} \\ \boldsymbol{f} = \boldsymbol{n}_0 \end{cases}$$

The bi-quadratic function is represented by screen coordinates x and y, instead of other coordinate systems, such as barycentric coordinates. The reason is that forward difference for this representation maps directly to scanline-based conversion on the raster display. Scanline-based hardware digital differential analyzer (DDA) can be applied here for scan conversion.

If the results of the zigzag detection in three edges are all arch-type, the bi-quadratic interpolation will be simplified to bilinear interpolation. In this situation, the operations of steps 1 to 4 can be reduced. If only one edge is arch-type, the normal vector at the midpoint of the edge will be generated as the average of the two end point normal vectors in step 1. Substituting the six sample points into the bi-quadratic function, the function values are linear along the edge but quadratic along other directions. Hence, C^0 continuity in normal vectors is maintained between triangles. The situation is analogical when two edges are arch type and one edge is serpentine type.

The drawback in (9) is that it does not include depth information about the triangle. Thus, it cannot simulate perspective foreshortening. This drawback occurs likewise in traditional Phong shading and the original quadratic interpolation, because interpolation is also performed in the screen coordinates. In Sect. 4, perspective correction is used to eliminate this drawback.

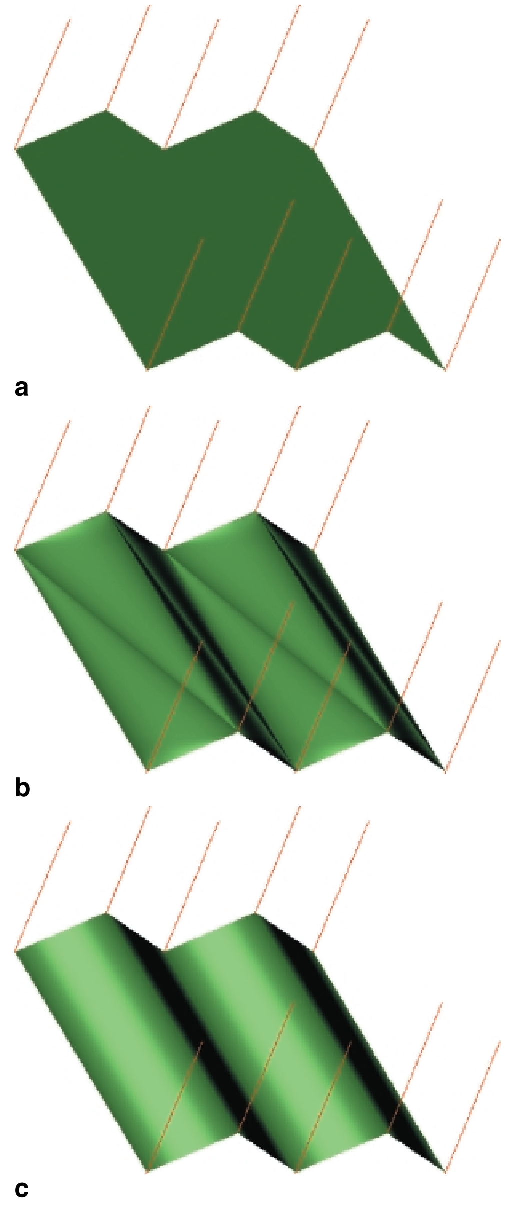


Fig. 8a–c. A surface with a zigzag profile shaded by different interpolation algorithms for normal vectors. **a** Traditional linear interpolation. **b** Original quadratic interpolation. **c** Bi-quadratic interpolation

3.4 Visual results and computation comparison

Visual quality comparison

All the above algorithms have been implemented within the Mesa 3D graphics library (Paul 2000), a famous OpenGL-like 3D graphics library. We add a new shading model into Mesa to increase its capability for Phong shading and bi-quadratic normal vector interpolation. Figure 8 shows the visual comparison between the original algorithms and bi-quadratic algorithm. The tested surface has a zigzag profile, and its vertex normals are computed by averaging the plane normals. The light is incident from

