An algorithm for polygon subdivision based on vertex normals

C.W.A.M. van Overveld
Department of Mathematics and Computing Science
Eindhoven University of Technology
Address: P.O.Box 513, 5600 MB, Eindhoven,
The Netherlands.
Email: wsinkvo@info.win.tue.nl

B.Wyvill
Department of Computer Science
University of Calgary
Address: 2500 University Drive N.W.,
Calgary, Alberta,Canada, T2N 1N4
Email: blob@cpsc.ucalgary.ca

Abstract

In order to achieve the impression of a smooth surface while rendering a polygon mesh, normal vector vectors may be provided in the vertices of the mesh that are the average of the surface normals of the adjacent polygons. Interpolation of these normal vectors while rendering of the polygons in the mesh, and using the interpolated normal vectors in the shading computations, yields a smoothly varying intensity distribution. There is an inherent mismatch, however, between the smoothness of the shading thus achieved and the non-smoothness of the geometry which is particularly visible at silhouettes, showing as straight edges and non-smooth edge junctions at the silhouette vertices. In this paper, a remedy for these artefacts is suggested. The remedy consists of subdividing each input polygon into a mesh of polygons prior to rendering. The shape of this resulting polygon mesh is controlled by the normal vectors that are provided in the vertices of the original polygon, unlike other subdivision schemes that make use of adjacent polygons. With our method, polygons equipped with vertex normal vectors can therefore be processed without further knowledge of neighbour polygons. This makes the method well-suited in the context of graphics libraries, such as OpenGl, that treat polygons typically on a per-polygon basis. So the proposed computation of the mesh which replaces the original polygon can be viewed as a filter which may operate as a process in front of a traditional polygon rendering pipeline.

Keywords geometric modelling, Bezier patches, subdivision, computer graphics

1. Introduction

Over the last years, efficient rendering algorithms, including hidden surface elimination and shading have found their way into the graphics work stations with the advent of sophisticated graphics hardware. This has emphasised the distinction between modelling and rendering: modelling takes place in the software domain, whereas rendering is taken care of by the underlying hardware. The interface between the two layers consists of a stream of data- and control signals. Several API's have been introduced recently, such as OpenGl, in order to standardise the format of the transfer of geometric modelling primitives. In particular, we will focus on polygons as modelling primitives in this paper: even though curved surfaces have been extensively used in geometric modelling, there is still a significant demand for visualisation of polygon models. Moreover, with the work of e.g. Mallet ([9]), Allan, Wyvill and Witten ([1]) and others, there even seems to be a revaluation for polygons in geometric modelling.

As far as geometry is concerned, the data that represents

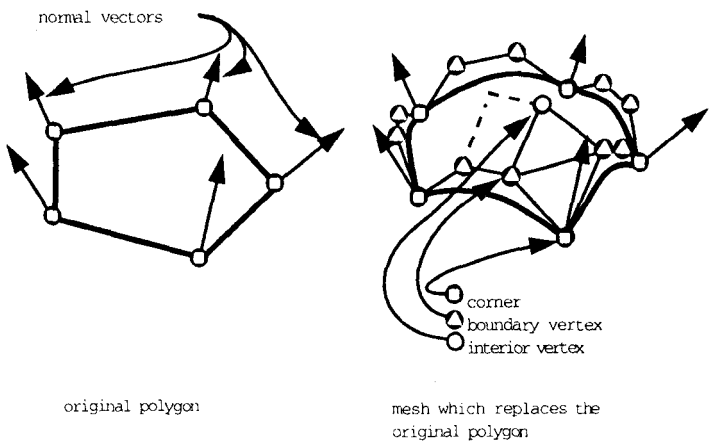
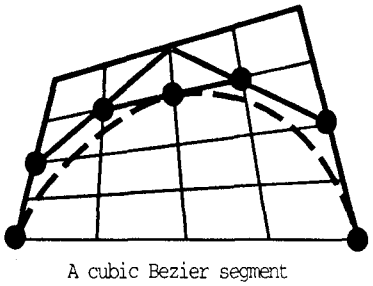


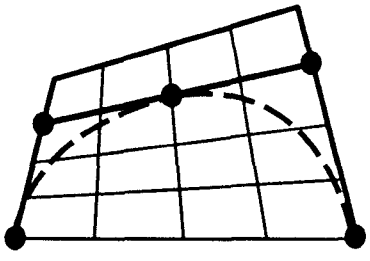
Figure 1. The original polygon; the resulting mesh; corners; boundary vertices; and interior vertices

0-8186-7825-9/97 $10.00 © 1997 IEEE

For color plate see page 246



A cubic Bezier segment
divided into two
cubic Bezier segments



A cubic Bezier segment replaced
by two quadratic Bezier segments with
equivalent boundary conditions

Figure 2. Reducing from a cubic to a quadratic subdivision scheme

polygons comprises vertex coordinates and vertex normal vectors. This data is passed through the graphics library API's in a pipe-lined fashion, i.e. each polygon is transformed, clipped and rendered in isolation without taking other polygons into account. In particular, the topological structure (neighbour relations) in the geometrical model is not represented in such API's[1], and hence not available, within the data structures of the rendering hardware. As a consequence, polygonal models will display straight silhouette edges and non-smooth edge junctions in silhouette vertices, even if their interior region is smoothly shaded. The most conspicuous aspect of straight silhouette edges are the non-G^1 edge junctions in the silhouette vertices.

Methods have been developed, such as the polygon subdivision algorithm by Catmull and Clark ([4]) that replace coarse polygon models by denser polygon meshes in order to remove straight silhouettes. However, these methods make use of the fact that for each polygon, its neighbours are available.

