A SCAN LINE ALGORITHM FOR DISPLAYING PARAMETRICALLY DEFINED SURFACES

James F. Blinn
Caltech/JPL

ABSTRACT

This paper presents a scan line algorithm for drawing pictures of parametrically defined surfaces. A scan line algorithm is characterized by the order in which it generates the picture elements of the image. These are generated left to right, top to bottom in much the same way as a picture is scanned out on a TV screen. Parametrically defined surfaces are those generated by a set of bivariate functions defining the X,Y, and Z position of points on the surface. The primary driving mechanism behind such an algorithm is the inversion of the functions used to define the surface. To keep the algorithm general enough to apply to a wide variety of functional forms, this inversion is done numerically. It is only required to provide a mechanism for evaluating the function and its derivatives at any parametric location.

The algorithm proceeds in two phases. First, a numerical search is made to find the local maxima of the Y definition function within the desired parameter ranges. These determine when portions of the surface first become visible as the scan plane progresses down the screen. Secondly, the actual scan conversion process is performed, maintaining a list of segments of the surface intersecting the current scan plane. As the scan plane passes local maxima of the Y function, new segments are added to the list. In additon, any existing segments are updated to reflect their intersection with the updated scan plane. All intersection calculations are performed by a bivariate Newton-Raphson solution of the defining equations. If the solution does not converge, it is due to the scan plane passing a local minimum, causing segments to be deleted from the active list. Finally, within one scan line, an X scan must be performed to generate the Z information about the surface for each picture element. This is also performed by a bivariate Newton-Raphson iteration with a different set of defining functions.

1. INTRODUCTION

Computer aided design has long been concerned with the design of surfaces represented parametrically. Such surfaces are those defined by three bivariate functions:

$$X = X(u,v)$$
$$Y = Y(u,v)$$
$$Z = Z(u,v)$$

As the parameters vary between 0 and 1, the functions sweep out the surface in question. The mathematical representation of these surfaces provides shapes with pleasing properites of continuity and smoothness. Until recently, the only method for drawing shaded pictures of such a surface has been to divide it into many polygonal facets and to apply any of several polygon drawing algorithms. A few years ago, Catmull[3] devised one of the first algorithms for drawing bicubic parametric surfaces directly from the mathematical surface formulation. While this algorithm generates images of superior quality, it still has some drawbacks. These have to do with speed and memory requirements and the ease of performing anit-aliasing operations. These drawbacks are eliminated by the class of algorithms known as scan line algorithms. Such algorithms generate the picture elements in order from left to right, top to bottom on the screen, much as a television might scan them out. The algorithm described here is a scan line based algorithm for generating such images which removes some of the dificulties of Catmull's algorithm without sacrificing substantially in picture quality. It is similar in form and intent to that of Whitted [8].

This paper presents a highly abbreviated version of the algorithm described in [1]. For further details on the algorithm and derivations of the equations the reader is referred to this publication. The presentation here will mainly consist of the overall process and show the special cases which must be considered.

2. SCAN LINE ALGORITHMS

The new algorithm is a generalization of more conventional scan line algorithms for drawing polygonal objects. It is therefore worth while to examine conceptually what is happening during a scan line algorithm for polygons. It is assumed for both the polygonal case and the parametric curve case that the objects to be drawn have been transformed to a screen space with X going to the right, Y going up and Z going into the screen. Furthermore, the perspective transformation is assumed to have been performed on all objects as described in [6] so that an orthographic projection of X and Y onto the screen is appropriate. In the case of parametric curved surfaces this serves to alter the form of the functions somewhat but the processing performed upon those functions remains the same.

A scan line algorithm basically consists of two nested loops, one for the Y coordinate going down the screen and one for the X coordinate going across each scan line of the current Y. For each execution of the Y loop, a plane is defined by the eyepoint and the scan line on the screen. All objects to be drawn are intersected with this plane. The result is a set of line segments in XZ, one (or more) for each potentially visible polygon on that scan line. These line segments are then processed by the X scan loop. For each execution of this loop a scan ray is defined by the eyepoint and a picture element on the screen. All segments are intersected with this ray to yield a set of points, one for each potentially visible polygon at that picture element. These points are then sorted by their Z position. The point with the smallest Z is deemed visible and an intensity is computed from it. The processing during the X scan is, then, fundamentally the same as the processing during the Y scan except for the change in dimensionality. During the Y scan, 3D polygons are intersected with a plane to produce 2D line segments. During the X scan, 2D line segments are intersected with a line to produce 1D points.