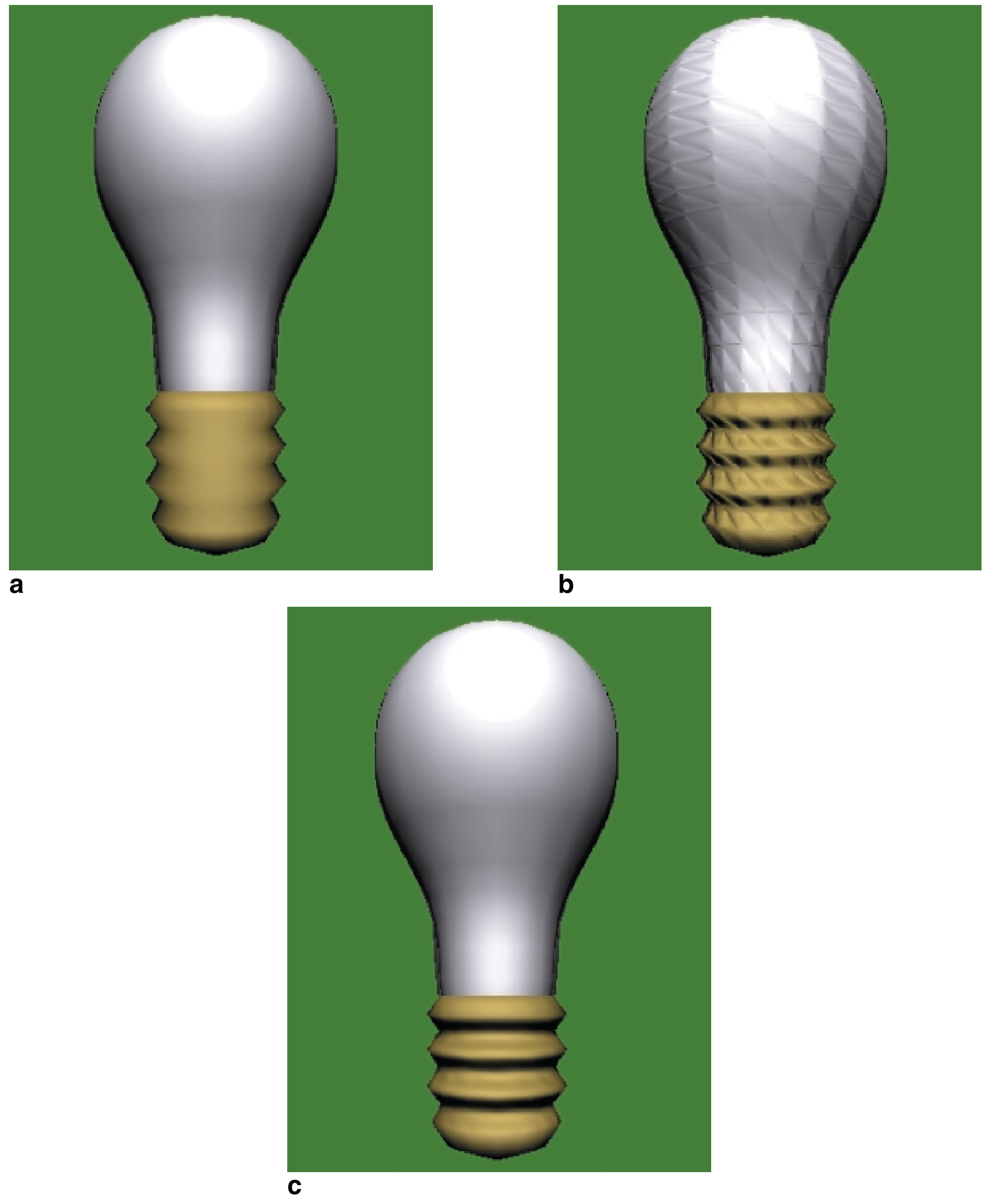


Fig. 9a–c. A bulb model shaded by different interpolation algorithms for normal vectors. **a** Traditional linear interpolation. **b** Original quadratic interpolation. **c** Bi-quadratic interpolation

the left side of the surface at an infinite distance. Gouraud shading and Phong shading produce the same result, namely that the whole surface is painted incorrectly with a single color. The original quadratic interpolation for normal vectors also produces incorrect illumination in triangle scan conversion, while bi-quadratic normal vector interpolation matches the ideal result.

Figure 9 is another example showing the visual improvement of bi-quadratic normal vector interpolation. In triangle scan conversion, the original quadratic interpolation is even worse than Phong