In this paper, we study the problem of removing straight-silhouette edges, and more in particular the non-G^1 edge junctions for geometric models that are communicated to the rendering hardware by means of a stream of isolated polygons. The only geometric information that is available, apart from the vertex coordinates, therefore, comprises the normal vectors in these vertices. Section 2 lists the requirements of the solution and reports on related work. A basic solution to the problem, which works for closed manifold polygon meshes, is presented in section 3. When the input polygons do not originate from a closed manifold mesh, but from a manifold mesh with one or more holes, a shortcoming of the method becomes visible, and we discuss a partial solution to this problem in section 4 (this solution is partial in the sense that it requires additional geometric information in the polygon vertices that should be provided by the application). Section 5 summarises the conclusions.

2. Requirements and conditions; Previous work.

An algorithm will be developed that converts a polygon, equipped with normal vectors in the vertices, into a triangle mesh. Assuming that the vertex normals of the input polygon are such that smooth shading of the original model would be achieved, the angles between adjacent triangles in the resulting mesh will be more obtuse than the angles between adjacent polygons in the original polygonal model.

[1] In some cases, regular neighbouhood structures such as triangle strips are supported for increased efficiency. In our study, however, we do not make any assumptions of the neighbourhood structure of the polygon to be rendered.

Therefore, the piecewise straight silhouettes in the resulting mesh will be less conspicuous. Also, the meshes of adjacent polygons will fit together in a C^0 fashion. The algorithm may operate recursively, by applying it again to this resulting mesh of triangles. A precondition for the algorithm is that normal vectors in shared edges of adjacent polygons are the same; this precondition is implied by conventional modelling techniques or post-modelling normal vector averaging, based on the neighbour relations between the polygons. The algorithm guarantees that the same condition holds for the output polygons.

Before listing the requirements and conditions for this algorithm, some nomenclature will be introduced (see also figure 1).

the polygon is the input polygon, i.e. a cyclic list of 3-D vertices with normal vectors (normalised). It is assumed that this polygon is part of a manifold polygon mesh, and for a given edge e in this mesh, the two polygons that share e have equal normal vectors in the two common vertices.

the mesh results from the polygon; the mesh vertices consist of the polygon vertices and additional new generated vertices. The locations of the new vertices are computed by taking the original vertices and normal vectors into account. The method for achieving this comprises the crucial aspect of the algorithm. In the sequel, polygons in the mesh will be called *sub-polygons*. After construction of the mesh, the sub-polygons will be triangulated.

the corners are those mesh vertices that equal the vertices of the polygon.

the boundary vertices are those mesh vertices that define, together with the corners, the boundaries of the mesh.

the interior vertices are the mesh vertices that are neither corners nor boundary vertices.

2.1. Requirements

The requirements that hold for a useful algorithm can be classified in three categories: architecture, functionality, and performance. Below we give these requirements; also, before explaining the algorithm in detail, some intuitive considerations are given as to how these requirements will be realised.

1. architecture: a pipeline-based, optional filter

   1.1 pipeline-based: the algorithm should deal with every polygon in isolation;

   1.2 optional filter: the algorithm should be a filter with congruent input- and output formats, i.e. both input and output should be a stream of polygons, equipped with normal vectors. This means that, e.g. for fast previewing, the filter can be passed by. Moreover it may be useful to have several instantiations of the filter in a row. This means that all sub-polygons that result after the first pass are replaced by sub-meshes of sub-sub-polygons after the second pass, etcetera.

2. functionality: continuity, consistency, affine invariance

   2.1 continuity: the filter's main purpose is to make the angles between adjacent triangles in the mesh more obtuse than the angles between the original polygons. This means:

   2.1.1 C^0 continuity: no cracks;

   2.1.2 G^0 normal vectors: no shading discontinuities;

   2.1.3 the shape of the mesh will be such that its periferal triangles will be (close) to perpendicular to the vertex normals of the original polygon, whereas the mesh will be 'smooth' in between (i.e. obtuse angles between adjacent triangles). Therefore, assuming that the normal vectors in the vertices of the input polygon resulted from averaging the surface normal vectors of this polygon and its neighbours, the meshes for adjacent input polygons will together form a 'smooth' mesh as well.

   2.2 consistency: the normal vector field in combination with the generated mesh is *consistent* with the geometry. This means the following. Linearly interpolating normal vectors over a polygon gives a shading pattern that suggests a monotonous curvature (either totally convex or totally concave) over this polygon. Such a monotonously curved surface, however, does not always exist (see e.g. figure 16.24 on page 741 in ([6]). Indeed, sometimesa curved surface that is perpendicular to the given vertex normals has to posses a curve of inflection (=the collection of points where the curvature changes sign). In this case, linear interpolation of normal vectors, suggesting a monotonous curvature is said to be *inconsistent*. The mesh generated by our algorithm, together with the vertex normals it provides, are such that each triangle represents either a totally convex or a totally concave surface fragment.

   2.3 affine invariance: the algorithm is transparent for linear transformations of the input vertices and normal vectors and for translations of the input vertices.

3. performance: generality, adaptiveness

   3.1 generality: the algorithm should work irrespective of the number of vertices in the input polygons. This means that the subdivision scheme used should be compatible both for triangles, quadrilaterals, and polygons with >4 sides, thereby taking into account continuity requirements as listed under 2.1.1-2.1.3,

also between polygons of different types. Though it will be assumed that the input polygons have convex projections[2], they need not be planar.

3.2 adaptiveness: the algorithm can be used repeatedly, yielding recursive subdivision. A stopping criterion may be imposed which is based on the length of a edge which is to be subdivided.