Many enhancements must be added to this basic scheme to make it practical. Most of these are referred to as taking advantage of the "coherance" of the picture. This basically means that many of the calculations are made incremental rather than absolute. The opportunity to do this is, indeed, much of the reason for generating pictures in scan line order in the first place. For example, the Y scan is responsible for constructing a list of all potentially visible segments which will be processed by the X scan. Rather than construct this list from scratch for each Y coordinate it is usual to keep the list around between scan lines and update it according how it has changed. Changes to this "active segment list" take three forms. As the scan plane drops below a vertex of the polygon which represents a local maximum, a new segment must be created and added to the list, figure 1a. As the scan plane drops below a vertex which represents a local minimum a segment must be deleted from the list, figure 1b. Finally, for those segments which remain in the list, the XZ coordinates of the endpoints of the segments must be updated to reflect their new position, figure 1c.

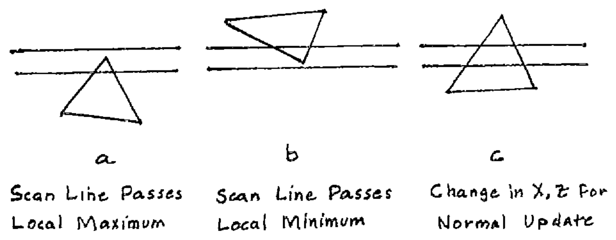
a
b
c
Scan Line Passes Local Maximum
Scan Line Passes Local Minimum
Change in X, Z For Normal Update

Figure 1 - Incremental Scan Line Operations

This latter operation can also be computed incrementally. The endpoint of an active segment is generated by the intersection of an edge of the polygon (a straight line segment) with the scan plane. The amounts of change in X and Z for a unit step in Y are constants along the entirety of the edge. The increments can be computed once when the edge first becomes active and just added to the XZ position for each step in Y.

The computation for the Y loop then reduces to the following processes. All endpoints are initially sorted in Y to determine the order in which they will pass through the Y scan plane. For each new Y, the X and Z coordinates of all existing segments are updated. If any polygon vertices have been passed new segments are created or old ones deleted according to the type of vertex. The calculations are analagously made incremental for the X scan. As it proceeds it maintains its own "active point list" of intersections.

It is useful to note an interpretation of the list of segment endpoints. Major events in the Y scan (indicating that elements are to be added or deleted from the active list) are signalled by the scan plane passing polygon vertices which are local extrema (maxima or minima) in Y. Similarly, major events in the X scan are signalled by the scan ray passing endpoints in the active segment list which are local extrema in X.

3. CURVED SURFACES

The generalization of the above algorithm to curved surfaces requires the handling of several new special cases. This section describes these cases and how to handle them. Most of the section is devoted to the Y scan portion of the algorithm. The X scan portion is handled similarly but is simpler due to its reduction in dimensionality.

3.1 Intersection Curves

The first new item about dealing with curved surfaces is that their intersection with the scan plane is not necessarily a straight line. For the parametric surfaces of interest here, the intersection curve in XZ space is defined in two stages. First we must find all parametric values (u,v) which satisfy:

$$Y(u,v) = Yscan$$

This equation defines some general curve in (u,v) space, called a "level curve". The actual XZ intersection curve is then generated by substituting all (u,v) values from the level curve into the X(u,v) and Z(u,v) surface definition functions. The big problem comes from the fact that, for most useful types of surface definition functions, there is no easy closed form representation for either of these curves.

The solution to the representation problem can best be done by examining what information about the curve is most desirable to represent. In the planar polygon case the intersections were all straight line segments and were represented by their endpoints. This list of endpoints was then used by the X scan to trigger changes in the active point list. They were thus used in two capacities; to represent the intersection curves and as a list of local extrema in X. For general intersection curves, of course, the endpoints are not sufficient to describe the entire curve. In particular there can be local extrema in X other than at the endpoints. These occur at the, so called, "silhouette edges" of a surface, figure 2.