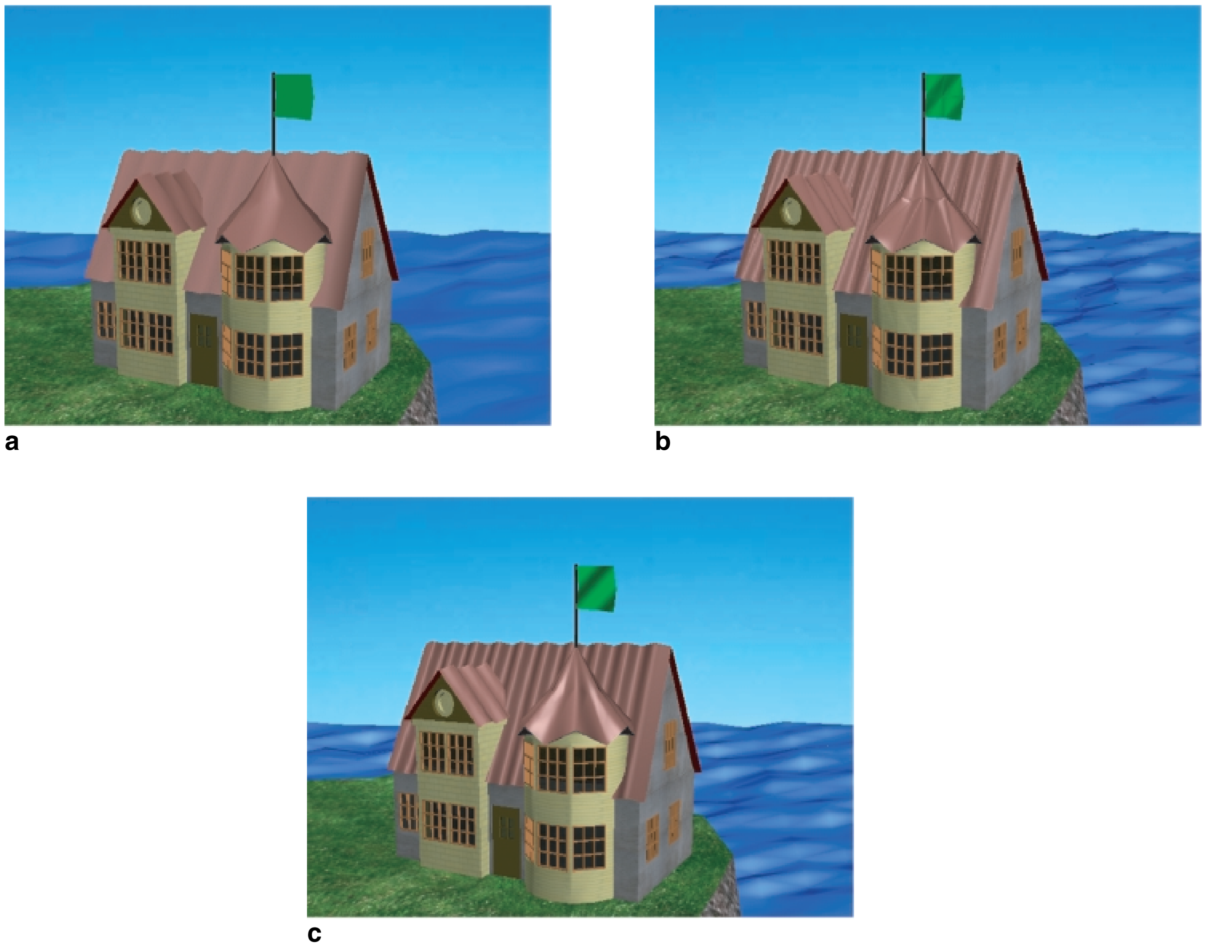


Fig. 10a–c. Another example shaded by different interpolation algorithms for normal vectors. **a** Traditional linear interpolation. **b** Original quadratic interpolation. **c** Bi-quadratic interpolation

shading. Bi-quadratic normal vector interpolation outperforms Phong shading in the base part of the bulb.

Figure 10 shows a more complex scene with different interpolation algorithms. The improvement is clearly evident for the roofing tiles, the waves, and the flag.

Although dense tessellation for the original curve surface may reduce the difference between these algorithms, this approach requires extra storage and computational overhead for additional polygons. By applying our algorithm, the number of necessary polygons can be reduced while keeping identical visual quality. Another point to notice is that the silhouettes of polygons are still not refined. Geometrical-modification methods mentioned in the introduction can be used to refine the silhouettes.

Computation comparison

First, we compare the setup cost between the original quadratic interpolation and bi-quadratic interpolation. Then, we estimate the ratio of computational overhead in bi-quadratic interpolation to the whole shading operations, including per-pixel normalization and illumination.

The original quadratic interpolation needs to perform setup operations not only for three edges but also for each scan line. The number of scan-line segments in a triangle is assumed to be Y. The total setup cost for the original quadratic interpolation consists of $(48 + 16 \times Y)$ multiplications, $(48 + 16 \times Y)$ additions, and $(3 + Y)$ divisions. In bi-quadratic interpolation, the total setup cost comes from steps 1–3. Step 1 requires 111 multiplications, 81 additions, 3 divisions, and 3 reciprocal square roots. Steps 2 and 3 require 75 multiplications, 78 additions, and 3 divisions. Thus, the total setup cost for bi-quadratic interpolation consists of 186 multiplications, 158 additions, 6 divisions, and 3 reciprocal square roots.

In the current hardware architecture, the ratio of execution latency among floating-point multiplication, addition, division, and reciprocal square root is about $1:1:3:3$. If the number of scan-line segments exceeds 7, our approach requires less computational cost than the original quadratic interpolation.

The overhead due to our approach is relatively little in comparison with other shading operations, such as per-pixel normalization and illumination. For example, per-pixel cost for normalization and illumination of a single light source at an infinite distance consists of 18 multiplications, 12 additions, 1 reciprocal square root, and 1 exponential. When the average pixel number in a triangle is 50, the overhead of our approach is only about 26.7% with a consideration of forward difference. If the pixel number is higher or multiple light sources are present, the ratio will drop quickly. Table 1 shows the ratio in detail. It is assumed that zigzag detection does not reduce the number of operations, so all the triangles need to be applied to the entire operations of bi-quadratic interpolation. Furthermore, in considering other rendering effects, such as texture mapping or anti-aliasing, the computational overhead of our approach is nearly negligible.

4 Perspective correction for shading

Since perspective projection does not preserve equi-spaced points along a line, any interpolation performed in screen coordinates must be perspective corrected. Otherwise, the foreshortening effect does not appear. The most visible artifacts occur in texture mapping, and similar errors also occur in shading. The errors in shading are usually less noticeable in still images. However, in animation, the errors may produce obvious color shifts, especially when the triangle is clipped or rotated.

