An Adaptive Subdivision Algorithm for Crack Prevention in the Display of Parametric Surfaces

David R. Forsey
Computer Graphics Laboratory,
University of Waterloo,
Waterloo, Ontario,
Canada N2L 3G1

R. Victor Klassen
Webster Research Center,
Xerox Corporation,
128-29E 800 Phillips Road,
Webster NY USA 14580

Abstract

An algorithm is presented for rendering parametric spline patches using adaptive subdivision. Criteria for the termination of subdivision are chosen to allow large approximating polygons whenever their use does not result in visible errors in patch boundaries or in lighting anomalies. User specified tolerances can be used to specify the size of acceptable boundary and lighting errors controlling the speed and the quality of the rendering.

Keywords: Spline surfaces, surface rendering, adaptive subdivision, screen space criteria.

Introduction

A number of algorithms exist for rendering parametrically defined curved surfaces. The purpose of such algorithms is to display surfaces that are smooth both in their interior and along silhouette edges, while expending a reasonable amount of computation. The two most common methods of displaying such surfaces are subdivision[4], [5], [10] and forward differencing[12], [19]. For each of these methods there is a speed/accuracy tradeoff. Because high performance graphics workstations can render polygons very quickly, the dominating cost is in finding the polygons to draw, rather than drawing them. Choosing polygons smaller than screen pixels gives a high degree of accuracy, with a correspondingly high cost. For this reason a number of systems choose the polygon size based on the local curvature of the surface, in order to draw flat regions using as few polygons as possible. Adaptive methods vary the parametric step size within a patch so that small polygons are only drawn where needed. When adjacent sub-patches are subdivided to different levels (either recursively or using forward differencing with different step sizes), cracks may appear between the two patches. Figure 1 shows an exaggerated view of how this can happen.

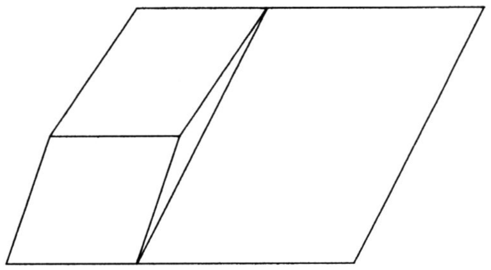

Figure 1: The patch to the left is subdivided more deeply than the patch to the right. As a result, a crack appears in the polygonal approximation.

This paper describes an adaptive subdivision scheme with screen-space crack prevention. Three principles guide the algorithm development. First, the criteria for terminating subdivision should be directly related to visible phenomena. Thus the factors that determine when a patch boundary is suitable for linear approximation do so within a screen-space tolerance. Secondly, the speed/quality tradeoff should be under user control. Edge error tolerance for a high-quality picture is set to avoid approximation errors larger than a half pixel thereby eliminating any visual evidence of error, but the user is free to increase the maximum edge error for faster previewing. To control lighting discontinuities with this approximation, a second parameter determines when the patch is considered flat enough for polygonal approximation. The third principle is simplicity. No complex data structures are needed as the algorithm recursively renders each patch separately without the need to pass boundary information between patches.

The body of the paper can be logically divided into three parts. First, section 2 outlines previous work related to our algorithm. Secondly, section 3 describes the algorithm itself, and section 4 describes a pitfall related to the way in which polygons are sometimes rendered that may cause the algorithm to appear to be working incorrectly. Finally, section 5 discusses results and performance issues.

1. Previous Work

There are three contexts in which crack prevention arises. First, cracks can appear in surfaces rendered using forward differencing if the step size in adjacent patches differs. Secondly, and most relevant to this paper, they may appear when adaptive subdivision is used to render surfaces. Thirdly, some surface-surface intersection algorithms use adaptive subdivision, and must avoid errors resulting from cracks.

Scanline algorithms[9], [11] can avoid cracks by keeping adjacency information explicitly. While scanline algorithms are asymptotically efficient when implemented in software, they cannot use hardware polygon rendering to full advantage.