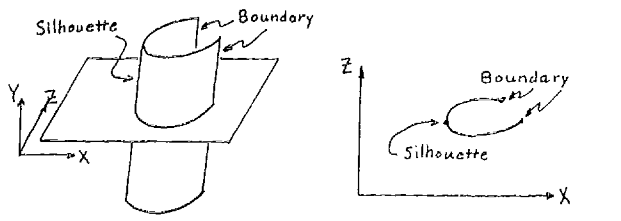

Figure 2 - Two Types of Edges For Curved Surfaces

The silhouette edge is defined mathematically as the locus of points on the surface which have a normal vector with a Z component of zero. A list of all points on the intersection curve which are the intersection of either silhouette edges or patch edges gives a good rough estimate of the shape of the curve and gives those locations in X where the X scan needs to change the contents of its X active list. These locations are most easily stored in terms of their (u,v) values. Each such point will be updated in (u,v) as the scan plane moves down the screen to track either a patch boundary edge or a silhouette edge. They will henceforth be referred to as "edge trackers". In the (u,v) space, the level curves are intersected by two kinds of lines, the patch boundaries and the Zn=0 curves. The edge trackers represent the intersections of these lines with the level curves as the Y scan plane moves down the screen. Several examples of typical sets of level curve intersections are shown in subsequent figures 6,7,8 and 9.

3.2 Operations On Intersection Curves

We now discuss the three major operations to be performed on intersection curves (and hence on the edge trackers whicn represent them) during the Y scan and how they are accomplished for curved surfaces. These will be described more in the order of their complexity rather than in their execution order.

3.2.1 Updating Edge Trackers Between Scan Lines

When the scan plane moves down by one raster element the (u,v) values of all edge trackers must be updated to represent the new Y position. For boundary edge trackers, e.g. for the v=0 edge, this requires solution of a univariate equation:

$$Y(u,0) = Yscan$$

In general we are not assuming this is solvable analytically. It can be easily solved numerically, however, by such common techniques as Newton-Raphson iteration. We even have a good value to use for the initial guess at the solution, viz. the u value on the previous scan line. The Newton iteration then refines this guess by the iterative scheme

$$unew = u - f(u)/f'(u)$$

$$\text{where } f(u) = Y(u,0)-Yscan$$

Various criteria can be used to determine when to stop iterating. The one used here is to iterate unti f(u) is less than one tenth raster element.

The silhouette edge trackers can be handled similarly. This time, however, there are two parameters to solve for and two defining equations:

$$Y(u,v)=Yscan$$

$$Zn(u,v) = 0$$

This can be solved by bivariate Newton-Raphson iteration using the current (u,v) values as initial guesses. This yields an iterative scheme

$$unew = u - (F\ Gv - Fv\ G)/(Fu\ Gv - Fv\ Gu)$$

$$vnew = v - (Fu\ G - F\ Gu)/(Fu\ Gv - Fv\ Gu)$$

$$\text{where } F(u,v) = Y(u,v) - Yscan$$
$$G(u,v) = Zn(u,v)$$

Again, this refinement of u and v is repeated until some convergance criterion is met.

3.2.2 Creating Edge Trackers

We now come to the question of where these edge trackers come from in the first place. The creation of intersection curves in general comes from the Y scan plane passing a local maximum of the surface. Strict local maxima occur only at (u,v) values where both the u and v derivatives of F(u,v) are zero. Several other things can happen instead at Fu=Fv=0 however. There might be a local minimum or a saddle point. These can be distinguished from local maxima by examining the second derivatives of F(u,v).

Saddle Point	$F^2uv - FuuFvv > 0$
Local Maximum	$F^2uv - Fuu\ Fvv < 0;\ Fuu,Fvv<0$
Local Minimum	$F^2uv - Fuu\ Fvv < 0;\ Fuu,Fvv>0$

All points where Fu=Fv=0 are called "stationary points" and are illustrated in figure 3.

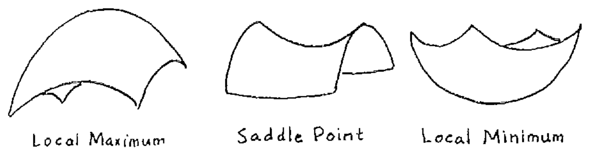

Figure 3 - Types of Stationary Points

Since the surface patch is bounded by the u=0, u=1, v=0, v=1 constant edges there can be local maxima within these boundaries that are not strict local maxima.

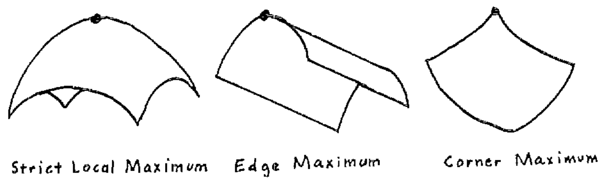

Figure 4 - Types of Constrained Local Maxima

For, e.g. the u=0 edge, these are characterized by Fv(0,v)=0 and Fu(0,v)<0 with similar relations for the other edges.