Rational linear interpolation (Heckbert and Moreton 1991) and hyperbolic interpolation (Blinn 1992) have solved perspective foreshortening for linear interpolation across a triangle. That is, define (x, y, z) as a point in eye coordinates, $(\hat{x}, \hat{y}, \hat{z}, \hat{w})$ as the point in homogeneous coordinates after the perspective projection, and $(\tilde{x}, \tilde{y}, \tilde{z})$ as the point in screen coordinates after the w division. The interpolation equation between two end points is:

$$u = \frac{\frac{u_0}{\hat{w}_0} + \beta\left(\frac{u_1}{\hat{w}_1} - \frac{u_0}{\hat{w}_0}\right)}{\frac{1}{\hat{w}_0} + \beta\left(\frac{1}{\hat{w}_1} - \frac{1}{\hat{w}_0}\right)} \tag{10}$$

where β is the interpolation parameter from 0 to 1 in screen coordinates, and u implies any parameter that is linear in eye coordinates, such as texture parameters or positions. Therefore, (10) means linear interpolating $u/\hat{w}$ and $1/\hat{w}$ in screen coordinates and dividing them to get the correct value.

4.1 Perspective correction for bi-quadratic normal vector interpolation

Equation (10) cannot be applied directly to quadratic interpolation. Thus, we extend the derivation of hyperbolic interpolation to construct new perspective-correct equations for bi-quadratic interpolation in triangle scan conversion. Let (x_0, y_0, z_0), (x_1, y_1, z_1),

Table 1. The ratio of overhead in bi-quadratic interpolation to other shading operations

Number of lights / Average number of pixels per triangle	1	2	4	6	8
25	45.7%	27.9%	15.6%	10.9%	8.3%
50	26.7%	16.3%	9.1%	6.4%	4.9%
100	17.2%	10.5%	5.9%	4.1%	3.1%

and (x_2, y_2, z_2) be the three vertices of a triangle in eye coordinates. All points in the triangle are expressed as

$$\begin{cases} x = x_0 + \alpha_1(x_1 - x_0) + \alpha_2(x_2 - x_0) \\ y = y_0 + \alpha_1(y_1 - y_0) + \alpha_2(y_2 - y_0), \\ z = z_0 + \alpha_1(z_1 - z_0) + \alpha_2(z_2 - z_0) \end{cases}$$
$$0 \le \alpha_1,\ \alpha_2 \le 1,\ \text{and}\ \alpha_1 + \alpha_2 \le 1$$

Since projection preserves planes, the points in homogeneous coordinates have the same relationship, as shown in the following:

$$\begin{cases} \hat{x} = \hat{x}_0 + \alpha_1(\hat{x}_1 - \hat{x}_0) + \alpha_2(\hat{x}_2 - \hat{x}_0) \\ \hat{y} = \hat{y}_0 + \alpha_1(\hat{y}_1 - \hat{y}_0) + \alpha_2(\hat{y}_2 - \hat{y}_0) \\ \hat{z} = \hat{z}_0 + \alpha_1(\hat{z}_1 - \hat{z}_0) + \alpha_2(\hat{z}_2 - \hat{z}_0) \\ \hat{w} = \hat{w}_0 + \alpha_1(\hat{w}_1 - \hat{w}_0) + \alpha_2(\hat{w}_2 - \hat{w}_0) \end{cases}$$

In screen coordinates, the step sizes are not the same after the w division. Thus, other interpolation parameters, β_1 and β_2, are imposed.

$$\begin{cases} \tilde{x} = \tilde{x}_0 + \beta_1(\tilde{x}_1 - \tilde{x}_0) + \beta_2(\tilde{x}_2 - \tilde{x}_0) \\ \tilde{y} = \tilde{y}_0 + \beta_1(\tilde{y}_1 - \tilde{y}_0) + \beta_2(\tilde{y}_2 - \tilde{y}_0), \\ \tilde{z} = \tilde{z}_0 + \beta_1(\tilde{z}_1 - \tilde{z}_0) + \beta_2(\tilde{z}_2 - \tilde{z}_0) \end{cases}$$
$$0 \le \beta_1,\ \beta_2 \le 1 \text{ and } \beta_1 + \beta_2 \le 1$$

Consider the equations for the x coordinates. Since

$$\tilde{x}_0 = \frac{\hat{x}_0}{\hat{w}_0}, \tilde{x}_1 = \frac{\hat{x}_1}{\hat{w}_1}, \tilde{x}_2 = \frac{\hat{x}_2}{\hat{w}_2},$$
$$\begin{aligned} \tilde{x} &= \frac{\hat{x}}{\hat{w}} \\ &= \frac{\tilde{x}_0\hat{w}_0 + \alpha_1(\tilde{x}_1\hat{w}_1 - \tilde{x}_0\hat{w}_0) + \alpha_2(\tilde{x}_2\hat{w}_2 - \tilde{x}_0\hat{w}_0)}{\hat{w}_0 + \alpha_1(\hat{w}_1 - \hat{w}_0) + \alpha_2(\hat{w}_2 - \hat{w}_0)} \\ &= \tilde{x}_0 + \frac{\alpha_1\hat{w}_1(\tilde{x}_1 - \tilde{x}_0)}{\hat{w}_0 + \alpha_1(\hat{w}_1 - \hat{w}_0) + \alpha_2(\hat{w}_2 - \hat{w}_0)} \\ &\quad + \frac{\alpha_2\hat{w}_2(\tilde{x}_2 - \tilde{x}_0)}{\hat{w}_0 + \alpha_1(\hat{w}_1 - \hat{w}_0) + \alpha_2(\hat{w}_2 - \hat{w}_0)} \end{aligned}$$

