This is to certify that the

dissertation entitled

SURFACE ASSESSMENT USING
COLOR GRAPHICS

presented by

Martin John Vanderploeg

has been accepted towards fulfillment
of the requirements for

Ph.D. **degree in** Mechanical Engineering

James E. Bernard

Major professor

Date 11/11/82

SURFACE ASSESSMENT USING COLOR GRAPHICS

By

Martin John Vanderploeg

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering

1982

ABSTRACT

SURFACE ASSESSMENT USING COLOR GRAPHICS

By

Martin John Vanderploeg

Modern techniques for the design and production of smooth skins, such as car bodies and aircraft wings and fuselage, depend upon the development of a database which accurately represents the surface. Before the database can be used for production, it must be checked for possible errors. Such errors may result from bad raw data, as from errors introduced in a digitizing process, or from designer errors. Typically these databases are very large, making the checking process tedious and expensive. Current checking methods include inspection of flow lines or sections, and may include building prototypes.

This dissertation develops a method for surface checking using shading and raster graphics. This method uses a scan line method which is based on improved numerical techniques and takes advantage of the rapidly decreasing cost of computer memory. These developments have resulted in a fast and accurate check procedure.

The dissertation illustrates the ability of the algorithm to detect specific surface flaws using a small test database. The ability of the algorithm to "assemble" objects on the screen is also demonstrated using a large database for an airplane fuselage and canopy.

ACKNOWLEDGEMENTS

I would like to extend special thanks to the people at General Dynamics for funding this work. Specifically, I thank Mr Bernard Breen and Mr. Larry Tucker for their technical assistance and support.

I also wish to thank the other members of my committee Drs. Erik Goodman and Ronald Rosenberg for their guidance and technical support.

Special thanks go to my parents, Marvin and Joann Vanderploeg for their loving support throughout my graduate work.

Finally, I would like to extend sincere thanks to my major professor and good friend Dr. James Bernard, for his guidance, constant encouragement, and especially for his extreme patience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

Introduction

Modern techniques for the design and production of smooth skins, such as car bodies and aircraft wings and fuselage, depend upon the development of an accurate mathematical representation of the surface. Before the surface definition can be used for analysis or production, it must be checked for possible errors. Such errors may result from bad raw data, as from errors introduced in a digitizing process, or from designer errors. Typically, the checking process is tedious and expensive.

This thesis develops a method for surface checking using surface shading and raster graphics that can reduce both cost and time required to check a surface. The next chapter discusses the mathematical representation of surfaces. Chapter 3 presents several categories of surface errors and current methods of surface checking. Chapter 4 reviews current methods of surface shading and display, and Chapter 5 presents an algorithm developed in this thesis especially for the purpose of surface checking. Chapter 6 then presents some examples, and Chapter 7 presents conclusions.

CHAPTER II

Curve and Surface Definitions

2.0 Generalized Curve and Surface Definitions

One way to define a curve in space consists of two transendental equations of the form:

$$F_1(x,y,z) = 0$$

$$F_2(x,y,z) = 0 \qquad (2.0.1)$$

where x, y, and z are three independent spacial coordinates. The curve consists of the locus of points which simultaneously solve these two algebraic equations.

Curves can also be defined parametrically. The curve becomes the locus of points whose coordinates are functions of a single independent parameter. The parametric form of a curve is:

$$x = f_1(s)$$

$$y = f_2(s) \qquad (2.0.2)$$

$$z = f_3(s)$$

The two forms may be illustrated using a straight line. The algebraic form consists of the equations for two planes, which form a line where they intersect. The algebraic form of the line is:

$$F_1(x,y,z) = A_1x + B_1x + C_1z + D_1 = 0$$

$$F_2(x,y,z) = A_2x + B_2y + C_2z + D_2 = 0 \qquad (2.0.3)$$

The parametric form of the line is:

$$x = E_1s + G_1$$

$$y = E_2s + G_2 \qquad (2.0.4)$$

$$z = E_3s + G_3$$

For simple curves it is possible to transform from one form to the other. As the equations become more complicated it is difficult or impossible to make this transformation.

Analogous forms also may be written to define surfaces. The algebraic form consists of one algebraic equation.

$$G(x,y,z) = 0 \qquad (2.0.5)$$

The surface is defined by the locus of points that solve this equation. The parametric form defines the surface as the locus of points whose

coordinates are functions of two independent parameters, s and t.

$$x = g_1(s,t)$$

$$y = g_2(s,t) \qquad (2.0.6)$$

$$z = g_3(s,t)$$

A plane may be used to illustrate the two different equation forms of a surface. The algebraic form is:

$$G(x,y,z) = Ax + By + Cz + D = 0 \qquad (2.0.7)$$

The parametric form is:

$$x = E_1 s + G_1 t + H_1$$

$$y = E_2 s + G_2 t + H_2 \qquad (2.0.8)$$

$$z = E_3 s + G_3 t + H_3$$

Again, transformations between the two forms are possible only for simple surfaces. The next section discusses the advantages of the parametric form, which will define surfaces throughout this thesis.

2.1 Parametric Curves and Surfaces

When producing shaded images from mathematical surface definitions, it is convenient to use the parametric form (1,2,3). One favorable property of the parametric form is the one-to-one mapping of spatial coordinates x, y, and z to the parametric coordinates s and t. A second advantage is the existence of well-behaved partial derivatives everywhere on the curve or surface. The importance of well-behaved partial derivatives will become evident as surface shading algorithms are discussed.

Throughout this thesis, the vector normal to the surface will be used for shading purposes. The normal vector of a parametric surface at a point is computed by taking the cross product of two vectors tangent to the surface at that point. A convenient pair of tangent vectors are:

$$\bar{V}_s^T = \frac{\partial x}{\partial s}\hat{i} + \frac{\partial y}{\partial s}\hat{j} + \frac{\partial z}{\partial s}\hat{k} \qquad (2.1.1)$$

$$\bar{V}_t^T = \frac{\partial x}{\partial t}\hat{i} + \frac{\partial y}{\partial t}\hat{j} + \frac{\partial z}{\partial t}\hat{k} \qquad (2.1.2)$$

The expression for the normal vector is then:

$$\bar{N} = \bar{V}_s^T \times \bar{V}_t^T \qquad (2.1.3)$$

Complex curves are typically broken up into several segments to facilitate fitting the curve with a series of low order polynomials.

In a similar way, complex surfaces are often broken up into several patches to facilitate fitting the surface with low order polynomials.

The boundaries of a patch are usually determined by limits on the parametric values s and t. A common convention allows these parameters to vary from zero to one.

2.2 Bicubic Parametric Surfaces

A popular surface representation is the bicubic polynomial (4). Although the ideas and analysis which are the basis of this thesis require only that the surface is parametrically defined, the bicubic surface is used throughout for illustrative purposes.

Equations for bicubic surfaces appear in many different forms. A general form of the equation is:

$$x(s,t) = \sum_{i=0}^{3} \sum_{j=0}^{3} X_{ij}\, s^i\, t^j \qquad (2.2.1)$$

where X_{ij} are constants. There are similar equations for y and z. Each patch is then delineated by 48 constants.

Equation 2.2.1 is often written in matrix form. The constants are contained in three 4-by-4 matrices, one for each equation. The matrix form is:

$$x(s,t) = [1 \; t \; t^2 \; t^3][X] \begin{bmatrix} 1 \\ s \\ s^2 \\ s^3 \end{bmatrix} \qquad (2.2.2)$$

where the matrix [X] contains the constants X_{ij}.

Bicubic parametric surfaces may also be represented in an alternative form which have been derived so that the constants defining the patch have geometric meaning. This form is

$$x(s,t) = \sum_{i=1}^{4} \sum_{j=1}^{4} X_{ij} * F_i(s) * F_j(t) \qquad (2.2.3)$$

where $F_i(s)$, so-called blending functions, are cubic polynomials. In this case the constants X_{ij} are geometric properties of the surface patch, such as spatial values or tangent vectors at various points on the patch. Similar equations are written for y and z.

One specific bicubic surface is the Coons surface. Coons surfaces are presented in detail in (4) and (5). Figures 1 and 2 are examples of surfaces defined by Coons patches.

CHAPTER III

Database Errors and Methods of Detection

3.0 Database Structure

A typical surface description is comprised of multiple surface patches. The database defining the surface consists of the constants which define the patches. For example, the bicubic patch is defined by 48 constants. The surface database for a bicubic surface then contains N blocks of 48 constants, where N is the number of patches comprising the surface.

When generating a database, errors may be introduced by the designer, or by the hardware. For example, mating parts may be designed by different people, introducing interference problems and slope mismatches along boundaries. Hardware related problems such as round off and hardware conversions may also create errors. In addition, design changes may create unexpected flaws. Detecting these errors, or so called, surface checking is an important design problem.

3.1 Surface Checking

There are three basic surface checking techniques which are commonly used.

One method is manual inspection of the constants defining the surface. This requires knowledge of the surface being modeled, and an understanding of how much each constant affects the shape of the patch.

This method is time consuming and inefficient for surfaces with a large number of patches, and considered impractical for production environments.

The second method of surface checking uses sections and flow lines to visually inspect the surface. Flow lines are constant s and t lines drawn at given intervals over a patch. They are easily computed and displayed. A viewpoint is chosen and the spatial coordinates of the constant s and t lines are projected onto a plane and displayed on a plotter or a graphics terminal. Figures 1 and 2 illustrate surfaces displayed using flow lines. Section cuts may also be displayed. Sections, which are more tedious to compute than flow lines, are useful for detecting surface flaws since they can be cut in any desired direction. However, it is usually not practical to make enough section cuts to properly check the surface.

A third method, which is often used serially with visual inspection via flow lines or sections, produces a three dimensional representation of the surface. This method uses the mathematical surface representation to drive a numerically controlled milling machine which produces a model of the surface. The model can then be inspected for flaws. This method is effective, but far more expensive than other techniques.

3.2 Types of Database Errors

When comparing the usefulness of different surface checking techniques, it is helpful to classify different types of errors that can exist in a database. The "bump" database, displayed in Figure 1, will be used to illustrate the different types of errors. The bump

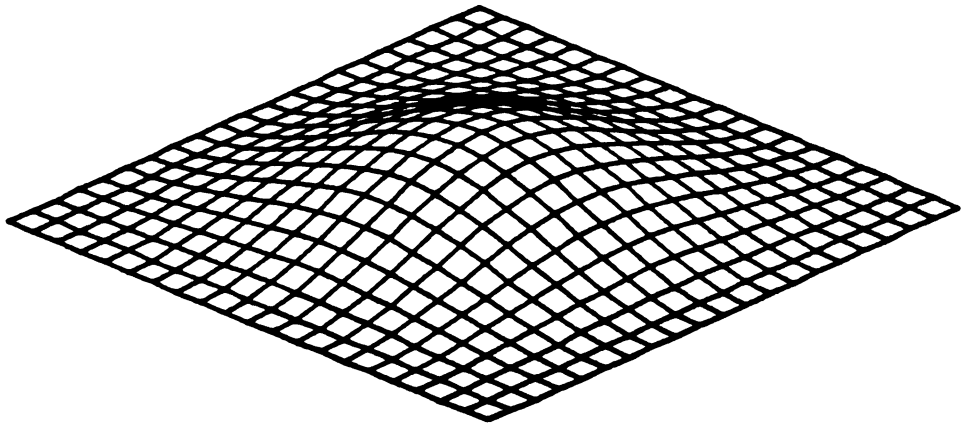


Figure 1. Flow Line Display of Bump

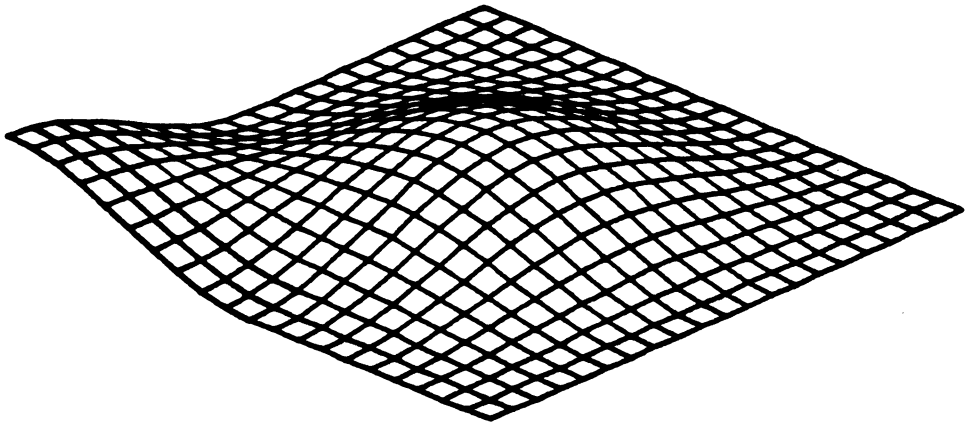


Figure 2. Gross Error in Bump

is made up of four patches which are based on 192 separate data entries in the database.

The first class of database errors contains gross errors. These errors are easily detected using flow lines. Figure 2 presents a gross error in the "bump", caused by one incorrect entry in the database.

The second error classification is missing data. Again, these errors are easily detected using flow lines. An example of missing data is presented in Figure 3, where one of the patches has been left out.

More difficult errors to detect are small slope discontinuities between patches. Figure 4 is an example of a surface containing an unwanted slope discontinuity. From this view it is impossible to visually detect the problem. With prior knowledge of the discontinuity location it is possible to choose a view looking down the patch boundary where the slope discontinuity exists. After magnification, this view is displayed in Figure 5, which is a closeup view of the patch centerline area as seen from the direction of arrow A in Figure 4. The slope discontinuity is now apparent, but finding the proper view depended on knowing the location of the slope discontinuity. For surfaces with many patches, the location of slope discontinuities are generally unknown. This makes identifying them with flow lines difficult, thus detection of this kind of error is often delayed until a model or prototype is constructed.