Algorithms based on forward differencing can render polygons that are all guaranteed smaller than a pixel [19], or approximately uniform in size [12], [18]. If they are approximately pixel-sized, visible cracks are unlikely. An alternative method [6] is to choose the step size based on the curvature, so that larger polygons can be used for patches of low curvature without introducing significant errors. In this case cracks may appear when different step sizes are chosen on two sides of a boundary. The remedy proposed by Filip *et al.* is to move points that lie on a patch boundary to the nearest vertex of a piecewise linear approximation of the boundary. Because the approximation to the boundary depends only on the boundary itself (and not the adjacent surfaces), vertices on both sides of the boundary are moved to points along the same curve, even if information about both patches is not available when either patch is rendered.

Adaptive subdivision was first proposed by Catmull [4], who suggested subdividing until all patches are less than one pixel in size. In this case cracks are completely avoided. A number of authors have suggested using a flatness measure, rather than screen-space size, as the stopping criterion for subdivision. (We defer discussion of specific flatness measures to section 3.4.) Adaptive methods using polygons larger than pixels need some means of preventing cracks if they are to give visually pleasing renditions.

Several strategies for crack prevention in the context of adaptive subdivision have been proposed. Nyddeger solved the problem by retaining a complete data structure of the subdivided control graph, and then inserting filler polygons wherever cracks were detected [14]. Clark uses Catmull's basis [4], subdivision of which requires fewer operations than the Bézier basis, and involves computing midpoints and then correcting them. Once a given edge satisfies the straightness test, the midpoint of the line segment connecting the edge endpoints is used in further subdivisions (the correction is omitted). Our method is similar, but provides more accurate shading than the straight line approximation.

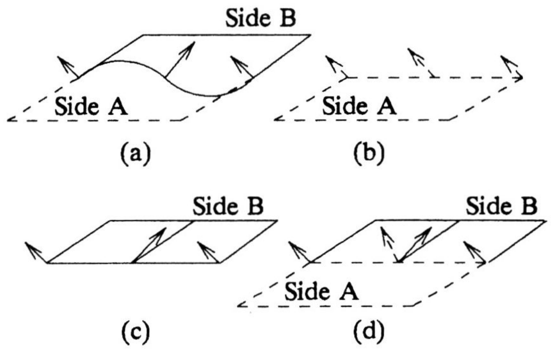

Figure 2: Lighting errors due to unequal sampling across patch boundaries. (a) patch boundary with correct normals, (b) linearly interpolated normals, (c) correct normals used on split side, (d) conflicting normals on boundary.

Barsky, DeRose and Dippé [2] proposed a method, which like Clark's, does not require information outside of the (sub)patch being rendered to avoid cracks. It differs in two significant ways. First, their method uses the actual patch boundary for subsequent subdivisions even once an edge is considered straight, and also keeps information indicating that final approximating polygon edges must lie along the line joining the endpoints. Thus polygon vertices generated at lower levels of subdivision are on the line at the points closest to the corresponding points on the true boundary. Secondly, lighting information is calculated from the actual control mesh. There is a potential lighting inconsistency across a boundary in the Barsky algorithm, although probably not visible at the tolerances the authors intend. It occurs where a curved boundary is shared between two patches, one of which is rendered as flat, while the other is split. Figure 2 (a) shows an (exaggerated) edge that, for the sake of illustration, is the boundary between two such sub-patches. Since the one side is approximated with a single polygon, and the normals at both ends of the boundary are the same, there is no variation in intensity along the edge on that side, for either Gouraud or Phong shading (b). On the other side the patch is split, and the actual behaviour of the normal is reflected in the shading of the two polygons (c). In this case there is a significant difference between the computed normals on the two sides of the boundary at the midpoint (d).

Adaptive subdivision is also used in several intersection algorithms [10], [15]. In these algorithms cracks must be avoided entirely, lest the intersection curves be disconnected. To do this, Peng's method replaces edges with mathematically straight lines once the edges have passed a straightness criterion. For rendering purposes, small cracks are deemed acceptable as long as their projection onto the screen is too narrow to appear on the raster. This allows us to trade a small amount of positional accuracy for greater lighting accuracy.