The relations between α_1,α_2 and β_1,β_2 are

$$\begin{cases} \beta_1 = \frac{\alpha_1\hat{w}_1}{\hat{w}_0+\alpha_1(\hat{w}_1-\hat{w}_0)+\alpha_2(\hat{w}_2-\hat{w}_0)} \\ \beta_2 = \frac{\alpha_2\hat{w}_2}{\hat{w}_0+\alpha_1(\hat{w}_1-\hat{w}_0)+\alpha_2(\hat{w}_2-\hat{w}_0)} \end{cases}$$

which are also identical in y and z coordinates.

The relations can be converted to

$$\begin{cases} \alpha_1 = \frac{\beta_1\frac{1}{\hat{w}_1}}{\frac{1}{\hat{w}_0}+\beta_1\left(\frac{1}{\hat{w}_1}-\frac{1}{\hat{w}_0}\right)+\beta_2\left(\frac{1}{\hat{w}_2}-\frac{1}{\hat{w}_0}\right)} \\ \alpha_2 = \frac{\beta_2\frac{1}{\hat{w}_2}}{\frac{1}{\hat{w}_0}+\beta_1\left(\frac{1}{\hat{w}_1}-\frac{1}{\hat{w}_0}\right)+\beta_2\left(\frac{1}{\hat{w}_2}-\frac{1}{\hat{w}_0}\right)} \end{cases} \quad (11)$$

In the raster display, the interpolation in screen coordinates is written as

$$\begin{cases} \tilde{x} = \tilde{x}_0 + \mu_x \\ \tilde{y} = \tilde{y}_0 + \mu_y \end{cases}$$

where μ_x and μ_y are integer numbers.
Representing μ_x and μ_y with β_1 and β_2, we obtain

$$\begin{cases} \mu_x = \beta_1(\tilde{x}_1 - \tilde{x}_0) + \beta_2(\tilde{x}_2 - \tilde{x}_0) \\ \mu_y = \beta_1(\tilde{y}_1 - \tilde{y}_0) + \beta_2(\tilde{y}_2 - \tilde{y}_0) \end{cases}$$

Or, converting the above equations, we obtain

$$c = (\tilde{x}_1 - \tilde{x}_0)(\tilde{y}_2 - \tilde{y}_0) - (\tilde{x}_2 - \tilde{x}_0)(\tilde{y}_1 - \tilde{y}_0)$$
$$\begin{cases} \beta_1 = \frac{\mu_x(\tilde{y}_2-\tilde{y}_0)-\mu_y(\tilde{x}_2-\tilde{x}_0)}{c} \\ \beta_2 = \frac{\mu_y(\tilde{x}_1-\tilde{x}_0)-\mu_x(\tilde{y}_1-\tilde{y}_0)}{c} \end{cases}$$

Substituting β_1 and β_2 into (11), we obtain

$$\begin{cases} \alpha_1 = \frac{\frac{1}{c}\frac{(\tilde{y}_2-\tilde{y}_0)}{\hat{w}_1}\mu_x+\frac{1}{c}\frac{(\tilde{x}_0-\tilde{x}_2)}{\hat{w}_1}\mu_y}{\frac{1}{\hat{w}_0}+\mu_x\frac{\mathrm{d}\left(\frac{1}{\hat{w}}\right)}{\mathrm{d}x}+\mu_y\frac{\mathrm{d}\left(\frac{1}{\hat{w}}\right)}{\mathrm{d}y}} \\ \alpha_2 = \frac{\frac{1}{c}\frac{(\tilde{y}_0-\tilde{y}_1)}{\hat{w}_2}\mu_x+\frac{1}{c}\frac{(\tilde{x}_1-\tilde{x}_0)}{\hat{w}_2}\mu_y}{\frac{1}{\hat{w}_0}+\mu_x\frac{\mathrm{d}\left(\frac{1}{\hat{w}}\right)}{\mathrm{d}x}+\mu_y\frac{\mathrm{d}\left(\frac{1}{\hat{w}}\right)}{\mathrm{d}y}} \end{cases} \quad (12)$$

where

$$\frac{\mathrm{d}\left(\frac{1}{\hat{w}}\right)}{\mathrm{d}x} = \frac{\left(\frac{1}{\hat{w}_1} - \frac{1}{\hat{w}_0}\right)(\tilde{y}_2 - \tilde{y}_0) - \left(\frac{1}{\hat{w}_2} - \frac{1}{\hat{w}_0}\right)(\tilde{y}_1 - \tilde{y}_0)}{c}$$

and

$$\frac{\mathrm{d}\left(\frac{1}{\hat{w}}\right)}{\mathrm{d}y} = \frac{\left(\frac{1}{\hat{w}_2} - \frac{1}{\hat{w}_0}\right)(\tilde{x}_1 - \tilde{x}_0) - \left(\frac{1}{\hat{w}_1} - \frac{1}{\hat{w}_0}\right)(\tilde{x}_2 - \tilde{x}_0)}{c}.$$