3.3 Surface Checking Using Shading

To avoid delays and costly proofing runs on numerical controlled machines, it is very desirable to facilitate detection of errors in

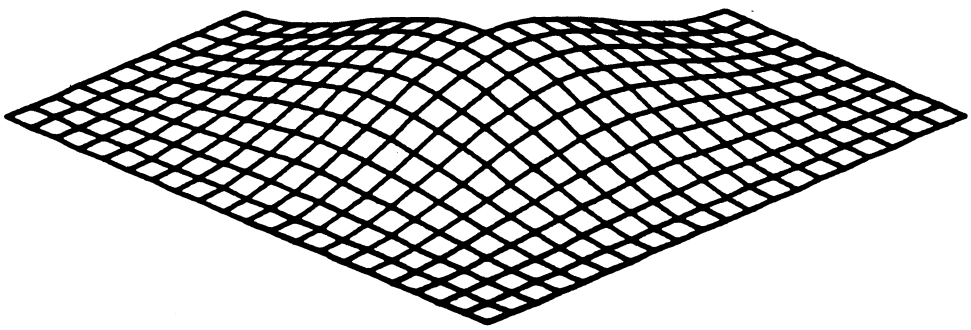


Figure 3. Missing Data in Bump

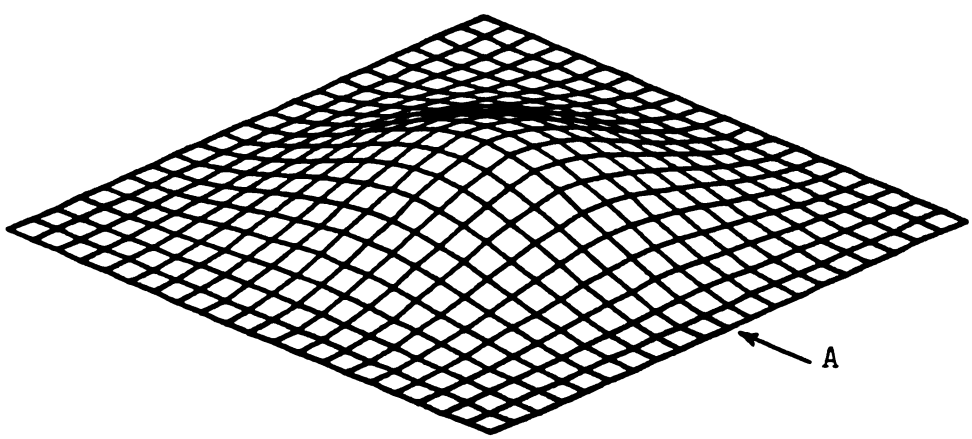


Figure 4. Slope Discontinuity in Bump

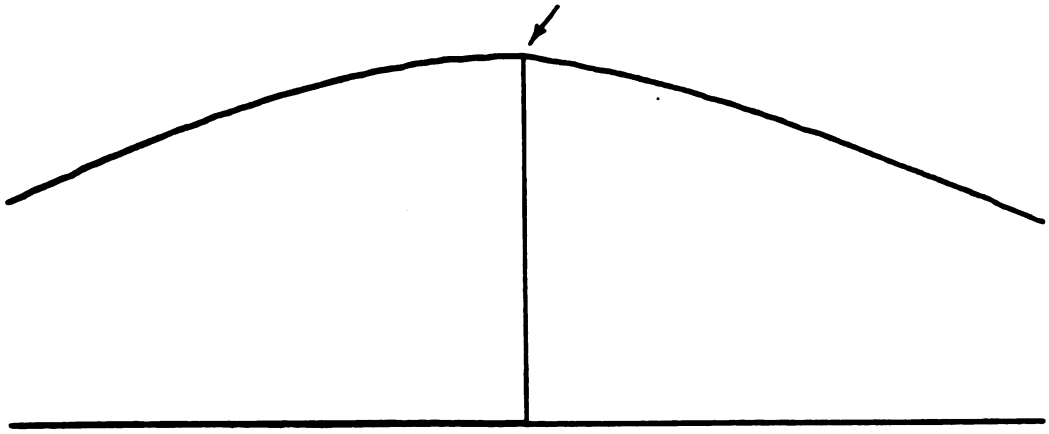


Figure 5. Magnified End View, Slope Discontinuity in Bump

the database using computer graphics. One promising technique is surface shading. By displaying the shaded surface on a variable intensity raster terminal, it is possible to locate subtle errors. The idea originates from the ability of the human eye to detect small deviations on smooth reflective surfaces. As an example, surface-related flaws on a car body are easily detected by the human eye, yet they are very difficult to find using only engineering drawings of sections or flow lines. If the surface can be represented visually by a computer as a reflective surface, these subtle errors can be detected.

The next section will introduce terminology and concepts concerning computer-generated pictures. Previous work done on surface shading will then be discussed.

CHAPTER IV

Introduction to Surface Shading and Display

4.0 Design Environment Requirements

When computer surface shading techniques are used for surface checking, the visual display must represent the surface properties accurately so that flaws may be identified. Shading algorithms often use approximations which make shading calculations easier, but these approximations may hide subtle flaws on the original surface. On the other hand, accurate processing of the details of the surface may require tedious calculations.

Since interactive programs are ideal for a design environment, fast shading algorithms are desirable so that a designer is not required to wait long periods of time for each image. The question of whether high accuracy or rapid calculation should have higher priority will be addressed in later chapters.

The next section describes how images are produced on a raster device. Basic concepts of surface display will then be presented, followed by a discussion of previous work done in the area of surface shading.

4.1 Raster Graphics

The raster screen is an increasingly popular method of displaying computer generated images. A common example of a raster device is the

television. Raster screens are made up of many small units called raster units or pixels. For a gray scale device, the user has control over the intensity of each pixel ranging from white to black. For color raster devices, the color and intensity of each pixel can be controlled.

The resolution of a raster device refers to the number of pixels on the screen. A screen with a large number of pixels is said to have good resolution, meaning each individual pixel is hard to discern, and the edges of objects being displayed appear smooth.

4.2 Raster Display of Surfaces

To aid in the discussion of surface display techniques, it is convenient to define a set of screen coordinates. Figure 6 illustrates the orientation of screen coordinates used in this discussion and throughout this thesis.

To display the color of surface on a raster screen requires that the intensity and color of each pixel be computed. This requires projecting a three dimensional surface to the viewing screen. Each pixel is then mapped to a point on the surface. The properties of the surface at that point are then used to compute the intensity of the pixel.

Typically, the mapping of a pixel to a point on the surface requires the construction of a sightline from the viewpoint through the pixel being mapped. All patches intersected by the sightline are determined. These patches are compared to determine which patch is visible along the sightline.

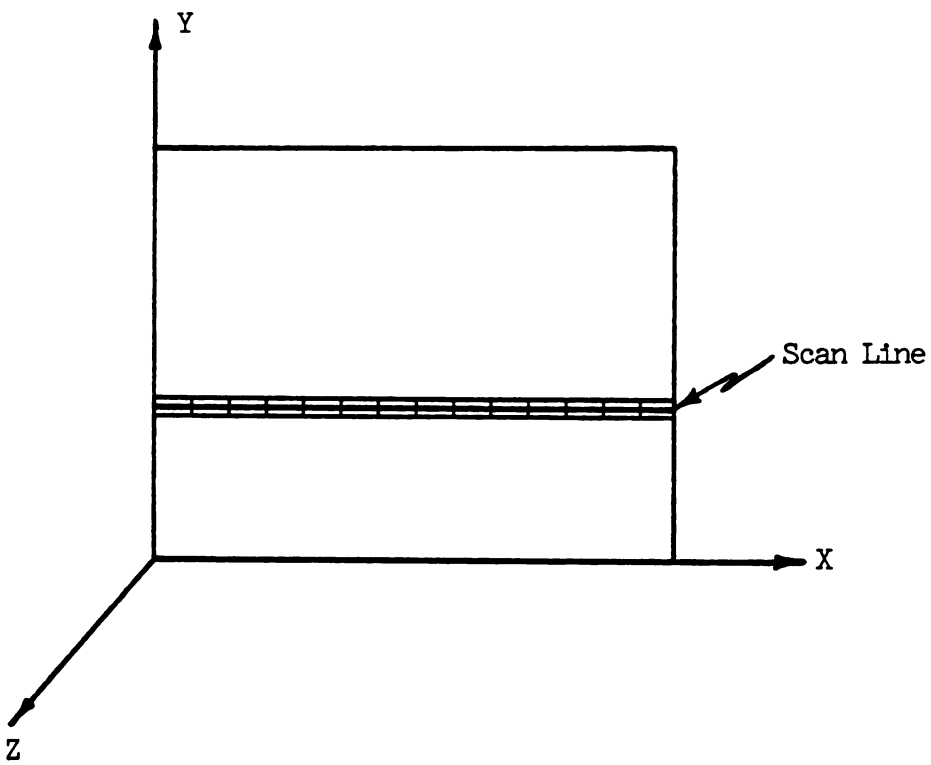


Figure 6. Viewing Screen and Screen Coordinates

Methods used to compute the sightline-patch intersection and normal vector at that intersection point depend on the type of surface patch. Existing algorithms for solving these problems will be discussed in the next section.

4.3 Previous Work in Surface Shading and Display

Literature in the area of computer-generated surface shading started appearing in the late 60's and early 70's. Since then the topic has been well represented in the literature. A good review of current literature in surface shading is presented in Reference 6.

Early shading techniques were based on surfaces consisting of flat patches (polygons).

4.3.1 Display Techniques for Polygon Surfaces

Curved surfaces may be approximated using many small polygons. For example, Figure 7 illustrates an approximation of a bi-cubic patch using polygons. Surface representations consisting of polygons are called polygon-type surfaces. The advantages and disadvantages of three well known polygon type surface display methods will be presented here.

The Warnock method (7) uses a depth priority algorithm to display surfaces constructed of polygons. This algorithm uses windows, or areas of the screen, which are successively subdivided into four smaller windows until the window is completely covered by a visible polygon. The window is then displayed. The subdivisions can continue until the windows are the size of one pixel. The major drawback of this method is that the pixels are not assigned in an orderly fashion, an inconvenient procedure for a raster device.

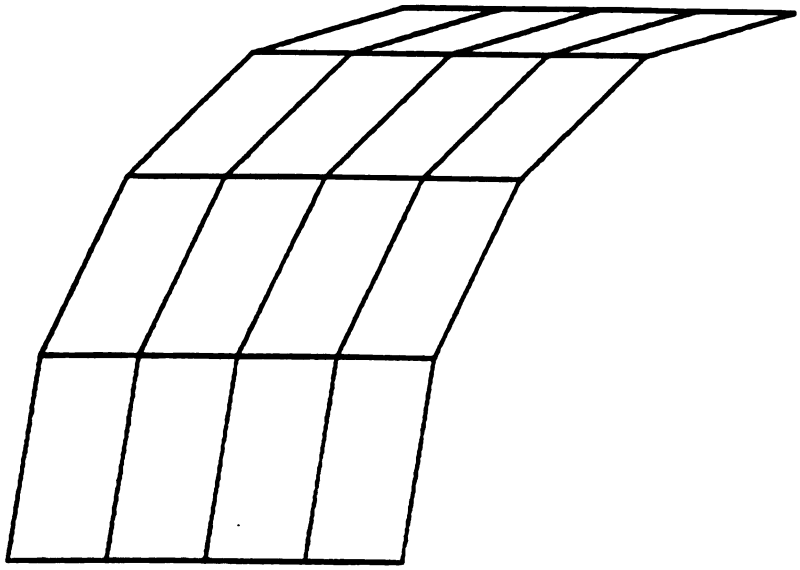


Figure 7. Polygon Approximation of Bicubic Patch

The method developed by Newell, Newell, and Sancha (8) determines z priority of all polygons in a scene by comparing the z coordinate at the centroid of each polygon. Of the polygons which are potentially visible at a given pixel, the polygon with the highest priority is assumed visible. This method works well for surfaces comprised of many small polygons. Problems arise when polygons penetrate each other, and when cyclic overlaps of polygons occur. Cyclic overlap involves polygons that are both in front and behind of another polygon. An extended algorithm was developed to handle these problems (8).

Watkins developed a scan line method for polygon display. This is the most widely used of the three methods and is available in commercial software (9,10). A general overview of the algorithm is presented below. Additional detail is presented in (11) and (12).

To aid in the discussion of scan line techniques, it is convenient to define several associated terms. A scan line is a constant y line representing a row of pixels (see Figure 6 and 8). The screen is comprised of many scanlines. The associated scan plane is defined by the viewpoint and the scan line. The scan plane cuts constant y sections through objects in the scene to determine what is potentially visible on the scan line, as shown in Figure 8. A sightline, passing through the viewpoint and the pixel, is defined for each pixel along the scan line. Intersections between the sightline and objects in the scene determine what is potentially visible at that pixel.

The scan line algorithm developed by Watkins (11) efficiently determines polygon sightline intersections by orderly processing of the sightlines. This makes possible the use of information about

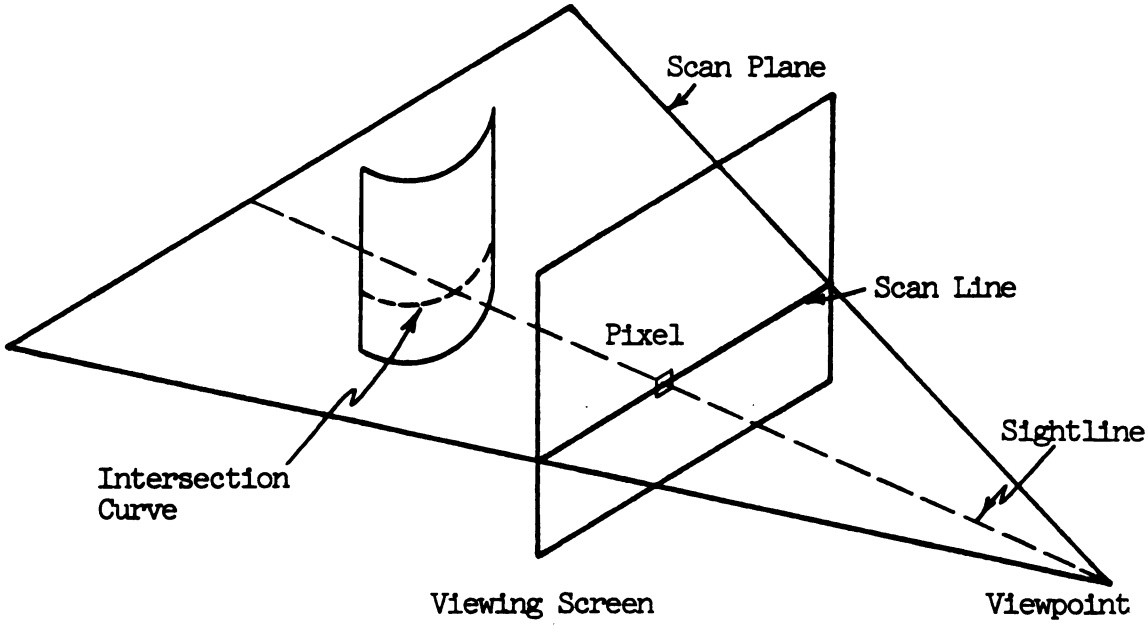


Figure 8. Scan Plane and Intersection Curve

intersections from the previous sightline when calculating intersections for the present sightline. This orderly processing, or scanning of sightlines, is the basis of scan line methods.

Scan line methods use two nested scans, the x scan and the y scan. The x scan, which is the inner loop, starts at the end of a scan line and steps along the scan line computing sightline polygon intersections for each sightline along the scan line. The y scan, starting at the top of the screen, steps through each scan line on the screen calling the x scan at each step.

When computing sightline polygon intersections, many calculations can be saved by knowing which polygons are intersected by the sightline. It is in the determination of the intersected polygons that the scan line methods derive their efficiency. This is done using a list of active polygons associated with each scan line, and a list of active line segments associated with each sightline.

The active polygon list contains all polygons intersected by a scan plane, therefore this list is updated for each scan line, or at each step of the y scan. The active polygon list is updated by using a list of maximum and minimum y values of polygons in the scene. The list of maxima are ordered such that they are decreasing in magnitude. A pointer is used to indicate the maximum or minimum of the next polygon to be encountered and added on the list, for a maximum, or dropped off the list for a minimum. Thus, as the y scan progresses, the y value of the scan line, Yscan, is compared to the next maximum or minimum. When Yscan is less than the current maximum or minimum, the associated polygon is added or deleted from the active polygon list, and the pointer is advanced to the next maximum or minimum. At this

point the active polygon list is updated for the scan line at Yscan, and the x scan can be initiated.

