A High-Level Computer Graphics Implementation

of

Three-Dimensional B-Spline Surface Generation

by

DENNIS MICHAEL BRYANT

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

University of Washington

1987

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Any other reproduction for any purposes or by any means shall not be allowed without my written permission.

Signature ______________________________

Date ______________________________

University of Washington

Abstract

A HIGH-LEVEL COMPUTER GRAPHICS IMPLEMENTATION OF THREE-DIMENSIONAL B-SPLINE SURFACE GENERATION

by Dennis Michael Bryant

Chairperson of the Supervisory Committee:
Professor D. E. Calkins
Department of Mechanical Engineering

The difficulty of accurately defining, representing and visualizing three-dimensional sculptured surfaces has concerned designers and engineers throughout history, but never more so than today. Technology has given man the ability, and the necessity, to build automobile bodies, ship hulls and aircraft fuselages with complex curvature and extremely smooth lines. The ability to efficiently develop and define the form of such surfaces has not kept pace with the requirement to manufacture them.

This study explores the application of a surface B-spline for surface definition coupled with high-level computer graphics to enhance visualization. A computer program, SPLINE, is introduced which facilitates interactive construction of the B-spline control graph and real-time manipulation of the model on an Evans and Sutherland PS 300 distributed graphics system. A uniform nonrational surface B-spline is utilized for interactive modeling, with the capability preprogrammed of implementing nonuniform rational surface B-splines upon

later addition of an inversion algorithm.

While the program is not fully functional, due to apparent shortcomings of the host computer operating system, the viability and desirability of such a system is fully demonstrated. Heuristic techniques are discussed for approaching model development, based on properties of the B-spline and characteristics of SPLINE. Step-by-step illustrated case studies further develop the modeling techniques. Examples include the generation of B-spline representations of a sailing yacht hull and of a Porsche automobile body, as well as the freeform development of a sculptured surface *ab initio*.

Structured and heavily commented program listings are included, with flow diagrams, function networks, and operating instructions. Program expansions and hardware improvements are recommended and existing provisions within the program for these improvements are identified.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Number Page

LIST OF TABLES

ACKNOWLEDGMENTS

The author wishes to express appreciation to Professor Calkins for his guidance in this study. Special thanks is also given to Mr. John Ishimaru for his patience and assistance, which were so needed during programming of the PS 300.

DEDICATION

To Linda,

for her patience.

CHAPTER 1

SCULPTURED SURFACE REPRESENTATION

1.1. Sculptured Surfaces

The computer program described herein was developed to demonstrate the feasibility of designing and defining sculptured surfaces through the use of a B-spline algorithm coupled with highly interactive computer graphics. The accurate definition, representation, and visualization of three-dimensional surfaces is a problem which has concerned designers and engineers throughout history. Even relatively simple three-dimensional shapes can often be difficult to depict on a two-dimensional drawing such that the viewer can readily generate a mental image of the object or the craftsman can construct the object true to the designer's intentions.

Particularly troublesome, however, are sculptured surfaces, those having curvature in more than one direction. Sculptured surfaces form a vital part of aircraft, ship, and automobile designs, being necessary in aircraft and ship design for aerodynamic and hydrodynamic streamlining and used extensively in automobile design primarily for styling reasons. Nor is the use of these troublesome surfaces for the sake of appealing to the eye restricted to such major market items as automobiles; as Rogers and Satterfield [1] point out, today even the

shampoo bottle is likely to sport a complex sculptured surface.

Given the difficulty of representing these surfaces in a two-dimensional medium, it is not surprising that the historical solution has often been simply to avoid producing such drawings at all. Letcher [2] presents a brief but interesting history of man's attempts to skirt the problem in the context of ship hull design. The form of the earliest hulls, and of many small craft in low-technology cultures today, was undoubtedly preserved by carving a solid model from wood. Later, the use of the master-section-and-batten method was developed and has continued to some extent until today. A few transverse sections are erected with the desired form, and the remainder of the hull is defined by the installation of the hull planking as it is sprung about the master sections.

Even as late as the early 1900's, the lines drawing was not the principal tool in developing and visualizing the lines of a ship's hull. While the lines were ultimately preserved with such a drawing, they were usually taken from a carved model. Often the model was even carved in a material with separable layers so that the lines could be directly traced onto the paper.

1.2. The Ship's Lines Drawing

The problem of defining and representing a sculptured

surface is perhaps best illustrated by a description of the lines drawing, that instrument which is used to develop and define the surface of ship and boat hulls. As is standard with the representation of virtually all three-dimensional objects, three views of the hull are presented on the lines drawing: a top view (known as the half-breadth plan, because only one half of the symmetrical hull is shown); a front view (known as the body plan, and which actually is a stern view as well), and a side or profile view (labeled the sheer plan).

Intersections of the vessel's hull with evenly spaced cutting planes are projected onto these three reference planes. On the body plan, the form of the hull at transverse sections is shown. Depending on the size and complexity of the vessel's hull, the sections are shown at ten, twenty or forty foot intervals, with a few intermediate stations shown in areas of greatest curvature. For the sheer plan, vertical cutting planes are taken at one to four foot intervals, yielding a set of buttocks lines. Waterlines are similarly generated on the half-breadth plan by intersecting the hull with vertical longitudinal cutting planes. Nonorthogonal cutting planes further serve to define the shape through the production of diagonals or cant lines. A sample lines drawing is shown in figure 1.

The task of the naval architect in developing a

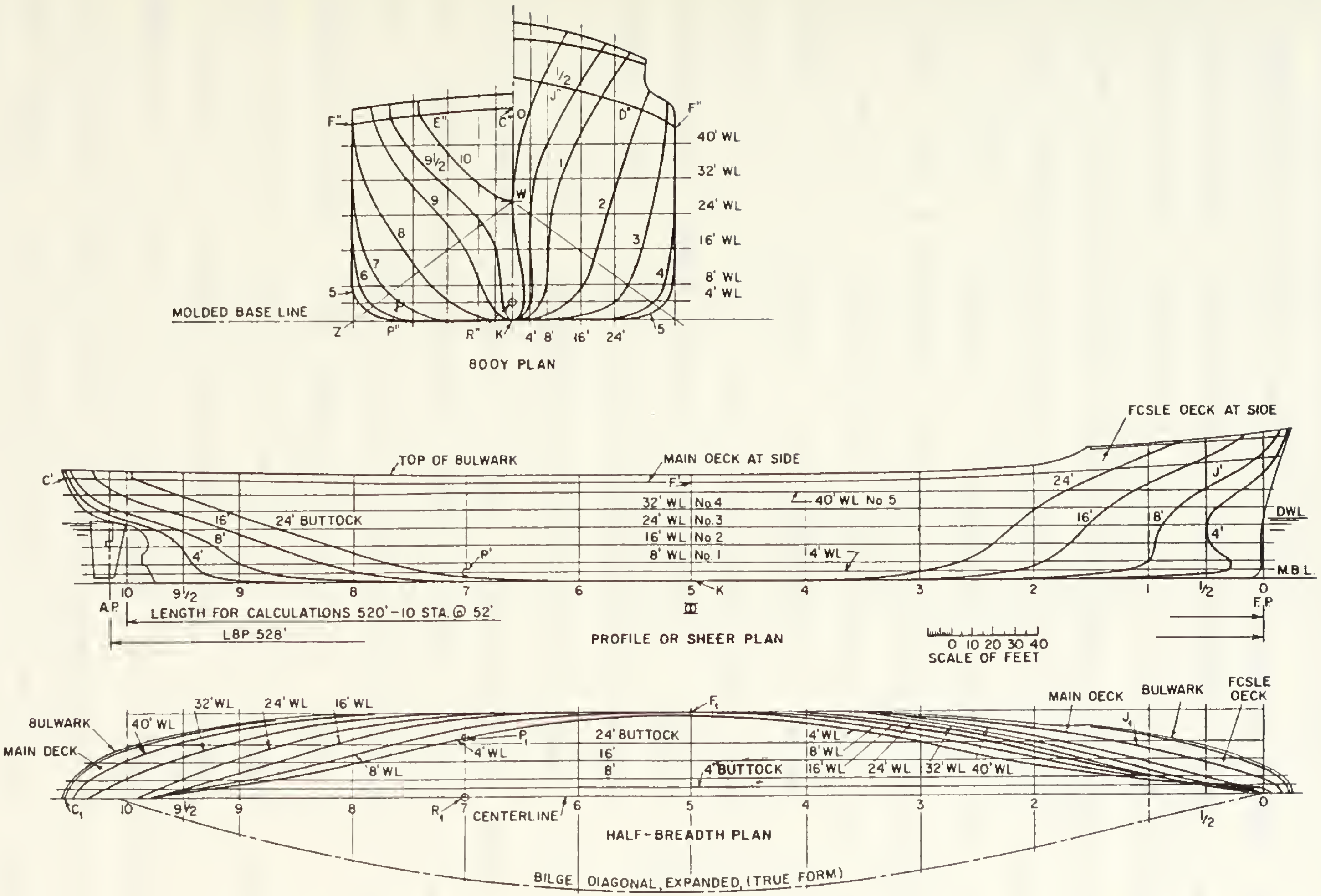

Fig. 1. Lines drawing for a Mariner class vessel. Source: Owen [3].

completed set of lines is not an easy one. Having visualized the desired form, the architect must accurately translate it into a set of lines on paper which satisfies quantitative, qualitative, and aesthetic considerations. Working with irregularly shaped curves, the designer must nevertheless maintain specific areas within the curves to yield proper ship stability and handling characteristics. In order to obtain fair (smooth) lines, the lines are drawn with a drafting spline, an elastic lath constrained by weights called ducks. The lines must not only be smooth and contain the appropriate areas, but the projection of all points on the lines onto all three reference planes must be consistent. These requirements often appear to be nearly mutually exclusive.

In short, because of the difficulties of (1) visualizing the sculptured surface, (2) obtaining smooth lines, (3) maintaining agreement of all views, and (4) achieving and measuring the required form characteristics such as section areas, the design of a ship's lines is a tedious and difficult iterative task for even the most experienced architect. Letcher [2] cites Kinney as saying, "'It takes between five and ten working days for an experienced man to produce a good set of lines,' regardless of the size of the vessel." He also points out that anything more than a slight change requires starting over almost from the beginning.

When the naval architect has finished the lines drawing, however, the task of hull definition is not yet complete. The lines are next translated into a tabulated form called a table of offsets. This table lists the distances, at each station, of waterlines from the centerline and of buttocks lines above the baseline. An example is shown in table 1.