3.3 Consider one edge of the input polygon, together with the two associated vertex normals. The mesh boundary that is to replace this edge may have to contain an inflection point, depending on the directions of the vertex normals in combination with the direction of the edge. It is possible, however, to replace the edge by *two* adjacent mesh boundaries that have both a monotonous curvature (either convex or concave). Therefore, in constructing the mesh, we first to introduce split points in the edges of the input polygon so that this polygon can be split in sub-polygons where each sub-polygon is guaranteed to be representable by a mesh which has monotonous curvature. In this manner, we can assure requirement 2.2. This first splitting can be compared with replacing the four control points that specify a cubic Bezier curve by two sets of three control points that specify two adjacent quadratic (and hence monotonous) Bezier curves, where the latter two curves are joined in a C^1 fashion. The two quadratic curves have the same geometric boundary conditions as the original cubic curve, as can bee seen in figure 2.

2.2. Previous work

A large amount of work has been done in relation to the conversion from polyhedra to curved patches, which also alleviate the problem of straight silhouettes. A survey of much of this work can be found in ([3]). Most subdivision schemes that yield cross-boundary continuity, however, rely on the availability of vertices of adjacent polygons (such as the Catmull-Clark subdivision scheme, [4]) rather than vertices and normal vectors of the polygon to be replaced proper. The requirements 1.1 and 1.2 render these approaches less suited. Recently, the problem of finding smooth surfaces that interpolate polygon vertices under the condition of being perpendicular to normal vectors has received attention from Bajaj et. al.. In ([2]), they give an algorithm which finds an algebraic representation of such a surface for a triangular mesh in the presence of normal vectors. Although ideally suited for ray tracing, algebraic surfaces do not lend themselves to rasterisation by means of scan conversion. More recently, Kuriyama ([7]) reported on the generalisation of Coons patches to arbitrary n-gonal networks of boundary curves, and Loop ([8]) introduced a patch representation which allows control point meshes with arbitrary topology. Similarly, the generalised b-spline patches of Fong and Seidel ([10]) are defined for arbitrary topologies. A polygon mesh approximation of the Kuriyama patches could form an alternative for our sub-meshes; the Loop patches and the generalised b-spline patches are different from our approach in the sense that they do not interpolate the original polygon mesh.

[2] The algorithm will produce a reasonable polygon mesh anyway, except in the case of non-convex input polygons which may produce incomprehensible result (e.g. a folded mesh).

2.3. A first approach

Our scheme can be described on a global level as follows:

1. Insert two boundary vertices in each edge of an input polygon, making use of the normal vectors in the vertices. This collection of 4 vertices can be regarded as the control points for a cubic Bezier curve segment which represents this edge. In the end points, this cubic Bezier segment is perpendicular to the normal vectors (see section 2.4)[3]

2. Cubic curve segments may possess inflection points. Since it is important for the sequel of our algorithm that the generated mesh correspond to patch segments with monotonous curvature (to allow consistent normal vector interpolation, according to requirements 2.2 and 3.3), we have to apply one subdivision step to the cubic Bezier segments, replacing the curve by two halves that are monotonously curved. This would result in a set of 7 control points for one boundary curve, and hence 7 boundary vertices for each boundary segment in our mesh. For our purposes, however, we don't need that many degrees of freedom. Therefore, instead of splitting the cubic segment into two new cubic segments for the two halves, we construct 5 control points that define two quadratic curves having the same geometric boundary conditions as the original cubic curve: so we maintain the condition of perpendicularity to the normal vectors. Note that quadratic curves are guaranteed monotonously curved.

3. For each corner, create one interior vertex. We need these interior vertices to guarantee that, after repeated recursive application of our algorithm, corner triangles of the resulting mesh will be perpendicular to the normal vectors in the corner vertices of the original

[3] Note that the requirement of the Bezier curve being perpendicular to the normal vectors does not uniquely define the locations of the two new vertices (control points). Further on we will discuss a method to find unique locations.

polygon. Therefore there has to be one independent interior point for each corner point.

If our original polygon was a triangle, we now have 5+4+3+2+1=15 vertices, consisting of 5 vertices per polygon edge (=12 vertices) plus the 3 interior vertices. These 15 vertices can be regarded as the control points of a 4th-degree triangular Bezier patch. This patch is next evaluated in 6 points, namely the three corners and the three points halfway the three border curves. Also, the 6 normal vectors in these points can be obtained from the 4-th order triangular Bezier patch (three of them are the orignal input normal vectors in the original vertices), and the 6 points with their normal vectors can be used to define 4 triangles with each three vertex normals that together form the resulting mesh replacing the input triangle.

If the input polygon had more than 3 vertices, similar constructions can be devised (see section 2.5).

Several more or less independent aspects of the algorithm deserve attention. These will be dealt with in the subsequent subsections.

2.4. Finding boundary vertices depending on one edge only

From requirements 1.1, 1.2, 2.1.1, and 2.3, it follows that the boundary vertices for one boundary segment (i.e. the part of the boundary of the mesh that sits between two subsequent corners) may depend only on the information that comes with the associated polygon edge, say e. Indeed: two adjacent polygons share only this edge, so all remaining geometric information of these two polygons is not allowed to contribute. The amount of allowed information comprises two vertices and two (normalised) normal vectors, i.e. a 10-dimensional datum. If we interpret the collection of boundary vertices for one boundary segment as the control points of a Bezier curve (that, for the course of our algorithm, is considered to represent the border segment associated to e), we can observe the following. A quadratic curve is defined by an element of $\mathbb{R}^9$ only, this means that the Bezier curve to which the boundary segment corresponds is at least a cubic curve. But then we have to find two additional boundary vertices, that will be regarded as control points, in between the two corners for each boundary segment. Therefore we may impose some additional conditions to the locations of the new boundary vertices: in particular, we may impose some form of smoothness criterion. Two methods to do so are given here. We adopt the convention that p_0 and p_3 are the extremes of the edge e we consider; p_1 and p_2 are the new control vertices to be computed. The normals in the extremes are n_0 and n_3.