Once the active polygon list is established for a scan plane, the x scan for that scan line can begin. For each x scan, the intersection of the scan plane with the active polygons create straight line segments which are defined by their endpoints. An active line segment list contains the subset of these line segments which are intersected by a given sightline. The active line segment list is updated for each sightline using a list of line segment endpoints. The list is sorted starting with the endpoint with the smallest x coordinate. A pointer is used to indicate the next line segment endpoint to be encountered, or so-called current endpoint. When the x value of the sightline, Xscan, is incremented for the next sightline, it is compared to the current endpoint. If Xscan is greater than the current endpoint, the active line segment list is updated.

The z coordinates of the intersections between the sightline and the active line segments are compared to determine which polygon is visible. The normal vector associated with the visible polygon is then used to compute the intensity of the pixel defining the sightline.

Scan line methods applied to polygons have proven very time-effective in producing shaded images. But if these methods are to be applied to contoured surfaces, the surfaces must be approximated by polygons before shading begins. For applications where flat surface approximations are unacceptable, direct shading of contoured surfaces may be required. The next section will examine methods of directly shading parametric surfaces.

4.3.2 Display Methods for Parametric Surfaces

Parametric surfaces were first displayed using polygon techniques. A curved parametric surface can be approximated by many small polygons as shown in Figure 7. The polygons may then be displayed using techniques discussed in the previous section. This method is effective for visualizing gross surface characteristics, but detailed slope and curvature information is lost in the image. To maintain exact slope and curvature information at each pixel, a direct method of shading parametric surfaces is required. Three methods of direct parametric surface display will be discussed here.

In 1974, Catmull developed one of the first algorithms for the direct display of parametric curved surfaces (13). This algorithm subdivides each patch until each subdivision covers only one pixel. The large number of subdivisions required can be time consuming. Catmull developed an efficient method to subdivide one specific surface type, the parametric bi-cubic patch. The subdivisions are made along flow lines. To test if the subdivided patches are the size of one pixel, an approximating polynomial is constructed using the four vertices of the patch. This method effectively handles patches with little curvature, but problems may be encountered with highly curved patches or patches with poor orientation. These problems arise from the fact that the polygon constructed may not totally contain the patch subdivision.

Lane and Carpenter (12) modified this method into a scan line type method. As patches are subdivided, pieces which do not fall on the scan line are placed in an inactive patch list. Subdivision

continues until the patch is within a set tolerance of being a four-sided planar polygon. The active patches along the scan line may then be processed using the x scan from the polygon scanline method. Sweeping in this fashion is convenient for raster display.

In 1978, Blinn developed a scan line method (11) for displaying curved parametric surfaces. The scan line algorithm used is similar to the one developed by Watkins (11) discussed earlier in this chapter, but the complexity of the algorithm is increased when polygons are replaced by curved parametric patches. The algorithm contains the y scan and x scan typical of scan line methods, with the addition of a patch preprocessor.

The preprocessor finds and orders local y maxima/minima for each patch. Unlike polygons, for which maximum and minimum values occur at the corners, maxima and minima for contoured surfaces can occur on the boundary or on the interior of the patch. The local maxima and minima of contoured surfaces contain the corners and solutions to the following equations:

$$\frac{\partial y}{\partial s}(s,t) = 0$$

$$\frac{\partial y}{\partial t}(s,t) = 0 \qquad (4.3.1)$$

Obtaining the solutions of these nonlinear equations requires an iterative technique. Blinn uses Newton's method for equations of this type.

Curved parametric patches create additional complications during the y scan. Intersections of the scan plane, defined by the scan line

and viewpoint, with a parametric patch results in a curve rather than the straight line which must result from polygons. We will refer to these curves as intersection curves.

Another complication results from the fact that the scan plane may intersect the patch in more than one place, as shown in Figure 9. Therefore more than one intersection curve may come from a single patch.

The number of intersection curves on each patch can be determined as a function of local minima and maxima. When a maximum or minimum of a patch is encountered during the y scan, the number of intersection curves associated with the patch is increased or decreased. This method is illustrated in Figure 9. The ith scan line has zero intersection curves associated with the patch. Moving from the ith scan line to the ith+1 scan line a global maximum is encountered at the point A, therefore the number of intersection curves on the patch

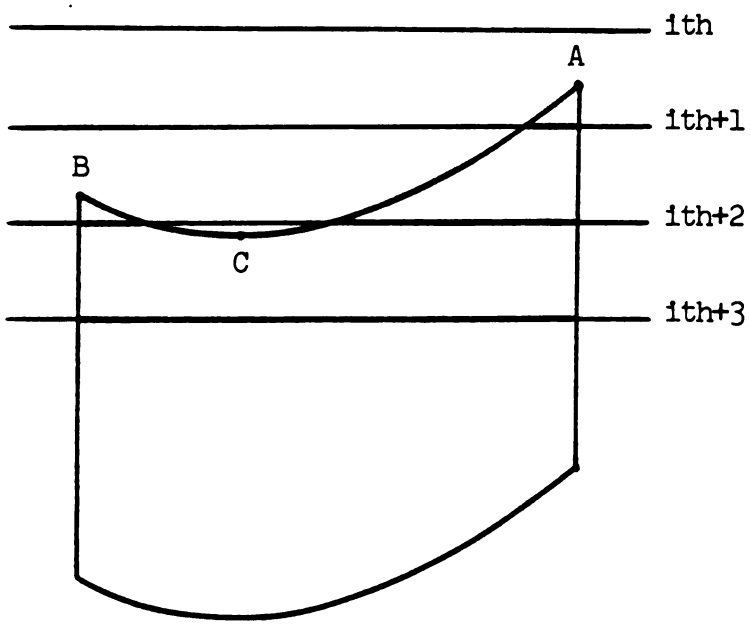


Figure 9. Number of Intersection Curves on a Patch

increases to one. Moving to the ith+2 scan line a second local maximum is encountered at point B increasing the number of intersection curves to two. The next scan line passes the local minimum at C, reducing the number of intersection curves to one.

Another problem associated with curved patches is the so-called silhouette edge. Silhouette edges are defined as curves on the surface where the z component of the normal vector is zero (see Figure 10). An intersection curve that intersects a silhouette edge is not single-valued in z. This problem may be handled by dividing the intersection curve at the silhouette edge into two intersection curves. Thus as the y scan progresses, the end of the silhouette edge implies a change in the number of intersection curves associated with the patch. This idea is illustrated in Figure 10. As the y scan moves from the ith scanline to the ith+1 scan line, the beginning end of a silhouette edge is encountered at A, therefore the number of intersection curves increase from one to two.

To facilitate the x scan, the endpoints of each intersection curve must also be computed. Endpoints can occur at the patch boundaries or at silhouette edges. If endpoints occur at patch boundaries, they may be found from one of these four equations.

$$
\begin{aligned}
&1)\ y(0,s) = Yscan \\
&2)\ y(1,s) = Yscan \\
&3)\ y(t,0) = Yscan \\
&4)\ y(t,1) = Yscan
\end{aligned}
\qquad (4.3.2)
$$

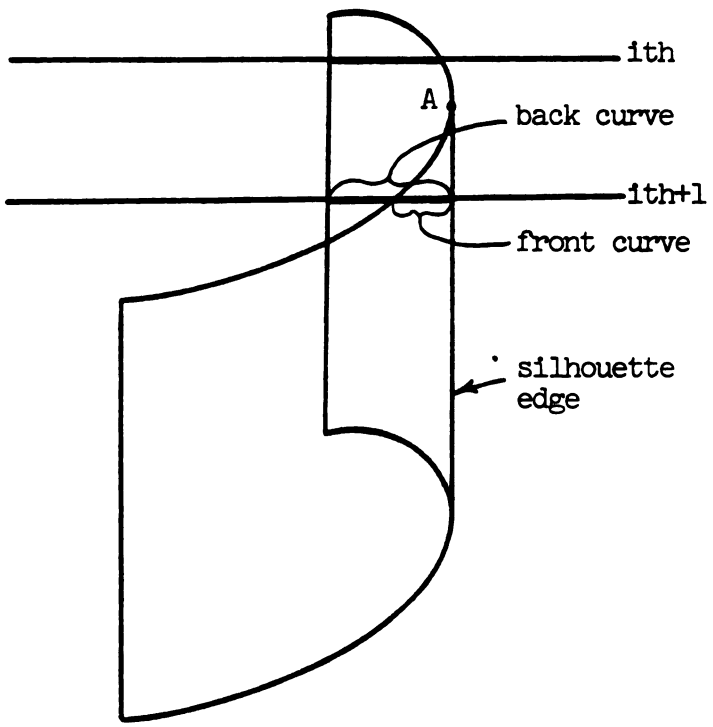


Figure 10. Effect of Silhouette Edge on the Number of Intersection Curves

Endpoints at silhouette edges are obtained by simultaneously solving:

$$y(s,t) = Yscan$$
$$N_z(s,t) = 0 \qquad (4.3.3)$$

where N_z is the z component of the normal vector. These endpoints are then used during the x scan to update the active curve list.

During the x scan, the visible intersection curve is determined by comparing the z coordinates of intersections between the sightline and the active intersection curves. Intersections of each sightline with the intersection curves are found by simultaneously solving:

$$x(s,t) = Xscan$$
$$y(s,t) = Yscan \qquad (4.3.4)$$

Typically, equation 4.3.4 is solved using Newton's method, but Blinn notes that singularities or cusps in the patch may occur, and that for these cases Newton's iteration is not appropriate.

Blinn also notes that, although the technique is accurate and avoids polygon approximations, the x scan portion of the algorithm is slow, and he developed an alternative algorithm. Along the visible intersection curve, Blinn computes sightline intersections only at so-called key visual points and uses linear interpolation to compute the normal vectors between them. This reduces the number of times equation 4.3.4 is solved, thus improving computation time. A

description of key visual points and how the algorithm works can be found in Reference 14.

Another method of directly displaying curved surfaces was developed in 1980 by Whitted (15). This algorithm, which was developed specifically for the bi-cubic surface, is a scan line algorithm similar to Blinn's, with modifications to handle silhouette edges. Silhouette edges, which are usually of a higher order than cubics, are approximated by one or more cubics. A patch containing a silhouette edge is then divided along the silhouette edge into two patches having boundaries which are cubics in one variable. The preprocessed patches, whose edges are cubics in one variable, are displayed using a scan line technique similar to Blinn's algorithm without having to compute silhouette edges at every scan line.

Each of the surface shading algorithms discussed has its advantages, either in accuracy or speed. But the demands of the designer required both speed and accuracy. The next chapter presents an algorithm developed to apply color graphics to surface checking and explains the steps taken to retain both speed and accurary.

CHAPTER V

A Surface Shading Algorithm for Design

5.0 Introduction

Since surface checking as defined here has the purpose of finding subtle flaws in the data base, it is clear that approximations to the surface must be scrutinized carefully. If the surface approximation washes out the sought-after flaws, all hope for successful surface checking is lost. On the other hand, exact methods lead to an increased computing burden, perhaps rendering the check process so time consuming as to rule out the kind of interactivity so desirable to the designer.

To attain the desired accuracy, the algorithm developed in this thesis builds on ideas developed by Blinn (14). This type of algorithm is referred to as an exact algorithm because the normal vectors are computed at each pixel. The algorithm is basically a scan line method with several modifications to help fulfill the dual needs of the design environment, namely, rapid calculation and accuracy. The three major components of this algorithm, the patch preprocessor, the y scan, and the x scan, are presented in the following sections.

5.1 The Patch Preprocessor

A major difference between the algorithm developed here and the algorithm developed by Blinn is that many of the calculations done by Blinn during the y scan are replaced by careful patch preprocessing.

The first calculation done by the patch preprocessor is finding global maxima and minima in x, y, and z for each patch. Conservative approximations are used, meaning that the patch is always contained by the approximate extrema. The algorithm developed to approximate these extrema is much faster than solving for the exact extrema, and errors are small for most patches. Appendix A presents a detailed description of this algorithm.

The x and y extrema are used to determine which patches are on the screen. These patches are then placed in an active patch list, alleviating any search involving patches not on the viewing screen. As an example, consider an instance when small areas of an object are magnified and displayed, leaving the majority of the patches outside of the viewing screen. The preprocessing saves work with patches that are not on the screen.

From this point on, calculations discussed are for active patches only. The z extrema for these patches are stored to be used later during the x scan.

The second calculation done during patch preprocessing is the calculation of all scan plane intersections with patch edges, including both patch boundaries and silhouette edges. These intersection points are the endpoints of intersection curves. The algorithm developed to compute these endpoints uses Newton's method to solve equations 4.3.2 and 4.3.3 presented by Blinn. But unlike Blinn's algorithm which computes all the endpoints on one scanline, this algorithm computes all of the intersection curve endpoints for each patch. Appendix B presents a detailed discussion of this algorithm. The endpoints computed for a typical patch are illustrated in Figure 11.

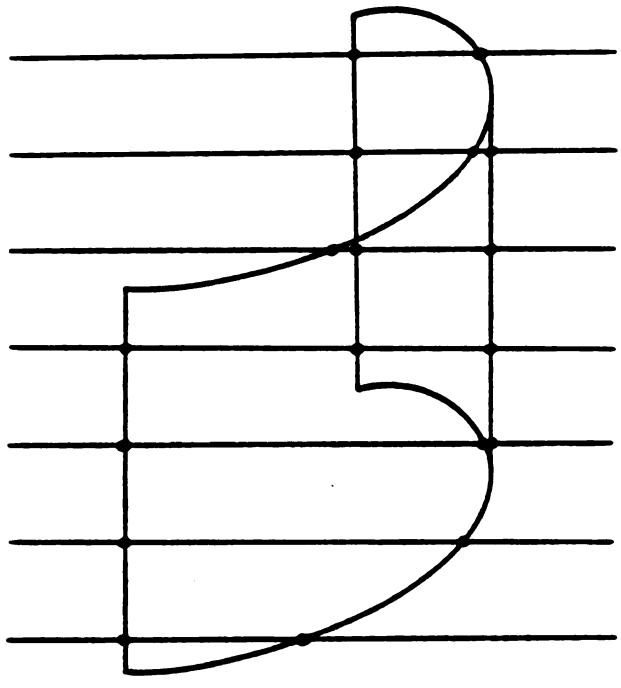


Figure 11. Typical Intersection Curve Endpoints

After all endpoints are computed for each patch, endpoints of the same intersection curve are paired. Pairing methods are also discussed in Appendix B. As the patches are processed, endpoint pairs are sorted by their y values and placed in vectors associated with each scan line. After all active patches are processed, the endpoints are then ordered in x in preparation for the x scan. At this point, any effects of the approximation used to calculate global patch maxima and minima have been corrected. No "search" beyond exact patch boundaries has been done.

A disadvantage of this method is the large amount of memory required to store the endpoints of the entire screen, as opposed to Blinn's method which requires memory for only one scan line at a time. But computing intersection curve endpoints a patch as a time eliminates the need for the exact local minima and maxima which are required by Blinn to compute intersection curve endpoints a scan line at a time, a calculation similar to cutting constant y sections for each y value in turn. Another advantage of storing all endpoints arises from the desire of the designer to "assemble" an object on the screen. Displaying objects a piece at a time may enable the designer to detect how well pieces are mating and possible interference. In such applications, patches already displayed need not be processed again by the preprocessor, and the scans need only cover areas where new patches have been added. Also, single patches may be corrected or altered and the entire scene can be displayed by only preprocessing the altered patch, deleting the original patch, and scanning in the neighborhood of the altered patch. In both of these instances there are large computational savings.