The numerators and denominators of α_1 and α_2 are linear functions in screen coordinates. In addition, these denominators are equal to the denominators of perspective-correct equations for linear interpolation in triangle scan conversion. The computational overhead is much reduced because of calculation sharing. The bi-quadratic function should thus change to the expression involving α_1 and α_2, as shown in (13).

$$\boldsymbol{n}(\alpha_1, \alpha_2) = \boldsymbol{a}\alpha_1^2 + \boldsymbol{b}\alpha_1\alpha_2 + \boldsymbol{c}\alpha_2^2 + \boldsymbol{d}\alpha_1 + \boldsymbol{e}\alpha_2 + \boldsymbol{f}. \tag{13}$$

Therefore, the coefficients become

$$\begin{cases} \boldsymbol{a} = 2\boldsymbol{n}_1 + 2\boldsymbol{n}_0 - 4\boldsymbol{n}_5 \\ \boldsymbol{b} = 4\boldsymbol{n}_3 + 4\boldsymbol{n}_0 - 4\boldsymbol{n}_4 - 4\boldsymbol{n}_5 \\ \boldsymbol{c} = 2\boldsymbol{n}_2 + 2\boldsymbol{n}_0 - 4\boldsymbol{n}_4 \\ \boldsymbol{d} = 4\boldsymbol{n}_5 - \boldsymbol{n}_1 - 3\boldsymbol{n}_0 \\ \boldsymbol{e} = 4\boldsymbol{n}_4 - \boldsymbol{n}_2 - 3\boldsymbol{n}_0 \\ \boldsymbol{f} = \boldsymbol{n}_0. \end{cases}$$

Finally, the perspective correction for bi-quadratic normal vector interpolation can be achieved via (12) and (13) in triangle scan conversion.

4.2 Perspective correction for light vector and view vector

The illumination model utilizes the cosine of the angle between the normal vector and light vector for diffuse reflection, as well as the cosine of the angle between the normal vector and halfway vector for specular reflection. The halfway vector is the bisector of the light vector and view vector. For local light sources, the light vectors should be linearly interpolated across the triangle. Similarly, for a local viewpoint, the view vectors should also be linearly interpolated across the triangle. When the view projection is perspective, all light and view vectors need to be perspective corrected with (10). The division in (10) is usually approximated by a reciprocal and a multiplication. However, if there are multiple local light sources, it is better to interpolate the eye coordinates of pixel positions with perspective correction, instead of individual light vectors and view vectors. Their correct vectors are then produced by subtracting the positions of light sources or the viewpoint. This will save several per-pixel multiplications in the division operations.

Figure 11 is an example showing the improvement with perspective correction. The rectangle, which approximates a portion of a cylinder surface, begins near the viewpoint and extends to the far end. The shading effect generates a light band on the rectangle. Notably, the light band without perspective correction bends in the center, while the one with perspective correction is perfectly straight.

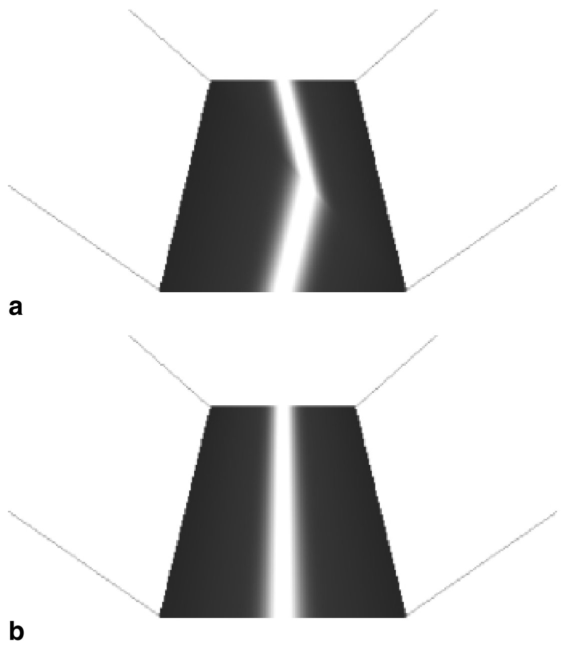


Fig. 11a,b. The improvement of perspective correct interpolation. **a** Original version. **b** Perspective correct version

5 Conclusion

We have presented an improved quadratic normal vector interpolation. Our algorithm eliminates the shading artifacts in traditional linear interpolation and original quadratic interpolation. The most significant benefit is to avoid dense tessellation of curve surfaces, which requires large storage capacity and mass computation. The improved quadratic interpolation requires little computation overhead. At the beginning of interpolation operations, the zigzag detection in the algorithm can decide dynamically whether to apply linear or quadratic interpolation. Moreover, when the triangle should be shaded by quadratic interpolation for normal vectors, the improved quadratic interpolation has less computation complexity than the original quadratic interpolation for eight or more scan-line segments. In addition, we have presented a perspective correction method for our bi-quadratic interpolation. With our algorithm, the shading effect can be more realistic.