2. Algorithm Outline

Our algorithm takes as input a set of patches, and begins by converting them to the Bézier basis. For the sake of presentation, we assume cubic patches, although the algorithm is in no way limited to degree three. (Catmull's basis could be used for cubics, giving greater efficiency in subdivision, but the Bézier basis makes the termination tests simpler). Each patch has any modelling and viewing transformations (but not perspective) applied to its control vertices, and is then rendered recursively as follows:

```
Render( patch )
    Save the unprojected control points
    Project the control points to screen space
    if Small( patch, $\gamma$ )
        draw patch as a polygon
        return
    mustsplit $\leftarrow$ false
    for each boundary of patch
        if $\neg$ boundary.straight
            Straight uses projected points
            if $\neg$ Straight( boundary, $\alpha$ )
                mustsplit $\leftarrow$ true
            else
                Straighten( boundary )
                boundary.straight $\leftarrow$ true
    if mustsplit or $\neg$ Flat( patch, $\beta$ )
        Split unprojected patch into
            four subpatches: p1..p4
        Render( p1 )
        Render( p2 )
        Render( p3 )
        Render( p4 )
        return
    else The patch has straight edges and is flat
        draw patch as a polygon
```

The key elements of this algorithm are hidden in the primitives Small(), Straight() and Flat(); and in the Split() and Straighten() routines.

2.1. Smallness

The size test is primarily intended for protection against particularly curved patches. Assuming the threshold size is small enough, deviations from straightness or flatness in such polygons cannot be rendered accurately anyway, and so no further time should be wasted on them.

Several criteria are possible; since patches are tested first by size it is important that the test is inexpensive. Our test is as follows: if the projections of the sixteen control vertices fit within an axis-aligned square of a user-specified size (γ) on the screen, then Small() evaluates to true.

2.2. Straightness

The main reason for straightness testing is crack prevention. Because edges are tested for straightness independent of any other patch properties, the result is consistent for both patches sharing an edge. Since such edges are straightened in a consistent manner, cracks are avoided entirely. There are several reasons for testing straightness in screen space. Foremost among them is the fact that an edge is considered straight as soon as it is safely possible, regardless of the depth of the edge.

Suppose a boundary edge is defined by the projected points p_0,p_1,p_2,p_3. Straight() is true if p_1 and p_2 are within α pixels of the pixels rendered along a line from p_0 to p_3, measured along a Manhattan grid. If the line is more horizontal than vertical, the vertical distances from the interior control points to the line are compared with α; otherwise the horizontal distance is used. This way of measuring deviation from the line is motivated by Bresenham's algorithm for selecting pixels on the line approximating the boundary [3].

```
Straight( $P_0$, $P_1$, $P_2$, $P_3$, $\alpha$ )
  translate polygon so $P_0$ is at the origin
  if morehorizontal( $P_0$, $P_3$ )
        Check for x-monotonicity (to within ½ pixel)
    if $P_0.x < P_1.x < P_3.x$ and $P_0.x < P_2.x < P_3.x$
      shear control polygon making $P_3.y \equiv 0$
      if $P_1.y$ and $P_2.y$ have different signs
        return $4/9 \max(|P_1|, |P_2|) < \alpha$
      else
        return $3/4 \max(|P_1|, |P_2|) < \alpha$
    else
      return false
  else
    perform similar operations with x and y exchanged
```

The multipliers are based on the worst case control polygons for a Bézier curve. If the signs change the curve can stray no further from the axis than 4/9 of the distance to the further point: this corresponds to the nearer point being on axis. If the signs are the same, moving the interior control point nearest the axis to the position of the further point only makes the curve go further from the axis. With both points at the same distance from the axis, the curve goes as far as 3/4 of the distance to the interior control points.

It is worth noting that because the first part of the straightness test guarantees that the points are monotonic in the long axis, patches with edges perpendicular to silhouettes are split until these edges are shorter than a pixel. Thus edge straightness together with the flatness criterion guarantees that silhouette edges of this type are well approximated. (Flatness comes into play when the exterior of a patch is planar but a bulge appears in the centre).

2.3. Straightening Edges

Edges are "straightened" by modifying the interior control points of the edge. In the methods of Barsky *et al.* [2], Clark [5] and Peng [15], the edges are made mathematically straight. Contrariwise, in our algorithm the edges are "straightened" by moving the interior control points onto the plane containing the end control points and the eyepoint. Currently we use the perpendicular projection of the control vertex onto the plane to determine its "straightened" position. Any such projection can yield edges that are not at all straight in world space, but whose curvature is invisible from the eye. Since straightness is designed specifically for crack prevention, this meets that need, without flattening patches that have an appreciable amount of curvature in the z direction. In this way a minimum of lighting errors are introduced by straightening edges.

