Phong Normal Interpolation Revisited

Article *in* ACM Transactions on Graphics · October 1997

DOI: 10.1145/263834.263849 · Source: doi.acm.org

Brian Wyvill
University of Victoria

Phong Normal Interpolation Revisited

C. W. A. M. VAN OVERVELD
Eindhoven University of Technology
and
B. WYVILL
University of Calgary

Phong shading is one of the best known, and at the same time simplest techniques to arrive at realistic images when rendering 3D geometric models. However, despite (or maybe due to) its success and its widespread use, some aspects remain to be clarified with respect to its validity and robustness. This might be caused by the fact that the Phong method is based on geometric arguments, illumination models, and clever heuristics. In this article we address some of the fundamentals that underlie Phong shading, such as the computation of vertex normals for nonmanifold models and the adequacy of linear interpolation and we apply a new interpolation technique to achieve an efficient and qualitatively improved result.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: three-Dimensional Graphics and Realism—*color, shading, shadowing, and texture*

General Terms: Algorithms

Additional Key Words and Phrases: Computer graphics, geometric modeling, rendering, shading

1. INTRODUCTION

Computer graphics, when viewed as a technical discipline, is relatively young. Compared with mathematics or physics, for example, not much tradition has emerged as yet as to the directions in which specialized research will develop. An ongoing stream of new ideas and methods of a wide variety are being generated by workers in the field, whereas the eventual fruitfulness of these ideas and methods may only show up later. Only a few of the results that have emerged so far may be classified as paradigms, in that they are sufficiently mature and well established to

Authors' addresses: C. W. A. M. van Overveld, Eindhoven University of Technology, Department of Mathematics and Computing Science, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands; email: ⟨wsinkvo@info.win.tue.nl⟩; B. Wyvill, University of Calgary, Department of Computer Science, 2500 University Drive N.W., Calgary, Alberta, Canada, T2N 1N4; email: ⟨blob@cpsc.ucalgary.ca⟩.

continue as a template for further development. Among these established methods are concepts such as the Z-buffer algorithm, the use of parametric surfaces, ray-tracing, and Phong shading.

Identifying a paradigm in retrospect is not too hard; however, describing it at a sufficient level of abstraction in order to catch its crucial concept often requires some insight. This may help us to understand the cause for the success of that paradigm, and it helps in exploring new applications, alternatives, or ramifications.

For instance, the concept of Phong shading [Phong 1975] is introduced (e.g., in Foley et al. [1990], p. 738) as mostly an extension of Gouraud shading [Gouraud 1971] where one of the flaws of the latter is corrected, in that highlights in the interiors of polygons can be reproduced. So at the lowest level of abstraction, Phong shading could be seen as a method for the synthesis of highlights. However true, this formulation would do no justice to the creative process that gave rise to Phong shading as it is currently used in most rendering systems. First of all, the notion "Phong shading" refers to both an illumination model and an interpolation scheme for normal vectors. The illumination model relates the illumination in a point P to the local normal vector n_P in that point, and the interpolation scheme defines n_P in terms of the normal vectors in the vertices of the polygon to which P belongs. The observations, essential for devising Phong shading, are:

(1) the visual appearance of an object is not only determined by its geometrical shape but also by the distribution of reflected light;

(2) the amount of light that is reflected from a surface segment depends chiefly on the local surface normal;

(3) A C^0 (continuous) normal vector field matches with a C^1 (differentiable) surface geometry;

(4) piecewise linear interpolation is a simple way to achieve a C^0 vector field; and

(5) decoupling the orientation of the surface normals and the true shape of the underlying geometrical model may cause a simple polyhedral model to produce a visual appearance of a smoothly curved object, which otherwise could only be achieved with a much denser polygonal mesh.

Summarizing (1) to (5) yields an even more abstract formulation of the quintessence of Phong shading: normal vectors can be used to enhance the visual appearance of geometrical objects. Notice that in the latter formulation, we have abstracted away from some ingredients of the more commonly used meaning of Phong shading. The following points concern this abstraction.

(1) The notion of visual appearance does not necessarily refer to highlights only, but may include other aspects of illumination;

(2) apart from linearly interpolating normal vectors, there might be other means for achieving the light reflection distribution of a (piecewise) C^1 surface geometry;

(3) it is not necessary to refrain from geometrical modifications of the underlying surface geometry in order to enhance its visual appearance, or, stated differently: using the information of the normal vectors it might be possible to come up with a somewhat adapted geometry that matches better with a valid smooth object; and

(4) "enhancement" is not necessarily equivalent to "smoothing": alternative geometric operations may be thought of, parameterized by normal vectors, that generate more involved geometries starting from simple polyhedral objects.

The current article discusses some topics in relation to the items (1) and (2). Of course, many authors already have elaborated on Phong shading, methods for improving it, and have explored different applications for normal vectors in the context of rendering. Section 2 deals with this previous work. In Section 3 the question of where vertex normals come from is addressed, this question being fundamental to whatever interpolation or geometric modification technique will be used afterwards. Among other things, an algorithm for the computation of vertex normals for nonmanifold polyhedral models is presented. Next, in Section 4, an alternative to linear interpolation of normal vectors is investigated. A summary and the conclusions of this article appear in Section 5.

2. PREVIOUS WORK

Normal vectors have been used to enhance the visual appearance of (mostly polyhedral) geometrical objects in numerous ways ever since the introduction of Phong shading. Perhaps the best known example is bump mapping (introduced by Blinn [1978]) but also displacement mapping [Cook 1984] and environment mapping [Blinn and Newell 1976] may be thought of as part of the Phong heritage. With the introduction of refraction in raytracing [Whitted 1980], normals not only serve to compute the amount of reflected light, but also to compute the direction of the refraction rays.

Also in modeling, normal vectors play a crucial role: for example, in the polyhedron modeling system of Allan et al. [1989] modifications of a polygon mesh can be made by shifting vertices in a direction normal to the mesh. More recently, with the advent of constraint-based modeling systems (e.g., the Mallet system [Mallet 1992]), normal vectors serve as a shaping device in that a surface may be required to be perpendicular to a given direction in a given point. The use of normal vectors, or rather tangent vectors, in specifying tensor product surfaces has been well known since the introduction of Hermite patches [Boehm et al. 1984].

Some authors have questioned the adequacy of the traditional two-fold linear interpolation of normal vectors while scan-converting polygons for rendering. Shoemake [1985] recommends the use of quaternion rotation, rather than linear interpolation for the interpolation for orientations, to arrive at a smoother appearance; Duff [1979] notes that the traditional two-fold linear interpolation is not affine invariant which causes highlights to move in a nonrealistic manner over rotating objects. Duff describes a method based on barycentric coordinates for curing this flaw. Also the necessity of repeated renormalization of normal vectors has been subject to investigation; Bishop and Weimer [1986] give an algorithm that circumvents this costly operation.