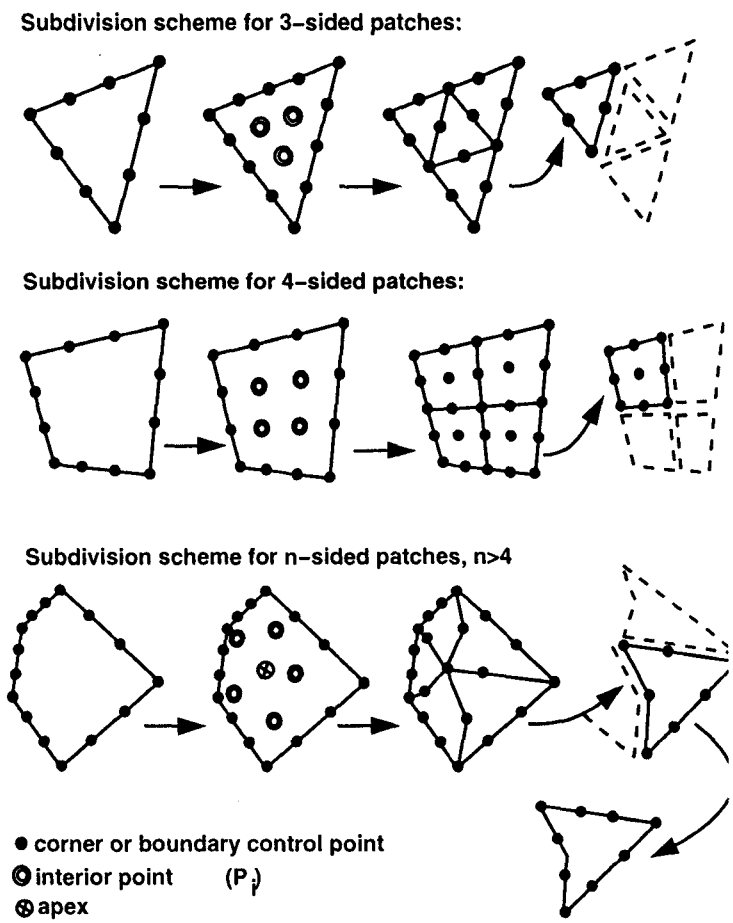


Figure 3. Subdivision schemes

A maximally stretched configuration

If we forget for a moment the requirement that the boundary vertices, regarded as control points, should direct the boundary curve to be perpendicular to the normal vectors, we could choose them on the edge $e = p_0p_3$. Then the control polygon becomes a straight line, which is in some sense maximally smooth. We call this configuration for $p_0...p_3$ 'a maximally stretched configuration'. In a maximally stretched configuration, p_1 and p_2 are on the segment p_0-p_3, and for an uniform parameterisation we can choose:

$$p_1 = \frac{2p_0 + p_2}{3}$$

$$p_2 = \frac{2p_3 + p_1}{3}$$

Of course, this configuration is at odds with the perpendicularity requirement, but it can serve to formulate a variational principle to arrive at p_1 and p_2 that are as close as possible to the maximally stretched configuration.

This variational principle has to fulfill the boundary conditions with respect to perpendicularity. If we denote the dot product of a and b by $(a \cdot b)$, then the boundary conditions are:

$$(n_0 \cdot p_1 - p_0) = 0 \tag{1}$$

$$(n_3 \cdot p_2 - p_3) = 0, \tag{2}$$

since n_0 and n_3 are normal vectors in p_0 and p_3, respectively. The variational principle becomes:

'minimise $M(p_1, p_2)$ where

$$M(p_1, p_2) = \mid 3p_1 - 2p_0 - p_2 \mid^2 + \mid 3p_2 - 2p_3 - p_1 \mid^2 . \tag{3}$$

subject to the boundary conditions 1 and 2'.

Using Lagrange multipliers, say μ_0 and μ_3, we have to solve

$$0 = \frac{\partial}{\partial p_1} M + \mu_0(n_0 \cdot p_1 - p_0) + \mu_3(n_3 \cdot p_2 - p_3)$$

$$0 = \frac{\partial}{\partial p_2} M + \mu_0(n_0 \cdot p_1 - p_0) + \mu_3(n_3 \cdot p_2 - p_3)$$

The result is (up to a multiplicative factor for the μ):

$$p_1 = \frac{p_3 + 3p_0}{4} - 5\mu_0 n_0 - 3\mu_3 n_3 \tag{4}$$

$$p_2 = \frac{p_0 + 3p_3}{4} - 3\mu_0 n_0 - 5\mu_3 n_3 \tag{5}$$

To establish the values of the multipliers, (5) is substituted back into (1), yielding the following set of linear equations to be solved:

$$5\mu_0(n_0 \cdot n_0) + 3\mu_3(n_0 \cdot n_3) = (n_0 \cdot p_3 - p_0)/4$$

$$3\mu_0(n_3 \cdot n_0) + 5\mu_3(n_3 \cdot n_3) = (n_3 \cdot p_0 - p_3)/4$$

It can be seen that if n_0 and n_3 are perpendicular to the edge p_0-p_3, the multipliers vanish and

$$p_1 = \frac{p_3 + 3p_0}{4}$$

$$p_2 = \frac{p_0 + 3p_3}{4},$$

which accords nicely with our intuition.

