A SCAN LINE ALGORITHM FOR COMPUTER DISPLAY OF CURVED SURFACES

Turner Whitted
Department of Electrical Engineering
North Carolina State University
Raleigh, NC 27650

Abstract

The conventional procedure for generating shaded images of curved surfaces is to approximate each surface element by a mosaic of polygons and to then apply one of several established polygon display algorithms. The method described here produces an excellent approximation of bi-cubic parametric surfaces in scan line order. Each surface patch is described in terms of edge curves which are intersected by successive scanning planes to form endpoints of scan line segments. Visibility is calculated for each segment by a hybrid priority/z-buffer scheme. Shading is computed using Phong's illumination model with interpolated surface normal values. CR Categories: 8.2.

Introduction

Most scan line algorithms for shaded display [1-3] can be applied only to polygonal objects. Curved surfaces are approximated by collections of many small polygons which may be smooth shaded via techniques developed by Gouraud [4] and Phong [5] to produce images of good quality except along the object's silhouette. Algorithms by Mahl [6] and MAGI [7] display quadric surfaces directly, but such surfaces cannot be smoothly joined to form arbitrary shapes. The curved surfaces of greatest general interest are bi-cubic parametric surfaces such as Coons, Bezier, or B-spline patches. Catmull's subdivision algorithm [8] produces images directly from bi-cubic surface descriptions, but requires a frame buffer since it does not proceed in scan line order.

This paper describes a technique for directly rendering a shaded display of bi-cubic surfaces in scan line order. The process yields an approximation to the actual surface that is superior in most respects to polygonal approximations.

Any scan line display process must intersect a three-dimensional object with a series of horizontal planes spaced so that each plane corresponds to a raster line on the display device, a process called scan conversion. The intersection of one of these planes with the object's surface is a series of line segments in the scanning plane. To generate a scan line from these segments a shader assigns intensity values to each pixel lying along the projection of the visible portion of the segment. If the surface is polygonal, each segment is a straight line which can be uniquely determined by the intersection of the polygon edges with the scanning plane.

If a bi-cubic surface is to be displayed the intersection and shading problems become more difficult. Calculating a single point of intersection between the curved patch and the scanning plane requires the simultaneous solution of two equations, each of which are cubic in two unknowns. The central idea of the algorithm presented here is to avoid these difficult calculations altogether while rendering a reasonably accurate picture of the curved object. As in the polygon display algorithms, each surface element is described in terms of its edges (in this case cubic curves) which are passed to a scan conversion processor to form the endpoints of scan line segments. This approach fails, naturally, if the surface element contains a silhouette on its interior or if it is excessively curved. To circumvent this problem the processor that generates edges also detects silhouettes and divides the patch along the silhouette curve. If a patch is excessively curved, the edge generator can produce additional curves on the interior of the patch to improve the accuracy of the image. These preprocessor stages, which may be regarded as "scene preparation", are tailored to the specific type of surface rendered. As outlined in Figure 1, the preprocessor is followed by a scan conversion processor and the shader.

Edge Description of Patches

Bi-cubic surface patches are defined by the expression

$$f(u,v)=[u^3u^2u\ 1]\ M\ P\ M^T\ [v^3v^2v\ 1]$$

for $u,v \in [0,1]$. P is a matrix of "control

*This work was supported by Grant MCS75-06599, Div. of Computer Systems Design, National Science Foundation.