2.4. Flatness

The size criterion is used to avoid runaways. The straightness criterion is used for crack prevention. A flatness criterion is used primarily to control errors in lighting, and (occasionally) correctness of silhouettes. If an edge is not straight, patches adjacent to it are not likely to be flat. Flatness tests are more expensive than straightness tests. Hence the flatness test is not performed until all four edges are found to be straight.

A number of flatness tests are given in the literature; we make no claim that ours is optimal. The best choice depends on the cost of the various operations involved. Riesenfeld and Lane were the first to suggest a flatness test [10]. In their method a patch is subdivided if the maximum distance of any control vertex to the plane through three of the corner control vertices is too large, or if the distance of any edge vertex to the line through the corresponding corner vertices is too large. Barsky and DeRose [1] use only the plane-point distance. Clark's method uses the parametric curvatures at the corners to decide whether the edges are approximately straight, splitting if they are not, but if they are, uses the parametric curvatures ($d^4f(u,v)/du^2dv^2$) to decide whether to split. In order to make the test resolution dependent, the depth component is divided out of the control vertices before the curvatures are computed. The method of Koparkar and Mudar [8] is to require straightness of all eight defining curves and co-planarity of the four corner control points. This criterion is more strict than necessary, as the four interior control points can be anywhere in the plane of the four corner points without affecting the planarity, as long as they are not outside the convex hull of the edge curves.

Wang has given an algorithm for finding the depth of subdivision required for a given degree of flatness, based on the control points of a patch [20]. Filip *et al.* improve on the bounds used by Wang to give a tighter estimate of the degree of subdivision required [6]. These last two methods are best suited for non-adaptive subdivision or forward differencing. A good test for adaptive subdivision uses a minimum of effort to decide whether no further subdivisions are required, rather than trying to estimate how many further subdivisions are needed.

Our flatness estimate is most nearly like that of Clark. It is motivated by the notion that the flatness test is entirely intended to avoid lighting errors. First the z components of the exterior edges are tested to see whether the outside control points are co-planar (x and y components must be acceptable for the straightness test to succeed). Then the normals of the corner rectangles and the centre rectangle (of the control graph) are compared until two of them are found that point in sufficiently different directions to indicate that splitting is required or all have been tested. If the cosine of the angle between two normals exceeds the parameter β, Flat() returns false.

2.5. Splitting

Patches are split into four subpatches by midpoint subdivision in both parametric dimensions [10]. Barsky [1] argues that it is more efficient to split in one parametric direction at a time. This approach certainly could be used here without adversely affecting the crack preventing quality of the algorithm and may be explored in a future implementation.

To minimize lighting discontinuities when a boundary curve is "straightened" but its related patches are not flat enough for polygonal approximation, the normals at the ends of the curve are calculated and linearly interpolated at lower levels of subdivision. Since boundary curves become straight at the same level of subdivision regardless of which adjacent patch is being rendered, the continuity of the normals between the patches is maintained in the final approximating polygons.

3. Caveat Implementor

This and other crack prevention algorithms do not always prevent single pixel errors along seams such as the one in Figure 1. While the point at the joint is mathematically on the line between the two endpoints, it may not be rendered as such using commonly implemented polygon display algorithms. This problem was noted by Pineda [17] and is an example of the need for a restartable DDA [13], [16]. If the hardware being used to render the polygons rounds the vertices to integers before computing the stepping coefficients, single pixel holes are likely to appear.

4. Results and Discussion

The algorithm exhibits reasonable behaviour over a wide range of tolerance values. The user controls two tolerance variables, α and β. The minimum polygon area, γ, is normally left at 1. The straightness tolerance, α, is measured in pixels and can take any positive value, although exceeding about half the size of the display screen results in distorted surfaces. The flatness tolerance, β may vary from 0 to 1. If the desired maximum normal variation is 10°, β would be set to $\cos(10°) = .9848$.