Minimal curvature

The demand for a maximally stretched configuration often gives satisfying results; it may nevertheless seem to be somewhat arbitrary. A more natural requirement (although somewhat more elaborate to derive) is the demand for minimal average curvature of a resulting cubic curve. Again, this curve will represent, for the course of our algorithm, the border segment of the mesh that is to replace a given edge of the input polygon. Before the resulting mesh is output, this curve is sampled in the desired number of points in order to yield border vertices of the mesh. Call the curve $p(t)$, $0 < t \leq 1$, then the criterion is expressed as "minimise $M(p_1, p_2)$", where (compare with 3)

$$M = \int_0^1 \mid \ddot{p}(t) \mid^2 dt + \mu_0(n_0 \cdot p_1 - p_0) + \mu_3(n_3 \cdot p_2 - p_3) \tag{6}$$

Since the two additional control points, p_1 and p_2 will serve in a cubic Bezier subdivision scheme (see sec. 2.5), the curve $p(t)$ has to be expressed into the cubic Bernstein polynomials:

$$p(t) = p_0(1-t)^3 + 3p_1t(1-t)^2 + 3p_2t^2(1-t) + p_3t^3$$

This yields for the integral (apart from a multiplicative factor)

$$\begin{aligned} \int_0^1 \mid \ddot{p}(t) \mid^2 dt \quad = \quad & 1/3 \mid p_3 - 3p_2 + \\ & 3p_1 - p_0 \mid^2 + \\ & \mid p_2 - 2p_1 + p_0 \mid^2 + \\ & (p_3 - 3p_2 + 3p_1 - p_0 \cdot p_2 - 2p_1 + p_0). \end{aligned} \tag{7}$$

Again, as above, we proceed by demanding $\partial M/\partial p_1$ and $\partial M/\partial p_2$ to vanish; this gives

$$p_1 = \frac{2p_0 + p_3 - 2\mu_0 n_0 - \mu_3 n_3}{3} \quad (8)$$

$$p_2 = \frac{2p_3 + p_0 - 2\mu_3 n_3 - \mu_0 n_0}{3} \quad (9)$$

To establish the values of the multipliers, (9) is substituted back into (1), yielding the following set of linear equations to be solved:

$$2\mu_0(n_0 \cdot n_0) + \mu_3(n_0 \cdot n_3) = (n_0 \cdot p_3 - p_0)$$

$$\mu_0(n_3 \cdot n_0) + 2\mu_3(n_3 \cdot n_3) = -(n_3 cdot p_3 - p_0)$$

Observe that, in the case of normalised normal vectors, the coefficient matrix of this set takes the form

$$\begin{pmatrix} 2 & \cos\phi \\ \cos\phi & 2 \end{pmatrix},$$

where ϕ is the angle between the two normal vectors. This matrix is always non-singular. (The same holds for the matrix in the maximally stretched configuration).

2.5. Subdivision schemes

Subdivision schemes for double-curved surfaces, designed for meeting various geometric constraints, have long been used (e.g. [5]). In principle, we could adopt their approach, i.e. treating the polygons as sets of Gregory patches. Although this is a sound approach for geometric modelling, the circumstance that we are merely dealing with straight silhouettes removing prior to rendering suggests that conceptually simpler and more efficient schemes might be applicable. Here we give a detailed account of our subdivision scheme corresponding to the global description of the algorithm in the beginning of section 3; the steps in the method are illustrated in figure 3:

(In the algorithm, the vertices of the input polygon are interpreted as control vertices for Bezier curve/patch segments. Subdivision schemes are used to create more control vertices; at the end, the resulting bi-quadratic triangular Bezier patches are output as single triangles of the resulting mesh.)

- initially, two boundary control points are generated between any two subsequent corners of the polygon, e.g. with one of the methods of section 2.4
  - for every side, consisting of 4 control vertices (2 corners and 2 boundary control points) generate 5 control vertices according to figure 3 (the configuration with two quadratic Bezier curves); the middle point equals the midpoint as would be obtained with a cubic subdivision scheme; the tangent constraints are met, but we can proceed with a quadratic subdivision scheme for both halves of the curve rather then with a cubic subdivision scheme.
  - for every corner, generate an interior control point (the points labeled p_i in fig. 2) by means of the method of section 3.3;
  - the resulting 15 control points form 4 bi-quadratic triangular Bezier consisting of 6 control points each;
  - the three corner control points of each of these 4 triangular bi-quadratic Bezier patches are output as the corner vertices of a single triangle of the output mesh; the mesh therefore consists of 4 triangles.
- for a patch with 4 sides:
  - generate an interior control vertex to go with every corner (see 3.3); this gives 4 rows of 4 vertices each.
  - replace every row of 4 control points with 5 control points in the same way as explained for 3-sided patches. This gives 4 rows of 5 control vertices each.
  - replace every column of 4 control vertices by 5 control vertices again in the same way as explained for 3-sided patches. This gives 5 rows of 5 control points; these form 4 four-sided bi-quadratic Bezier patches of 9 control points each, or equivalently, 8 triangular bi-quadratic Bezier patches of 6 control points each.
  - the corner control points of these 8 triangular bi-quadratic Bezier patches each are each output as the corner vertices of a single triangle in the output mesh; the mesh therefore consists of 8 triangles.
- for a patch with n, $n > 4$ sides:
  - generate an interior control vertex to go with every corner (see 3.3); now we have $4n$ control vertices.
  - generate a control vertex which will serve as apex; for the apex, we compute the average of the corners (a_c) and the average of the interior points (a_i). Next, we form the convex combination $apex = \lambda a_c + (1 - \lambda)a_i$. Changing the value of λ controls the global curvature of the resulting surface. We have to take care, however, to take λ such that inflection points between the

apex and the corner points are avoided as much as possible, since this may cause an unwanted buckling in the resulting surface. Nevertheless, choosing λ usually leaves considerable freedom to shape the final surface.