Since memory cost has been rapidly decreasing to current rather nominal levels, the time saving from this technique more than offsets the large memory requirements. In the case of a computer with virtual memory, there is, in fact, no trade-off at all since paging delays for memory access are very much smaller than the time saved by this preprocessing.

The next section will discuss the y scan. Subsequent sections give a detailed explanation of the x scan.

5.2 The Y Scan

Recall that the patch preprocessor has computed the endpoints of intersection curves for the entire scene. The endpoints are stored in vectors representing each scan line. The endpoints in each vector are stored in order of increasing x.

The y scan steps through all scan lines on the screen. At each scan line, the y scan calls the x scan and supplies the associated endpoints. The x scan computes the visible intersection curve and the point where the sightline intersects it for each pixel on the scan line. The y scan then moves on to the next scan line until all scan lines are displayed.

5.3 The X Scan

The x scan must determine, for each sightline along the scan line, which intersection curve is visible at the pixel. This problem is usually called the hidden line problem. After the visible curve has been identified, the intersection point between the sightline and the visible curve is computed. Because the intersection curves are not

linear, an iterative method such as Newton's method is required to solve for the intersection point. Solving nonlinear equations in the context of this task will be examined in a later section. The next section will discuss the hidden line problem.

5.4 The Hidden Line Problem

The hidden line problem refers to the identification of visible intersection curves or portions of intersection curves. This is done at each pixel as the x scan sweeps across a scan line.

The first step in determining which curve is visible is to find all intersection curves intersected by the sightline. (For the remainder of this chapter, the word curve will refer to an intersection curve.) This is accomplished using an active curve list which contains curves intersected by the sightline. The active curve list is updated for each pixel as the x scan moves across the scan line. The updating is done using a pointer and the ordered list of curve endpoints supplied for each scan line by the y scan and the patch preprocessor. The pointer is used to indicate the next endpoint to be encountered during the x scan. This endpoint will be called the current endpoint. As the x scan progresses, the x value of the pixel, Xscan, is compared with the x value of the current endpoint. If Xscan is greater than the x value of the endpoint, a curve is added or deleted from the active curve list, depending on whether the endpoint encountered is a leading or trailing endpoint of the curve. The pointer is then advanced to the next endpoint in the list, which is again compared to Xscan, and the process continues.

An example is presented in Figure 12. The example contains two patches A and B. Patch B contains one silhouette edge. The scan line has three associated intersection curves, $\bar{a}$, $\bar{b}$, and $\bar{c}$. Curve $\bar{a}$ lies in patch A, and curves $\bar{b}$ and $\bar{c}$ lie in patch B. The ordered list containing the x values of the curve endpoints is given beside the figure, and lines joining endpoints indicate a pair of endpoints on the same curve. The first endpoint of a pair appearing in the list is the leading endpoint of the curve and the second endpoint is the trailing endpoint.

The x scan starts at the left end of the scan line. At this point no patches are intersected by the sightline. Therefore there are no active curves when the x scan begins. The pointer starts at the top of the endpoint list, pointing at X1. As the x scan progresses and the value of Xscan is incremented, it is compared to the endpoint pointed at by the pointer, or the so-called current endpoint. Note that X1 is a leading endpoint, therefore when Xscan becomes greater than X1, curve $\bar{a}$ is intersected by the sightline, and therefore is placed in the active curve list. The pointer is then advanced to the next endpoint, X2, and Xscan is compared to X2. Because Xscan is less than X2, no additional changes are made to the active curve list. The x scan continues until Xscan becomes greater than X2, at which point curve $\bar{b}$ is added to the active curve list. The pointer is advanced to X3, but X3 is still greater than Xscan, so no additional changes are made to the active curve list, which now contains curves $\bar{a}$ and $\bar{b}$. Because X3 is a trailing endpoint, when Xscan becomes larger than X3, curve $\bar{a}$ is removed from the active curve list. The pointer is then advanced to X4, which is also less than Xscan, and therefore curve $\bar{c}$ is added to

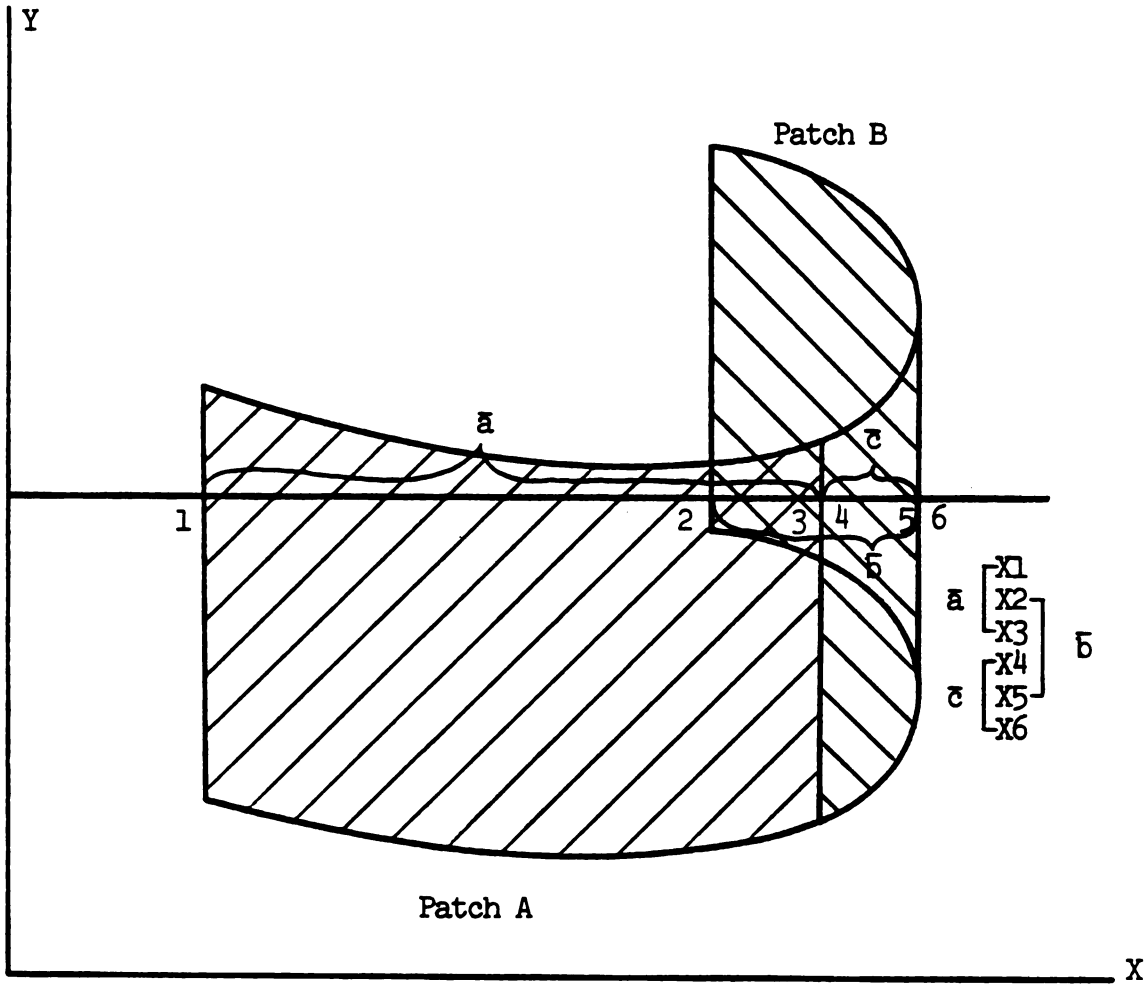


Figure 12. Illustration of Active Curve Updating

the active curve list. The pointer then advances to X5, which is greater than Xscan, and therefore, at this point the active curve list contains $\overline{b}$ and $\overline{c}$. As the x scan continues, it then passes endpoint values X5 and X6, both trailing, leaving the active curve list empty.

After determining all curves intersected by a given sightline, it must be determined which curve is first intersected by the sightline, or, in other words, which curve is visible at the sightline. The algorithm which determines the visible curve uses two steps. The first is a global depth comparison of patches from which the active curves originate. This step may determine if some curves cannot be visible. The second step determines which of the remaining curves is visible. The global depth comparion of patches will first be discussed.

Recall that global z extrema were computed for each patch during patch preprocessing. The z limits of patches containing active curves are compared to determine if any are globally behind others. This is typical for most objects being modelled, for example, solid objects generally have two disjoint sides. Active curves lying on patches globally behind others must be behind curves lying on the other patches. Such curves are immediately eliminated when determining the visible curve. Note that the global patch check is required only when there is a change in the active curve list.

From this point, determination of which active curve is visible is broken into two classifications, depending on whether or not patches intersect each other on their interiors. For example, Figure 13 illustrates two patches which intersect on their interiors.

The more general case, where intersecting patches may be present, requires a depth check at each pixel. The z coordinate of

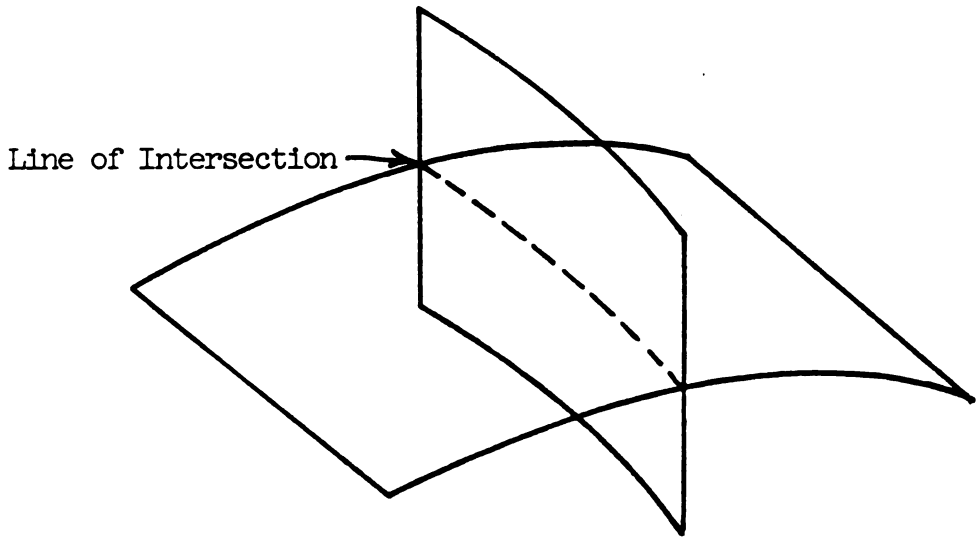


Figure 13. Interior Intersection of Two Patches

intersections between the sightline and the active curves are computed to determine which curve is visible at the pixel. Computing the z value of the intersection point requires solving the nonlinear equations 4.3.4 for parametric values s and t, given x and y of the pixel. (This solution is discussed in detail in the next section). The values of s and t are then used to compute z. Doing this calculation for all active curves is compute intensive, thus savings from the global z check, which reduces the number of active curves, is evident.

The more restrictive case, where patches are assumed not to intersect each other, requires a depth check only when the active curve list is changed. This is due to the fact that once a curve is determined to be in front of another, it remains in front. In fact, when a new curve is added to the active curve list it need only be compared to the

current visible curve to determine which curve is visible, and only when the visible curve is deleted from the active curve list is the depth comparison required.

It is also possible to convert cases with intersecting patches into those without. This is done by dividing the intersected patch into two patches by computing the intersection of the two patches, and dividing the associated intersection curves into two curves. This idea is illustrated in Figure 14. In this case, two intersection curves are divided into four curves which only intersect at their ends, thus the faster algorithm can be used to display these patches.

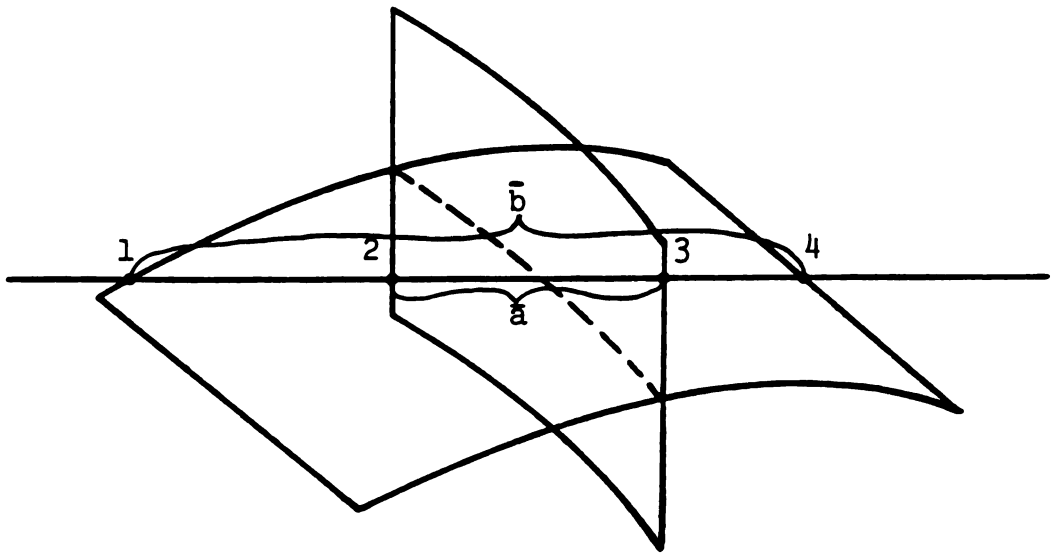
Clearly, if it can be assumed that patches do not have interior intersections, calculations are significantly reduced. Of course, such assumption precludes the detection of unwanted interference.

The next section will discuss solution methods for finding intersections of a sightline and an intersection curve.

5.5 Solutions of Nonlinear Equations

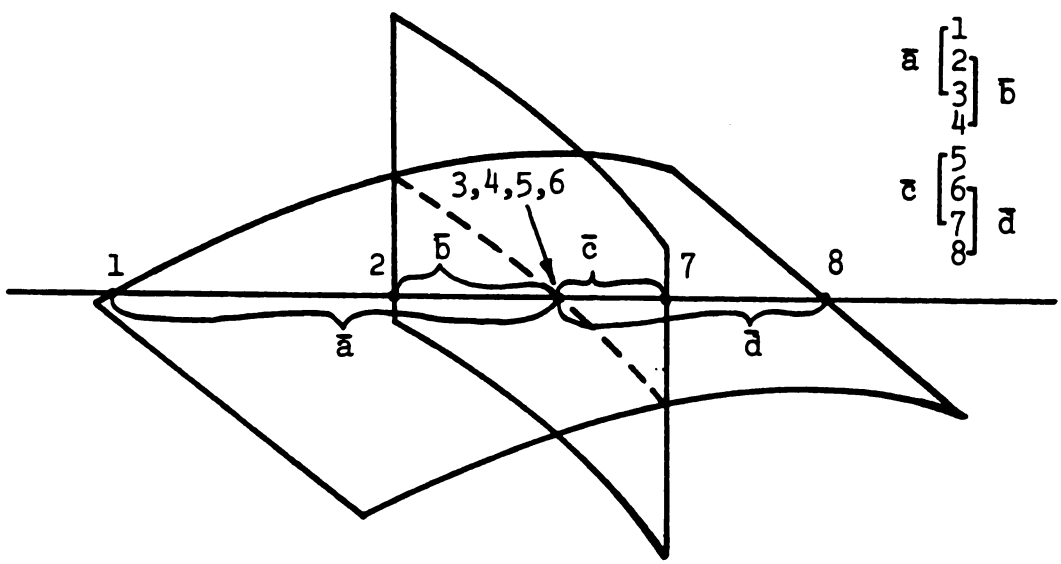
Up to this point, the algorithm has determined which curve is visible for a given sightline. The curve is identified by which patch it is in and its endpoints. The next step is to compute normal vectors for the associated pixels along the visible curve. The s and t values of each sightline intersection point are needed to compute the normal vectors. Therefore, an iterative technique is required to solve equation 4.3.4 for s and t.