Finally a local maximum may occur at the corner of a patch. (These are the only kind which occur for flat polygons). These are characterized at, e.g. the (0,0) corner, by Fu(0,0)<0 and Fv(0,0)<0 with similar relations for the other corners.

Given that we know what to look for we now need to know how to find them and what to do when the Y scan plane passes one.

3.2.2.1 Finding Local Maxima

This process can again be solved numerically. The iterative scheme here is a combination of two techniques. One, commonly called "hill climbing" involves moving some initial guess uphill in the direction of the gradient of the function, (Fu,Fv). The second is a Newton iteration on the derivatives of the function to solve the two equations

Fu(u,v)=0
Fv(u,v)=0

The first technique is appropriate when the current guess is far from the local maximum and the second is appropriate to refine the guess when it gets close. Dahlquist et al. [4] gives a good algorithm for carrying this out. Such an algorithm must be modified, however, to take into consideration the constraints on the values of u and v. Whenever an iteration tries to move the current guess outside the patch boundaries the increment must be clipped at the boundaries. If the current guess lies on a boundary and the iteration tries to drive it outside then an edge maximum or a corner maximum is suspected. The appropriate criteria are checked to see if this is indeed the case. After running this process on several initial guesses (e.g. the corners of the patch and the center point) a list of local maxima in Y can be constructed to be referenced during the Y scan.

3.2.2.2 Passing Local Maxima

When the Y scan passes one of these local maxima some new edge trackers must be created to represent the new intersection curve. What types to create depends on the type of local maximum. For strict local maxima the new intersection curve will be (locally) elliptical and will contain two silhouette edge trackers. For edge maxima the new intersection curve will be (locally) parabolic and will result in two edge trackers. For corner maxima the new intersection curve will be roughly linear and will result in an edge tracker for two different edges.

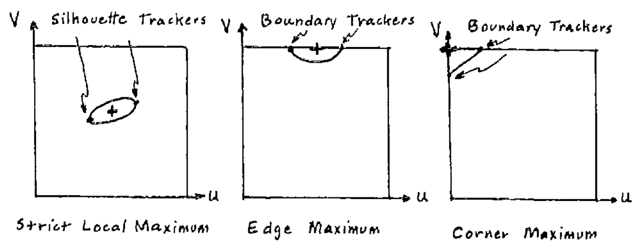

Figure 5 - Level Curves Just Below Local Maxima

An initial (u,v) value for these points must be found to be used for the Newton iteration which updates them to the next Y scan line. For the corner maxima this can just be the coordinates of the maximum itself (i.e. the coordinates of the corner). For the strict maxima and edge maxima this will not work. The Newton iteration which will operate on these points ultimately divides by derivatives of F and, at these maxima, these derivatives are zero. At such entering points, the second derivatives of F must be used to locally approximate the level curve of F, at the Y of the next scan line, by a parabola or an ellipse and solve this for an initial guess at (u,v).

3.2.2.3 Folded Edges

There is a case where a new edge tracker must be created other than at a local maximum in Y. This is the case where a new silhouette edge tracker coming within the patch boundaries due to a silhouette edge intersecting with a boundary edge. This is illustrated in figure 6.

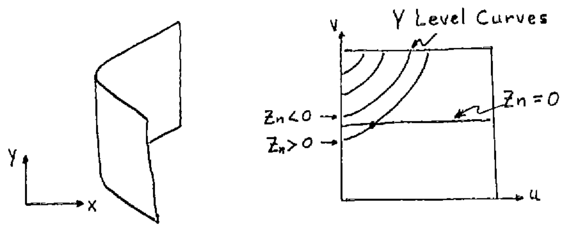

Figure 6 - Folded Edge Creating New Silhouette Tracker

To detect such cases, there must be a check of the sign of the Z component of the normal of each boundary edge tracker. If the sign changes from one scan line to the next the above situation has occurred. The new silhouette edge tracker may be created using the coordinates of the boundary edge tracker as its initial guess.

3.2.2.4 Saddle Points

Another place where silhouette edge trackers must be created is at saddle points of the Y function. Depending upon orientation silhouette edge trackers must be created or be deleted there. This is indicated by examining some second derivatives of the Y function at the saddle point.