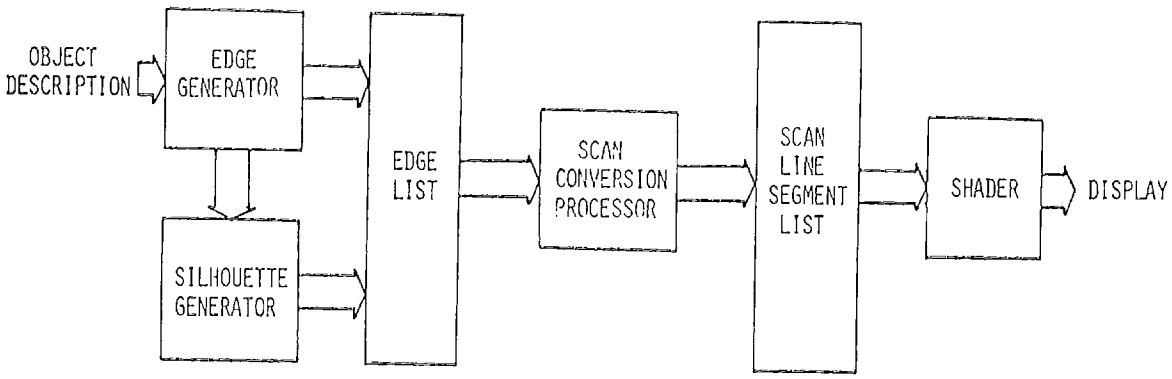

Figure 1 Display system block diagram

points" and M is a matrix of coefficients that define the interpolating function for the surface. Depending on the way that M and P are defined, the surface element may be a Coon's patch, Bezier patch, or B-spline patch. B-splines are used in all examples shown here. These patches have four natural "edge" curves: $E_0=f(u,0)$, $E_1=f(0,v)$, $E_2=f(u,1)$, and $E_3=f(1,v)$, each of which is cubic in one variable. If, as in the case of excessively curved patches, it is necessary to specify additional "edges" on the interior of the patch, these edges (parametric curves on the surface) are also cubic curves of one variable, defined by either

$$E_v=f(k_u,v) \quad \text{or} \quad E_u=f(u,k_v).$$

The addition of two such interior edges, specified by $k_u=0.5$ and $k_v=0.5$, has the effect of dividing the patch into four subpatches, as shown in Figure 2.

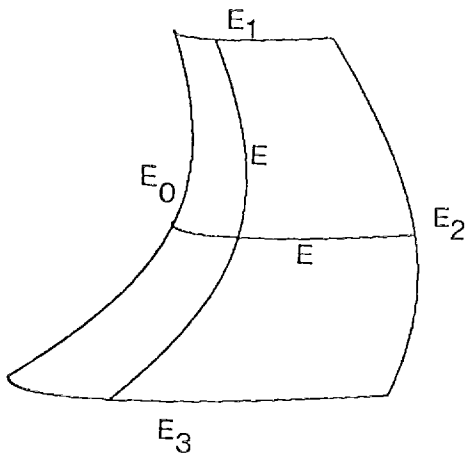

Figure 2

A third type of edge is the patch silhouette, i.e., the curve on the surface for which the z component of the normal vector is zero. In general, the order of the silhouette curve is greater than cubic, but it is approximated here by a piecewise cubic interpolant so that the silhouette can be treated the same as any other edge. If the silhouette curve passes through a patch, $f(u,v)$, there are two points, (u_a,v_a) and (u_b,v_b), each on an edge of the patch such that $N_z(u_a,v_a)=0$ and $N_z(u_b,v_b)=0$ where N_z is the z component of the normal vector. At each of these points a plane tangent to the surface is defined by the two vectors $\partial f/\partial u$ and $\partial f/\partial v$. Since any vector tangent to the surface must lie in this plane, the derivative of the silhouette curve can be expressed as a linear combination of the two vectors that define the plane. Then, a hermite interpolant joining the two endpoints can be specified by

$$p(t)=[t^3\ t^2\ t\ 1]\ M\begin{bmatrix} f(u_a,v_a) \\ f(u_b,v_b) \\ \alpha_u \partial f(u_a,v_a)/\partial u + \alpha_v \partial f(u_a,v_a)/\partial v \\ \beta_u \partial f(u_b,v_b)/\partial u + \beta_v \partial f(u_b,v_b)/\partial v \end{bmatrix}$$

where

$$M = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

The accuracy of the resulting silhouette curve depends on the number of cubic segments used in the piecewise approximation and on the choice of the α and β terms. Since each cubic segment spans the area between endpoints on the edges of a patch, the specification of additional edges on the interior of the patch containing the silhouette will improve the result. After the patch is subdivided by adding internal edges the silhouette generator examines each subpatch in turn to see if its edges are intersected by the silhouette and produces an approximating segment that spans the two endpoints. This approach to approximating the silhouette is similar to the one described in [9]. If the silhouette crosses the boundaries of a subpatch just once, or more than twice, or crosses any one boundary more than once, of it it is contained entirely within the subpatch, then the

silhouette generator defaults and an error occurs on the visible portion of that subpatch.

The choice of α and β terms in the interpolation formula determine both the direction and magnitude of the endpoint derivative vector. Since excessively curved patches are typically subdivided by the insertion of internal edges, one may assume that each subpatch examined by the silhouette generator is reasonably close to planar. Then a very simple approximation will suffice for α and β.