Blinn suggested bivariate Newton's method (16,17) for solving the equations. Newton's method requires the s and t values of a starting point for the iteration. Faster convergence is achieved when



Original Intersection Curves

$$\bar{a}\begin{bmatrix}1\\2\\3\\4\end{bmatrix}\bar{b}$$



$$\bar{a}\begin{bmatrix}1\\2\\3\\4\end{bmatrix}\bar{b} \qquad \bar{c}\begin{bmatrix}5\\6\\7\\8\end{bmatrix}\bar{d}$$

Newly Formed Intersection Curves

Figure 14. Division of Intersection Curves for Interior Intersections

the starting point is near the solution. Good initial starting points are available from the x scan. Interior to a visible curve, the s and t values from the previous pixel are used as a starting point for the present pixel. When the visible curve changes, a starting point is required on the new curve. In this case, the s and t values of the endpoint of the new visible curve closest to the x value of the sight-line is used as the starting point.

It is also important to note a singularity that may prevent convergence of Newton's method. Consider a situation wherein s and t for a pixel on a visible curve are known, and the ds and dt to arrive at the next pixel are sought. The following equation is solved for ds and dt

$$\begin{bmatrix} ds \\ dt \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial s} & \frac{\partial y}{\partial t} \end{bmatrix}^{-1} \begin{bmatrix} dx \\ dy \end{bmatrix} \tag{5.5.1}$$

where dx is the known x distance to the next pixel and dy is the known y distance to the next pixel. Solutions exist only when the matrix of partial derivatives is nonsingular, i.e., when the determinant is non-zero. The determinant of this particular matrix is proportional to the z component of the normal vector at the point. Therefore, Newton's iteration will fail on areas of the patch where the z component of the normal vector is near zero. This corresponds to normal vectors lying in the plane of the screen, i.e., to silhouette edges. Therefore, Newton's method must be modified in these areas.

In the neighborhood of a silhouette edge, an absolute maximum on Δs and Δt is imposed so that the iteration stays in the region of the patch where it started. Convergence can be obtained for most pixels near silhouette edges by heavily damping Newton's method, but still retaining the $\Delta s/\Delta t$ ratio. As the number of iterations increases, the step size is increasingly reduced. In the event that the iteration does not converge after a limited number of iterations, the pixel is assigned the same normal vector as the neighboring pixel.

Another class of methods, the generalized secant methods, can be used to solve this type of nonlinear equations. The advantage of these methods is that the partial derivatives used in equation 5.5.1 of Newton's method are not used after the first step of the iteration. Obtaining these derivatives requires numerous calculations. Although the convergence rate for secant methods is slower than Newton's method, extra iterations using secant methods are offset by the savings incurred from not having to compute partial derivatives at every iteration step. One particular generalized secant method, Broyden's method (16), which is used in this thesis, is discussed in detail in Appendix C.

5.6 Approximate Methods

A method was developed for approximating the normal vectors along a visible curve between its endpoints similar to the method of key visual points developed by Blinn. At key visual points, Blinn computes the exact normal vector. For points on the visible curve between key visual points, he linearly interpolates between the normals at the key visual points. One method used by Blinn to determine the location of key visual points is to compute a key visual point every time the normal

vector rotates a given number of degrees in the scanplane. This procedure was apparently derived to avoid the burden of calculating the s and t values to go with each pixel.

The approximating algorithm developed in this thesis also computes exact normals only in a few prescribed planes along the visible curve. But instead of linearly interpolating, a third order interpolation, or so-called blend, is used between the points. (This method preserves the slope at the endpoints of the curve, which is necessary if slope discontinuities between patches are to be visible.) The following equation is used to blend the normal vector:

$$N = N_0 * F_1(x) + N_1 * F_2(x) + N_{x0} * F_3(x) + N_{x1} * F_4(x) \qquad (5.6.1)$$

where N_0 is the normal vector at the leading endpoint of the curve and N_1 is the normal at the trailing endpoint. Also N_{x0} and N_{x1} are the partial derivatives of the normal vector with respect to x at the leading and trailing endpoints, and $F_1(x)$ through $F_4(x)$ are third order blending functions (5). These blending functions guarantee that the normal and the derivative of the normal with respect to x is maintained at the ends.

Since the normal vector is a complicated function of s and t, computing the derivative with respect to x is not straightforward. Appendix D presents the derivation for N_x.

It is usually sufficient to blend between the endpoints of the visible curve, thus no interior points need be computed. But for sharply curved patches it may be necessary to break the curve into shorter segments. It is also important to note that N_x is undefined

along silhouette edges. For this case, the blend is started a slight distance in from the silhouette edge, and points near the silhouette edge are computed exactly.

This approximating technique has been shown to be much more accurate than linear interpolation, with only a slight increase in computation time. When compared to exact techniques, a large time savings is realized.

5.7 Summary

A tradeoff of time versus accuracy is evident for all of the methods discussed. The most accurate technique, the exact method, requires calculation times which may be prohibitive for extensive use in an interactive environment. However, it is possible, through a combination of exact and approximate methods developed in this thesis, to create a practical design tool.

The next section will present results. Accuracy and calculation time will be compared for several algorithms. The effectiveness of each technique for detecting surface flaws will also be discussed. Finally, a possible surface checking scheme will be outlined.

CHAPTER VI

Algorithm Evaluation

6.0 Introduction

In the following section, comparisons are made between the exact and the approximate methods presented in this thesis. First the ability of the algorithm to detect surface flaws is examined. The calculation time is then compared for each algorithm. Finally, a large data base with many patches is used to illustrate the overall effectiveness of the modified y scan developed in this thesis.

6.1 Error Detection

An evaluation of the algorithm's ability to detect surface flaws makes use of the bump presented in Chapter 2. The bump with no slope discontinuities is shown in Figure 1. Figures 4 and 5 show the bump after a slope discontinuity was introduced.

The following examples compute the intensity for each pixel based on the absolute value of the dot product of the normal vector and the light vector. Taking the absolute value implies the light vector is in both directions, therefore all surfaces are illuminated. Although this is not realistic, it is effective when looking for surface irregularities.

The exact algorithm is used to shade the bump with and without the slope discontinuity. Figure 15 presents the bump without the slope

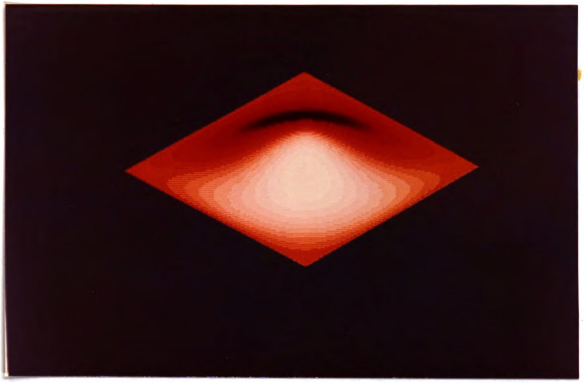


Figure 15. Shaded Bump

discontinuity. The color discontinuity in Figure 16 clearly indicates the slope discontinuity along the patch boundary. This could easily be detected by a designer who could then correct the error.

The approximate method can also be used to identify the slope discontinuity. The image of the bump computed by the approximate method is shown in Figure 17. Again, a color discontinuity is apparent along the patch boundary containing the slope discontinuity. It should be noted that since the interpolation scheme is accurate at patch boundaries, flaws on the interior of the patch may not be as accurately represented by the approximate method as those near the patch boundary.

It is of interest to note that not all views of the bump, or all light sources will highlight the slope discontinuity. For example, Figure 18 presents the same view of the bump as displayed in Figure 16,

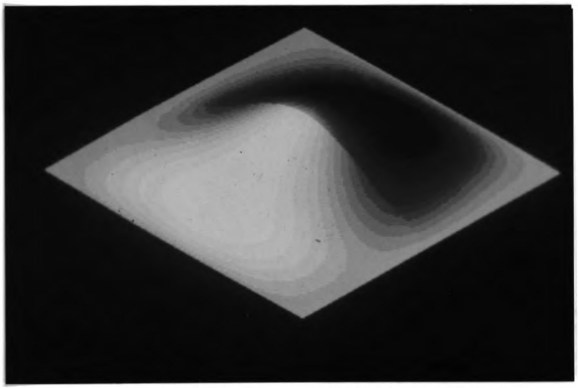


Figure 16. Shaded Bump with Slope Discontinuity

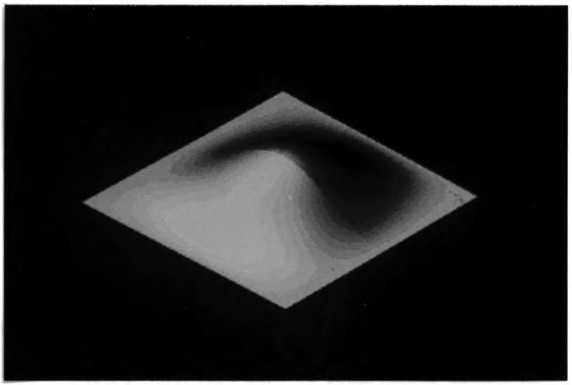


Figure 17. Shaded Bump with Slope Discontinuity, Approximate Method

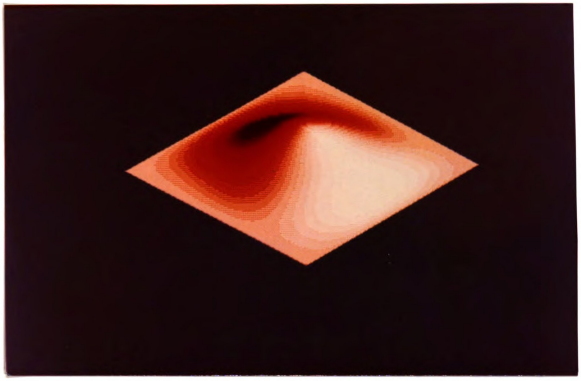


Figure 18. Shaded Bump with Slope Discontinuity, Different Light Source

but the light source has been moved. The slope discontinuity is not apparent in Figure 18. This is due to the fact that the change in the normal vector across the discontinuity is in the plane normal to the light vector. Thus the angle between the light vector and the normal does not change, and no color discontinuity is generated. Therefore, when checking a given view of a surface, it is necessary to observe images produced from several different light sources. The next section will discuss an efficient method of recomputing the image for each light source.

6.2 Image Generation for a New Light Source

Computing an image for a new light source without changing the viewpoint is a straightforward calculation. Surface normal vectors

computed for a given view remain the same for any light source. Therefore, moving the light source requires only the computing of the pixel intensity using the known normal vector and new light vector. This is a simple calculation, orders of magnitude less burdensome than computing the normal vectors themselves.

In this thesis, pixel intensity is computed by taking the dot product of the normal vector and the light vector. Therefore, moving the light source requires calculation of one dot product for each pixel. Such a calculation is potentially simple enough to be interactive. It is concievable that with the help of an array processor, the light source could be hard wired to a joy stick, and new light source locations could be viewed almost continuously. In contrast, changing the viewpoint necessitates recalculation of the normal vectors, with attendant silhouette and hidden surface problems.

The next section will benchmark several of the images presented above.

6.3 Compute Times

Calculation times for the bump are indicated in Table 1. Note that two different numerical techniques are listed for the exact method. These indicate that Broyden's method results in a significant time savings compared to Newton's method. As expected, the table also indicates that the approximate method requires less time than either of the exact methods. Finally, the time required to compute Figure 18 using normal vectors computed for Figure 16, which amounts to moving the light source, is listed to illustrate the simplicity of this procedure.

TABLE 1

Compute Times

Exact method (Newton's)	42 sec
Exact method (Broyden's)	37 sec
Approximate method	19 sec
Light source change	5 sec

All routines used in these comparisons are in Fortran and are being run on a Prime 750.

6.4 Object Assembly

The following example is used to demonstrate how objects can be assembled on the screen without reshading the entire scene. Data representing an aircraft wing are used for the example. A flow line representation of the wing is shown in Figure 19.

The following is a possible scenario for a designer checking the wing. Assume portions of the wing were designed by two different designers. Thus, the designer may first choose to inspect the lower half of the wing. The resulting image is shown in Figure 20. At this point many different light sources would be used to inspect the

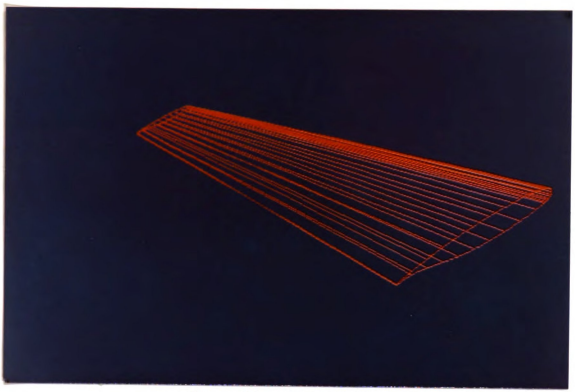


Figure 19. Aircraft Wing

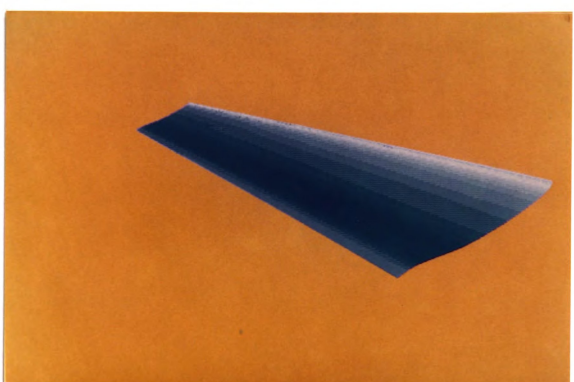


Figure 20. Shaded Aircraft Wing, Lower

surface. Upon completion of this check, the designer may want to add a portion of the wing as shown in Figure 21. Again, many different light sources are used to inspect the newly added portion and to check how it fits with the lower half. At this point he may choose to complete the assembly as shown in Figure 22 and continue the checking. Later, while checking the fuselage, the wing could be added to insure a proper fit. The total aircraft is shown in Figure 23.