3. WHERE THE VERTEX NORMALS ORIGINATE

For polyhedral models, considered as a stream of polygons, vertex normals are often calculated as a process separate from Phong shading. Phong shading normally consists of, for every scanline, linear interpolation of vertex normals to produce edge normals at the extremes of that scanline, and subsequently, for every pixel, interpolating between these edge normals to find the normal vector in the pixel. Indeed, providing for normal vectors in the vertices can be argued to be part of the modeling process.

Especially in the case of CSG modeling [Requicha 1980] or other types of modeling that are based on assembling a scene from instantiations of a limited (parameterized) set of geometrical polyhedral primitives, generating the normal vectors together with the locations of the vertices is the most obvious thing to do, provided that the geometrical primitives are closed shapes. For example, for a sphere, irrespective of the number of polygons that is used to approximate its shape, the normal vectors are generally all assumed to point in radial directions.

Even if the geometric model is not assembled from a set of closed polyhedra, normal vector assignments can be rather straightforward. Consider, for example, a geometrical model that is the isovalue surface of a well-chosen potential function (as in Wyvill and Wyvill [1989]). Here it is guaranteed that the surface is orientable; moreover, in a polyhedral approximation of such a surface, every vertex is completely surrounded by polygonal facets that are all part of the surface. Then a normal vector can be assigned to every vertex by appropriate averaging [Rogers and Adams 1976] of the normal vectors of the surrounding facets. (There is, however, a problem here: it is not a priori clear what weights should be applied to the normals when averaging.)

Also in the case where the polygons are consistently oriented (all clockwise or all counterclockwise), assigning normal vectors is a straightforward task, and the standard algorithms such as in Newman and Sproull [1979] and Rogers and Adams [1976] are perfectly adequate.

It is not always possible, however, to guarantee consistent orientation for an arbitrary polygon mesh. First of all, if the polygon mesh is not a manifold, there may not exist one global consistent orientation; as we discuss later, it may be necessary then to identify several subsets of the mesh that are manifolds and orient each manifold individually. Second, even if the polygon mesh represents one orientable manifold, globally consistent orientation may be hard to guarantee a priori, depending on the modeling technique used. For instance, in a free-hand 3D geometric design environment where a user may add vertices, edges, and polygons in an arbitrary order, nothing is known about the relative orientation of these polygons, and achieving a globally consistent orientation requires a separate processing step.

This is the situation we consider in the current section: the geometric model studied here consists of a set of vertices and a set of faces; a face is just a cyclic list of vertices. This means that nothing is known a priori of the topology: an edge of a face may be shared not only by two but also by one, three, or more faces.[1]

The algorithm suggested here for obtaining vertex-normal vectors consists of three passes. These are explained intuitively; a (somewhat more) formal definition of the geometry is introduced, and the second and third passes of the algorithm are described in terms of the formal definitions.

3.1 Intuitive Description

Normal vector interpolation between adjacent faces may only be expected to yield an intuitively clear result if these faces share exactly one edge where this edge is not shared by any other face. Such a pair of faces, say f_1 and f_2 are called *good neighbors*, abbreviated by $f_1 \bowtie f_2$. The relation $\bowtie$ induces a transitive closure $\overset{*}{\bowtie}$. Here $f_1 \overset{*}{\bowtie} f_2$ means that either

$$f_1 \bowtie f_2$$

or

$$f_3 \colon f_1 \bowtie f_3 \colon f_3 \overset{*}{\bowtie} f_2 .$$

In other words, the relation $f_1 \overset{*}{\bowtie} f_2$ exists if f_1 and f_2 are connected eventually via a series of $\bowtie$ relationships. Note that if f_1 and f_2 share an edge, say e, and also $f_1 \overset{*}{\bowtie} f_2$, it is still possible that a third face f_3 is incident with e and at the same time $f_1 \overset{*}{\bowtie} f_3$ and $f_2 \overset{*}{\bowtie} f_3$. So in this case $f_1 \bowtie f_2$ does not hold. This happens if the surface "loops back," as one of the configurations in Figure 1 illustrates.

The relation $\overset{*}{\bowtie}$ is symmetric, reflexive, and transitive; it causes the set of faces to be partitioned into equivalence classes. Normal vector interpolation between two adjacent polygons that share an edge is allowed provided that they belong to the same equivalence classes. In Figure 1, a schematic configuration is depicted that gives rise to several equivalence classes.

If it were only for establishing if normal vector averaging between two adjacent polygons is allowed, the actual construction of the equivalence classes could be omitted. There is a second reason, however, which makes it very convenient still to construct the equivalence classes. Indeed, computing a normal vector for a polygon leaves its sign undetermined. In order to

[1] One can adopt a somewhat liberal point of view towards edges where more than two faces meet. One possibility is to argue that normal vector averaging is allowed at such edges provided that the smooth surface which is being simulated is differentiable. Since this would significantly complicate a quantitative analysis of the normal vector interpolation, we stick to the more strict point of view that normal vector averaging is only allowed at edges where two faces meet.

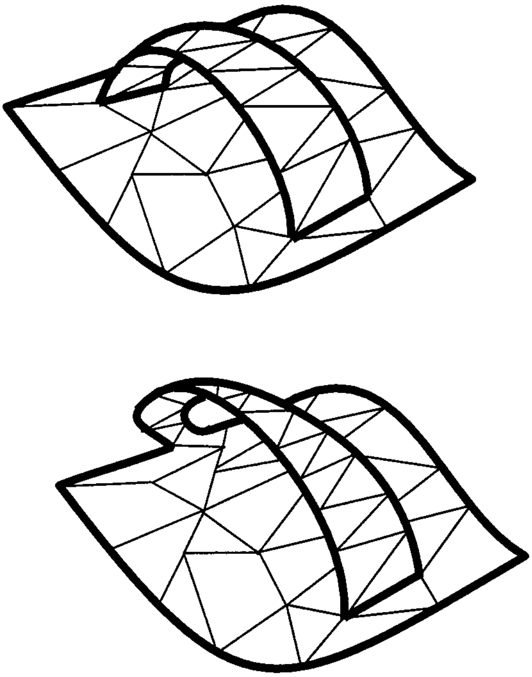

Fig. 1(a)—top: a polygon mesh that gives rise to two equivalence classes; Fig. 1(b)—bottom: a polygon mesh that gives rise to one equivalence class. (Notice the three-face edge.)