Greater accuracy is needed for production purposes than is obtainable from the relatively small scale lines drawings. For this reason the lines are redrawn full size on a large wooden floor (the mold loft) and faired again. In very recent practice, the full scale lofting may be replaced by drawing the lines in one-sixteenth scale on a marble slab with a very hard pencil or by entering the offsets into a computer for generation of a large scale drawing on a plotter. In any case, the large scale fairing is used to identify necessary revisions to the lines and to produce a more accurate table of offsets.

1.3. Mathematically Defined Surfaces

It is not hard to understand, then, the benefits which would be derived if the hull form, or shampoo bottle, could be defined by a precise mathematical relationship. The need for full scale lofting would be eliminated, since accuracy could be obtained initially to whatever degree was desired. Precision would no longer depend upon the ability to measure small increments on a

TABLE 1

Table of Offsets for a Mariner Class Vessel

Source: Owen [3]

(Dimensions given in feet, inches and eighths)

Station (FWD END)	Half Breadths at Waterlines and Decks			
	0'-0" WL	4'-0" WL	8'-0" WL	16'-0" WL
0 FP	---	3-0-5	2-8-2	1-0-1
1/2	---	4-8-7	5-0-7	4-3-5
1	---	6-10-2	7-11-6	8-6-1
2	2-4-0	13-7-6	16-9-3	19-9-0
3	9-2-0	24-1-0	27-7-1	30-10-3
4	21-4-0	33-3-0	35-8-1	37-2-2
5	28-5-6	36-2-6	37-10-3	38-0-0
6	24-10-4	34-8-1	37-0-7	37-11-7
7	13-6-4	27-1-4	31-2-6	35-8-1
8	4-5-2	15-0-3	19-3-3	26-2-2
9	---	6-0-7	7-9-4	11-2-4
9 1/2	---	2-8-7	3-4-1	4-3-0
AP 10	---	---	---	---
(AFT End)				

Station (FWD End)	Heights Above Molded Baseline			
	4-ft buttock	8-ft buttock	16-ft buttock	24-ft buttock
0 FP	47-5-6	57-0-7	---	---
1/2	2-3-0	40-8-3	55-4-4	---
1	0-11-7	8-0-7	43-4-1	56-8-1
2	0-0-4	0-8-7	6-9-2	31-5-7
3	---	0-0-0	0-6-7	3-11-1
4	---	---	0-0-0	0-1-1
5	---	---	---	0-0-0
6	---	---	---	0-0-0
7	---	---	0-0-6	2-1-0
8	0-0-0	0-3-5	4-9-6	13-3-3
9	1-2-1	8-6-6	21-11-3	30-0-4
9 1/2	14-3-4	23-4-1	30-3-6	38-4-5
AP 10	28-5-0	31-6-5	38-11-3	---
(Aft End)	37-1-4	41-7-0	44-1-6	---

drawing, but could be achieved by simply outputting a few more digits from the computer's memory.

Preparation of a table of offsets would not be a manual task, but simply a matter of solving the mathematical relationship at any desired point on the surface. Thus the representation would not only be more accurate, but could be more complete as well. A lines drawing only accurately defines the surface at points which fall exactly on the lines drawn. Determination of intermediate points requires drawing and fairing additional lines or performing some manner of interpolation. Interpolation will yield only a rough approximation, given the complex curvature of the form. A mathematical representation can be equally accurate at any point which is defined by the mathematical relationship.

A mathematical representation could also provide direct input for the calculation of stability data, as well as for fabrication. The need for scale model testing would be greatly reduced with the substitution of computer simulations, which would no longer need to rely upon mere approximations of the form.

Perhaps most exciting are the prospects for using a mathematical representation of a sculptured surface as input to a numerically controlled machining (NCM) process. Rogers, Rodriguez, and Satterfield have already reported substantial progress in this application [4,5,6]. A

system of computer programs was developed on a microprocessor-based graphics system interfaced with a CNC controller to drive a milling machine for the production of wooden towing tank models. The numerical control milling data base is actually generated directly by a computer program from a mathematical definition of the hull. This mathematical definition is created by the program in response to manipulation of a graphical image on the computer screen by the user. Using this system, towing tank models have been produced in only two days.

Considering the beleaguered state of the shipbuilding and automotive industries in the United States today, it is impossible not to become excited with the possibilities of larger scale implementations of this concept.

1.4. High-Level Computer Graphics

The implementation of mathematical definition of sculptured surfaces does not require enormous computing power. The FAIRLINE series of computer programs developed by Letcher [2,7,8] for the design of sailing yacht hulls was designed for use on a Texas Instruments pocket calculator. On the other hand, defining the surface is only part of the problem. Closely related is the problem of representing the surface during the surface development process and subsequent to surface definition. It may be difficult to visualize a three-dimensional surface while looking at a two-dimensional drawing, but it is even more

difficult to visualize a complex sculptured surface by reviewing a computer-generated table of coordinates.

It is when coupled with high-level computer graphics that the mathematical representation begins to be as useful as it is powerful. High-level computer graphics have been defined by Calkins [9] as being characterized by:

(1) the ability to display three-dimensional objects,

(2) real-time image manipulation of the three-dimensional objects displayed, and

(3) color-shaded object surfaces.

Calkins summarizes high-level computer graphics as the "capability of real time manipulation of realistically rendered 3-dimensional objects." (It will be shown later that while the computer system utilized for the research described in this paper does not meet the third criterion, it certainly fulfills the intention of this summation.) With high-level computer graphics, the designer can rotate the object about to obtain another perspective or zoom in on a troublesome detail. The results of changes made in one view can be immediately reviewed from other angles to determine their suitability.

It is this merging of visualization of sculptured surfaces through high-level computer graphics and definition by mathematical representation which has been

explored in this research.

CHAPTER 2

B-SPLINE SURFACES

The form of mathematical surface representation selected for implementation in the computer program under consideration is the surface B-spline. In particular, a nonuniform rational surface B-spline algorithm has been utilized, though in its present form the program only allows the user direct control of a subset of this, the uniform rational surface B-spline.

The B-spline concept was developed by Schoenberg [10] in 1946 and introduced to computer-aided geometric design by Riesenfeld [11] in 1973. Several properties of the B-spline curve or surface generation algorithms make them very suitable for sculptured surface modeling using interactive computer graphics. Briefly, the B-spline offers inherent curve smoothing, considerable data compression (complex surfaces can be fully defined with a small amount of data), local control (a portion of a curve or surface can be modified without affecting the entire surface), the flexibility of including sharp discontinuities in a curve or surface, and a convenient means of interactively manipulating the curve.

This last property will be shown to be a particularly useful characteristic. Barsky and Greenberg [12] discuss the difficulty of performing shape design using

mathematical methods which require the designer to directly specify values for algorithm variables. It can require an intimate knowledge of complex mathematics for a designer to be able to readily perceive the necessary adjustment to a mathematical algorithm which will effect the desired change in a portion of a surface. In contrast, modeling with B-splines requires virtually no knowledge of mathematics by the program user. Fish [13] observed, "The nice thing about spline curves is that they allow you to treat a collection of pieces of various polynomials as a single entity, maintaining continuity of tangent directions, curvatures, and so on without working at it." Undoubtedly the same considerations led Rogers [4], working with a number of mathematical curve representations, to conclude that the B-spline was "most generally useful."

While a detailed understanding of the mathematics behind B-splines is not necessary in order to use them, some understanding will nevertheless aid the user in developing the ability to more quickly model a desired surface. A brief introduction follows. More extensive treatments of the subject can be found in Rogers and Adams [14], who provide useful examples, and in Bartels, Beatty, and Barsky [15], who provide a particularly methodical and understandable approach.

2.1. Parametric Curves

The Cartesian coordinate system is essential to most surface design. It is in this coordinate system that most of the formulations are presented with which the naval architect or aircraft engineer will work to determine the performance characteristics of the craft. It is also Cartesian coordinates in which the graphics computer will require data to be presented for display and manipulation on the screen. At the same time, the Cartesian coordinate system has significant liabilities when working with sculptured surfaces.

Most significantly, many sculptured surfaces or curves cannot be defined by an explicit function (a function having only one value of y for each value of x). The transverse section of a ship with tumble-home is a ready example. Also, it will be shown that calculating one Cartesian coordinate for a curve at even increments of another Cartesian coordinate can result in very irregular definition resolution. For these reasons, a parametric representation is preferable. The B-spline can be parametric. In a parametric representation, the Cartesian coordinate for a point on a curve is not defined in terms of the other Cartesian coordinate (i.e. $y = f(x)$), but coordinates are instead defined as functions of a parametric variable: thus,

$$x = f(u)$$

and

$$y = g(u),$$

where *u* is the parametric variable.

The parametric representation addresses both of the objections to the purely Cartesian coordinates. Implicit functions in *x* and *y* can be handled without ambiguity and the resolution of the curve representation is generally much more uniform with equal parameter spacing than with equal Cartesian coordinate spacing.

Rogers and Adams [14] provide an illustration similar to figure 2, which compares representations of a circular arc using a nonparametric representation with data points at equal *x* intervals, using uniform parametric spacing, and using equal arc lengths. The nonparametric representation uses the relationship,

$$y = \sqrt{(1-x^2)}. \tag{1}$$

The uniform parametric spacing uses a parameter, *u*, which varies from zero to one. The points on the curve are defined by

$$x = (1-u^2)/(1+u^2)$$

and

$$y = 2u/(1+u^2). \tag{2}$$

The equal arc length representation defines the points on the curve as

$$x = \cos\theta$$

and

$$y = \sin\theta . \tag{3}$$

In each case, the arc is represented by a straight-line connection between six points.