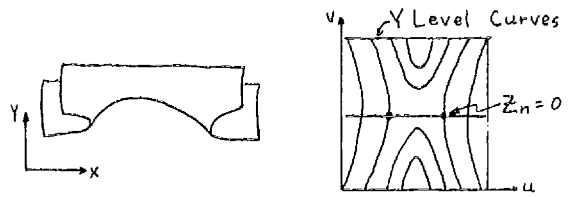

Figure 7 - Creation of Silhouette Edges at Saddle Point

3.2.3 Deleting Edge Trackers

An edge tracker is created either by passing a local maximum in Y or having a silhouette edge crawl in across the boundary. The edge trackers must be deleted in just the inverse situations, when passing a local minimum in Y or when a silhouette edge crawls out of the boundaries. The second of these is easy to detect. For each update of a silhouette edge tracker the resultant (u,v) values are checked against the patch boundaries. If either of them lies outside the range [0,1] the tracker is deleted. The local minimum case can be solved in sevaral ways. A list of local minima of Y can be made similarly to the local maxima. Then, upon updating Yscan, this list can be checked for the occurrence of points to delete. There are two problems with this. First, the list technique only tells where the local minimum is in (u,v) space. A matching process must be used to find the corresponding point in the list of edge trackers. Second, numerical operations on points near local minima can fail to converge. The problem has been solved in this algorithm by a modification to the Newton iteration. A few simple checks can be added to the iteration to detect failure to converge. Any points whichh fail to converge are then removed from the list of edge trackers.

3.3 The X Scan

The X scan is responsible for maintaining a list of active intersection points as a scan ray sweeps from left to right across the screen. Intersection points are those formed by the intersection of the X scan ray with a cross section curve in XZ. Points enter and exit at boundary edges and silhouette edges of the patch. These are just those points generated by the Y scan portion of the algorithm. These points should then be kept sorted according to their X values to ease the X scan. Between entering and exiting, points are updated in (u,v) (and thus in Z) by a similar bivariate Newton iteration as that for silhouette edges during the Y scan. In this case, however, the two functions which must be solved for zero are:

$$F(u,v) = Y(u,v) - Yscan$$

$$G(u,v) = X(u,v) - Xscan$$

3.4 Singularities

There are still some cases which can cause problems with the algorithm. These typically occur at singularities of the Y function. A singularity is basically a place where some quantity goes to zero where it needs to be divided by later. The most common of these is the parabolic cylinder maximum. This occurs when F^2uv - Fuu Fvv equals zero. It is thus an intermediate case between a strict local maximum and a saddle point, see figure 8.

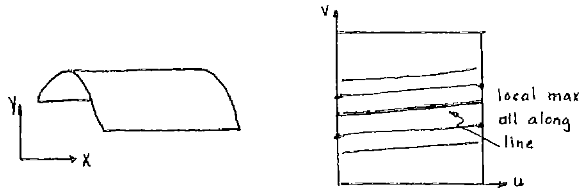

Figure 8 - Parabolic Cylinder

When this occurs, the level curve just below it consists of two straight lines. Thus, rather than two silhouette edge trackers, four boundary edge trackers must be created. The trick here is to determine how close to zero the above quantity must be in order to be considered a parabolic cylinder.

Another tricky case is the curtain fold problem. This is illustrated in figure 9.

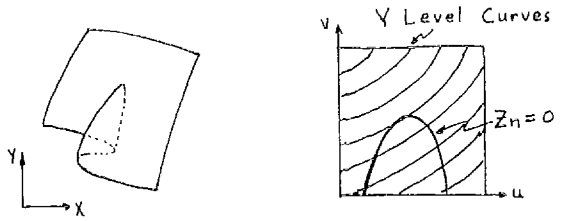

Figure 9 - Curtain Fold Singularity

This occurs when the Zn=0 curve in (u,v) space comes tangent to a level curve somewhere within [0,1]. The only correct way to solve this is to provide another pre-scan for such locations similar to the scan for local maxima in Y. A heuristic for avoiding this is given in section 4.

Finally, there is the general problem of singularities intersecting other singularities. For example a strict local maximum may lie on the boundary edge of a patch. The initial creation of silhouette edge trackers or boundary edge trackers is not so straightforward. Attempts to make such situations fall naturally out of their general cases have so far proven unsuccessful.