6.5 Surface Checking Scheme

A possible surface checking scheme could entail the following steps. A designer may start the surface check by displaying the surface using flow lines. This would be used to check for missing data or gross errors in the database. During this procedure, the

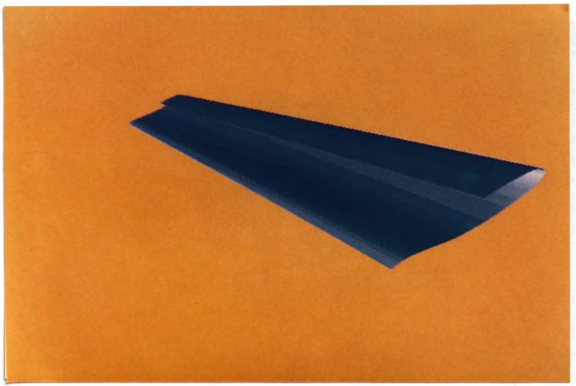


Figure 21. Shaded Aircraft Wing, Lower plus Half of Upper

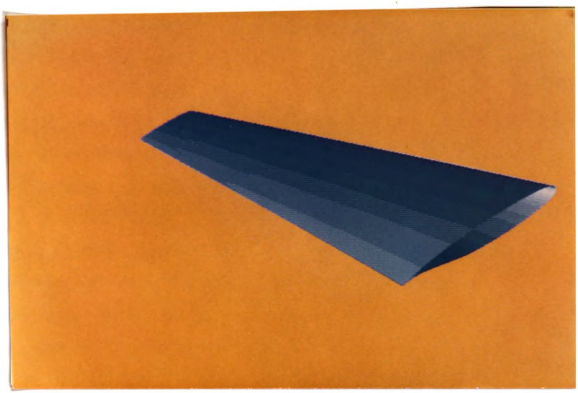


Figure 22. Shaded Aircraft Wing, Entire Surface

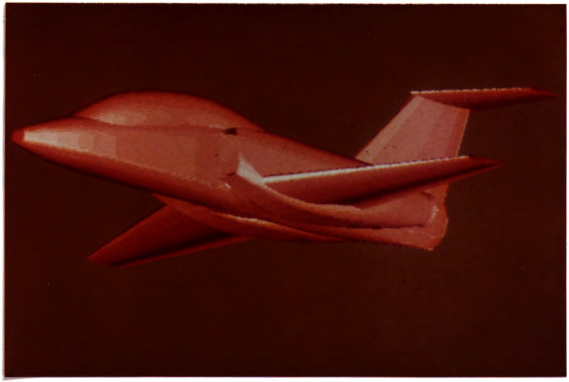


Figure 23. Shaded Aircraft

designer would note which views clearly display areas of interest. At this point he may choose to examine a view more carefully. This view would then be shaded using either the approximate method or the exact method. From a time standpoint, it is advantageous to use the approximate method whenever possible, but which algorithm should be used would depend on several other considerations.

For example, the type of surface flaws which are to be identified would affect the choice of the algorithm. If the slopes along patch boundaries are being checked, the approximate method would be very effective. On the other hand, if flaws involving curvature inflections on patch interiors are suspected, the exact method would better identify these errors.

The designer may also have prior knowledge of possible problem areas on the surface, or may have located possible problem areas using the approximate method. In this case, these problem areas could be magnified and viewed using the exact method.

Regardless of which algorithm is used, each view is then inspected using many different light sources. A combination of these procedures and the assembly procedure would be used repeatedly until the designer is satisfied that the surface is correct.

CHAPTER VII

Conclusions

Surface display methods developed prior to this work were primarily aimed at producing aesthetically pleasing images. The goals of this thesis, to develop a display algorithm to be used for surface checking, required a different approach to the problem.

The algorithm developed in this thesis has proven to be effective at locating surface flaws, and has been shown to be time effective from a designer's standpoint, facilitating a much more efficient check procedure than using various views from line drawings.

Future work should include speedup of the exact method through the use of better hardware. For example, many of the shading calculations can be done in parallel, lending themselves to array processing. Also, improvements in the accuracy of the approximate method may be possible through the use of higher order interpolation schemes. Other shading techniques could also be investigated, such as shading by curvature, which could highlight different types of surface flaws.

APPENDICES

APPENDIX A

Calculation of Approximate Patch Extrema

Patch extrema are approximated by expanding the matrix expressions for the spatial coordinates, and summing maxima of terms. The expressions for the z extrema will be derived in the following discussion. Identical expressions exist for x and y.

Figure A1 illustrates a typical patch. Parametric variables s and t run from zero to one. The patch boundaries, which are curves of one parametric variable, are numbered from one to four as shown in the figure. For example, $L_1(s)$ is the boundary curve at $t = 0$. The following discussion specifically deals with Coons surface definitions, but similar techniques can be applied to other parametric surface definitions.

The matrix equation for z as a function of s and t on the surface is:

$$z = (F_1(s)F_2(s)F_3(s)F_4(s)) \begin{bmatrix} Z_{00} & Z_{01} & \frac{\partial z}{\partial t}00 & \frac{\partial z}{\partial t}01 \\ Z_{10} & Z_{11} & \frac{\partial z}{\partial t}10 & \frac{\partial z}{\partial t}11 \\ \frac{\partial z}{\partial s}00 & \frac{\partial z}{\partial s}01 & \frac{\partial^2 z}{\partial s \partial t}00 & \frac{\partial^2 z}{\partial s \partial t}01 \\ \frac{\partial z}{\partial s}10 & \frac{\partial z}{\partial s}11 & \frac{\partial^2 z}{\partial s \partial t}10 & \frac{\partial^2 z}{\partial s \partial t}11 \end{bmatrix} \begin{bmatrix} F_1(t) \\ F_2(t) \\ F_3(t) \\ F_4(t) \end{bmatrix} \quad (A.1)$$

where F_1 through F_4 are cubic blending functions, and data subscript appearing in the matrix related to the patch corners, i.e., Z_{01} is the

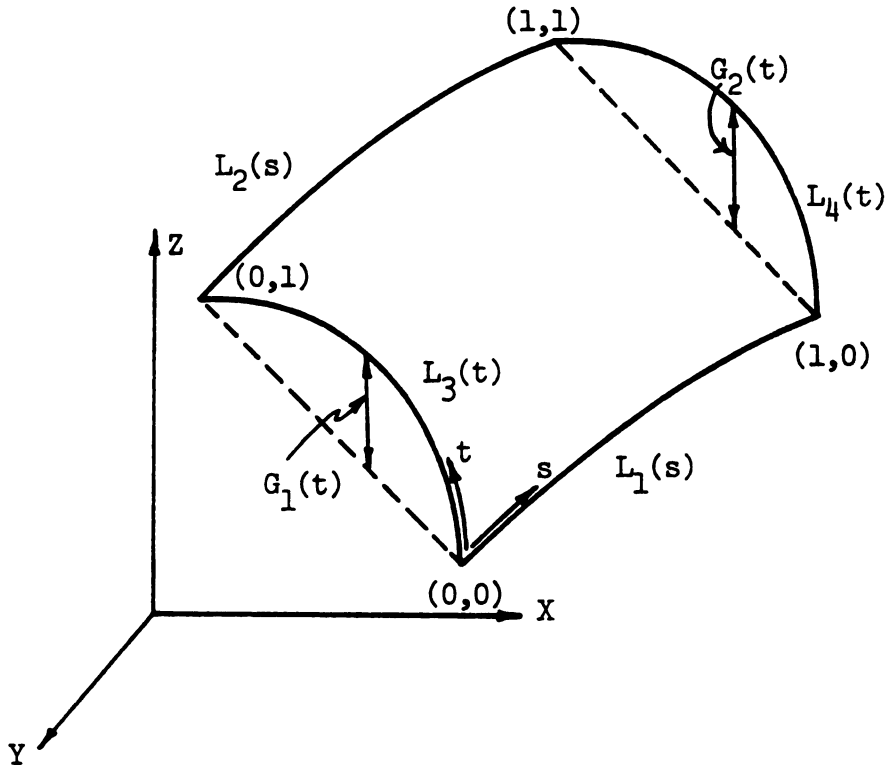


Figure A1. Definition of Patch Boundaries

z value at the corner s=0, t=1. Partially expanding equation A.1 yields:

$$z = L_1(s)F_1(t) + L_2(s)F_2(t)$$

$$+ \frac{\partial z}{\partial t}00 * F_3(t)F_1(s) + \frac{\partial z}{\partial t}01 * F_4(t)F_1(s) \qquad (A.2)$$

$$+ \frac{\partial z}{\partial t}10 * F_3(t)F_2(s) + \frac{\partial z}{\partial t}11 * F_4(t)F_2(s)$$

where $L_1(s)$ and $L_2(s)$ give the z coordinates on the boundaries 1 and 2 respectively. If $G_1(t)$ and $G_2(t)$ are defined as the following:

$$G_1(t) = \frac{\partial z}{\partial t}00 * F_3(t) + \frac{\partial z}{\partial t}01 * F_4(t) \qquad (A.3)$$

$$G_2(t) = \frac{\partial z}{\partial t}10 * F_3(t) + \frac{\partial z}{\partial t}11 * F_4(t)$$

Then equation A.2 becomes:

$$z = L_1(s)F_1(t) + L_2(s)F_2(t) \qquad (A.4)$$

$$+ G_1(t)F_1(s) + G_2(t)F_2(s)$$

The first two terms are associated with t blending of the s varying boundaries, while the last two terms represent s blending of additional surface contours due to partial derivatives with respect to t (see Figure A1). Using the fact that $\max(a+b) \leq \max(a) + \max(b)$, equation A.5 becomes:

$$\max(z) \leq \max(L_1(s)F_1(t) + L_2(s)F_2(t)) \qquad (A.5)$$

$$+ \max(G_1(t)F_1(s) + G_2(t)F_2(s))$$

Also note that since $F_1 + F_2 = 1$ the $\max(aF_1 + bF_2) = \max(a, b)$, i.e., the larger of the two values a and b. Therefore the approximate maximum of z can be written as:

$$\max(z) \leq \max(L_1(s), L_2(s)) + \max(G_1(t), G_2(t)) \qquad (A.6)$$

Equation A.6 is easily solved. The expressions L_1, L_2, G_1, and G_2 are cubics of one variable, therefore locations of local extrema can easily be found by differentiation and solving of the resultant quadratics. Local extrema located between 0 and 1 are then compared to end values to obtain global extrema over the range of 0 to 1. A similar derivation can be done to determine a lower bound for the minimum z.

APPENDIX B

Calculation and Pairing of Intersection Curve Endpoints

B.0 Calculation of Intersection Curve Endpoints

All intersection curve endpoints of a patch are computed at one time. Endpoints can occur on patch boundaries or on silhouette edges. Calculation of endpoints on patch boundaries will be discussed first.

B.0.1 Patch Boundaries

Patch boundaries are functions of one parametric variable. Endpoints are computed using Newton's method to solve the following equation:

$$Y(s) = Yscan_i \tag{B.1}$$

where Y(s) is the expression for the y value of the patch boundary, and $Yscan_i$ is the y value of the ith scan line. Any number of scan lines may cross a patch boundary, therefore this calculation must be performed for all intersections. There may also be more than one intersection of a patch boundary with a given scan line. Both of these difficulties are handled using the following algorithm.

The algorithm starts at the end of the patch boundary representing the lower limit of the parametric value. For this discussion the

parametric value will be s and therefore the search starts at s=0. Both y and dy/ds are computed at the starting point. The value of Yscan is then incremented in the direction of dy/ds, starting next to the y value of the starting point, until the upper limit of the parametric value is encountered. Each solution of equation B.1 uses the solution of the previous Yscan as a first guess for Newton's method. If anytime during the Newton iteration the sign of dy/ds changes, the iteration is stopped and Yscan is incremented in the new direction, and Newton's iteration is continued.

These ideas can be illustrated in the example presented in Figure B1. Due to the positive derivative at s = 0, the first Yscan value greater than Y_0, $Yscan_1$, is used in equation B.1 to solve for endpoint 1. The derivative at point 1 is also positive, therefore Yscan is incremented to $Yscan_{i+1}$ to compute endpoint 2. Yscan is again incremented to $Yscan_{i+2}$, but the first step of Newton iteration moves s past s_{max} and a derivative sign change is detected. Therefore Yscan is decremented to $Yscan_{i+1}$, and Newton's method is resumed, using as a starting point the last s value computed during the aborted iteration. Because s was already incremented past s_{max}, Newton's iteration converges to endpoint 3. This process is continued until s = 1 is encountered. This operation is performed for all four sides of the patch.

B.0.2 Silhouette Edges

The algorithm used to compute intersection curve endpoints along a silhouette edge is based upon two assumptions: 1) all silhouette edges come in contact with a patch boundary in at least one point, and

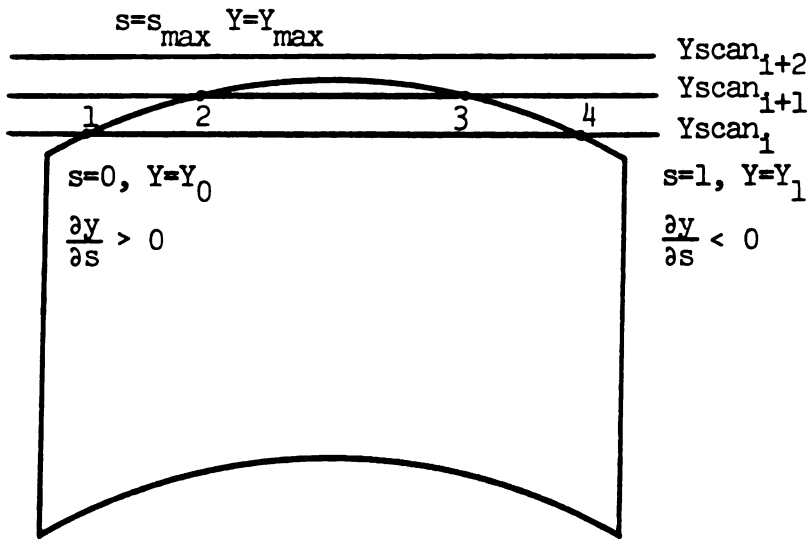


Figure B1. Calculation of Boundary Endpoints

2) each silhouette edge has no more than one sign change in dy/dx, or so-called inflection point, along the silhouette edge. Figure B2 illustrates both of these conditions.

The algorithm first locates intersections between silhouette edges and patch boundaries. A silhouette edge is defined as the locus of points on a patch whose z component of the normal vector is zero. Therefore points on the boundaries whose z component of the normal vector is zero can be thought of as silhouette edge starting points. The z component of the normal vector on a patch boundary can be expressen in terms of one parametric value, therefore zeros can be found using one-dimensional Newton's method. These points are stored for the next step of the algorithm.

Using a silhouette edge starting point, the algorithm increments Yscan and computes the intersection points. Using partial derivatives at the starting point, the direction to increment Yscan is determined. For example, on a constant t edge, t=0, if dy/dt is positive, Yscan would be incremented in a positive direction. The initial Yscan is determined by the y value of the starting point and the increment direction. Intersection points are found using bi-variate Newton's method to solve the following simultaneous equations:

$$N_z(s,t) = 0$$

$$Y(s,t) = Yscan_i \qquad \text{(B.2)}$$

where N_z is the z component of the normal vector. Good initial values for Newton's iteration are supplied by the solution at the previous Yscan.