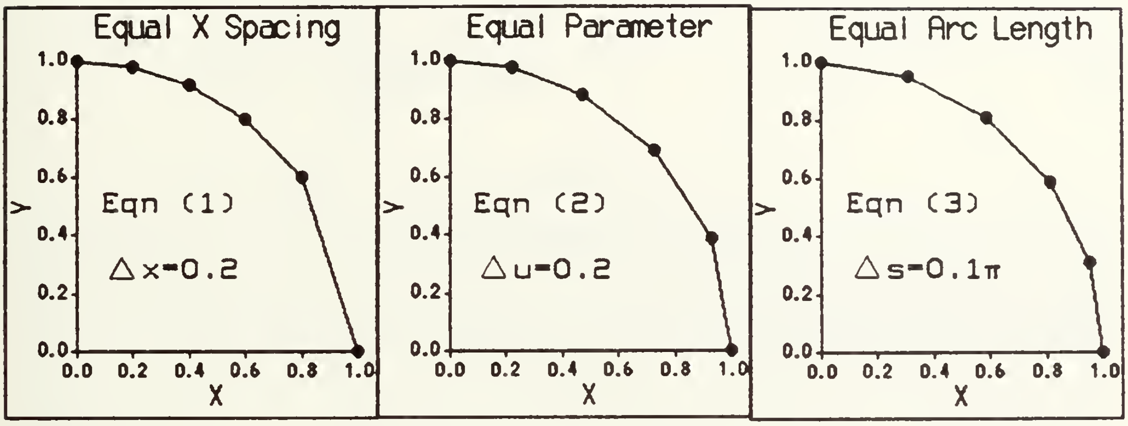

Fig. 2. Comparison of parametric and nonparametric representations.

While the uniform parametric spacing does not produce a resolution quite as uniform as that of equal arc lengths, it is clearly superior to the equal *x* coordinate spacing. Sharpe and Thorne [16] discuss an approach for using the equal arc length spacing with a parametric representation, but more straightforward equal parameter spacing was felt to be sufficient for the purposes of this program.

2.2. Two-Dimensional B-Splines

The B-spline is most easily understood initially by looking at a two-dimensional uniform nonrational case, the simplest form of the B-spline. The B-spline curve is a piecewise polynomial curve. That is, the curve is broken into segments which are defined by separate polynomials.

This curve is everywhere continuous, both in position and in up to the $(K-2)$ derivative, K being the order of the spline.

The general B-spline curve is defined by:

$$Q(u) = \sum_{I=1}^{n} B_I N_{I,K}(u) \qquad (4)$$

$$N_{I,1}(u) = 1 \text{ if } x_I \leq u < x_{I+1}$$

$$= 0 \text{ otherwise}$$

$$N_{I,K}(u) = \frac{(u-x_I)N_{I,K-1}(u)}{x_{I+K-1}-x_I} + \frac{(x_{I+K}-u)N_{I+1,K-1}(u)}{x_{I+K}-x_{I+1}}$$

where

$Q(u)$ = $[x(u),y(u)]$, the curve,

n = number of control vertices,

B_I = control vertices' coordinates,

K = order of the B-spline curve (implies degree $(K-1)$),

$N_{I,K}(u)$ = B-spline basis functions,

and

x_I = elements of the knot vector.

While it can be seen that the B-spline algorithm can be somewhat confusing mathematically, the use of it is actually quite intuitive. The curve being formed is shaped by controlling a relatively small number of points known as *control points* or *vertices*. The curve will not interpolate (pass through) the control vertices, but will mimic the overall shape of the array of vertices. For a

better visualization of this, the vertices are frequently joined by straight line segments, forming a *control polygon*. An example control polygon with its associated B-spline curve is shown in figure 3. The vertices are represented by the squares, the control polygon by the connecting straight line segments, and the B-spline curve by the curved line.

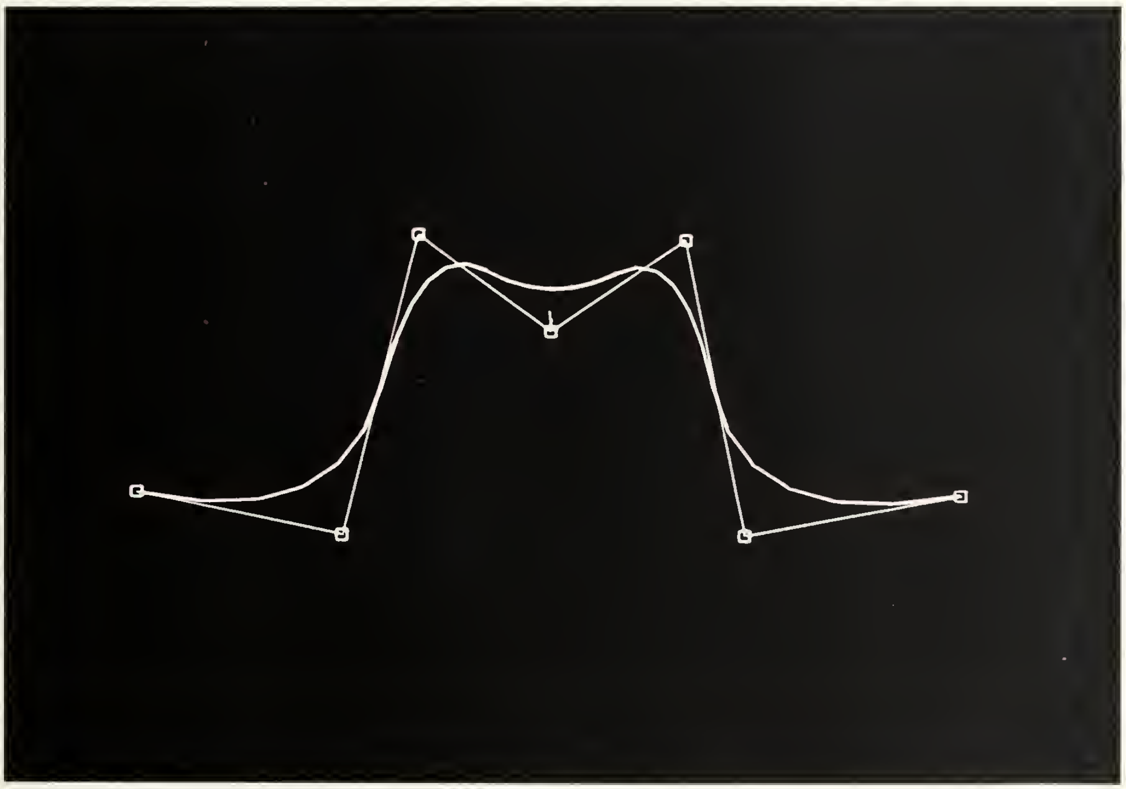

Fig. 3. Third order B-spline curve with control polygon.

An apt analogy to the effect of the control vertices upon the curve is that of a set of magnets to a ferrous wire. While the curve does not pass directly through the control vertices, it is molded by their placement, being drawn toward them as if toward magnets. Moving a control

vertex pulls the curve in the vertex's local region in the direction of the movement. Placing vertices close together creates a stronger effect on the curve in the region of those vertices.

Every point on the curve is a weighted average of the position of a number of control vertices. Each vertex has a unique basis function, which defines the weight of the vertex with respect to the parameter u. These basis functions, polynomials in the parametric variable, are independent of the shape of the controlling polygon, being completely defined by the order (k) and a knot vector.

The knot vector divides the curve parameter (u) into a series of discrete values. This subdivides the B-spline curve into a series of segments defined by separate polynomials. The joined segments form a piecewise polynomial curve. If the divisions in the curve are equal (the knot vector specifies uniform step width between knots), the B-spline is said to be uniform.

The parameter u varies between an initial value and some final maximum as the curve is traversed. Values of u that correspond to joints between the spans are called ***knots***. In the case of a uniform B-spline, the knots will typically correspond to integer values of u. The knot vector gives the sequence of these knots and is constrained to be non-decreasing. Thus the knot vector defines the range of the parameter u which corresponds to

each segment of the curve.

When the knot vector is specified, a multiplicity *K* at each end of the knot vector ensures that the ends of the curve interpolate the first and last vertices in the control polygon. This is a very convenient constraint when performing modeling with a B-spline. The knot vector for a uniform third order B-spline with four vertices, which varies from zero to (n-K+1), is [0,0,0,1,2,2,2].

The basis functions, then, indicate the degree and extent of the influence of each control vertex. This provides the local control which was mentioned above. The maximum extent of influence of a single control vertex in

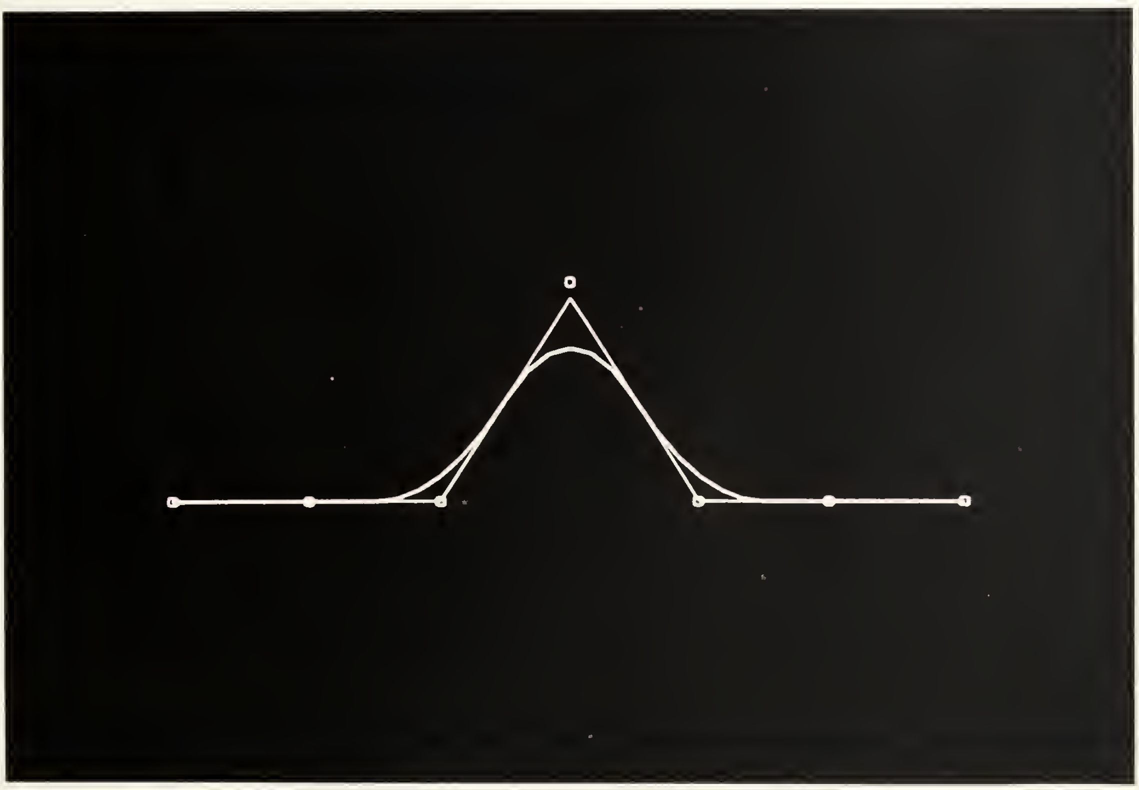

Fig. 4. Localized effect of moving a single control vertex.

parameter space is plus or minus half the order of the basis function. Conversely, the shape of every span is affected by K successive vertices. Figure 4 illustrates the localized effect of moving a single control vertex.