First let $\frac{|\alpha'_u|}{|\alpha'_v|} = \frac{|u_2-u_1|}{|v_2-v_1|}$

with $|\alpha'_v| \quad 1-|\alpha'_u|$

Then the first expression can be rewritten as

$$|\alpha'_u| = \frac{1}{1 + \frac{|v_2-v_1|}{|u_2-u_1|}}$$

To adjust for the arclength of the interpolant

$$|\alpha_u| = |\alpha'_u|\sqrt{(u_2-u_1)^2+(v_2-v_1)^2}$$
$$|\alpha_v| \quad |\alpha'_v|\sqrt{(u_2-u_1)^2+(v_2-v_1)^2}$$

with signs given by

$$\text{sign}(\alpha_u) = \text{sign}\ (u_2-u_1)$$

and

$$\text{sign}(\alpha_v) = \text{sign}\ (v_2-v_1).$$

Finally let $\beta_u=\alpha_u$ and $\beta_v=\alpha_v$.

Figure 3 shows the resulting silhouette approximation superimposed on a set of sectional curves.

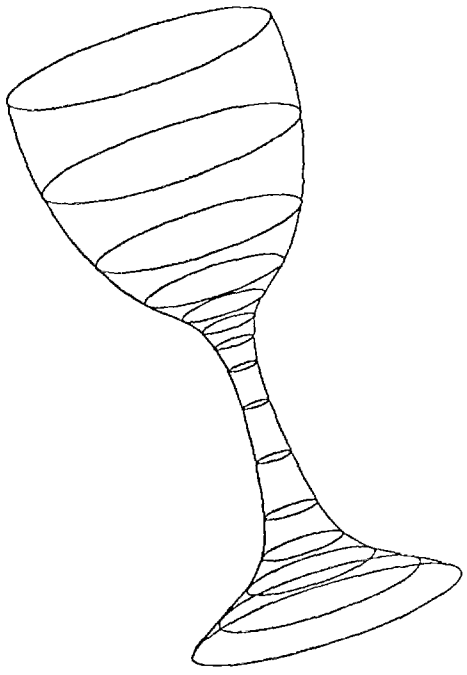

Figure 3

The definition of a cubic edge requires 12 coefficients: four each for the x, y, and z components. In addition, surface normal information along each edge must be provided for use by both the shader and the silhouette detector. If only an orthographic view is required in the final display, all of the required information can be obtained from the coefficients of the derivative with respect to the constant parameter* along each edge. The derivative with respect to the variable parameter (the curve's tangent vector) can be derived readily from the curve coefficients. The cross-product of these two derivatives yields an exact normal at each point on the edge.

Ordinarily a perspective view is required. A technique described by Catmull [8] uses a bi-cubic equation to approximate the normal vector on the surface, yielding a cubic normal function along each edge. The perspective view of the surface is generated by transforming the control points for the surface and using the resulting control points to form an approximation to the transformed bi-cubic surface. The bi-cubic normal approximation is not passed through the perspective transform since proper shading depends on preserving the object space illumination direction. Use of the approximate normal function has the added advantage of speeding the scan conversion process since it is not necessary to calculate cross products at every intersection point on an edge to find the surface normal. Edges are stored in a y-sorted list of modules, each containing 24 coefficients (12 for the edge curve and 12 for the cubic normal approximation). As noted before, the inclusion of interior edges effectively subdivides the surface element into smaller and more nearly planar subpatches. There are interesting differences between this approach and the subdivision of patches for approximation by polygons. Figure 4a shows a polygonal approximation of a bi-cubic patch created by evaluating the patch equation at 25 equally spaced vertex points. Assuming that the patch is surrounded by four neighbors with which it shares vertices and

*for the edge $E_u=f(u,k_v)$, $\partial f/\partial v$ is a cubic function with respect to u.