3.5 Accellerating Convergance

The average number of iterations necessary for convergance of the edge trackers has proven to be roughly 2.5. This can be improved substantially by an extrapolation process on the path of the point in (u,v) space. If the (u,v) position of the point on the provious Yscan value is saved an initial guess fed to the Newton iteration is computed, not as the current position byt as the the position extrapolated from the previous two. The result is an average number of iterations per point of less than 1 since some initial guesses already satisfy the termination criterion.

4. SPEEDING THE X SCAN

The X scan portion of the above process, while completely accurate, is quite time consuming. A refinement of the algorithm replaces this X iteration with the dynamic selection of a set of sample points in this dimension and interpolating the intensity (Gouraud [5]) or surface normal (Phong [7]) between them. This section tells how these sample points are defined and manipulated and some of the implications of using them.

4.1 Definition of X sample points

The effect of the selection of sample points is to approximate the intersection curve in XZ with straight line segments. We wish to choose these in a manner which most closely approximates the curve. A good technique is to chose them at equally spaced intervals of the angle of the normal to the curve. This tends to cluster the sample points at areas of high curvature. For example, a circle will be approximated by a regular polygon while an ellipse will have its points clustered near the sharp ends.

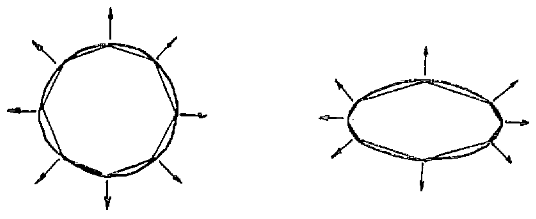

Figure 10 - X Z Sample Point Definition

The normal to the intersection curve can be found from just the X and Z components of the surface normal. The mathematical definition of a normal pointing at angle ϑ is

$$\vartheta = \mathrm{atan}(Zn/Xn)$$
$$\text{or} \quad Xn \tan\vartheta + Zn$$
$$\text{or} \quad Xn \sin\vartheta - Zn \cos\vartheta = 0$$

This is a generalization of the concept of the silhouette edge tracker where the angle ϑ is 0° and 180°. We then define new types of edge trackers for equal spaced increments of ϑ from 0° to 360°. For the pictures shown here, ϑ is in increments of 22.5 degrees giving 16 samples around a complete circle. Each new type of edge tracker has its own defining G function calculated as above with the appropriate value of ϑ. It is not necessary, of course, to re-evaluate thhe sines and cosines whenever G is evaluated. They can be pre-computed and stored in a table. The index into this table then labels the type of edge tracker. (A somewhat different formulation for the Gi is necessary,as detailed in [1], if perspective pictures are desired.) When a local maximum in Y is passed an edge tracker must be created for each G function in the same manner as silhouette edge trackers were created before.

4.2 Linking Edge Trackers Together

The set of edge trackers for all the Gi functions are the sample points of the intersection curve. A list of just their locations in X,Z does not, however, provide sufficient information to reconstruct a polygonal approximation to the curve. Information about how the trackers are connected together must be maintained explititly by a set of likns between them. In particular, they must be linked together in the order in which they occur while tracing along the curve. This is easily done when new curves are created at local maxima since the curves are simple ellipses with one of each type of G function around the perimiter. When the scan plane passes over a saddle point or other such singularity some of the edge trackers will cease to exist. This is because a saddle point is a local minimum in Y for some normal vector directions. This results in some broken chains of level curve spans. These breaks are repaired by a "sideways" iteration, so named because it moves along the level curves of constant Y but varying ϑ. THis is sideways from the ordinary iteration which follows curves of constant ϑ but varying Y. Whenever this sideways iteration encounters a value of ϑ corresponding to one of the tabulated directions (and thus a G functions) it creates a new edge tracker and links it to the broken chain. The process stops when it encounters an already existing edge tracker. This process, while slow, only needs to be performed when the scan plane passes some singularity of the Y function.

4.3 Turn Points

The level curves at a particular scan line are represented by a chain of edge trackers, each one corresponding to an index into a table of sines and cosines of ϑ. This angle rotates continuously as the level curve is traced out. We therefore expect that the index flags of the edge trackers will differ by exactly 1 between any two connected points (except for the end of the table wrapping around to the beginning). One other case that can occur, however, is that the function ϑ can have local maxima/minima along the level curve, in which case the index value changes by 0 between the points which straddle this location. This change in the direction of rotation of the angle will occur at inflection points in the cross section curve. This often happens in the vicinity of saddle points where the curve could appear as below.