average the normal vectors of two adjacent polygons, however, the normal vectors of these two faces have to be consistently oriented. Now when using an algorithm that builds an equivalence class starting from one polygon and next recursively searches all adjacent polygons for good neighbors, normal vector orientation can be done at the same time. Based on the preceding considerations, a first estimate of a three-pass algorithm for computing vertex normals looks like the following.[2]

(1) Read the geometric model; for all faces, compute the normal vector (apart from its sign);

(2) partition the set of faces into equivalence classes and come up with a consistent orientation for all normal vectors within each equivalence class;

(3) for all faces, loop over their vertices; for each vertex, find the neighboring faces that belong to the same equivalence class and average the normal vectors of these faces; assign the resulting normal vector to this vertex; output the vertices and normal vectors for each face. If we find a vertex where faces from different equivalence classes meet, normal vector averaging is only performed with the normals from faces that

[2] In many systems, the cyclic order of the vertices defines the sign of the surface normal vector direction. The algorithm discussed here makes no such assumption.

belong to the equivalence class of the current face. So in this case, one vertex will receive as many different normal vectors as there are different equivalence classes among the incident faces of that vertex.

Before turning to a more detailed description, some remarks should be made.

(1) Few as possible requirements should be imposed onto the input geometry. This means, for example, that the method should still yield reasonable results in the case of moderately warped polygons (a "moderately warped polygon" is a polygon where the vertices do not lay in a plane, but the distance between the vertices and a best-fit plane is not too large).

(2) The method used here for recursively orienting all normal vectors within an equivalence class assumes that the associated (piecewise polygonal) surface is orientable. If this is not the case (e.g., if the geometry models a Moebius ring), a priori orienting the normal vectors cannot work.[3] Instead, the much more elaborate method needs to be used of orienting the normal vectors each time when computing the vertex normals, that is, during pass 3 of the algorithm.

(3) Assuming that polygons may have nongeometrical attributes as well (e.g., color, transparency, texture, . . .) it may make sense to strengthen the meaning of $\bowtie$ in that $f_1 \bowtie f_2$ only holds if f_1 and f_2 also have the same attribute set. A similar way to strengthen the meaning of $\bowtie$ is by imposing a maximum angle between the normal vectors of two adjacent polygons for allowing normal vector averaging.

(4) As we observed before, the recursive construction of the equivalence classes does not guarantee the absence of edges that are shared by three or more faces belonging to the same equivalence class. This and similar "pathological" cases are dealt with during the third pass of the algorithm. Fortunately, pass 3 is much more local in nature than pass 2 in that it is concerned with vertices in isolation rather than with the entire geometric structure. This means that the computational burden, involved with checking for pathological cases, is mostly quite modest.

3.2 Some Definitions

To give a more detailed version of the preceding algorithm, it is convenient to introduce some notation first. Algorithms are written in a combination of C-language (extended with the **listof**-keyword) and conventional set-theory notation. Some extensions that are used are as follows:

Let S be a set, then $|S|$ denotes its number of elements;

[3] The second pass of the algorithm for constructing the equivalence classes, which also computes a consistent orientation, can detect if the surface is nonorientable; if we are about to assign a given orientation, say upwards, to a polygon that has earlier received the other orientation (downwards), we conclude that the surface is nonorientable.

Let L be a list (= an ordered set), then $L.i$ is the ith element, $0 \leq i \leq |L| - 1$;

Looping over all elements of a set S and assigning these to the dummy variable a is denoted by **for** $(a \in S)$;

Selecting an element of a set S and assigning it to a is denoted by $a :\in S$;

Joining two sets S_1 and S_2 while maintaining the order is denoted by $S_1 \sqcup S_2$. This applies as well to lists.

The types that are used are:

```
vector      (float x, y, z;)          /* the 3D coordinates */
face        (list of vertex vl;       /* vertices of this face */
            vector n;                 /* the normal vector */
            int s;)                   /* the number of the
                                       * equivalence class to which
                                       * this face belongs or
                                       * −1 if this face has not
                                       * yet been encountered */
vertex      (vector p;                /* the location */
            list of face fl;)         /* the faces f for which
                                       * this vertex occurs
                                       * in f.vl */
geometry    (list of vertex vl;)      /* the list of constituent
                                       * vertices */
            list of face fl;)         /* the list of constituent
                                       * faces */
```

It is assumed that one global variable g, of type *geometry*, holds the complete model.

3.3 The Algorithm for Computing Vertex Normals

The algorithm is presented in a top-down fashion. At the highest level, it consists of merely three function calls.

```
InputGeometry( )
/* the data structure g is completely built;
 * all faces f have the directions of the normal
 * vectors f.n, apart from their signs;
 * the f.s-attributes are all set to −1 */
BuildEquivalenceClasses( )
/* all faces belong to an equivalence class;
 * all normal vectors f.n are consistently oriented
 * within each equivalence class */
OutputFaces( )
/* all faces have been output, including the vertex normals. */
```

Implementing the first pass is rather straightforward; it is not given explicitly here.

The second pass, *BuildEquivalenceClasses*( ), looks as follows.

```
BuildEquivalenceClasses( )
  {
    int $i$;
    face $f$;
      $i = 0$;
      for($f \in g.fl$)
      if($f.s == -1$)
      {
        $f.s = i$;
        BuildEquivalenceClass($f$);
        $i = i + 1$;
      }
  }
  /* Every face belongs to an equivalence class;
   * for the faces in each equivalence class,
   * the normal vectors are consistently oriented. */
```

Next, the recursive function *BuildEquivalenceClass* is introduced.

```
BuildEquivalenceClass($f_1$)
  face $f_1$;
  int $i$;
  {
    vertex $v$;
    face $f_2$;
  /* precondition: $f_1.s \neq -1$ */
    for($v \in f_1.vl$)
    for($f_2 \in v.fl \backslash \{f_1\}$)
  /* $f_1$ and $f_2$ share vertex $v$ */
    if($|f_1.vl \cap f_2.vl \backslash \{v\}| = 1 \wedge f_1$ and $f_2$ share an edge $\wedge f_2.s = -1$)
  /* $f_2 \overset{*}{\bowtie} f_1$, but $f_2$ has not yet been added to
   * this equivalence class. Notice that not necessarily $f_2 \bowtie f_1$:
   * there may be more faces in the current equivalent class that
   * share an edge with $f_1$ and $f_2$. */
    {
      Orient($f_1$, $f_2$);
      $f_2.s = f_1.s$;
      BuildEquivalenceClass($f_2$);
    }
  }
```

The function *Orient*(f_1, f_2) computes the normal vector of f_2. Of the two possible normal vectors $f_2.n$ and $-f_2.n$, the one oriented consistently with $f_1.n$ is chosen. This means that if f_1 and f_2 meet at a sharp angle, the dot product $(f_1.n, f_2.n)$ is less than 0; otherwise it is greater than 0. This function can be implemented rather straightforwardly. Notice that *Orient* is called at most once for every polygon. For simplicity, the check for a nonorientable polygon mesh has been omitted in the preceding listing.