A comparison of figures 5 and 6 will demonstrate the effect of curve order upon the basis functions. Figure 5 shows the values for the basis functions for a third order curve with five vertices. Figure 6 shows the basis function values for a fourth order curve, also with five vertices. In each case, the parameter u has been normalized to unity. Notice that the basis functions for the fourth order curve are generally lower in value at any given value of u, but are nonzero over a greater range. Thus a higher order basis function yields a curve which is less markedly influenced locally by the movement of a single control vertex, but which is smoother and more taut than a lower order curve. This effect can be readily visualized by examining figures 7 and 8, which show third and fourth order B-spline curves created by the same control polygons.

A few characteristics of the B-spline curve will be convenient to keep in mind when manipulating a curve:

(1) The B-spline curve will have at most as many extreme and inflection points as the control polygon.

(2) A straight line will be produced in the B-

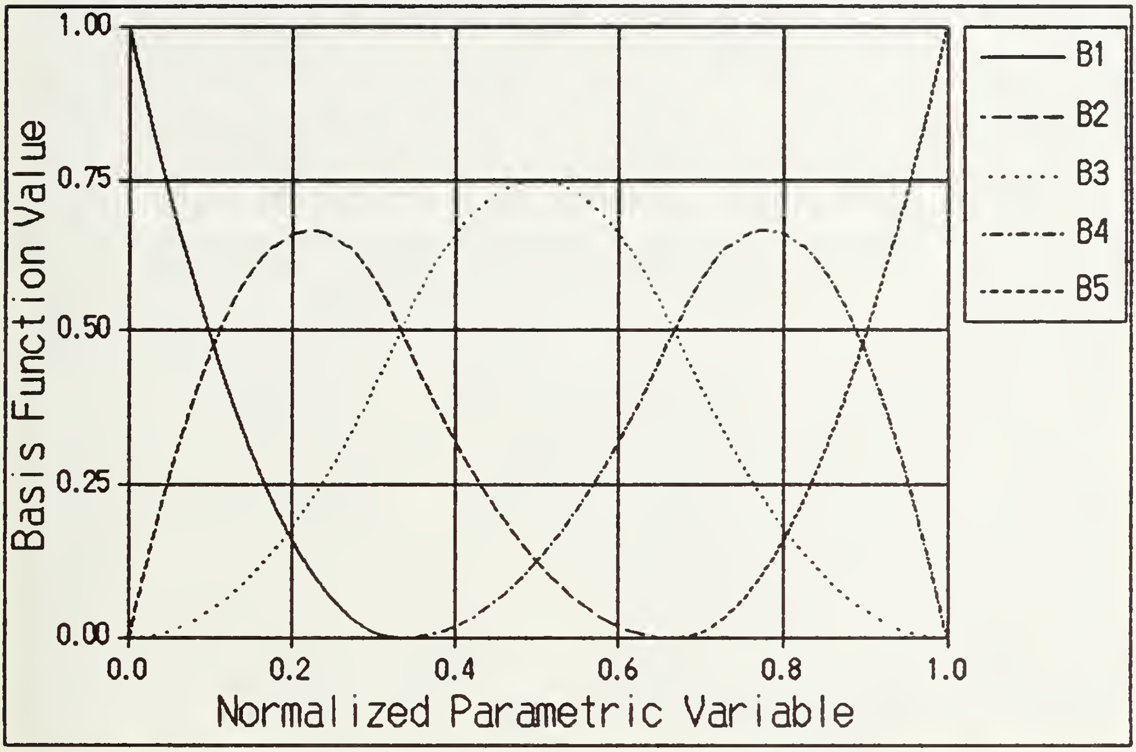

Fig. 5. Third order basis functions for five vertices.

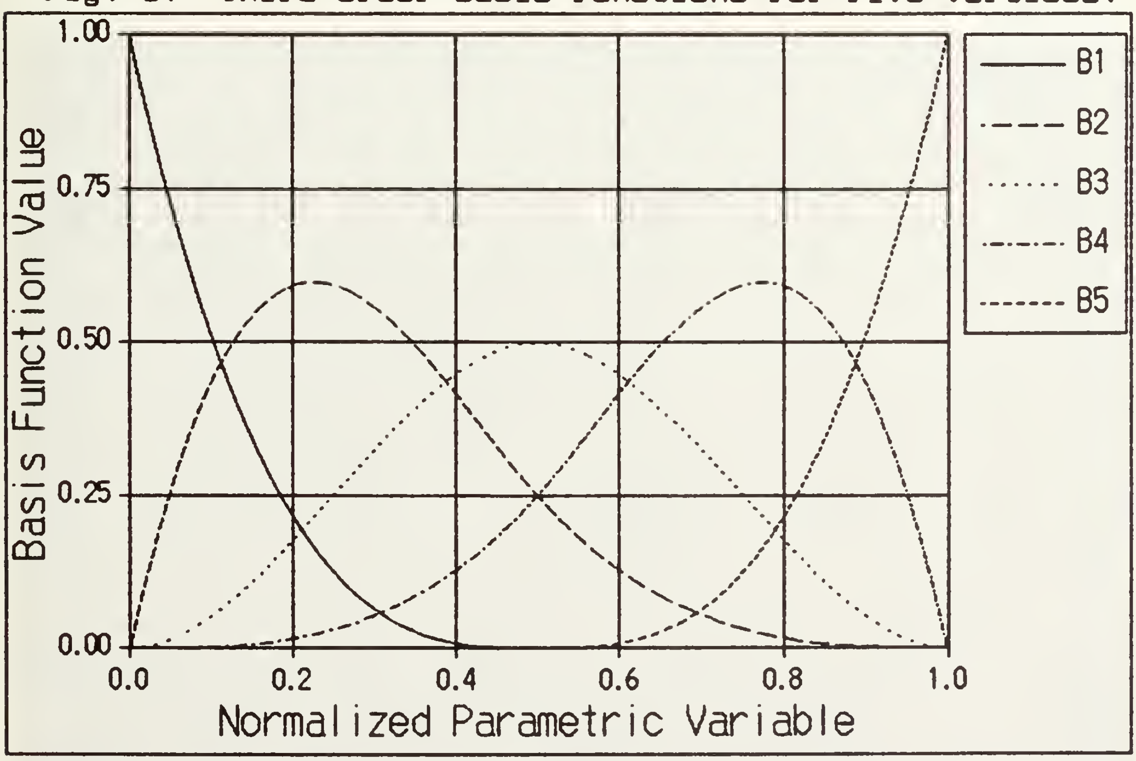

Fig. 6. Fourth order basis functions for five vertices.

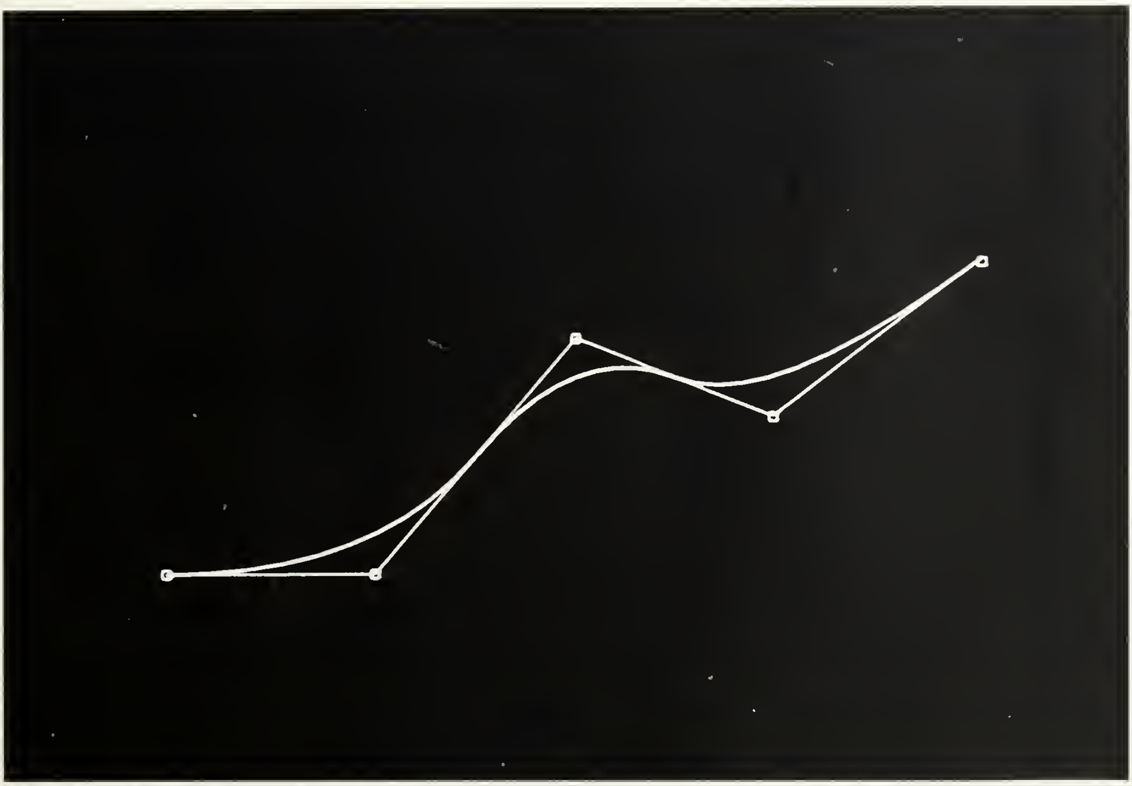

Fig. 7. Third order B-spline curve.