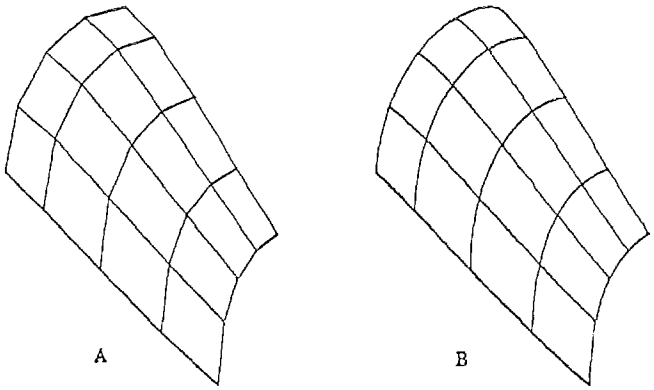

Figure 4

edges, the number of vertices per patch is 16 and the number of edges is 32. Each vertex is defined by six coefficients (three for position and three for the surface normal) and each edge description requires two pointers (one to each of the endpoint vertices). The total number of words required per patch is 160. Figure 4b shows the same patch in terms of its boundary curves and six internal edges. Assuming that the boundaries are shared, the total number of edges per patch is 8, requiring 192 words of memory. In general, if a patch is subdivided M times in the u direction and N times in the v direction and represented by quadralateral polygons, the number of edges required is 2MN per patch. For the same level of subdivision, only M+N cubic edges are required. (For Figure 4 M=4 and N=4.) Furthermore, every cubic edge lies entirely on the surface (except for the silhouette approximation) whereas if a polygonal approximation is used, the edges coincide with the surface only at the vertices.

Intersection Processor

The intersection (scan conversion) processor is the heart of this algorithm; it operates on the edge list and outputs scan line segments in reverse order of visibility.

The first stage of the procedure examines each edge to insure that it is monotonic in y, segmenting those that are not, and avoiding the problem of finding multiple intersections of the edge with a single scan line. The presence of extrema along an edge can be detected rapidly by examining the coefficients of the y component of the edge curve, and since the derivative of the curve is quadratic, their location is found using the quadratic formula to solve for zeros of the derivative.

The equation $E_y(t)=y_n$, where y_n is the y value of scan line number n and $E_y(t)$ is the component of the edge curve, is solved using Newton's iteration to yield t_n. In turn, $E_x(t_n)$, $E_z(t_n)$, and the components of the normal vector at that point on the edge are computed. Making use of scan line coherence, a first order estimate of the solution for the next scan line,

$$t_{n+1} \approx t_n - \frac{(y_n - y_{n+1})}{E'_y(t_n)}$$

results in rapid convergence of the next solution, usually in the first iteration. Because t is restricted to the [0,1] interval, Newton's method will occasionally fail to converge. In this case the scan conversion routine resorts to a brute force search for the solution.

Because internal edges and concavities lead to multiple pairs of intersection points, edges of a given patch must be sorted into ascending x order to insure the generation of proper segments. Note that this is a relatively cheap sort since a patch is typically intersected only a few times on any given scan line. As each segment is formed, it is inserted into a depth ordered list of all segments for the current scan line. In the interest of high speed processing, the depth separator test is limited to comparing the average depth of segment endpoints to establish the priority of segments. The test is performed in two dimensions instead of three and involves only scan line segments rather than entire objects or patches. Z ordering of the segments is included to enable the simulation of transparency, but it can be eliminated if only opaque surfaces are considered, since final visibility is established by z-buffer comparisons that are incorporated into the shader.

Shader

The shader operates on the list of scan line segments, assigning intensity values to the pixels spanned by each segment. With the segment list stored in reverse depth order, the shader can overlay segments into a pixel buffer in the manner of Newell's algorithm [2], but since the depth separator test does not always yield the correct result, a z-buffer is included for point-by-point visibility comparisons as in Myer's algorithm [3]. The z value of a scan line segment is linearly interpolated between endpoints. If two patches intersect, a piecewise cubic approximation of the curve of intersection should be included in the list of edges since point-by-point comparison between linearly interpolated z values will not usually yield a smooth curve of intersection. (The current preprocessor has no provision for generating the approximate intersection curves.)

Intensity calculations are performed by the method described by Phong [5] in which the normal vector is linearly interpolated between endpoint values. The shading function used is

$$S=\text{background}*T*f(N)+(1-T)*\text{hue}*(.25+.75(N\cdot I))+\text{glossiness}*(R\cdot I)^{60}$$

where N is the surface normal, R is the specular reflection vector, I is the incident light vector, and T is the transmission coefficient (equal to zero for opaque objects). The function $f(N)=(1-N_z^2)$ is used to create an illusion of thickness in transparent surfaces by attenuating the background in silhouette regions. Transmission coefficient, hue, and glossiness are stored in tables that are indexed by codes stored in the patch description, allowing each patch to have a different set of attributes. The constants shown in the shading equation have been chosen through a process of trial and error.

Implementation

The display algorithm is implemented as a series of modules written in Fortran and assembly language on an Adage AGT-30 computer. Display is on a standard 19" color television refreshed from an analog video disc. A specially designed interface allows the computer to write an image on

the disc one scan line at a time. These display routines are part of an experimental graphics animation display system (EGADS) being developed at North Carolina State University.