Acknowledgements. The authors would like to acknowledge the valuable suggestions from Wen-Chang Yeh and Kun-Bin Lee as well as the reviewers. This work was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC89-2215-E009-052.

Appendix

We use the same assumption and notation herein as van Overveld and Wyvill (1997b) did. In their paper, they claimed that except for the case of mirror-symmetric normal vectors at the two opposite end points of an edge, no curve passing through the end points is perpendicular to the normals and has linearly changing tangents. Although their claim is correct, their inference contains some flaws. First, they proved that linearly interpolated tangents could not match a smoothly varying curve, except for the mirror-symmetric case. Then, they immediately concluded that no surface would yield the normal vector distribution used by Phong shading, except for the mirror-symmetric case. However, it is quite obvious that all quadratic curves have linear tangents, if the tangents at the end points are calculated directly from the curves. The inconsistency of Phong shading is simply because the normal vectors at the vertices are always forced to unit length.

We correct their inference in the following. First, we verify that linearly interpolated tangent vectors are appropriate for all quadratic curves, not only those that are mirror-symmetric. Then, we prove that linearly interpolated normal vectors are suited only to mirror-symmetric cases because the normal vector length at the vertices is set to a unit length.

Interpolation is assumed herein to take place in 2D, as van Overveld and Wyvill did in their paper, so that a normal vector determines the direction of a tangent vector uniquely. To facilitate the analysis, tangent vector interpolation occurs over the edge from $(0, 0)$ to $(1, 0)$. However, they used an explicit curve function in their inference. The first-order derivative of the function is a tangent slope, instead of a tangent vector. The inference is hence unable to prove the mistake of linear interpolation. We use parametric representation here to correct their inference.

The tangent vectors are denoted as $\boldsymbol{\phi}(t)$. The curve with a smoothly varying profile is represented by

$$\Phi(t) = \int_{\xi=0}^{t} \boldsymbol{\phi}(\xi)\,\mathrm{d}\xi.$$

The tangent vectors at the two end points are given as $\boldsymbol{\phi}_0$ and $\boldsymbol{\phi}_1$ by first-order differentiation. For linear interpolation, the tangent vectors along the curve are

$$\boldsymbol{\phi}(t) = \boldsymbol{\phi}_0 + t(\boldsymbol{\phi}_1 - \boldsymbol{\phi}_0).$$

Since the curve passes through two end points $(0, 0)$ and $(1, 0)$ with linearly interpolated tangent vectors, we obtain

$$\Phi(1) - \Phi(0) = (1, 0) = \int_{\xi=0}^{1} [\boldsymbol{\phi}_0 + \xi(\boldsymbol{\phi}_1 - \boldsymbol{\phi}_0)]\,\mathrm{d}\xi.$$

After evaluating the integral, the results become

$$\Phi(1) - \Phi(0) = (1, 0) = \text{edge vector} = \frac{1}{2}(\boldsymbol{\phi}_0 + \boldsymbol{\phi}_1).$$

Obviously, the above condition is satisfied by all quadratic curves. The fact that van Overveld and Wyvill (1997b) used tangent slope in their inference already restricts the x coordinate of the tangent vectors to 1. As a result, they only noticed the mirror-symmetric cases.

Next, we analyze in which condition the linearly interpolated normal vectors will match a quadratic curve.

The two normal vectors are perpendicular to the tangent vectors at the two end points. The corresponding inner products of the two vector pairs are zero.

$$(\boldsymbol{n}_0, \boldsymbol{\phi}_0) = 0 \text{ and } (\boldsymbol{n}_1, \boldsymbol{\phi}_1) = 0.$$

Since the linearly interpolated normal vectors match the curve, we obtain

$$((1-t)\boldsymbol{n}_0 + t\boldsymbol{n}_1, (1-t)\boldsymbol{\phi}_0 + t\boldsymbol{\phi}_1) = 0, \quad \text{for all } t.$$

After rearrangement,

$$(1-t)^2(\boldsymbol{n}_0, \boldsymbol{\phi}_0) + t(1-t)(\boldsymbol{n}_0, \boldsymbol{\phi}_1) + t(1-t)(\boldsymbol{n}_1, \boldsymbol{\phi}_0) + t^2(\boldsymbol{n}_1, \boldsymbol{\phi}_1) = t(1-t)[(\boldsymbol{n}_0, \boldsymbol{\phi}_1) + (\boldsymbol{n}_1, \boldsymbol{\phi}_0)] = 0.$$