We observe that, in the case of a nonmanifold mesh, the assignment of polygons to equivalence classes is nondeterministic. There seems to be no obvious local criterion to choose, for a polygon, "the most adequate equivalent class" in case of several possible equivalence classes. This ambiguity typically occurs at an edge where three or more polygons meet. The consequence of the ambiguity may be that a connected submesh of the nonmanifold mesh is oriented independently from the rest of the mesh, although the submesh will be internally consistently oriented.

Next we arrive at the third pass of the algorithm: the function *OutputFaces*. It looks as follows.

```
OutputFaces( )
  {
    face f;
    for(f ∈ g.fl)OutputFace(f);
  }
```

The function *OutputFace* outputs one face; it consists essentially of a loop over the vertices of that face.

```
OutputFace(f)
face f;
{
    vertex v;
    Output("polygon heading");
    for(v ∈ f.vl)
    {
    Output(v.p);
    Output(VertexNormal(v, f));
    }
}
```

Finally, the function *VertexNormal* is, of course, the heart of the algorithm: there the actual normal vector averaging takes place. If the geometric model were to consist of one single manifold (i.e., every edge is shared by precisely two polygons), such a function would be trivial, now that pass 2 of our algorithm has computed a consistently oriented set of face normals. In that case, one may want to continue reading from Section 4. In the general case, however, we have to deal with several "pathological" cases. Two of these cases are the configuration where an edge is shared by three or more faces (of the same equivalence class) and the case where faces meet in one vertex rather than in a complete edge (see Figures 2(a) and 2(b)).

The function *VertexNormal* operates by first building a list of candidate faces for normal vector averaging, and next pruning this list while searching for pathological cases.

```
VertexNormal(v, $f_1$): vector
  vertex v;
  face $f_1$;
  {
    edge e;
    list of face fl;
    int i, j, k, m, progress;
    face $f_2$;
  /* the face list fl will hold the candidate faces for
   * normal vector averaging, namely, all faces that surround v and that
   * are in the same equivalence class as $f_1$.
   * The first candidate, of course, is $f_1$. */
    fl = {$f_1$};
    for($f_2$ ∈ v.fl)
```

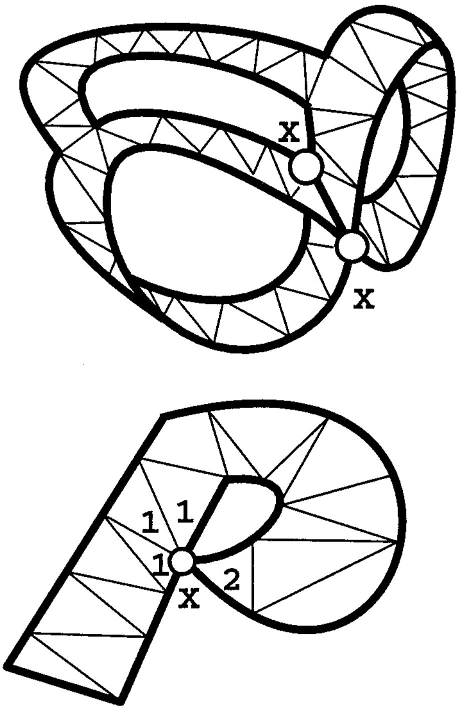

Fig. 2(a)—top: Even though the polygon mesh corresponds to one equivalence class, normal averaging should not take place in the vertices labeled "X"; Fig. 2(b)—bottom: Even though the polygon mesh corresponds to one equivalence class, normal vector averaging in vertex "X" should only take the polygons "1" into account; the normal vector in "X" due to polygon "2" is different from the normal vector due to polygon "1".

```
    if($f_1.s == f_2.s$)$fl = fl \sqcup \{f_2\}$;
/* Now the list $fl$ of candidate faces for normal vector averaging is
 * complete.
 * Start checking if some faces have to be excluded from $fl$.
 * The only faces $f_s$ that may stay are those that share an edge with
 * one of the other faces in $fl$, and for which no third face exists
 * in $fl$ that is incident in this same edge.
 * We define three segments in the list: elements $fl.i$ with $0 \leq i < k$
 * have been checked and they have been approved for averaging.
 * Elements $fl.i$ with $k \leq i < m$ have not been checked. Elements $fl.i$
 * with $m \leq i < |fl|$ have been rejected. Initially,
 * $k = 1$ (since we know that $f_1$ is approved for averaging),
 * and $m = |fl|$. The algorithm terminates either when $k = m$ (i.e., when
 * all elements have been checked) or when no other faces $fl.i$ in the
 * segment $k \leq i < m$ exist that share an edge with one of the faces in
 * the segment $0 \leq i < k$. The first segment will be called the "OK-
 * segment,"
 * the second segment the "TODO segment," and the third segment the
 * "NON-OK-segment." The algorithm proceeds by inspecting elements
 * from the
 * TODO-segment and moving them either to the OK-segment or to the
 * NON-OK segment. Also, if none of the faces in the TODO segment can
 * be decided upon, the algorithm terminates. The variable
```

```
   * progress serves to monitor this last condition. */
    k = 1;
    m = |fl|;
    progress = 1;
    while(k < m ∧ progress == 1)
    {
      progress = 0;
      j = k;
  /* search the TODO-segment to see if we can move an element either to
   * the OK-segment or the NON-OK segment */.
      while(j < m)
      {
        if(fl.j shares an edge e with one of the fl.i in the OK-segment)
        {
          progress = 1;
  /* we will now decide if fl.j goes to the OK-segment or the NON-OK
   * segment,
   * but in any case, progress is made since we can reduce the TODO-
     segment. */
          if(there is an $f_3$ in fl, other than fl.j and fl.i, that also shares e)
          {
  /* fl.j goes to the NON-OK-segment */
            Swap(fl.j, fl.m − 1);
            m = m − 1;
          }else
          {
  /* fl.j goes to the OK-segment*/
            Swap(fl.j, fl.k);
            k = k + 1;
          }
        }
        j = j + 1;
      }
    }
    return(1/k $\Sigma_{i=0}^{k-1}$ vl.i.n);
  }
```

Note that each face and each vertex is only visited once. Hence this algorithm has complexity O(number of faces + number of vertices).