The modular construction of the software allows the use of several configurations for different applications. For the pictures of Figures 5 and 6, each object was processed separately and overlaid into a frame buffer. Figures 7 and 8 were generated by applying the preprocessor and intersection stages to each object separately and then merging all scan line segments into a large list. The entire list of segments was then passed to the shader. Each object in these figures is composed of B-spline patches. Generation times for such simple scenes vary from two minutes to ten minutes, depending on the configuration of the processor, with the shader occupying about 80% of the CPU time. A new shader, currently being developed [10], will improve the processor's performance by a factor of at least two.

Figure 5

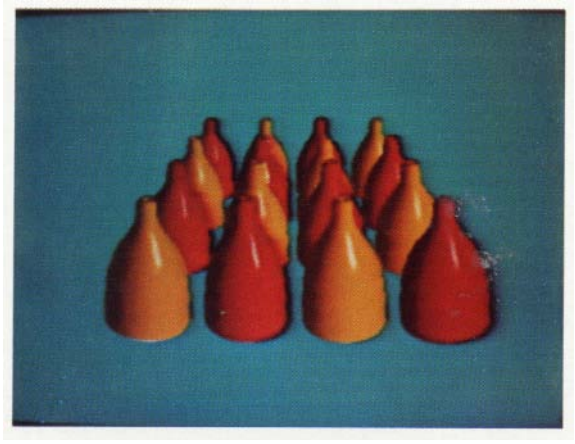

Figure 6

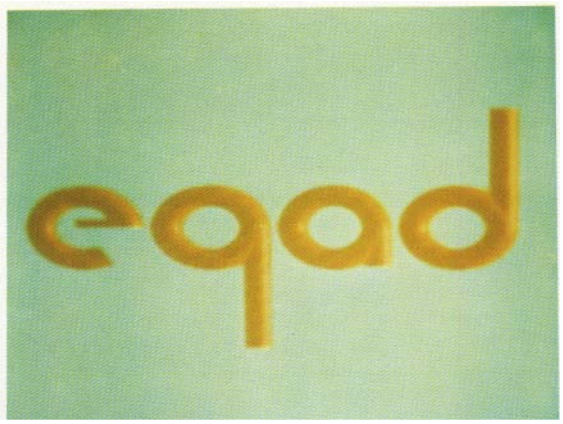

Figure 6

Figure 7

Summary

In the algorithm described here, the scan conversion process is performed only at the edges of surface elements; shading and visibility calculations on the interior of the surface are performed using interpolated data. To insure the quality of the final image, additional edges may be added along silhouettes, intersections of patches, and on the interior of the patch if necessary. The use of edges to describe the surface elements allows the display processor to operate only on univariate functions, resulting in a reasonably fast algorithm. In a sense, this algorithm represents an extension of existing polygon-based display methods, but it overcomes many of their shortcomings.

References

[1] Watkins, G.S. A real-time visible surface algorithm. UTEC-CSc-70-101, Computer Science Dept., Univ. of Utah, June 1970.

[2] Newell, M.E., Newell, R.G. and Sancha, T.L. A solution to the hidden surface problem. Proc. of the ACM Nat'l. Conf., 1972, pp. 443-450.

[3] Myers, A.J. An efficient visible surface algorithm. Computer Graphics Research Group, Ohio State University, July 1975.

[4] Gouraud, H. Computer display of curved surfaces. UTEC-CSc-71-113, Computer Science Dept., Univ. of Utah, June 1971.

[5] Bui Tuong Phong. Illumination for computer generated images. UTEC-CSc-73-129, Computer Science Dept., Univ. of Utah, July 1973.

[6] Mahl, R. Visible surface algorithm for quadric patches. IEEE Trans. on Computers, Vol 21, p. 1, Jan. 1972.

[7] MAGI, Mathematical Applications Group Inc. 3-D simulated graphics. Datamation, 14, Feb. 1968, p. 69.

[8] Catmull, E. A subdivision algorithm for computer display of curved surfaces. UTEC-CSc-74-133, Computer Science Dept., Univ. of Utah, Dec. 1974.

[9] Yoshimura, S., Tsuda, J., and Hirano, C. A computer animation technique for 3-D objects with curved surfaces. Proc. of the 10th Annual UAIDE meeting, 1971, pp. 3.140-3.161.

[10] England, J.N. and Whitted, J.T. A high speed shader for color computer graphics. Signal Processing Lab, Electrical Engineering Dept., NC State University, unpublished report.