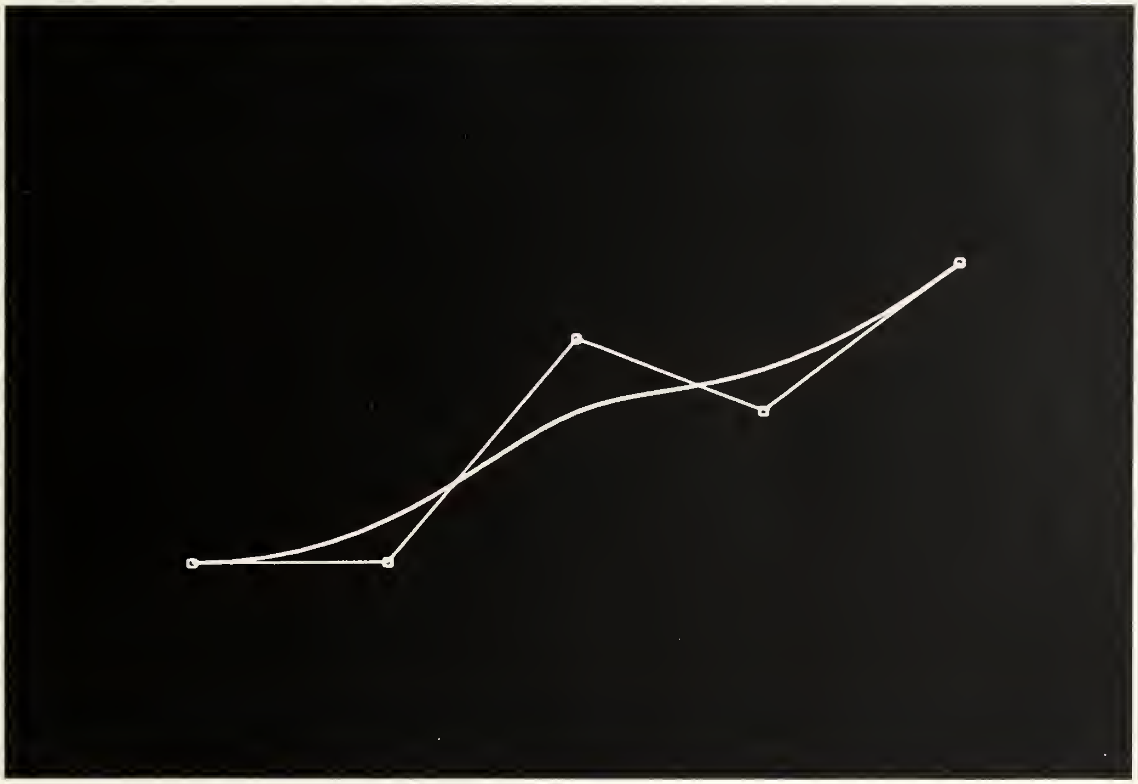

Fig. 8. Fourth order B-spline curve.

spline curve by *K* collinear control vertices.

(3) *K-1* collinear control vertices will yield a curve tangent to the control polygon.

(4) A knuckle, or sharp discontinuity, will occur at the point where *K-1* control vertices are colocated. An example is shown in figure 9.

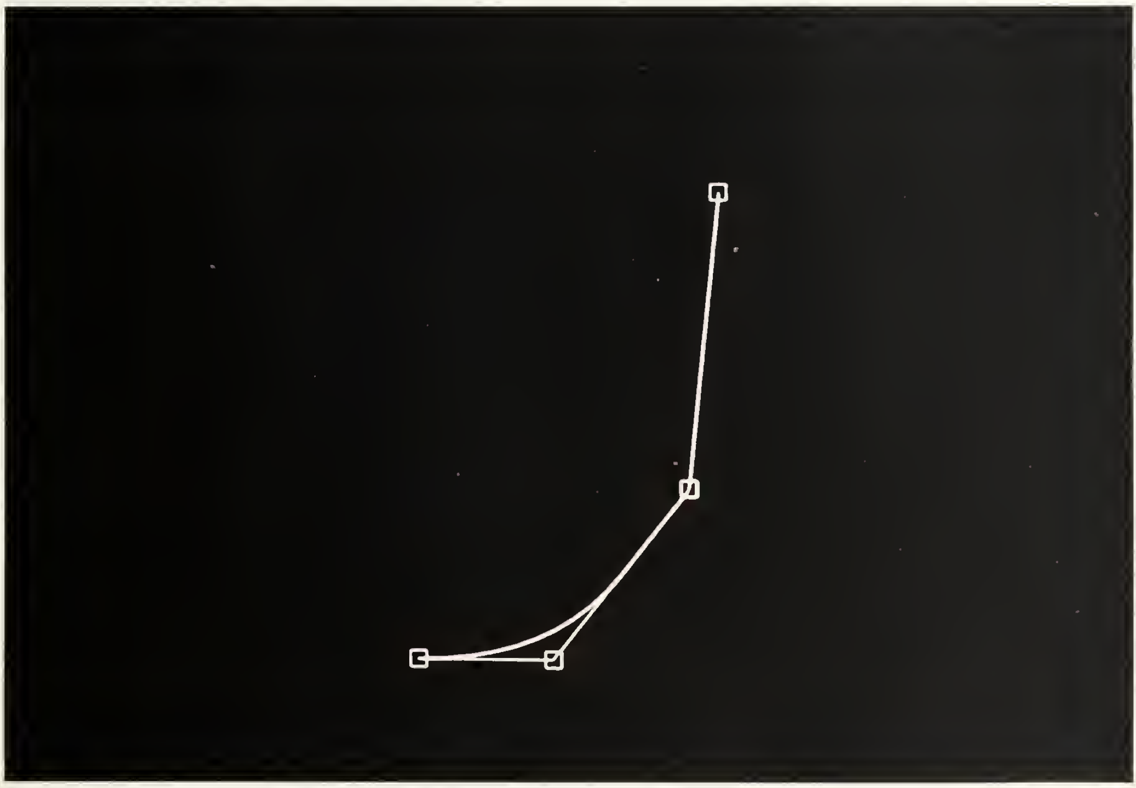

Fig. 9. Knuckle produced by colocated vertices.

(5) Each span of a B-spline curve of order *K* lies within the convex hull of its *K* associated vertices. An illustration of this concept is shown in figure 10, where the B-spline curve would lie in the shaded

area.

As previously mentioned, an advantage of the B-spline is the resultant data compression. Rogers and Satterfield [1] present an example wherein they successfully represented the forebody of a U.S. Navy ammunition ship using only a five by eight polygonal net with two-dimensional uniform B-splines.

2.3. Surface B-Splines

Nearly all systems developed for computer-aided sculptured surface design have defined the surface in terms of a net of lines. Typically, a series of B-spline curves in parallel planes is intersected with another series of curves in planes perpendicular to the first. The result is not a true surface definition, but definition of a net of lines lying on the surface. To define any point not lying on a net line requires an interpolative approximation.

The program which is the subject of this study uses a true surface algorithm rather than such a network of two-dimensional representations. The surface created is a mosaic of surface patches, just as the two-dimensional curve was a piecewise polynomial curve, and the surface, like the curve, is everywhere continuous with continuity of the derivatives to a degree dependent upon the order of the basis functions. Control is achieved through manipulation of a *net* of control vertices, with the

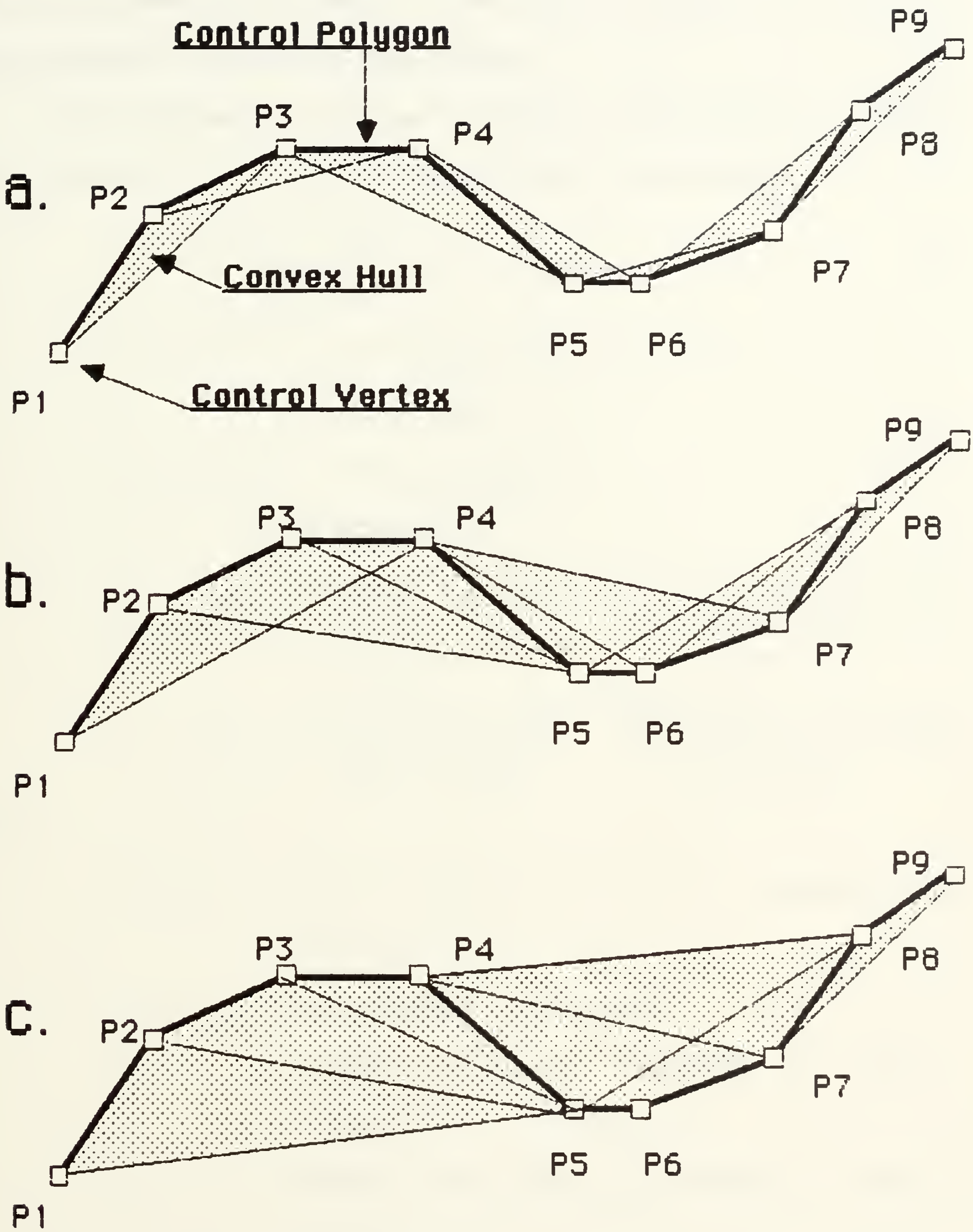

Fig. 10. The convex hull of a third order (a), fourth order (b) and fifth order (c) polygon.

control graph being analogous to the control polygon of the two-dimensional case. The control graph, as well as the surface, is three-dimensional.