4. WHY LINEARLY INTERPOLATING NORMAL VECTORS IS WRONG

Images that are generated by means of the traditional Phong shading algorithm typically look very good provided the polygonal mesh is sufficiently dense. For larger polygons, several conspicuous artifacts may arise. The silhouette edge problem is probably the most notorious one, but also the shading may be thoroughly wrong. This is remarkable, since the *size* of the polygons is not an input parameter of the algorithm. Closer observation shows that indeed it is not so much the polygon's size but rather the rate of change of the normal vectors over the surface that may cause shading artifacts. Consider, for example, the configuration of Figure 3. Here a surface is depicted with a zigzag profile. Applying the normal vector averaging algorithm of Section 3 (or indeed any other sensible approach for obtaining vertex normals for this surface) yields pairs of adjacent polygons

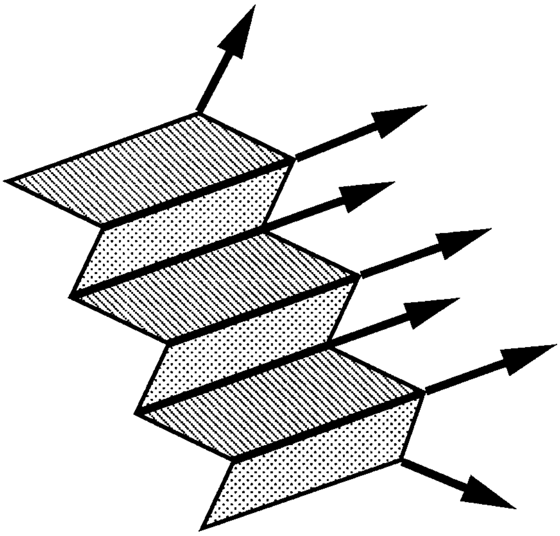

Fig. 3. A surface with a zigzag profile gives rise to unnatural normal vector assignment.

where the normal vectors do not vary at all over the surface, resulting in a completely flat appearance. This is at odds with the appearance of the silhouette. This would be even more conspicuous if the surface were to be moving: in this case the visual cues from the varying intensity distribution and the geometry of the moving silhouettes would be contradictory, which would be highly confusing to the perception of a viewer. Apparently, cases exist where the traditional linear interpolation of normal vectors is not such a good idea. More specifically, in Section 4.1 a quantitative analysis shows that, in fact, linear interpolation generally leads to inconsistencies between the geometric representation of the surfaces and the illumination distribution; Section 4.2 suggests a simple remedy.

4.1 The Diagnosis

Part of the problem is related to the fact that a polygon mesh, which is supposed to approximate a curved surface, is actually discretely sampling this surface. By taking the sample density sufficiently high, the shape difference between the curved surface and the mesh can be made arbitrary small. The illumination distribution then may be obtained by assigning a constant illumination value to each polygon, and still the resulting illumination will look natural.[4] Since this would require extremely high sampling densities, and hence large data volumes and processing capacity, it is advantageous to reduce the sampling density, while at the same time compensating for the illumination artifacts (i.e., undersampling artifacts) by some form of illumination interpolation. Gouraud shading is the sim-

[4] Assuming that Mach bands will vanish if the intensity discontinuities are sufficiently close to each other and sufficiently small.

plest of these compensation strategies, and it is adequate if the sample density is sufficiently high to capture all local maxima (highlights) and minima of the shading distribution over the polygon mesh.

When proposing increasingly more advanced illumination or normal vector interpolation strategies, we have to be aware that we are, in fact, aiming to reconstruct a surface from a discretely sampled version. Reconstruction cannot add information, but at least we can try to come up with a reconstructed surface that is consistent with the sampled data, that is, that both interpolates the vertices of the polygon mesh and is perpendicular to the normal vectors. In the remainder of this section we show that the linear interpolation of normal vectors in Phong shading is not consistent in this sense; in the next section we propose an interpolation method that is consistent.

In order to facilitate the analysis, we assume for a minute that normal vector averaging takes place in 2D rather than 3D. This means that a normal vector is directly equivalent to a tangent vector (whereas in 3D a normal vector is equivalent to a tangent plane). Consider normal vector interpolation along an edge. Choose the coordinate system such that the x-axis lies along the edge, and (without loss of generality) the edge extends from $x = 0$ to $x = 1$. With $\phi(x)$ we denote the tangent of the curve that is perpendicular to the interpolated normal vector anywhere. Indeed, the curved surface (here, the curve) that will cause the smoothly varying reflection is given by

$$\Phi(x) = \int_{\xi=0}^{x} \phi(\xi) d\xi.$$

Now the normal vectors (and hence ϕ) are given for $x = 0$ and $x = 1$, say ϕ_0 and ϕ_1, respectively. The function $\phi(x)$ therefore reads

$$\phi(x) = \phi_0 + x(\phi_1 - \phi_0).$$

The curve should pass through (0,0) and (1,0) since we have chosen the x-axis along the edge. Thus $\Phi(0) = \Phi(1) = 0$. This yields:

$$0 = \int_{\xi=0}^{x} \phi_0 + \xi(\phi_1 - \phi_0) d\xi.$$

Evaluating the integral on the right-hand side, we find

$$0 = \frac{1}{2}(\phi_0 + \phi_1),$$

for which case

$$\Phi(x) = \phi_0 x(1 - x).$$

We conclude from this that $\Phi(x)$ only exists if $\phi_0 = -\phi_1$; in all other cases there simply exists no curve (surface) that passes through the vertices, is perpendicular to the normals in the vertices, and has a linearly changing tangent (normal vector). So except for the case of mirror-symmetric normal vectors in the two opposite vertices of an edge, *there is no surface that would yield the normal vector distribution used by traditional Phong shading.*[5] Now let us study the consequence of this observation for the resulting illumination distribution. The illumination distribution is a function of the normal vector distribution, but in the case of diffuse reflection and only one light source, this function is not necessarily bijective. In that case, all normal vectors n that yield the same values for the dot product (n, l), where l is the direction of the light source, give rise to the same illumination. However, with three or more light sources i with different directions l_i there are in general no two different normal vectors n_1 and n_2 such that for all i, $(n_1, l_i) = (n_2, l_i)$. So if in that case the light sources have different colors, the illumination distribution is indeed a bijective function of the normal vector distribution. This means that the fact that the Phong normal vector distribution does not correspond to any possible surface leads to the conclusion that *there is, in general, no surface that would yield the illumination distribution generated by traditional Phong shading*. Since most renderers use linear interpolation, most of the pictures we have been producing are simulating surfaces that cannot exist![6] We investigate in the next section what alternative interpolation scheme could be used.