We illustrate the performance of the algorithm using a simple uniform bi-cubic B-spline surface with 64 patches, shown in Figure 3 rendered with 117232 sub-pixel sized polygons. Figure 3 serves as our reference image. A naive attempt to speed up the rendering process by simply increasing the acceptable minimum polygon size (without applying the crack prevention techniques of this or other papers), yields cracks across unequally sampled boundaries. Figure 4 is an enlargement of such a region showing several cracks.

The affect of α can be isolated by setting $\beta = 0$ (so that the flatness test never fails) while varying α. Figure 5 shows the polygons selected when the reference surface is rendered with α values of 100. The enlargement without the polygons outlined shown in Figure 6 shows the resulting errors. With α set to 1.0, no further improvements are possible. Figure 7 and 8 show the cases of $\alpha = 1.0$ and $\alpha = 0.5$ respectively.

Similarly, α can be held constant while β changes (Figures 9 to 12) In both cases the number of polygons varies from a few hundred to a few thousand with a corresponding improvement in the quality of the final image. The increase in quality is negligible from Figure 11 to Figure 12, in spite of a considerable increase in polygon count.

Because the act of straightening an edge also tends to flatten the patch, α and β are not independent variables. Different tolerance values can produce the same visual effect while using a considerably different number of polygons. The "optimum" setting depends on the nature of the surface and the relative importance of speed and accuracy.

In general we have found that α has the greater influence on the performance of the algorithm (measured in number of polygons generated) and the quality of the lighting, but this may be a side effect of the way edges are straightened. Future work will explore alternative projection methods.

5. Summary

This paper presents an algorithm for rendering parametric spline surfaces using adaptive subdivision. Patches that are smaller than a user-specified limit are rendered as polygons; patches are normally found to be flat and rendered before this test can succeed.

To avoid cracks, edges are tested for straightness and then straightened if they are nearly straight. Patch boundaries are tested until their projections into screen space are within a user-specified tolerance of being x-y straight, and their curvature in z is within a second user-specified tolerance of being flat. Once edges are straight in this sense, their interior control vertices are moved to the plane through the eye and the end control vertices, thereby guaranteeing that the patch on either side of the edge can be rendered as a single polygon without creating a crack should the other patch be further subdivided. Normal information is calculated before the edges are straightened and interpolated in further subdivisions, guaranteeing that if either side is Phong shaded immediately, the other may be subdivided and subsequently shaded without introducing any lighting discontinuities. When an edge straightness test succeeds, a flag is set so that no successful test need be repeated.

Once all four edges of a patch are tested the flatness is measured. This test is more expensive than straightness testing, but it seldom fails once the edges are all straight. Highly curved patches are subdivided until they have a small projection onto the screen; less curved patches can be approximated with large polygons.

Three user specifiable tolerances make it possible to vary the quality of the result in terms of silhouette degradation (maximum polygon size), geometric errors (deviations from straightness) and lighting errors (flatness). When the polygon size is small, a very strict tolerance in either straightness or flatness produces a high quality image. Relaxing the tolerances results in significant speedups for the purposes of previewing.

Regardless of the tolerances for errors, consistency across patch boundaries is maintained. Cracks do not appear, regardless of the straightness tolerance, and lighting discontinuities do not appear, regardless of the flatness tolerance. The images produced run the range of very reasonable in very little time to excellent in the same or less time than required by previous crack prevention algorithms.

6. Acknowledgements

Partial financial support for the research reported in this paper was provided by operating, equipment, strategic, and infrastructure grants from the Natural Sciences and Engineering Research Council of Canada, by the Province of Ontario's Information Technology Research Centre, by Digital Equipment of Canada and by Silicon Graphics of Canada. All the surfaces were created and rendered on an IRIS graphics workstation with a locally developed spline editor [7].