The surface (or tensor-product) B-spline is defined very similarly to the two-dimensional spline as:

$$Q(u,w) = \sum_{I=1}^{n} \sum_{J=1}^{m} B_{I,J} N_{I,K}(u) M_{J,L}(w) \qquad (5)$$

$$N_{I,1}(u) = 1 \text{ if } x_I \leq u < x_{I+1}$$
$$= 0 \text{ otherwise}$$

$$M_{J,1}(u) = 1 \text{ if } y_J \leq u < y_{J+1}$$
$$= 0 \text{ otherwise}$$

$$N_{I,K}(u) = \frac{(u-x_I)N_{I,K-1}(u)}{x_{I+K-1}-x_I} + \frac{(x_{I+K}-u)N_{I+1,K-1}(u)}{x_{I+K}-x_{I+1}}$$

$$M_{J,L}(w) = \frac{(w-y_J)M_{J,L-1}(w)}{y_{J+L-1}-y_J} + \frac{(y_{J+L}-w)N_{J+1,L-1}(w)}{y_{J+L}-y_{J+1}}$$

where

$Q(u,w)$ = $[x(u,w), y(u,w), z(u,w)]$, surface data points,

u,w = parametric variables,

n = number of control vertices in the *u* direction,

m = number of control vertices in the *w* direction,

$B_{I,J}$ = control graph points,

K = order of the surface in the *u* direction,

L = order of the surface in the w direction,

$N_{I,K}(u)$ = basis functions in the u direction,

$M_{J,L}(w)$ = basis functions in the w direction,

x_I = elements of the knot vector in the u direction,

and

y_J = elements of the knot vector in the w direction.

The computer program developed for this study implements a very direct approach to surface calculation, computing all basis functions initially, then performing the summation of equation (5). A more efficient algorithm, based on the work of Cox and de Boor, is presented in appendix C-27 of Rogers and Adams [14].

The surface B-spline is directly analogous to the two-dimensional B-spline in virtually every way, having very similar properties extended into three dimensions. For example, just as the effect of moving a single control vertex was seen in two dimensions in figure 4, the effect in three dimensions is shown in figure 11. (The effect extends toward the back of the picture the same as toward the front, although some of that effect is lost to view in this illustration due to depth cueing.)

One difference between the two and three-dimensional B-splines is the effect of a repeated knot vector. A

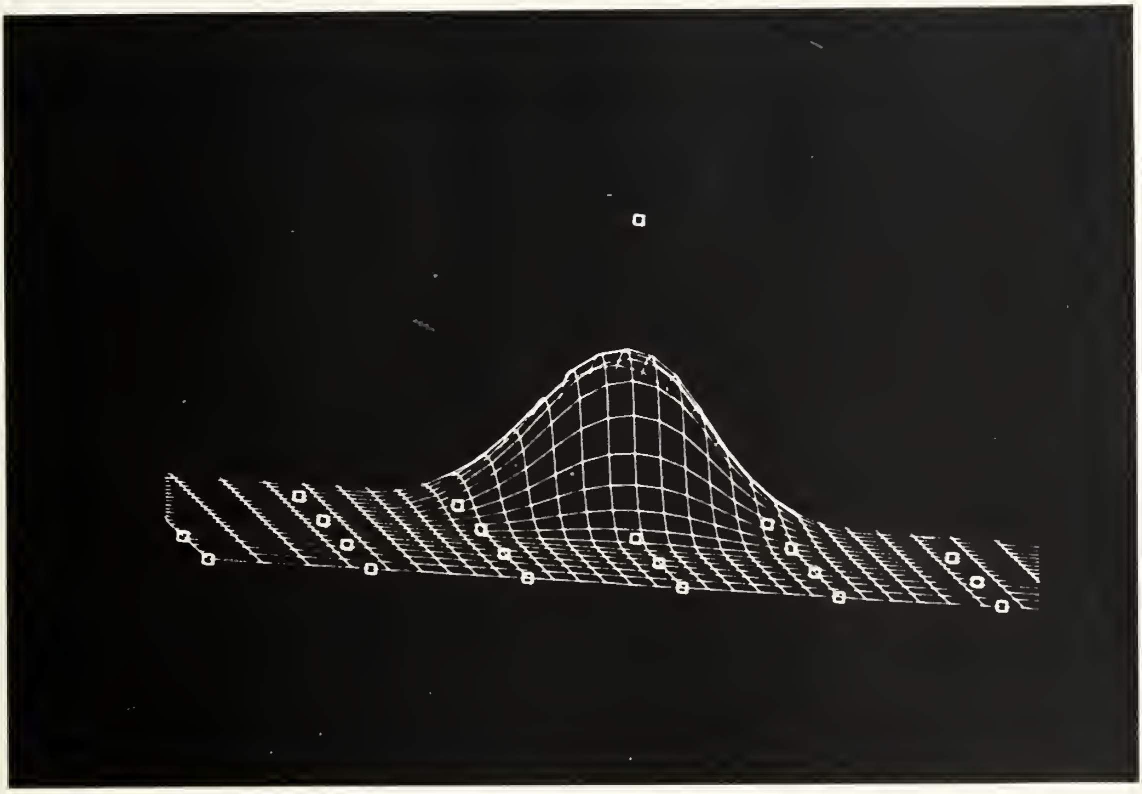

Fig. 11. Localized effect of moving a single control vertex in a surface B-spline.

knuckle can be introduced into a two-dimensional curve by use of a repeating knot value in the knot vector. The same is not true with a three-dimensional surface, unless the knuckle is desired across the entire surface, since the same knot vector is used for all sets of vertices in the *u* (or *w*) direction. Repeating a knot for one or two positions in the control graph will result in repeated knot values for the corresponding positions in all sets of vertices in the net. For this reason, knuckles will be introduced into the surface in this program only by colocating vertices. Multiple vertices will not indicate repeated knot values. This concept is dealt with in more

depth by Rogers and Satterfield [1,17].

2.4. Nonuniform Rational B-Splines

The B-spline algorithm implemented in this program is a still more general case of the surface B-spline, the nonuniform rational B-spline (NURB). Both non-uniformity and rationality extend the power of the algorithm, as explained by Tiller [18].

A limitation of nonrational B-splines is the inability to accurately represent circles, conics, and quadric primitives. The rational B-spline can do so. Rationality refers to the addition of a fourth coordinate. Given a point in three-dimensional Euclidean space, $P=(x,y,z)$, a corresponding point is now defined in four-dimensional space, $P^H=(Hx,Hy,Hz,H)$. The H is the homogeneous coordinate and is constrained to be non-zero. By adding a homogeneous coordinate to the definition of each control vertex, the effect of that vertex is effectively scaled. Thus the B-spline algorithm now becomes:

$$Q^H(u,w) = \sum_{I=1}^{n} \sum_{J=1}^{m} B^H_{I,J} N_{I,K}(u) M_{J,L}(w). \qquad (6)$$

Application of the rational B-spline to the development of surfaces is further dealt with by Tiller.

As alluded to above, the knot vector spacing can also be used in some cases to control the surface shape. Thus the algorithm implemented in this program will allow

nonuniform knot vectors.

Because of their tremendous power and flexibility, nonuniform rational B-splines have been recognized now as an IGES standard for curve and surface definition [19]. The Alpha_1 project at the University of Utah Computer Science Department, working with Riesenfeld, has done much work in exploring the potential of NURB's [13].

On the other hand, manipulation of the homogeneous coordinates and of the knot vector does not produce intuitively predicted results. Since this negates one of the greatest advantages of the B-spline for interactive modeling, the user of the computer program developed in this work has not been given direct control over these variables. Uniform knot spacing, and a homogeneous coordinate of one, are default values for interactive modeling. By including these variables in the algorithm, though not under user control, the potential of utilizing nonuniform rational B-splines in an inversion algorithm (discussed in chapter 6) is retained.

CHAPTER 3

SPLINE: A B-SPLINE SURFACE MODELING PROGRAM

The end product of this study is a computer program for the Evans and Sutherland PS 300 computer graphics workstation which facilitates the design and representation of sculptured surfaces using a surface B-spline. This program is known as SPLINE.

3.1. Hardware

The PS 300 is a distributed graphics system, employing its own microprocessor for the processing of display-related data and relying upon a host computer for other operations. The host used in the development of this program is the PDP-11/44 with an RSX-11M+ operating system. The computer is the University of Washington's Mechanical Engineering Department computer and is set up on a time sharing system. Memory for execution of individual programs is limited to 64 kilobytes, with extension to 128 kilobytes possible by specifying an *ID* flag during compilation. This flag overrides the reservation of memory normally restricted to system housekeeping tasks.

The PS 300 station which was used is the single-user version of the Evans and Sutherland graphics workstation. Properly designated the PS 330, this unit is the most basic system of the PS 300 family and is commonly referred

to by the family designation. The unit upon which this program was developed was running *PS Version 8* firmware, a preliminary version of the operating system and proprietary programming language. The PS 300 workstation is pictured in figure 12. As can be seen there, the workstation is comprised of the control unit, display screen, keyboard, data tablet, and control dials.