4.2 The Remedy

Before turning to the 3D case, we first try to come up with an improved interpolation scheme to cure the problem as observed in Section 4.1. We are in search of a function $\phi(x)$ with $\phi(0) = \phi_0$ and $\phi(1) = \phi_1$ subject to the boundary condition

$$0 = \int_{\xi=0}^{1} \phi(\xi) d\xi.$$

There is, of course, a continuum of functions ϕ that satisfies the preceding requirements; an often successful approach consists of demanding that ϕ

[5] Although we do not claim this observation to be new, we have not found any references to this observation in the literature.

[6] One may question the impact of this observation. Remember that a polygon mesh is a discrete sampling of a curved surface. With a high sample density, the normal vectors in the vertices will be close to perpendicular, and hence both ϕ_0 and ϕ_1 will be close to 0, so the condition for symmetry $\phi_0 = -\phi_1$ is not violated very badly. Hence the inconsistency may not be too conspicuous for dense polygon meshes.

should have a minimal average curvature. This is formulated as

$$\int_{\xi=0}^{1} \dot{\phi}^2 d\xi$$

is minimal.
(Here $\dot{\phi}$ means differentiation with respect to ξ.) Using variational calculus, the problem of finding such a function that meets the boundary conditions is formulated as

$$\delta\left(\int_0^1 (\dot{\phi} + \epsilon\dot{\eta})^2 d\xi + \lambda \int_0^1 (\phi(\xi) + \epsilon\eta(\xi)) d\xi \right) = 0.$$

The δ means that the entire expression should be independent for changes in the perturbation function η for arbitrary η with $\eta(0) = \eta(1) = 0$ up to first order in the small parameter ϵ. The Lagrange multiplier λ is a priori unknown, but its values will be fixed by substituting ϕ back into the boundary conditions. Evaluating the integrals up to first order in ϵ and partial integration yield

$$\int_0^1 (-2\ddot{\phi} + \lambda)\eta(\xi) d\xi = 0.$$

This only holds irrespective of the choice of η iff

$$\ddot{\phi} = \lambda/2$$

or

$$\phi(\xi) = a + b\xi + \lambda\xi^2/4.$$

The values of a, b, and λ are readily obtained by the boundary conditions for ϕ and $\int_0^1 \phi d\xi$; the final result is

$$\phi(\xi) = \phi_0 - (4\phi_0 + 2\phi_1)\xi + 3(\phi_0 + \phi_1)\xi^2.$$

We conclude that the old linear interpolation scheme should be replaced by a quadratic interpolation scheme; moreover, in the single case where the linear interpolation worked ($\phi_0 = -\phi_1$), the two interpolation schemes are identical. Moreover, in the case where $\phi_0 = \phi_1$ (i.e., the infamous example from the introduction of this section where we considered the surface with the zigzag profile), the result is

$$\phi(\xi) = \phi_0(1 - 6\xi + 6\xi^2),$$

that is, a parabola which is symmetric with respect to $\xi = 1/2$. In other words, the quadratic interpolant as suggested here results in a normal vector bent in the opposite direction halfway, which nicely matches with our intuition for a curved surface that has to fit with the zigzag profile.

Unfortunately, the previous results do not generalize seamlessly to 3D. This is due to the fact that the boundary condition

$$0 = \int_{\xi=0}^{1} \phi(\xi) d\xi$$

cannot be formulated in terms of the normal vectors instead of the tangents. We use an approximation instead, which we call the *plane curve approximation*, which is based on the following observation. Consider a parameter curve $\Phi(s)$, for $0 \le s < 1$. In 2D, the relation between a normal vector and a tangent is given by

$$\begin{pmatrix} n.x \\ n.y \end{pmatrix} \propto \begin{pmatrix} \dfrac{d\Phi.y}{ds} \\ -\dfrac{d\Phi.x}{ds} \end{pmatrix}.$$

Or, with $\Delta.x = \Phi(1).x - \Phi(0).x$ and similarly for $\Delta.y$,

$$\left(\int_{0}^{1} n(s) ds,\ \Delta \right) = 0.$$

(Notice that Δ actually is the vector that represents the edge along which normal vector interpolation has to take place.) This relation does not in general hold in 3D, but it obviously does if the curve is planar. Our approximation now holds that we assume the interpolation curves to be sufficiently planar to use the preceding formula.

If we again adopt a quadratic interpolation scheme[7] (which has been shown to yield curves with minimal average curvature when the normals happen to be coplanar), we have to solve for the vectors a and b in

$$n(s) = n_0 + as + bs^2$$

by demanding that

$$n(1) = n_1$$

[7] As with linear interpolation schemes, quadratic interpolation can be computed incrementally; see, for example, Foley et al. [1990], p. 511–512.

and

$$\left(\int_0^1 (n_0 + as + bs^2) ds,\ \Delta \right) = 0,$$

gives

$$a = n_1 - n_0 - b.$$

Eliminating a yields

$$(b,\ \Delta) = 3(n_0 + n_1,\ \Delta). \tag{1}$$

This means that we are allowed to impose one additional constraint; again we demand the average curvature to be as small as possible.[8] So we minimize

$$\int_0^1 (\ddot{n}(s))^2 ds$$

subject to the boundary condition (1). Again using a Lagrange multiplier (this time called μ), this results in the following expression for b to be minimized:

$$2(b,\ b) + \mu(b,\ \Delta - 3(n_0 + n_1,\ \Delta)).$$

Demanding the derivative with respect to b vanish, yields

$$4b = -\mu\Delta,$$

and substituting this back into 1 gives

$$\mu = -3\,\frac{(n_0 + n_1,\ \Delta)}{\Delta^2},$$

and hence

$$b = 3\,\frac{(n_0 + n_1,\ \Delta)}{\Delta^2}\,\Delta.$$

[8] This time, we demand the average curvature of the normal vectors to be minimal rather than the average curvature of the curve proper. This has two reasons: in 3D the curve itself is not easily accessible; and the sequel of the derivation otherwise would be somewhat more involved. Of course, which optimization criterion is chosen is largely arbitrary.

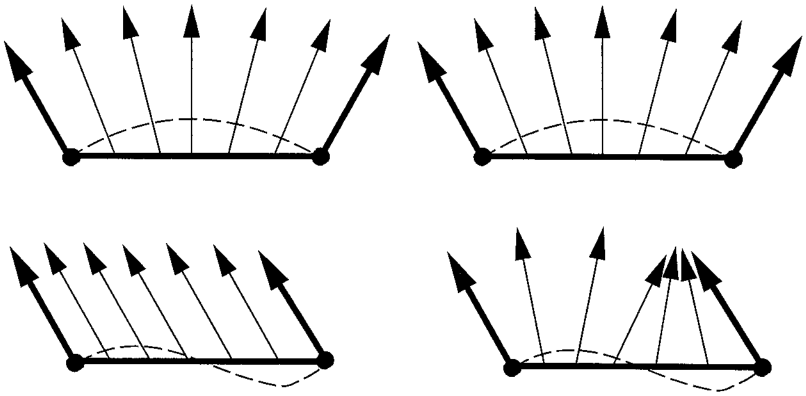

Fig. 4. Some examples of normal vector averaging over an edge. Left: traditional linear interpolation. Right: quadratic interpolation. The dashed curve indicates the profile of the surfaces that should be simulated.