- group the control points in n triangular configurations, each configuration defined by 7 control vertices: the apex plus the two interior control vertices plus two subsequent corners and the 2 boundary vertices between these two corners.
- upgrade that triangular configuration to a 9-control vertex triangle[4] by means of degree elevation (quadratic to cubic) of the 3-point curves that are connected to the apex.
- deal with this 9 control point-triangle as described above.

2.6. Finding interior control points

At several instances during the subdivision process, it is necessary to produce interior control points. The location of these interior control points is responsible for fulfilling the requirement that is related with smoothness near the corners (2.1.3). Consider two adjacent bi-cubic or bi-quadratic (sub-) patches, π_1 and π_2, somewhere during the subdivision process.

Let p, p_{12}, p_1 and p_2 be corner or boundary vertices. Let p be a common corner, and $p - p_{12}$ is tangent to the shared edge of the two patches. $p - p_1$ is tangent to another edge of π_1 and $p - p_2$ is tangent to another edge of π_2. The two patches should have to have a smooth junction in the shared corner, thus p, p_{12}, p_1 and p_2 should be co-planar. Observe that during the first stage of the subdivision process, this condition is guaranteed by (1) from section 3.1.2[5] and the proper location of the interior points in combination with the normal vector averaging after every subdivision phase has to guarantee this during the rest of the process. Now it can be seen that choosing the new interior points, say p_{i1} and p_{i2} in π_1 and π_2, respectively, also to be co-planar with p, p_{12}, p_1 and p_2 is a sufficient condition for continuous cross-boundary derivatives when subdividing. The simplest choice is

$$p_{i1} = p_{12} + p_1 - p;$$

$$p_{i2} = p_{12} + p_2 - p.$$

2.7. New normals

The output polygons have to be equipped with normal vectors as well, in order to meet requirement 1.2. and to yield a smooth surface after repeated subdivision. Moreover, requirements 2.1.2, 2.2 and 3.3 apply. For arbitrary normal vectors, associated with possibly non-monotonously curved patches, we observed that the conventional linear interpolation approach for normal vectors may yield significant errors in the shading. In the case of a monotonous curvature, however, linear interpolation works (even though normals that are obtained via linear interpolation will not be exactly perpendicular to the surfaces; most polygon rendering systems (both hardware and software) do not deal with exact normal vectors anyway.) As a conclusion, normal vector averaging produces good results in the quadratic subdivision regime, but since we don't know if the condition for monotonous curvature is met at the first pass, more sophisticated method has to be used there.

Since a minimal curvature criterion is used to find the intermediate boundary control points for the first step, a quadratic interpolation scheme for normal vectors rather than linear interpolation may be used to get the normal vectors in the first subdivision points: indeed, the quadratic interpolation formula (in the approximation where the edge and the two normals are co-planar) yields the correct normal vector in the case of a curve with minimal average curvature (see [11] for a derivation of quadratic normal vector interpolation).

Let p_0 and p_3 be the extremes of a curve segment, $\Delta = p_3 - p_0$, and n_0 and n_3 are the normals in these points. Then the normal vector matching the minimal curvature-curve reads (this is easily derived from the formula for the tangent of the minimal curvature curve in [11]):

$$n(s) = n_0 + (n_3 - n_0 + n_m)s - n_m s^2$$

Hence, in the midpoint for $s = 1/2$:

$$n(1/2) = \frac{n_0 + n_3}{2} + \frac{1}{4} n_m$$

where

$$n_m = -3 \frac{(n_0 + n_3 \cdot \Delta)}{(\Delta \cdot \Delta)} \Delta.$$

2.8. Evaluation; extending the range of applicability

The method as presented above generates a smooth polyhedral mesh for an input mesh that representes a closed man-

[4] These 9 control vertices are only the border control points of a triangular patch; indeed: a bi-cubic triangular Bezier patch is defined by 10 control points.

[5] This holds for 3-sided and 4-sided patches. In the case of >4 sided patches, the location of the apex influences the geometry near the apex. In general, it will not be possible to have all control points adjacent to the apex co-planar. However, after one recursive subdivision step, all polygons are triangular, so this problem will not affect the smoothness of the surface after repeated application of the subdivision.

ifold. It is essential that at every edge of the input mesh, exactly two polygons meet. In this section we demonstrate what may happen if this condition is not met, e.g. when there are one or more holes in the manifold. Colour plate 1 (upper left) depicts a simple polygonal model of a vase: this is a manifold which is topological equivalent to a cylinder, and hence it has two holes. Normal vectors in the vertices have been obtained by means of traditional normal vector averaging. In plate 2 (upper right) it has been rendered with Phong shading based on the traditional linearly interpolated normal vectors. Colour plate 3 (lower left) shows the result after replacing the polygons by means of the subdivision structure of section 3.2., where the new boundary vertices have been computed by means of the second method of section 3.1. As is clearly visible, the straight silhouettes have vanished. In contrast with what one might expect, however, the upper rim of the object is not a planar convex curve: instead, it displays an undulating behaviour which has cusps near the vertices of the upper rim. A remedy to this problem is to have normals in vertices that are on the border of a polygon mesh (i.e. vertices that are not fully surrounded by polygons) not to result from simply averaging the surface normals in the adjacent polygons, but to force them to lay in the plane through the two incident mesh border edges. In this manner, in the example of the vase all normals in border vertices will lay in a plane through the rim. Unfortunately, this alternative normal vector averaging scheme has to make use of 'global' knowledge of the topology of the input polygonal mesh. This global knowledge has to be provided during the modelling stage of the object to be rendered. The version where the normal vectors have been forced to lay in the plane through the incident mesh border edges is depicted in colour plate 4 (lower right).