7. References

1. B.A. Barsky and T.D. DeRose, "The Beta2-spline: a special case of the Beta-spline curve and surface representation," *IEEE Computer Graphics and Applications*, **5**(9) pp. 46-58 (September 1985).
2. B.A. Barsky, T.D. DeRose, and M.A. Dippé, "An Adaptive Subdivision Method with Crack Prevention for Rendering Beta-Spline Objects," Technical Report UCB/CSD 87/348, Computer Science Division, University of California at Berkeley (March 1987).
3. J.E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM System Journal*, **4**(1) pp. 25-30 (1965).
4. E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, University of Utah PhD Thesis 1974.
5. J. Clark, "A fast algorithm for rendering parametric surfaces," *Proceedings of SIGGRAPH'79*, pp. 7-12 (1979).
6. D. Filip, R. Markot, and R. Magedson, "Using bounds on derivatives in computer aided geometric design," *Computer-Aided Geometric Design*, **3** pp. 295-311 (1986).
7. D.R. Forsey and R.H. Bartels, "Hierarchical B-spline refinement," *Proceedings of SIGGRAPH'88*, pp. 205-212 (August 1988).
8. P.A. Koparkar and S.P. Mudar, "Computational techniques for processing parametric surfaces," *Computer Vision, Graphics and Image Processing*, **28** pp. 303-322 (1984).
9. J. Lane and L. Carpenter, "A generalized scan line algorithm for the computer display of parametrically defined surfaces," *Computer Graphics and Image Processing*, **11**(3) pp. 290-297 (1979).
10. J. Lane and R. Riesenfeld, "A theoretical development for the computer generation and display of piecewise polynomial surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-2**(1) pp. 35-46 (January 1980).
11. J. Lane, L. Carpenter, J.T. Whitted, and J. Blinn, "Scanline methods for displaying parametrically defined surfaces," *Communications of the ACM*, **23**(1) pp. 23-34 (January 1980).
12. S-L. Lien, M. Shantz, and V. Pratt, "Adaptive forward differencing for rendering curves and surfaces," *Computer Graphics*, **21**(4) pp. 111-118 (July 1987).
13. M.L.P van Lierop, C.W.A.M. van Overfeld, and H.M.M van de Wetering, "Line rasterization algorithms that satisfy the subset line property," *Computer Vision, Graphics and Image Processing*, **41** pp. 210-228 (1988).
14. R.A. Nyddeger, "Data minimization algorithm of analytical models for computer graphics," Master's thesis, Department of Computer Science, University of Utah (1972).
15. Q.S. Peng, "An algorithm for finding the intersection lines between two B-spline surfaces," *Computer Aided Design*, **16**(4) pp. 191-196 (July 1984).
16. R. Pike, "Graphics in overlapping bitmap layers," *ACM Transactions on Graphics*, **2**(2) pp. 135-160 (April 1983).
17. J. Pineda, "A parallel algorithm for polygon rasterization," *Proceedings of SIGGRAPH'88*, pp. 17-20 (August 1988).
18. A. Rockwood, K. Heaton, and T. Davis, "Real-time rendering of trimmed surfaces," *Computer Graphics*, **23**(3) pp. 107-116 (July 1989).
19. A.P. Rockwood, "Generalized scanning technique for display of parametrically defined surfaces," *IEEE Computer Graphics and Applications*, **7**(8) pp. 15-26 (August 1987).
20. G. Wang, "The subdivision method for finding the intersection between two Bézier Curves or Surfaces," *Zhejiang University Journal, special issue on Computational Geometry (in Chinese)*, (1984).

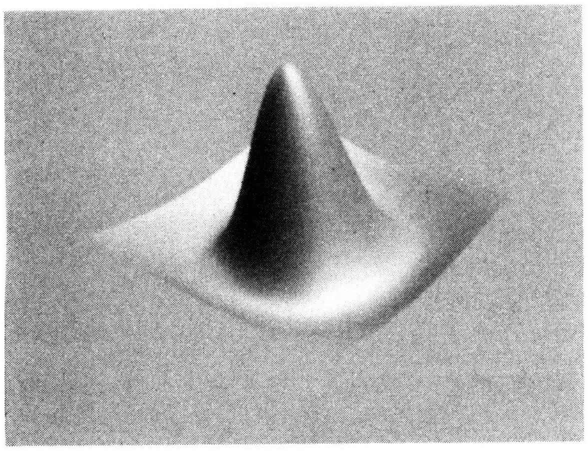

Figure 3: A bi-cubic B-spline surface with 64 patches rendered with 117232 sub-pixel polygons.