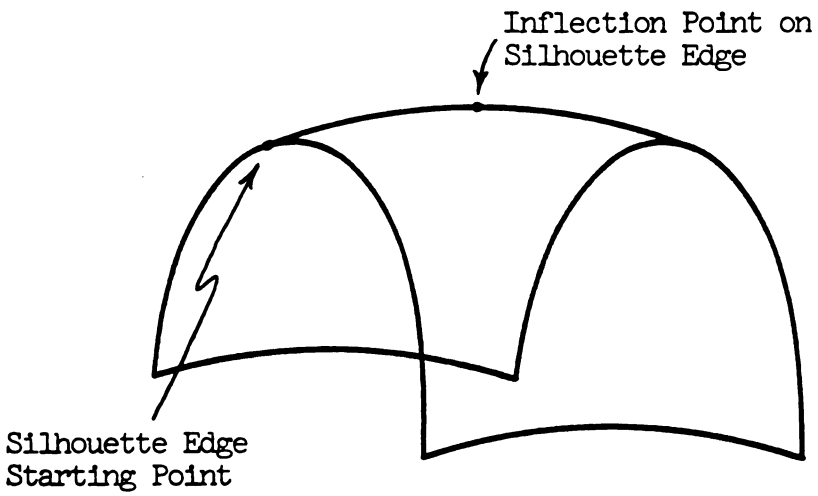


Figure B2. Critical Points on a Silhouette Edge

Yscan is incremented until one of two termination criteria is met: 1) Newton's iteration does not converge. This condition occurs when a silhouette edge has ended, or an inflection point has been passed. Endpoints that exist past the inflection point are computed when the opposite silhouette edge starting point is used. This idea is illustrated in Figure B3. Starting at point A, endpoints 1 and 2 are computed and termination occurs when Newton's iteration does not converge for $Yscan_{i+2}$. Similiarly, endpoints 3 and 4 are computed when starting from point B. 2) A limit on s or t is exceeded, meaning the silhouette edge reaches a patch boundary. When this criterion is met, the starting point associated with the termination point is removed from the starting point list to avoid duplicate calculations.

B.1 Pairing of Intersection Curve Endpoints

After all intersection curve endpoints are computed for a patch, endpoints of the same intersection curve are paired. Endpoints are sorted by scan line and pairing is done for each scan line. The logic used to pair the endpoints depends on the number and type of endpoints for the given scan line, i.e., whether the scan line originates from a patch boundary, or from a silhouette edge. The majority of pairing results from several simple cases discussed below.

Two common cases, two or four boundary endpoints, are illustrated in Figure B4. In these cases the pairing is straightforward. Another common case, two boundary endpoints and one silhouette endpoint, is presented in Figure B5. Note that the silhouette endpoint actually plays the role of two endpoints. Pairing is again straightforward.

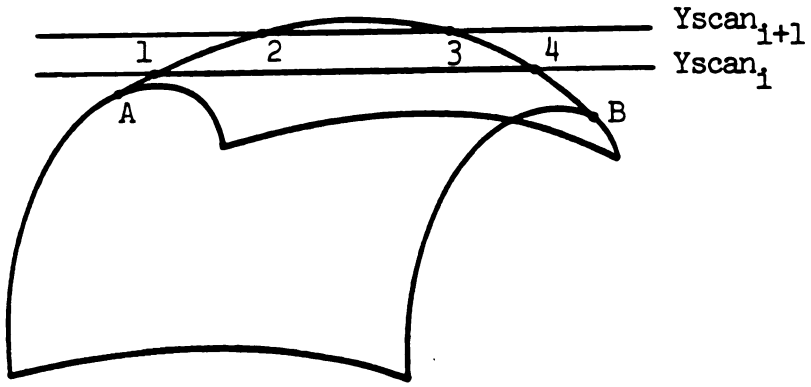


Figure B3. Method of Calculating Silhouette Endpoints

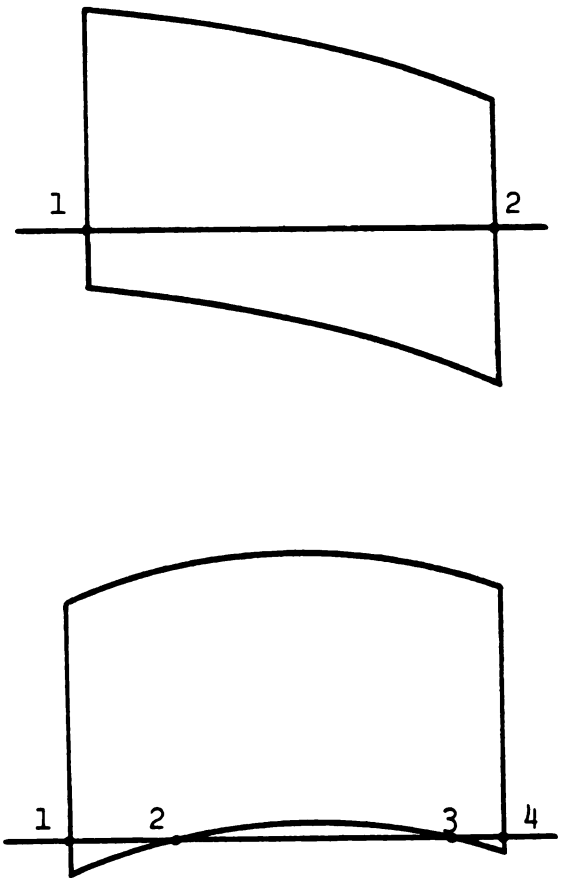


Figure B4. Simple Pairing Cases

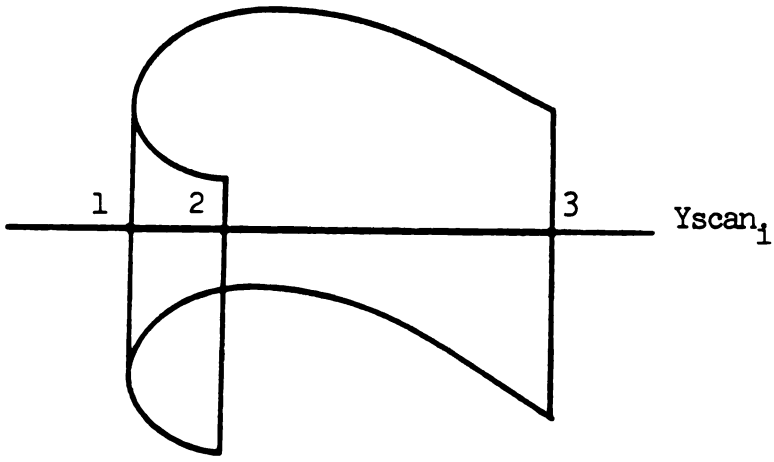


Figure B5. Pairing with One Silhouette Endpoint

When only one boundary endpoint exists on a scanline, it is assumed to be at a patch maximum or minimum, as shown in Figure B6, and the endpoint is deleted. Because these cases make up the majority of pairing, the time required for pairing is small.

More complicated cases are paired by determining if each endpoint is a leading or trailing endpoint. The algorithm which defines the leading or trailing status uses the idea that once the status of the patch boundary is defined, it retains that status until: 1) a silhouette edge starting point is passed, or 2) a change in the sign of dy/ds occurs along the boundary. These ideas are illustrated in Figure B7. At s=0, the boundary is a leading edge, but after the first critical point, i.e., the silhouette edge is passed, the patch boundary becomes a trailing edge. Note that the silhouette edge takes on the status the patch boundary had before it crossed the silhouette edge, thus the silhouette edge in Figure B7 is leading. When the second critical point, i.e., the sign change in dy/ds, is passed the patch boundary becomes a leading edge.

Therefore the status of endpoints is defined leading or trailing and they are stored as they are being computed. The algorithm starts by computing the status of the four patch boundaries at their beginnings. Next the silhouette edge starting points are computed. As endpoints of the patch boundary are computed, their status remains the same as the beginning until an 1) inflection point is passed, which corresponds to a change in the Yscan increment direction, or 2) a silhouette edge starting point is passed. In the second case, the silhouette edge starting point is assigned the same status as the

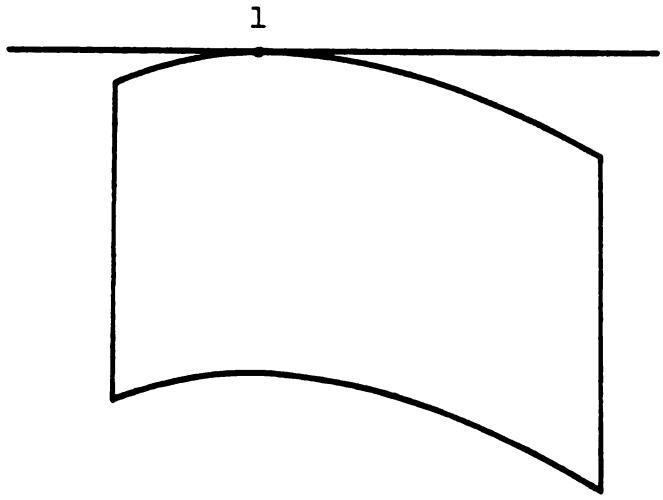


Figure B6. Single Boundary Endpoint

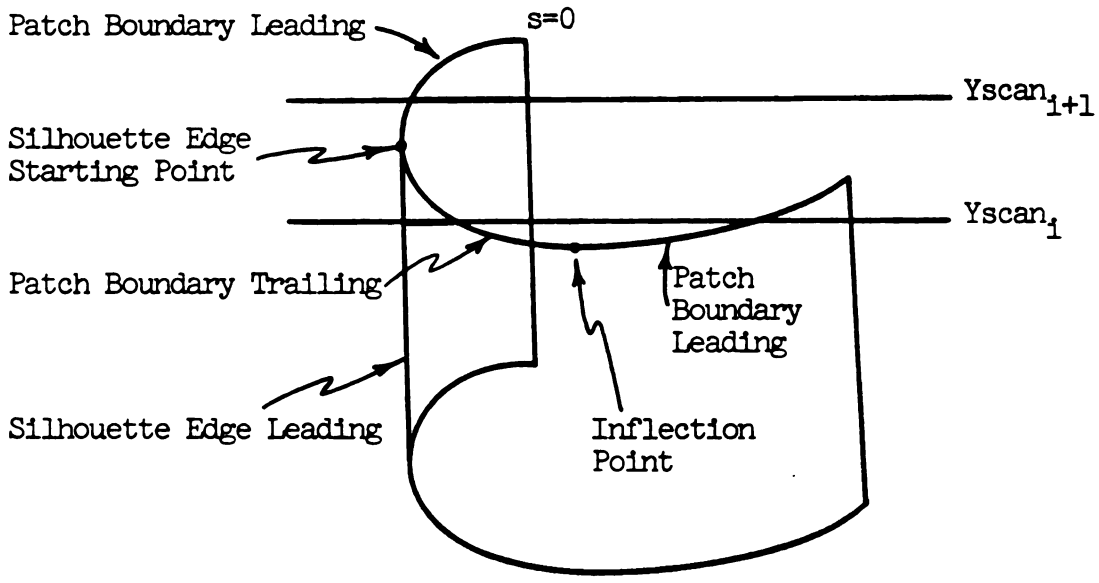
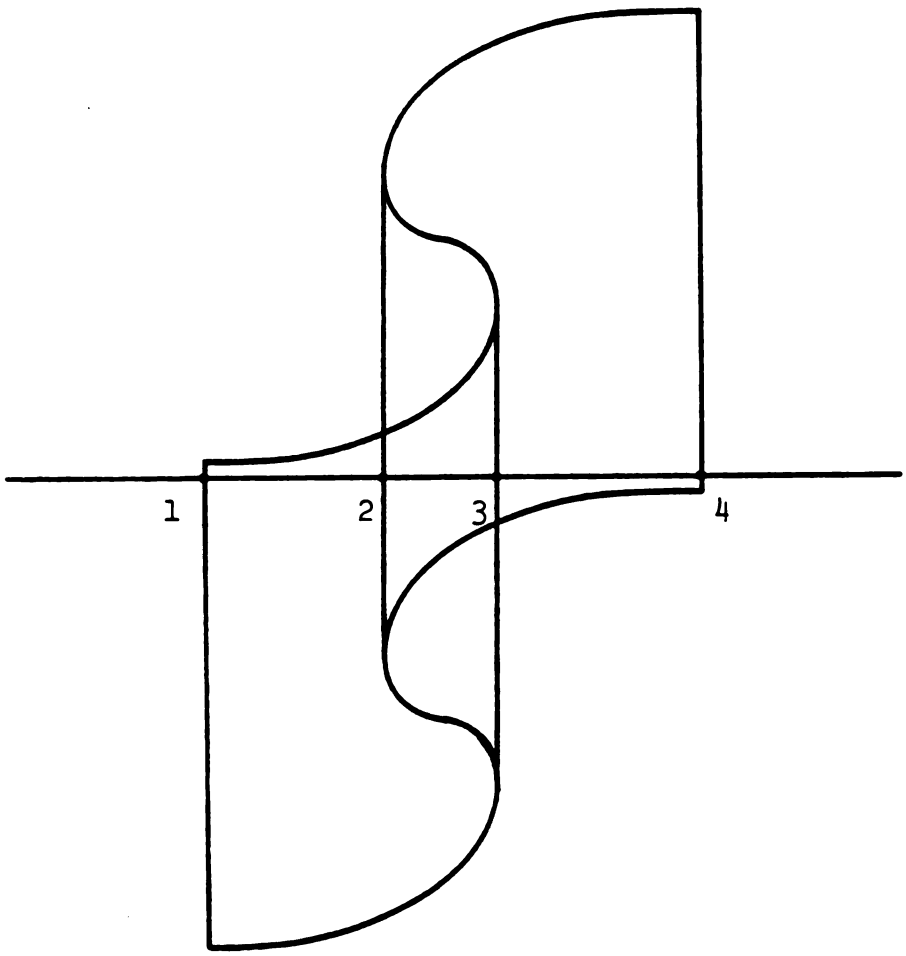


Figure B7. Determination of Leading or Trailing Status

boundary it started from. Silhouette edge endpoints then have the same status as their starting points.

Pairing endpoints is done by pairing leading endpoints in the order they occur from left to right with the closest trailing endpoints, until all endpoints are used. As endpoints are used they are deleted from the list so they will not be paired twice. Silhouette endpoints are paired twice. Figure B8 illustrates this process. Unpaired endpoints are displayed in the figure after each pairing.



Beginning Endpoint List; L-leading, T-trailing
1L, 2L, 2L, 3T, 3T, 4T
First Pairing 1L-3T
2L, 2L, 3T, 4T
Second Pairing 2L-3T
2L, 4T
Third Pairing 2L-4T

Figure B8. Pairing Example

APPENDIX C

The Generalized Secant Method for Solving Nonlinear Equations

Newton's method, which is a popular technique for solving nonlinear equations, is addressed in detail in (16) and (17). In general, for the one-dimensional case, a solution t_s, is sought for the nonlinear equation:

$$X(t) = 0 \tag{C.1}$$

Given a starting value t_0, t is incremented each iteration according to the equation:

$$t_{n+1} = t_n - \{\frac{dx(t_n)}{dt}\}^{-1} * X(t_n) \tag{C.2}$$

An example of Newton's method is graphically illustrated in Figure C1. The iterations are continued until $X(t_n)$ is less than some prescribed error, and then t_n is assumed to be a solution.