This completes the derivation of the quadratic interpolation scheme for normal vectors. Applying it again to the example with $n_0 = n_1 = n$ with a coordinate system such that $\Delta = (1, 0, 0)^T$ gives for $n(s).x$:

$$n(s).x = n.x(1 - 6s + 6s^2),$$

again a symmetric parabolic behavior results where the interpolated normal vector in $s = 1/2$ points in the x-direction, opposite to $n.x$. In Figure 4, the difference between linear and quadratic interpolation of normal vectors is shown for interpolating normal vectors over an edge.

Color Plate A (upper left) shows a flat shaded geometric model of an extruded and rotation-swept zigzag profile. In the upper right image the conventional linear interpolant has been used, whereas the lower left was rendered with the quadratic interpolant of this section. Notice that the straight silhouettes of course are still visible, since up to now we have not changed the geometry. The lower right image shows the same object modeled with curved patches, approximated by a dense polygon mesh. Notice that the intensity distribution is rather similar to the distribution obtained with quadratic interpolation. Similar results are depicted in Plate B.

Performance Issues of Quadratic Normal Vector Interpolation. Linear normal vector interpolation requires the computation of an expression of the form $n_0 + as$, where n_0 and a are vectors and s is a floating point number between 0 and 1. Since s is incremented in equal steps over a scanline segment, this has been implemented as one vector addition, or three floating point additions per pixel.

Quadratic normal vector interpolation requires the computation of an expression of the form $n_0 + as + bs^2$, where n_0, a, and b are vectors, and s is again a floating point number between 0 and 1. Using forward

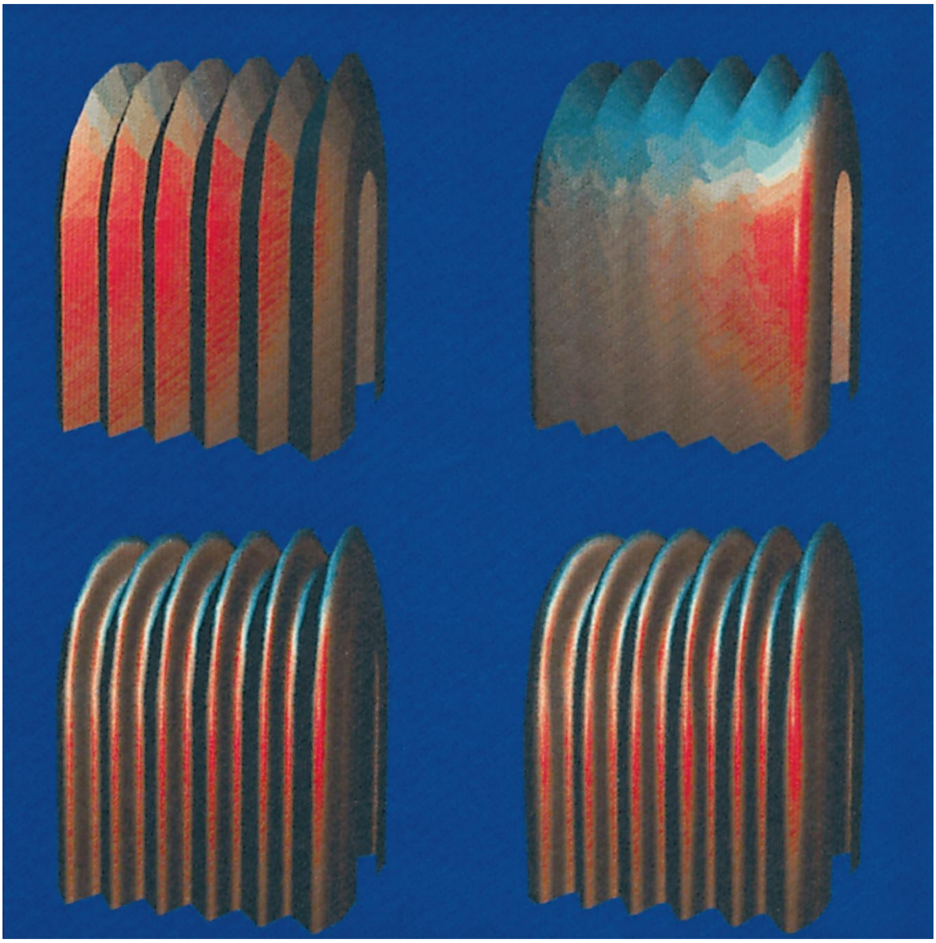

Plate A. Upper left: Flat shaded polygons on a zigzag profile. Lower left: Quadratic interpolation of normal vectors on a zigzag profile. Upper right: Traditional linear interpolation of normal vectors on a zigzag profile. Lower right: Reduction of straight silhouettes using a dense polygon mesh approximation of a curved patch model of the same object.

differences, an incremental implementation requires two vector additions, or six floating point additions per pixel.[9]

The three additional floating point additions per pixel for quadratic interpolation instead of linear interpolation should be compared with the other calculations that have to take place for one pixel. For a very simple illumination model this is:

—normal vector normalization: six multiplications plus a square root;

[9] For sufficiently long scanline segments, the setup computation of about 12 multiplications for one scanline segment can be ignored; on the other hand, for very short scanline segments, the difference between linear and quadratic interpolation is hardly visible, so in that case linear interpolation can be used after all.