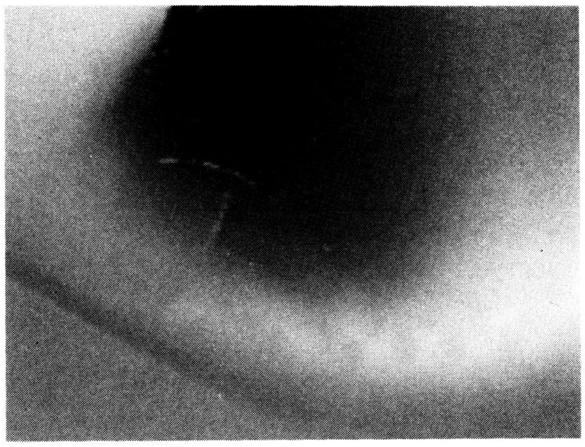

Figure 4: Cracks resulting from a naive polygonal rendering of the surface in Figure 3.

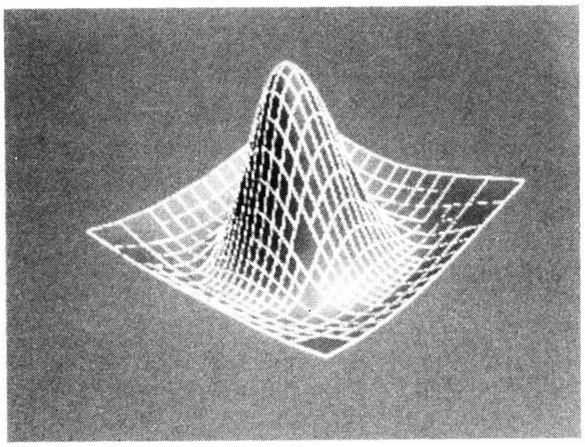

Figure 5: The 460 polygons created when rendering with $\alpha=100$ and $\beta=0$.

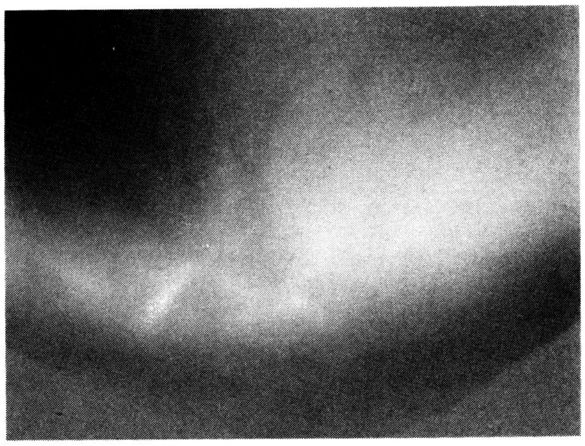

Figure 6: An enlargement of Figure 5.

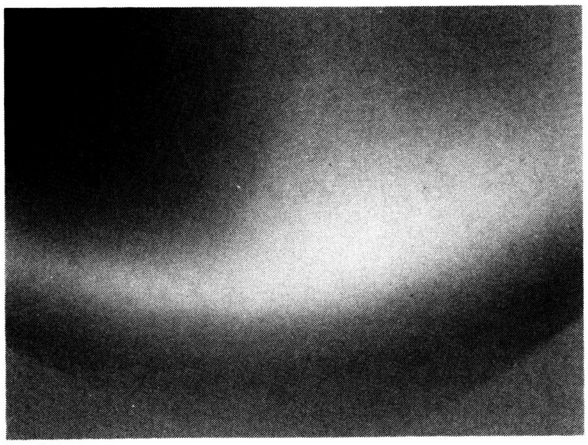

Figure 7: $\alpha=1$, $\beta=0$, 2881 polygons.

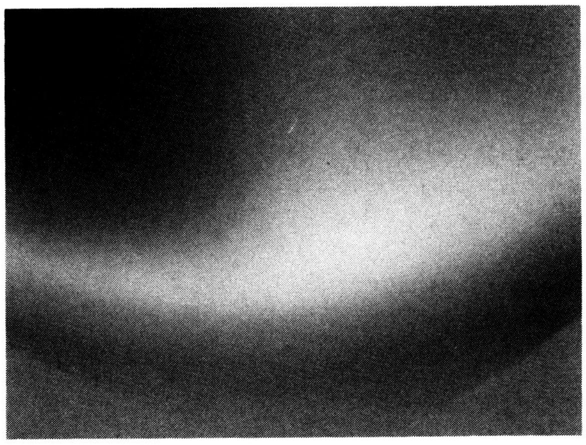

Figure 8: $\alpha=0.5$, $\beta=0$, 6193 polygons.