3. Summary and conclusions

A method has been given for removing straight silhouettes of polyhedral scenes, based on the normal vectors in the vertices. The method works by replacing the polygons in the scene by triangle meshes; things are arranged such that the algorithm may be configured as a filter which operates prior to (hardware)-rendering of a stream of polygons. Some results of the method may be seen in the colour plates 1...8. Colour plate 1 shows a vase which is, in colour plate 2, rendered using phong shading based on linearly interpolating the normal vectors. In colour plate 3 the same vase is shown after application of the straight silhouettes filter. The artefacts that arise due to a conflict between the directions of the normals and the geodetic curves on the surface are removed by enforcing the vertex normals on the rim of the input mesh to lay in the plane through the incident mesh border edges, which may be seen in colour plate 4. Colour plates 5-8 repeat a similar sequence for an object with a somewhat more complex topological structure; notice the topmost polygon which is 8-sided rather then 4-sided.

References

[1] J. Allan, B. Wyvill, and I. Witten. A methodology for polygon mesh modelling. *Proc. CG International 89*, pages 451–470, 1989.

[2] C. Baja and I. Ihm. Smoothing Polyhedra using Implicit Algebraic Splines. *Computer Graphics (Proc. SIGGRAPH 92)*, 26(2):79–88, 1992.

[3] W. Boehm, G. Farin, and J. Kahman. A survey of Curve and Surface methods in CAGD. *Computer Aided Geometric Design*, 1:1–60, 1984.

[4] E. Catmull and J. Clark. Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design*, pages 350–355, November 1978.

[5] H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. *Computer Graphics (Proc. SIGGRAPH 83)*, 17(2):289–298, 1983.

[6] J. D. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics Priniciples and Practice*. Addison-Wesley, 1990.

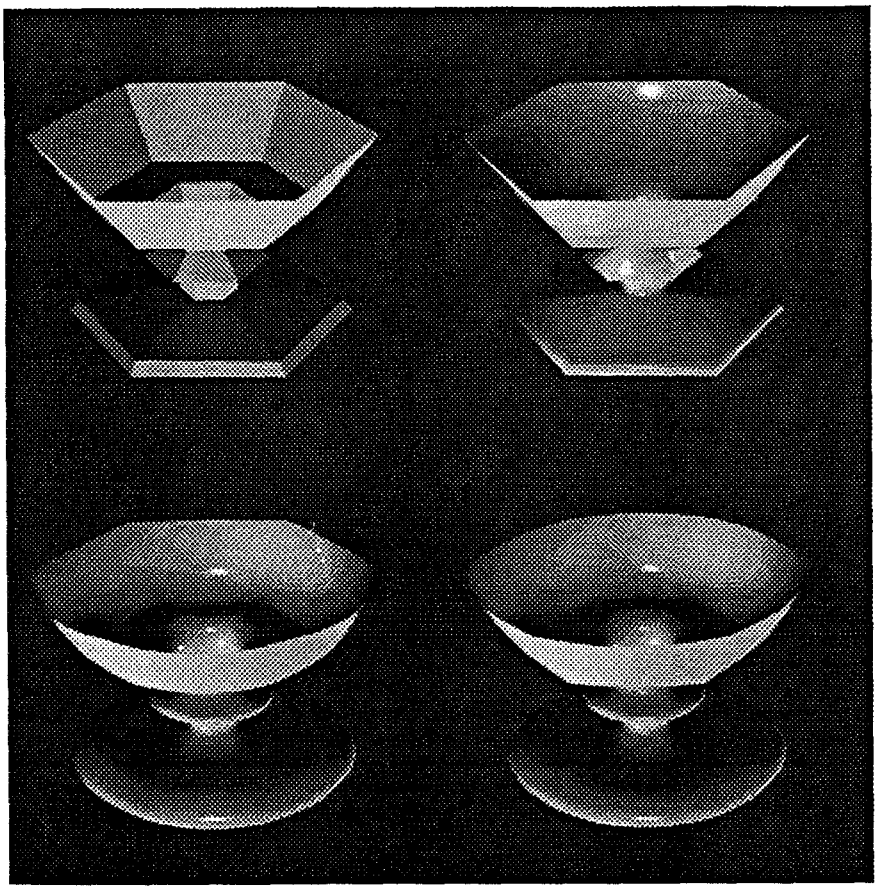
[7] S. Kuriyama. Surface modelling with an irregular network of curves via sweeping and blending. *Computer Aided Design*, 26(8):597–606, August 1994.

[8] C. Loop. Smooth Spline Surfaces over Irregular Meshes. *Computer Graphics (Proc. SIGGRAPH 94)*, 28:303–309, July 1994.

[9] J. Mallet. Discrete smooth interpolation in geometric modelling. *Computer Aided Design*, 24(4):178–191, April 1992.

[10] P.Fong and H.-P. Seidel. An implementation of multi-variate b-spline surfaces over arbitrary triangulations. In *Graphics Interface '92 (Morgan Kaufman)*, pages 1–10, 1992.

[11] C. van Overveld and B. Wyvill. Phong normal interpolation revisited. *ACM Transaction of Graphics*, XX(XX):XX, XX to be published.