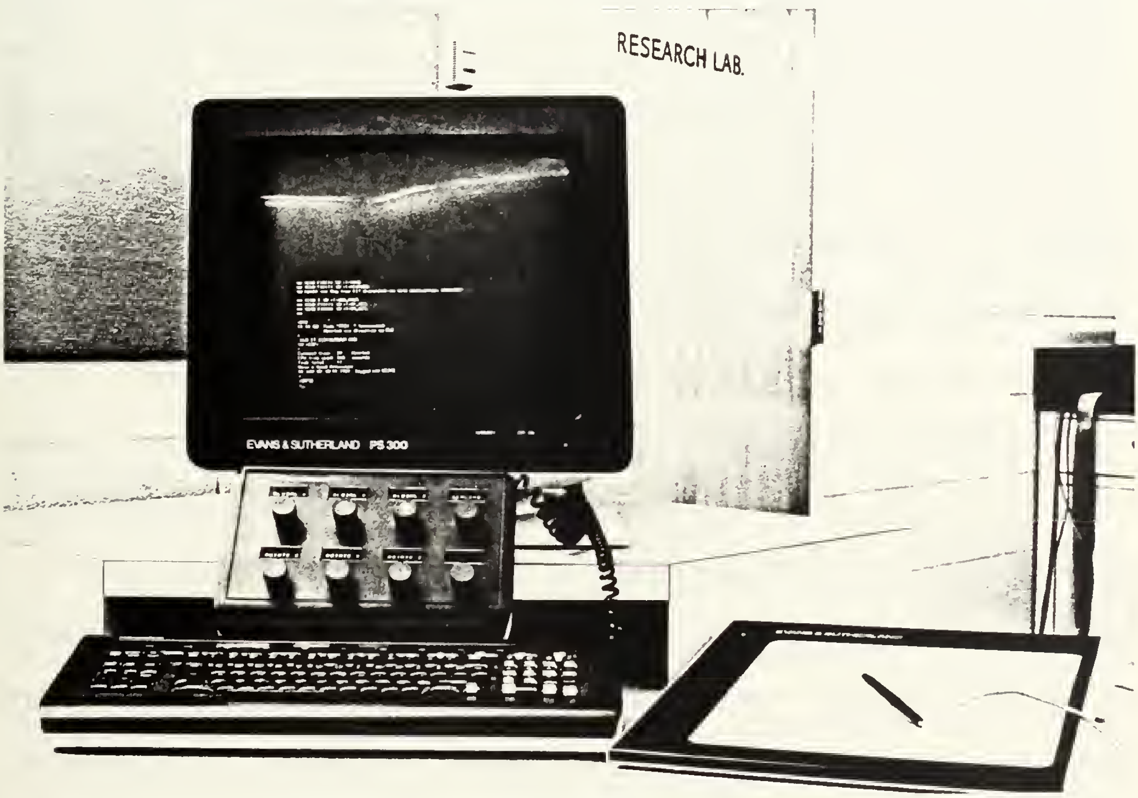

Fig. 12. PS 300 workstation.

The control unit consists of a Graphics Control Processor, mass memory, and the Display Processor. The Graphics Control Processor is a Motorola 68000 microprocessor with 24-bit address space and 256 kilobytes of local memory for the graphics firmware. A full megabyte is available in mass memory, with 16-bit

communication with the Graphics Control Processor and 32-bit communication with the Display Processor, for very fast display access.

The PS 300 display screen is a monochrome vector refresh display with 8192 by 8192 resolution. White lines and characters are displayed on a black screen with sixty-four levels of intensity. While solids with color shading cannot be displayed, the fine scaling of intensity allows display of wire frame models which are unusually easy to interpret. Through a feature called *depth cueing*, the intensity of lines is gradually decreased as their distance behind the plane of the screen is increased. This fading with depth helps to give the perception of three dimensions while viewing a model on the two-dimensional screen.

An additional benefit of depth cueing is that wire frame lines in the background can be clipped from view, eliminating confusion with foreground lines. This is especially important since the PS 300 does not provide hidden line removal. A comparison of figures 13 and 14 will illustrate this advantage.

The keyboard doubles as an input device for the PS 300 and a terminal for the host computer. Features include a full keyboard and twelve programable function keys with LED labels. A separate pad with eight programable control dials also features LED labels.

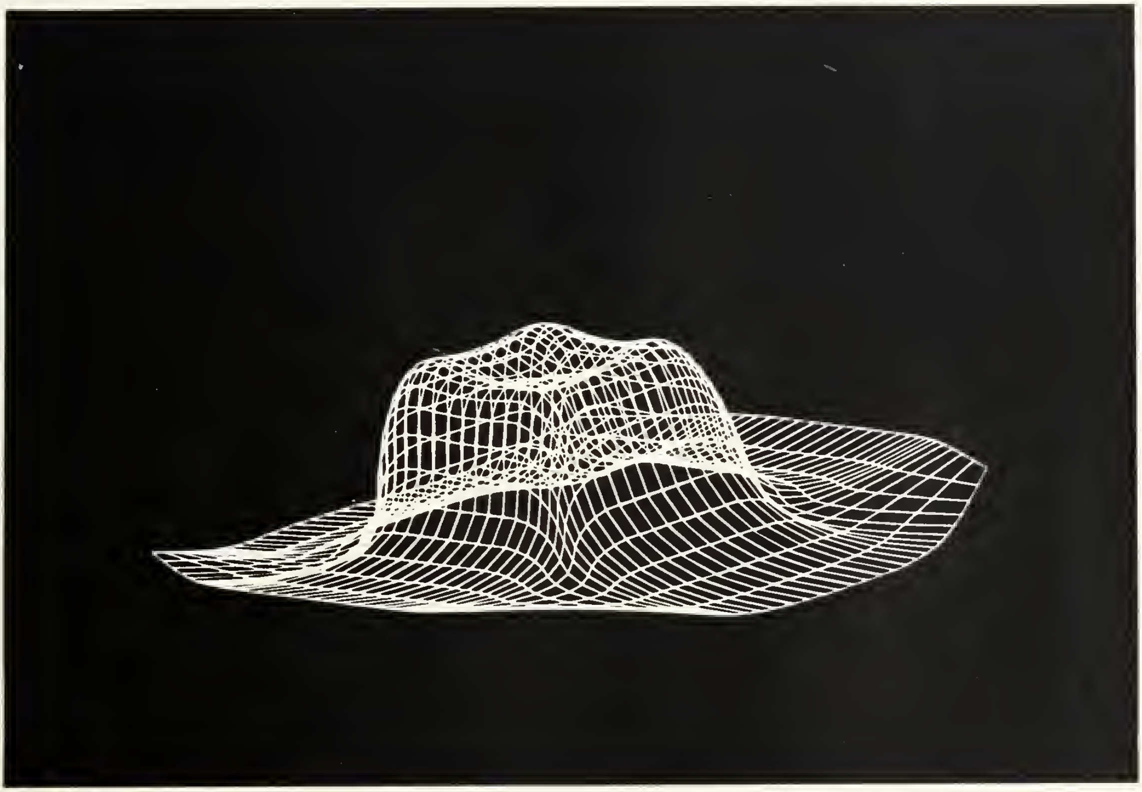

Fig. 13. Wire frame model without depth cueing.

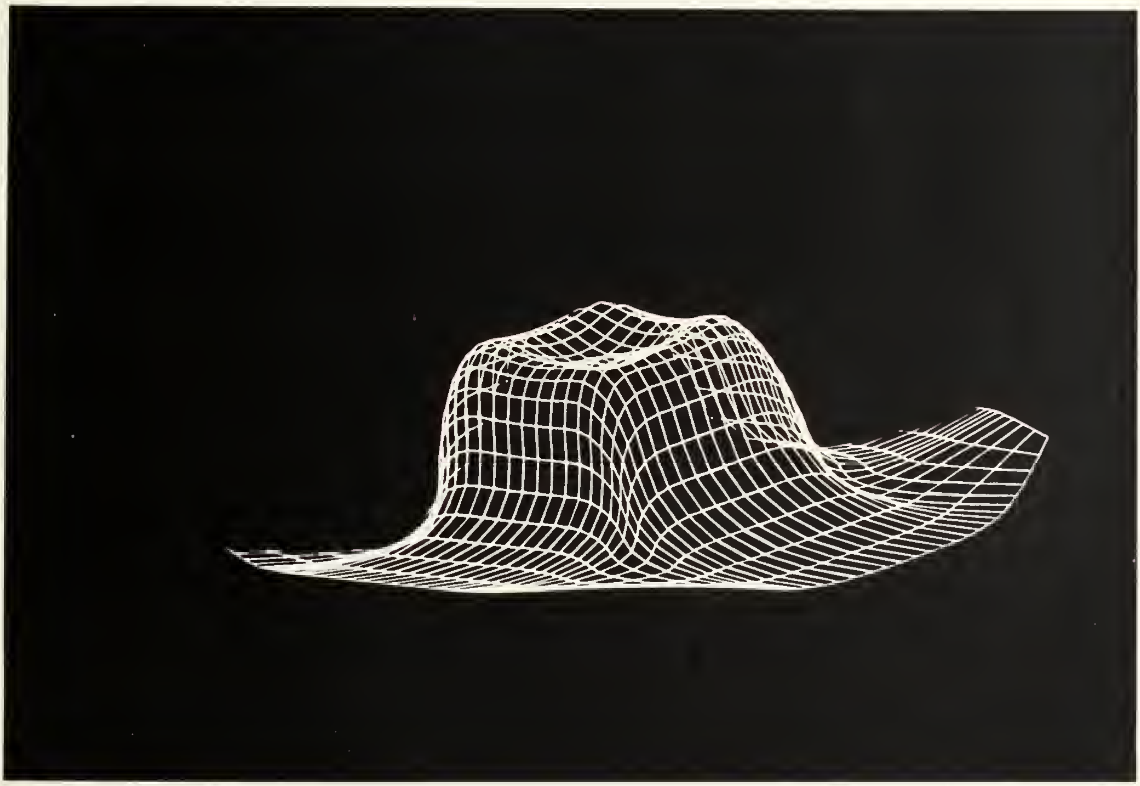

Fig. 14. Wire frame model with depth cueing.

The coordinate position of a stylus on the data tablet is converted to a digital equivalent for the positioning of a cursor on the screen. This can be programmed for menu selection or picking of points or lines on a displayed model.

The PS 300 performs graphical manipulations and the associated mathematical transformations independently from the host. The transformations and matrix concatenations are accomplished without the user having to program the matrix arithmetic.

While graphics manipulations can be performed without host interaction, communication between the host computer and the PS 300 is essential for virtually all applications. The PS 300 has no local storage capability, so its program must be stored on the host system and passed to the PS 300's memory as ASCII data. The programming language available on the PS 300, which is unique to Evans and Sutherland systems, is highly efficient for graphics display and manipulation, but other mathematics and data processing which would be straightforward in another high level computer language can be very difficult. The programmer must make a division of the graphics application into graphics processing and analysis functions in accordance with each computer's strengths. An intelligent division can minimize communication between the computers, but

substantial communication will remain necessary.

Communication between the PS 300 and the host is accomplished through a 9600 baud interface. Data transfer protocol is established through the PS 300 Host-Resident I/O Subroutines (PSIO's). These are host-resident FORTRAN subroutines which provide two-way communication between the computers.

3.2. Programming the PS 300

The PS 300 has its own high level ASCII command language with English-like commands, detailed in the Evans and Sutherland manual, PS 300 Computer Graphics System [20]. Structured objects are created and accessed by name with commands such as VECTOR_LIST (to create an object as a list of vectors), TRANSLATE, ROTATE, and SCALE (to perform the named operations).