Thus, the linearly interpolated normal vectors match a quadratic curve, if and only if $(\boldsymbol{n}_0, \boldsymbol{\phi}_1) + (\boldsymbol{n}_1, \boldsymbol{\phi}_0) = 0$.
Let the angle between $\boldsymbol{\phi}_0$ and $\boldsymbol{\phi}_1$ be θ. Therefore, the above condition can be rewritten as

$$\|\boldsymbol{n}_0\|\,\|\boldsymbol{\phi}_1\| \cos\left(\theta+\frac{\pi}{2}\right)+\|\boldsymbol{n}_1\|\,\|\boldsymbol{\phi}_0\| \cos\left(\theta-\frac{\pi}{2}\right)=0$$

or

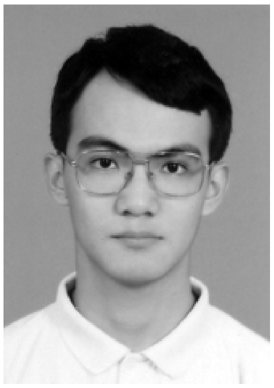
$$\|\boldsymbol{n}_0\|\,\|\boldsymbol{\phi}_1\| \cos\left(\theta-\frac{\pi}{2}\right)+\|\boldsymbol{n}_1\|\,\|\boldsymbol{\phi}_0\| \cos\left(\theta+\frac{\pi}{2}\right)=0$$

If $\|\boldsymbol{n}_0\| = \|\boldsymbol{n}_1\| = 1$, the only solution is $\|\boldsymbol{\phi}_0\| = \|\boldsymbol{\phi}_1\|$, which implies mirror-symmetric cases for quadratic curves. Conversely, provided that we can set $\|\boldsymbol{n}_0\| = \|\boldsymbol{\phi}_0\|$ and $\|\boldsymbol{n}_1\| = \|\boldsymbol{\phi}_1\|$, the linearly interpolated normal vectors will correctly match a quadratic curve in all situations. However, the exact length of the tangent vector is not easily available in the shading stage. Furthermore, the vertex normal vectors in 3D meshes are always generated by the weighted average of neighboring face normals. It can be said that linearly interpolated normal vectors correspond to an existing surface only in mirror-symmetric cases. Fortunately, the highlight effect in Phong shading is very similar to the ideal effect in quadratic surfaces.

References

1. Akeley K, Jermoluk T (1988) High performance polygon rendering. Comput Graph 22(4):239–246
2. Bishop G, Weimer DM (1986) Fast Phong shading. Comput Graph 20(4):103–106
3. Blinn J (1992) Jim Blinn's corner: hyperbolic interpolation. IEEE Comput Graph Appl 12(4):89–95
4. Catmull E, Clark J (1978) Recursively generated B-spline surfaces on arbitrary topological meshes. Comput Aided Des 10(6):350–355
5. Duff T (1979) Smoothly shaded renderings of polyhedral objects on raster displays. Comput Graph 13(2):270–275
6. Foley J, van Dam A, Feiner S, Hugues J (1996) Computer graphics principles and practice second edition in C. Addison Wesley, Massachusetts, pp 739–741
7. Gouraud H (1971) Continuous shading of curved surfaces. IEEE Trans Comput 20(6):623–629
8. Heckbert PS, Moreton HP (1991) Interpolation for polygon texture mapping and shading. In: Rogers DF, Earnshaw RA (ed) State of the art in computer graphics: visualization and modeling. Springer, New York, pp 101–111
9. Kuijk A, Blake EH (1989) Fast Phong sShading via angular interpolation. Comput Graph Forum 8:315–324
10. Lathrop O, Kirk D, Voorhies D (1992) Quadratic interpolation for shaded image generation. US patent no. 5109481
11. Lee YC, Jen CW (2000) On-line polygon refining using a low computation subdivision algorithm. In: Martin R, Wang W (eds) Proceedings of Geometric Modeling and Processing 2000, Hong Kong. IEEE Computer Society Press, Los Alamitos, pp 209–219
12. Max N (1989) Smooth appearance for polygonal surfaces. Visual Comput 5:160–173
13. van Overveld CWAM, Wyvill B (1997a) An algorithm for polygon subdivision based on vertex normals. In: Van Reeth F, Vince J (eds) Proceedings of IEEE Computer Graphics International 1997, Belgium. IEEE Computer Society Press, Los Alamitos, pp 3–12
14. van Overveld CWAM, Wyvill B (1997b) Phong normal interpolation revisited. ACM Trans Graph 16(4):397–419
15. Paul B (2000) The Mesa 3D Graphics Library. Url: http://www.mesa3d.org/
16. Phong BT (1975) Illumination for computer generated pictures. Commun ACM 18(6):311–317
17. Seiler L (1998) Quadratic interpolation for near-Phong quality shading. Comput Graph: conference abstracts and applications 1998, pp 268
18. Shantz M, Lien SL (1987) Shading bicubic patches. Comput Graph 21(4):189–196

Photographs of the authors and their biographies are given on the next page.



Yuan-Chung Lee received the B.S. degree in electronics engineering from National Chiao Tung University in 1997. He is currently pursuing the Ph.D. degree at National Chiao Tung University. His research interests include real-time rendering, graphics architecture, VLSI system, and digital IC design.

Chein-Wei Jen received the B.S. degree from National Chiao Tung University, Hsinchu, Taiwan, in 1970, the M.S. degree from Stanford University, Stanford, CA, in 1977, and the Ph.D. degree from National Chiao Tung University in 1983. He is currently with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, as a Professor. During 1985-1986, he was with the University of Southern California at Los Angeles as a Visiting Researcher. His current research interests include VLSI design, digital signal processing, processor architecture, and design automation. Dr. Jen is a member of the IEEE and of Phi Tau Phi.