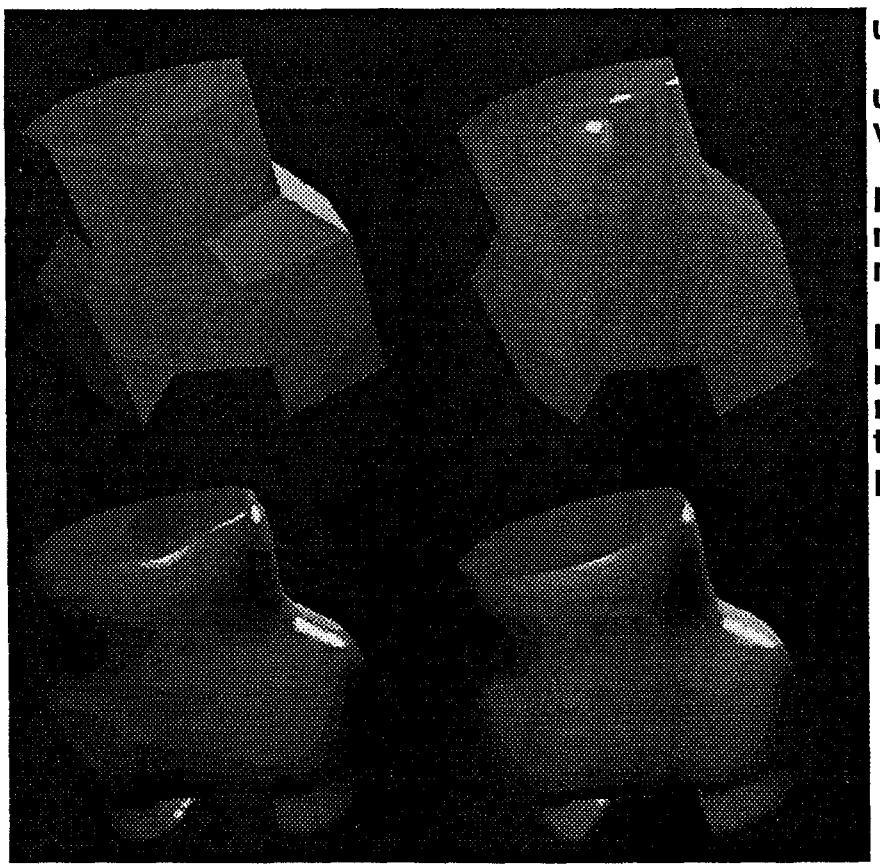


upper left: flat shading

upper right: Phong normal vector interpolation

lower left: straight silhouettes removed; polygons replaced by meshes. No precautions against cusps.

lower right: straight silhouettes removed; polygons replaced by meshes. Cusps are removed using enforced normal vector averaging.

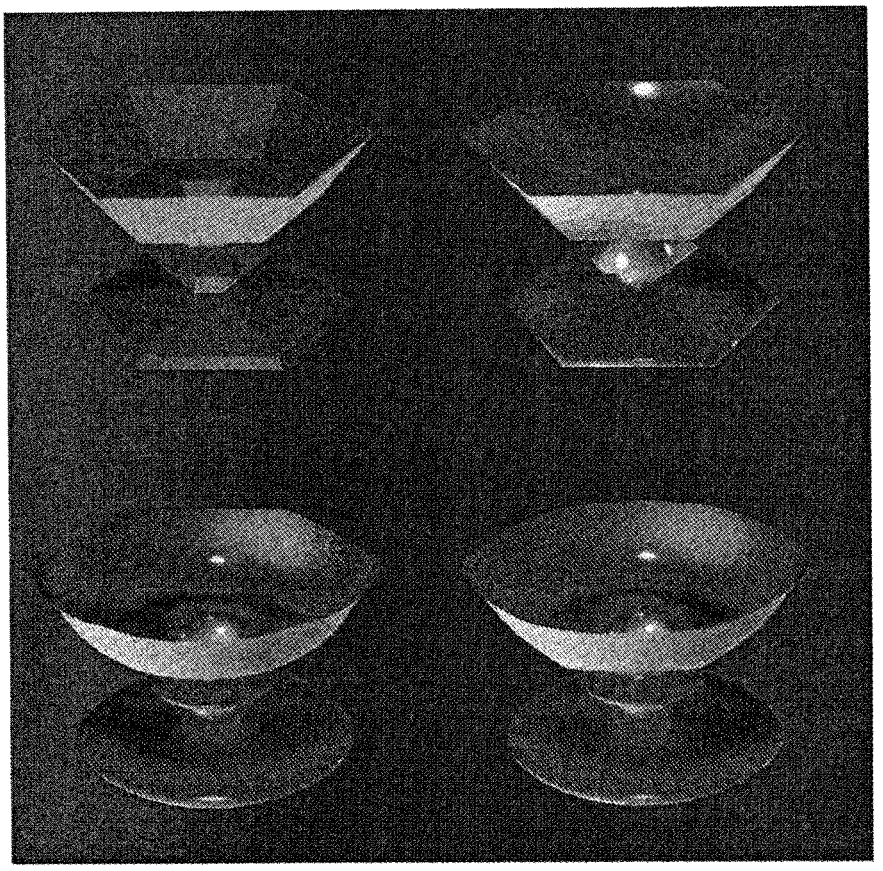


upper left: flat shading

upper right: Phong normal vector interpolation

lower left: straight silhouettes removed; polygons replaced by meshes.

lower right: straight silhouettes removed; polygons replaced by meshes. Normal vectors in the top plane were enforced perpendicular to the top plane.



upper left: flat shading

upper right: Phong normal vector interpolation

lower left: straight silhouettes removed; polygons replaced by meshes. No precautions against cusps.

lower right: straight silhouettes removed; polygons replaced by meshes. Cusps are removed using enforced normal vector averaging.

upper left: flat shading

upper right: Phong normal vector interpolation

lower left: straight silhouettes removed; polygons replaced by meshes.

lower right: straight silhouettes removed; polygons replaced by meshes. Normal vectors in the top plane were enforced perpendicular to the top plane.