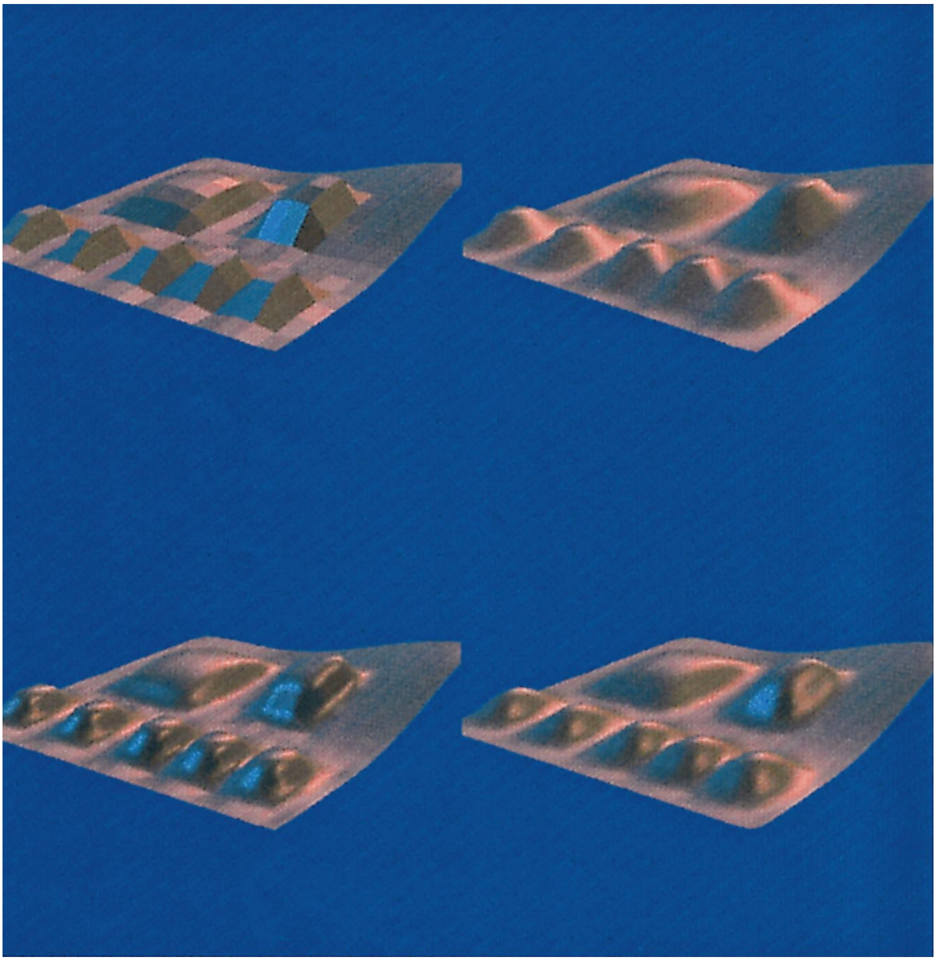

Plate B: Same as Plate A for another 3D object.

—diffuse illumination contribution: dot product (= three multiplications) plus three multiplications for the red, green, blue components;

—specular illumination contribution: dot product (= three multiplications), exponential plus three multiplications for the red, green, blue components; and

—six additions to accumulate all illumination components.

If we count a floating point multiplication just as expensive as a floating point addition, and exponential and square root calculation each as five multiplications, we find the equivalent of 34 additions. So the three extra additions are less than 10% of the computational effort per pixel. For every additional feature (texture mapping, anti-aliasing, distance attenuation, . . .) the relative overhead of quadratic normal vector interpolation drops to an even smaller amount.

5. SUMMARY AND CONCLUSIONS

We study topics that are closely related to Phong shading. First, we propose an algorithm to obtain normal vectors in the vertices of a polygon mesh that are suitable for normal vector interpolation. The algorithm does not use any assumptions on the topology of the polygon mesh: it may consist of one or more orientable or nonorientable manifolds, and nothing is assumed about the relative orientation of the polygons. It runs in O(number of polygons + number of vertices).

Second, a modification of normal vector interpolation is suggested. It replaces the traditional linear interpolation by a quadratic interpolation. Using forward differences for incremental computation of the quadratic terms in the interpolant, this can be implemented within the control structure of standard polygon scan conversion algorithms. In that case, the overhead consists of three additions per shaded pixel. For sophisticated rendering algorithms, including, for example, texture mapping and/or environment mapping, experiments show that this is an increase of (estimated) less than 5% processing time.[10] If anti-aliased texture is used, the overhead turns out to be less than 1% in our renderer. The most significant benefit of quadratic normal vector interpolation is that polygon models of curved surfaces may be coarser without the risk of inconsistencies between the shape and the illumination distribution.

REFERENCES

ALLAN, J., WYVILL, B., AND WITTEN, I. 1989. A methodology for polygon mesh modeling. In *Proceedings of CG International 89* (Leeds, U.K., June 27–30), Springer-Verlag, New York, 451–470.

BISHOP, G. AND WEIMER, D. M. 1986. Fast Phong shading. *Comput. Graph. 20*, 4 (Aug.), 103–106.

BLINN, J. AND NEWELL, M. E. 1976. Texture and reflection in computer generated images. *Commun. ACM 19*, 10 (Oct.), 542–547.

BLINN, J. 1978. Simulation of wrinkled surfaces. *Comput. Graph. 12*, 3 (Aug.), 286–292.

BOEHM, W., FARIN, G., AND KAHMAN, J. 1984. A survey of curve and surface methods in CAGD. *CAGD Comput. Aided Geom. Des. 1,* 1, 1–60.

COOK, R. L. 1984. Shade trees. *Comput. Graph. 18*, 3 (July), 223–232.

DUFF, T. 1979. Smoothly shaded renderings of polyhedral objects on raster displays. *Comput. Graph. 13,* 3 (Aug.), 270–275.

FOLEY, J. D., VAN DAM, A., FEINER, S., AND HUGHES, J. 1990. *Computer Graphics Principles and Practice.* Addison-Wesley, Reading, MA.

GOURAUD, H. 1971. Continuous shading of curved surfaces. *IEEE Trans. Comput. C-206* (June), 623–629.

MALLET, J. L. 1992. Discrete smooth interpolation in geometric modeling. *Comput. Aided Des. 24,* (April), 178–191.

NEWMAN, W. M. AND SPROULL, R. F. 1979. *Principles of Interactive Computer Graphics.* McGraw-Hill, New York.

PHONG, B.-T. 1975. Illumination for computer generated pictures. *Commun. ACM 18*, 6 (June), 311–317.

REQUICHA, A. A. G. 1980. Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv. 12,* (4) (Dec.), 437–464.

[10] Of course, this depends somewhat on the chosen implementation of the other shading calculations and the hardware characteristics.

ROGERS, D. F. AND ADAMS, J. A. 1976. *Mathematical Elements for Computer Graphics.* McGraw-Hill, New York.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. *Comput. Graph. 19*, 3 (July), 245–254.

WHITTED, T. 1980. An improved illumination model for shaded display. *Commun. ACM 23,* 6 (June), 343–349.

WYVILL, B. AND WYVILL, G. 1989. Field functions for iso-surfaces. *Visual Comput. 5,* 1/2 (March), 75–82.

Received December 1993; revised August 1995; accepted April 1997.