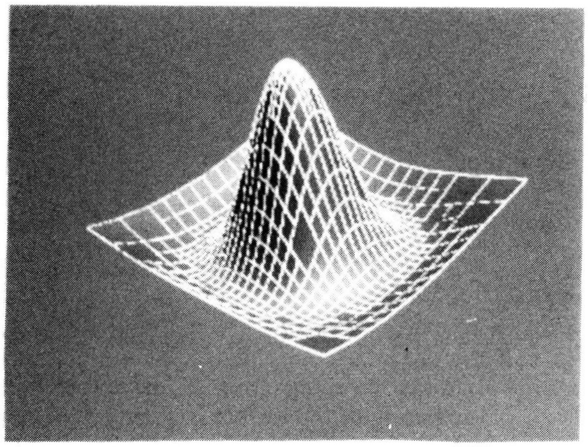

Figure 9: The 709 polygons created when rendering with $\alpha=10$ and $\beta=0.96$.

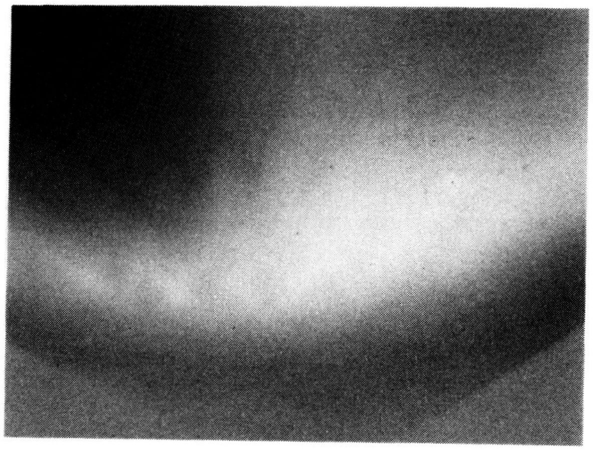

Figure 10: An enlargement of Figure 9.

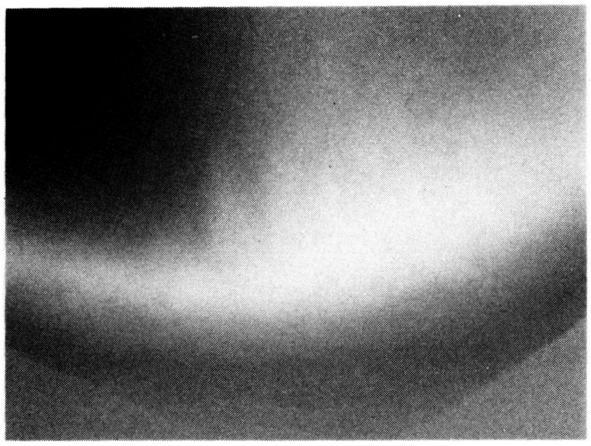

Figure 11: $\alpha=10$, $\beta=0.995$, 4582 polygons.

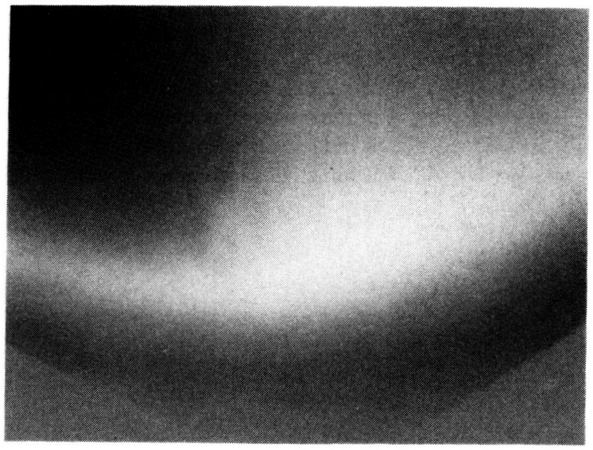

Figure 12: $\alpha=10$, $\beta=0.997$, 7342 polygons.