The mathematical definition of such turnabout points is derived from setting the directional derivative of ϑ (in the direction tangent to the level curve) to zero. This leads to the definition function:

$$T(u,v) = Fu(Zn\ Xnv - Xn\ Znv) + Fv(Xn\ Znu - Zn\ Xnu)$$

By creating trackers which follow along the zeroes of this function we can ensure that various crinkles and folds in the XZ curve are accurately represented. These are also the points at which the curtain fold problem occurs so watching the Xn,Zn direction there can enable detection of this problem.

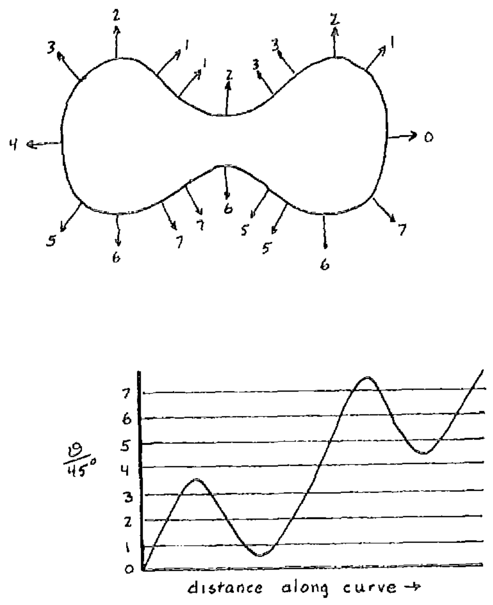

Figure 11 - Turn Point Definition

4.4 Results

The X sampling version of the algorithm may be thought of as dynamically slicing the surface up into one picture element high polygons. It can still yield some noticable errors in the image in some circumstances. This occurs notably at intersections of surfaces where the straight line approximateion to the intersections curves becomes obvious. In general, however, the approach combines some of the speed of polygon algorithms with some of the accuracy of full blown numeric patch algorithms to yield good pictures in a reasonable amount of time. For smooth shading, most of the time is spent in the Y iteration routines. This causes the computation time to be proportional to the number of active edge trackers times the number of scan lines during which they are active. The time to draw a sphere, for example, was roughly .4 seconds per scan line. Some of the more interesting applications of parametric surfaces, however, is the use of texture mapping, see [2]. In this case, there is a term in the timing which is proportional to the number of pixels covered by the object, and thus the square of the number of scan lines. For objects larger than about 100 pixels, this term dominates the timing of the algorithm and the time taken in the Y iteration becomes lost in the noise.

5. CONCLUSIONS

The curved patch algorithm generates much smoother looking pictures of curved surfaces than can be generated by polygonal approximation. Some pictures resulting from the algorithm appear in [2]. One disadvantage of the algorithm is its complexity. Several more special cases can arise than there is room to report on here. These are more thoroughly covered in [1]. The complexity issue can be improved considerably by further work on refining the techniques involved. There are still some heuristics involved in the algorithm which could be made more rigorous.

One final advantage of experimenting with such algorithms is that, by looking at scan line algorithms in a more general light, new insight may be gained in the properties of polygon based algorithms.

REFERENCES

[1] Blinn, J. F., "Computer Display of Curved Surfaces", Computer Science Department, University of Utah, Thesis, 1978.

[2] Blinn, J. F., "Simulation of Wrinkled Surfaces", Proc. 5th Conference on Computer Graphics and Interactive Techniques, 1978.

[3] Catmull, E. E., "Computer Display of Curved Surfaces", Proc. IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structures, Los Algeles (May 1975),11.

[4] Dahlquist, G., Bjorck, A., and Anderson, T., Numerical Methods, Prentice Hall, 1974.

[5] Gouraud, H., "Continuous Shading of Curved Surfaces", IEEE Transactions, C-20, (June 1971) 623.

[6] Newman, W. M. and Sproull, R. F., Principles of Interactive Computer Graphics, New York, McGraw Hill, 1973.

[7] Phong, Bui-Tuong, "Illumination for Computer Generated Pictures", Comm. ACM, 18 6(June 1975) 311.

[8] Whitted, J. T., "A Scan Line Algorithm for Computer Display of Curved Surfaces", Proc. 5th Conference on Computer Graphics and Interactiive Techniques, 1978.