The secant method, which is similar to Newton's method, uses one iteration of Newton's method to start the sequence, and then increments t using the following equation:

$$t_{n+1} = t_n - [\frac{t_n - t_{n-1}}{X(t_n) - X(t_{n-1})}] * X(t_n) \tag{C.3}$$

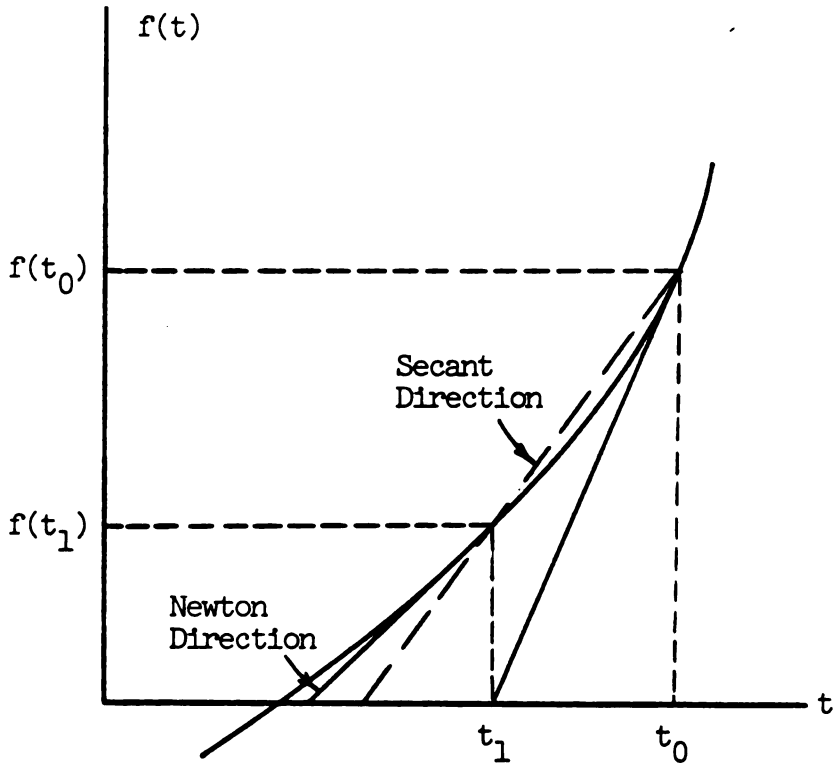


Figure C1. A Comparison of Newton's Direction and the Secant Direction

Note that the secant method approximated dx/dt with a finite difference, which is referred to as the secant direction. The secant direction is contrasted with Newton's direction in Figure C1. The figure indicates a decreased convergence rate for the secant method. The convergence rate for the secant method is $(1 + \sqrt{5})/2$ which is less than the quadratic convergence rate of Newton's method. The major advantage of the secant method is that derivatives are not required after the first step. The choice of the better method depends on the relative difficulty of computing the needed derivative.

Newton's method is easily extended to an N-dimensional space. In this case, there are N nonlinear equations of N variables of the form:

$$X_k(t_1, t_2, t_3 \ldots . t_N) = 0 \quad k = 1,2,3,\ldots N \tag{C.4}$$

or in vector form:

$$\bar{X}(\bar{t}) = 0 \tag{C.5}$$

where $\bar{X}$ and $\bar{t}$ are N vectors. The equation which updates the $\bar{t}$ vector becomes:

$$\bar{t}_{n+1} = \bar{t}_n - \tilde{J}(t_n)^{-1} \bar{X}(\bar{t}_n) \tag{C.6}$$

where $\tilde{J}(t_n)$ is the Jacobian matrix defined below.

$$\tilde{J}(t_n) = \left[\begin{array}{ccc} \frac{\partial x_1}{\partial t_1} & \frac{\partial x_1}{\partial t_2} & \frac{\partial x_1}{\partial t_N} \\ \frac{\partial x_2}{\partial t_1} & \frac{\partial x_2}{\partial t_2} & \\ \vdots & & \vdots \\ \frac{\partial x_N}{\partial t_1} & & \frac{\partial x_N}{\partial t_N} \end{array}\right]_{t_n} \tag{C.7}$$

The vector $\bar{t}_n$ converges quadratically toward $\bar{t}_s$, the solution vector.

Generalization of the secant to N dimensions is more difficult. A matrix $\tilde{B}_n$ is desired such that:

$$\bar{t}_{n+1} = \bar{t}_n - \tilde{B}_n^{-1}\bar{X}(\bar{t}_n) \tag{C.8}$$

converges to a solution vector $\bar{t}_s$. By definition of the secant direction, $\tilde{B}_n$ must also solve the equation:

$$\tilde{B}_n \Delta\bar{t}_n = \bar{X}_n \tag{C.9}$$

where $\Delta\bar{t}_n = \bar{t}_n - \bar{t}_{n-1}$ and $\Delta\bar{X}_n = \bar{X}(\bar{t}_n) - \bar{X}(\bar{t}_{n-1})$. There are many $\tilde{B}_n$s which solve equation C.9, but the assumption is made that $\tilde{B}_n$ is close to the previous $\tilde{B}_{n-1}$, and the difference between them is $\tilde{C}_{n-1}$:

$$\tilde{B}_n = \tilde{B}_{n-1} + \tilde{C}_{n-1} \tag{C.10}$$

Generalized secant methods are classified by the rank of the updating matrix, $\tilde{C}_n$. The discussion that follows deals with rank 1

updating methods, in particular Broyden's method (17). The derivation is as follows.

Substituting equation C.10 into equation C.9 yields:

$$(\tilde{B}_{n-1} + \tilde{C}_{n-1})\ \Delta\bar{t}_n = \Delta\bar{X}_n \tag{C.11}$$

Choosing a vector $\bar{W}$, such that $\bar{W}^T\Delta\bar{t}_n$ is not equal to zero, multiplying both sides of equation C.11 by $\bar{W}$, and rearranging yields:

$$\tilde{C}_{n-1} = \left(\frac{1}{\bar{W}^T\Delta\bar{t}_n}\right) * (\Delta\bar{X}_n - \tilde{B}_{n-1}\Delta\bar{t}_n) * \bar{W}^T \tag{C.12}$$

A common choice for $\bar{W}$ is:

$$\bar{W} = \Delta\bar{t}_n \tag{C.13}$$

This choice of $\bar{W}$ minimizes the change in $\tilde{B}_n$. Another choice for $\bar{W}$, which minimizes the change in $\tilde{B}_n^{-1}$, is:

$$\bar{W} = \tilde{B}_{n-1}^T\Delta\bar{X}_n \tag{C.14}$$

The effect of $\bar{W}$ on the convergence rate depends on the application.

It is useful to compare Newton's method and Broyden's method in the context of the x scan. The nonlinear system of equations being solved at each pixel is:

$$X(s,t) = Xscan_i \tag{C.15}$$

$$Y(s,t) = Yscan_i$$

The starting point used for each iteration is the solution at the previous pixel. The first step of Newton's method, which is required by both solution methods, is readily available from the known partial derivatives computed when finding the normal vector at the previous pixel. Therefore, comparisons of the two methods will begin after the first step.

The majority of the calculations required for solving equation C.15 using Newton's method arise from solving for x, y, and the Jacobian matrix at each step of the iteration. The Jacobian matrix consists of the four partial derivatives, dx/ds, dy/ds, dx/dt, and dy/dt. Computing each of these six values requires multiplication of three matrices, a 1x4 containing blending functions in s, times a 4x4 containing patch data, times a 4x1 containing blending functions in t. This product can be thought of as five vector dot products of length four, but the calculation of x and y has common intermediate products with two of the partial derivatives, therefore after computing x and y, only two addition dot products are required to obtain two partial derivatives. The remaining two partial derivatives cost five dot products each, therefore the approximate cost of Newton's method for one step is 22 dot products. On the other hand, Broyden's method only requires calculation of x and y at each iteration, along with calculations for the matrix updating approximately equal to two of the dot products discussed earlier. Thus the approximate cost for one step of Broyden's method is 12 dot products.

A direct comparison of the two methods for a given problem then depends on the average number of iterations required for each solution. Several test cases have indicated that for each pixel Newton's method

uses an average of .8 additional steps after the initial step, while Broyden's method required an average of one additional step. This indicates an approximate savings of 30 percent for solving equation C.15 during the **x** scan.

APPENDIX D

Calculation of $\frac{\partial N}{\partial x}$

The normal vector is a function of the parametric values s and t, therefore derivatives with respect to s and t are easily obtained. (see eq. 2.1.3) Computing derivatives of the normal vector with respect to x is not straightforward.

The first step in computing $\partial N/\partial x$ is to compute the derivative of the normal vector along the intersection curve cut by the scan plane. The intersection curve is a constant y curve, therefore the following equation is true along the curve:

$$dy = \frac{\partial y}{\partial s} * ds + \frac{\partial y}{\partial t} * dt = 0 \qquad (D.1)$$

This equation gives the ratio of ds to dt to move along the curve, as illustrated in Figure D1.

The derivative of the normal along the curve can be found using the following equation:

$$\frac{\partial N}{\partial u} = \frac{\partial N}{\partial s}\frac{\partial s}{\partial u} + \frac{\partial N}{\partial t}\frac{\partial t}{\partial u} \qquad (D.2)$$

where u is the arch length along the curve.

But the u direction contains components of x and z as shown in Figure D2. From Figure D2, dx and du are related by the following expression:

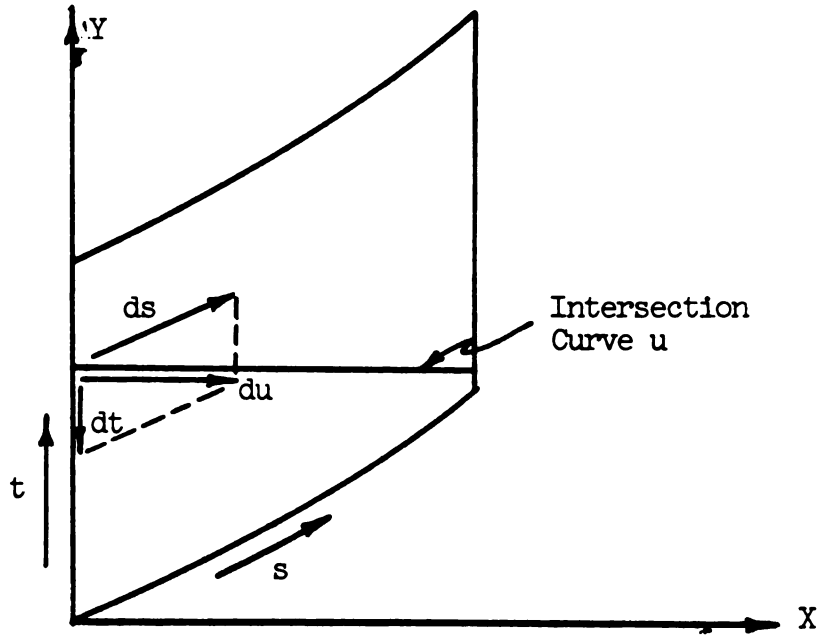


Figure D1. The Ratio ds/dt Along the Intersection Curve u

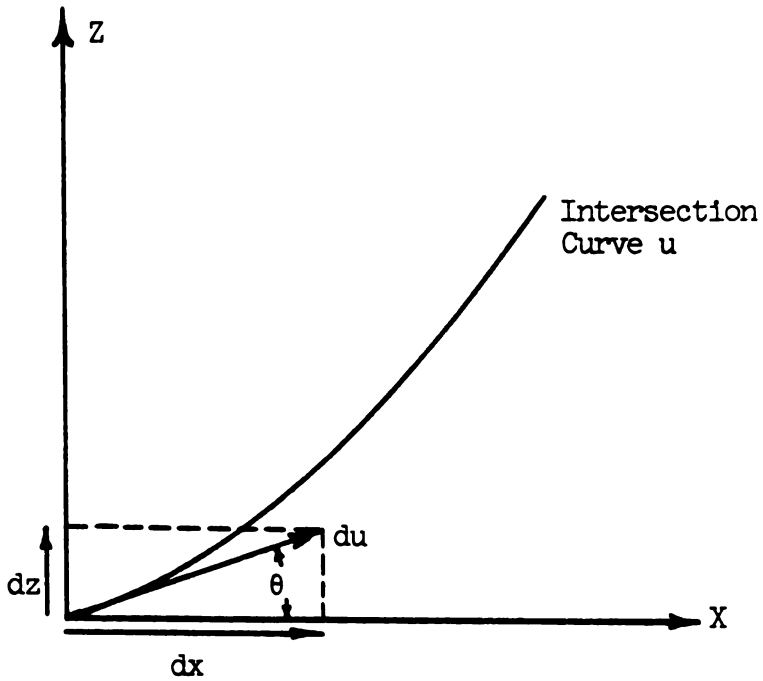


Figure D2. Relationship of the u and x Directions

$$dx = du * \cos(\theta) \tag{D.3}$$

where

$$\theta = \arctan\left(\frac{\partial z}{\partial x}\right) \tag{D.4}$$

where

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial u} \Big/ \frac{\partial x}{\partial u} \tag{D.5}$$

and

$$\frac{\partial x}{\partial u} = \frac{\partial x}{\partial s}\frac{\partial s}{\partial u} + \frac{\partial x}{\partial t}\frac{\partial t}{\partial u}$$

$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial s}\frac{\partial s}{\partial u} + \frac{\partial z}{\partial t}\frac{\partial t}{\partial u} \tag{D.6}$$

Thus, equation D.3 yields:

$$\frac{\partial N}{\partial x} = \frac{\partial N}{\partial u} * \frac{1}{\cos(\theta)} \tag{D.7}$$

LIST OF REFERENCES

LIST OF REFERENCES

1. Struik, D.J., "Differential Geometry", Addison-Wesley, Cambridge, 1950.

2. Weatherburn, C.E., "Differential Geometry", University Press, Cambridge, 1927.

3. Osserman, R., "A Survey of Minimal Surfaces", Van Nostrand Reinhold, 1969.

4. Forrest, A.R., "On Coons and Other Methods for the Representation of Curved Surfaces", Computer Graphics and Image Processing, 1972, pp. 341-359.

5. Rogers, D.F. and Adams, J.A., "Mathematical Elements for Computer Graphics", McGraw-Hill, 1976.

6. Coviak, R.A., "Color Graphics in Engineering Design", Masters Thesis, Dept. of Mechanical Engineering, Michigan State University, E. Lansing, MI., 1981.

7. Warnock, J.E., "A Hidden Surface Algorithm for Computer Half-Tone Generated Pictures", Computer Science Dept., University of Utah, TR4-15, June 1969.

8. Newell, M.E., Newell, R.G., and Sancha, T.L., "A Solution to the Hidden Surface", 1972.

9. "Graphics Utah Style - 80", Manual, University of Utah, Salt Lake City, Utah, 1980.

10. Sutherland, I.E., Sproull, R.F., and Schumacher, R.A., "A Characterization of Ten Hidden Surface Algorithms", Computing Surveys, Vol. 6, No. 1, March 1974, pp. 1-55.

11. Watkins, G.S., "A Real-Time Visible Surface Algorithm", Computer Science Dept., UTECH-CSC-70-101, June 1970.

12. Lane, J.M., Carpenter, L.C., Whitted, T., and Blinn, J.F., "Scan Line Methods for Displaying Parametrically Defined Surfaces", Communications of the A.C.M., Vol. 23, No. 1, Jan. 1980, pp. 23-34.

13. Catmull, E., "A Subdivision Algorithm for Computer Display of Curved Surfaces", UIEL-CSC-74-133, 1974.

14. Blinn, J.F., "Computer Display of Curved Surfaces", Ph.D. Diss., Computer Science Dept., University of Utah, Salt Lake City, Utah, 1978.

15. Whitted, T., "An Improved Illumination Model for Shaded Display", Communications of the A.C.M., Vol. 23, No. 6, June 1980, pp. 343-349.

16. Ortega, J.M. and Rheinbolt, W.C., "Iterative Solutions of Non-linear Equations in Several Variables", Academic Press, London and New York, 1971.

17. Conte, S.D. and deBoor, C., "Elementary Numerical Analysis", McGraw-Hill, New York, 1980.