Models are created as hierarchical groupings of graphical data, transformations applied to the data, and attributes such as intensity and level of detail. Primitive objects are defined and used to build more complex objects. For example, an automobile wheel might be defined by means of a vector list, then instances of that wheel translated and rotated to create four wheels and position them relative to a separately defined body, forming a representation of a car. A sample display tree is shown in figure 15. The three basic forms of display tree nodes can be seen there: data nodes (such as vector

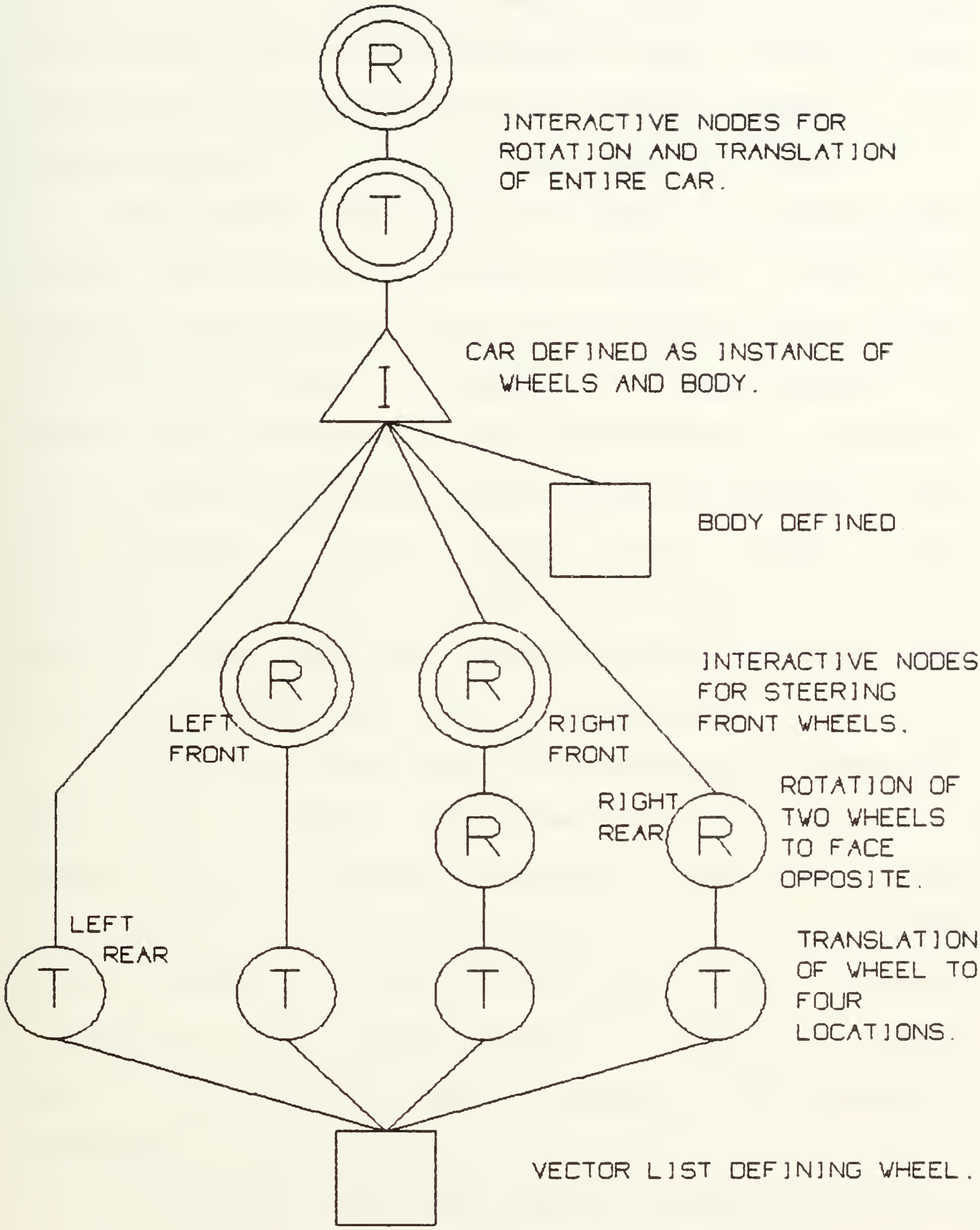

Fig. 15. Display tree for an automobile.

lists defining objects), operation nodes (such as translations and rotations), and instance nodes. Instance nodes group other elements together under a single name. Applying a transformation to an element results in the transformation of all subordinate elements as well.

Manipulation and control of models is accomplished through the creation of function networks. Values from interactive devices such as function keys or control dials are processed through user-designed function networks and applied to interaction points in the model's structure. The function networks are combinations of connected black box functions. These function blocks accept data, manipulate it according to predefined rules for the given function, and output new values to another function input or to interaction points in the display tree.

For example, ADDER might be defined as an instance of the function, F:ADDC, which sums any number sent to its first input and a constant value held on its second input. If the function keys are connected to input one of ADDER and the integer "2" is sent to the second input, whenever a function key is pressed ADDER will output an integer equal to two plus the number of the key. The programming statements to accomplish this would be:

```
ADDER:=F:ADDC; {Defines ADDER as an instance of
                the function, ADDC.}
SEND FIX(2) TO <2>ADDER;{Puts the value, 2, on
```

```
                                        ADDER's constant input.}
          CONNECT FKEYS<1>:<1>ADDER;{Connects function key
                                        output   to   ADDER's
                                        trigger input.}
          CONNECT ADDER<1>:[input and name of the  element
                                 to  which  the sum  is  to  be
                                 sent]
                              {Routes    output   to    desired
                              location.}.
```

Available functions include mathematical operations, Boolean logic operations, routing functions, and data conversion.

3.3. SPLINE

Figure 16 presents an overview of SPLINE, showing the interrelationship of major hardware and software components. Items marked by dashed lines have not yet been implemented, but provision for future addition of these features has been made within the program architecture and source coding. All programming on the PDP-11 was performed in FORTRAN language and all programming on the PS 300 was in the language described above. A more detailed description of the programs is contained in appendix A and the full FORTRAN and PS 300 program listings are contained in appendices B and C, respectively.

Program execution will vary slightly depending on

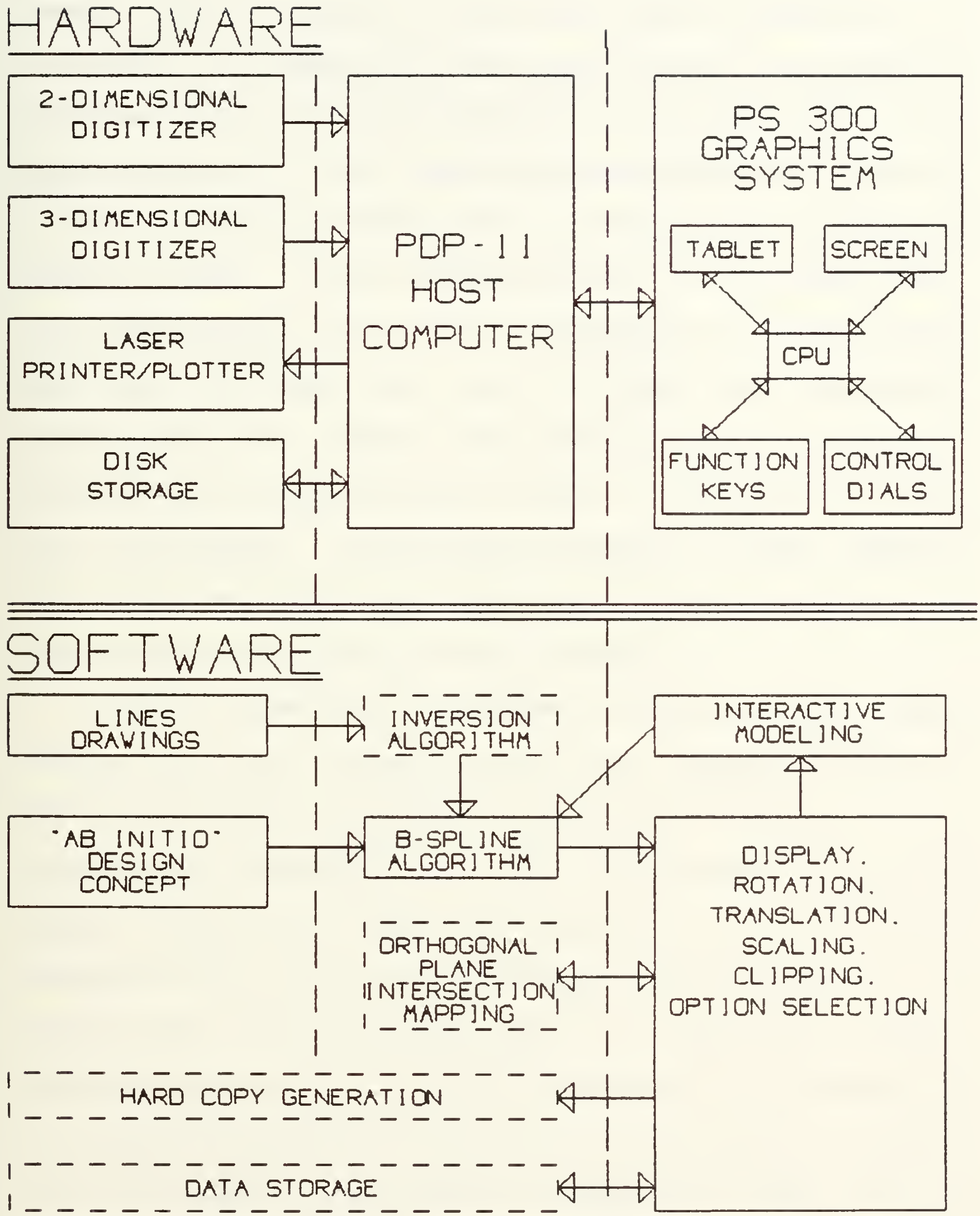

Fig. 16. Overview of SPLINE.

whether it is desired to perform a freeform *ab initio* design or to represent a previously known surface with a B-spline. In respect to this distinction, Wu, Abel and Greenberg [21] distinguish between *shape design* and *shape representation*. They identify shape design as having a primary concern of conceiving, displaying and modifying complex forms. Shape representation, on the other hand, is characterized by a need to describe or approximate existing surfaces for the purpose of graphical manipulation to enhance visualization or for geometric calculation to permit quantification. SPLINE will allow both approaches, shape design (freeform surface design) and shape representation (modeling a B-spline surface to approximate a previously known surface).

If it is desired to represent a known surface, data points on that surface may be input through a two or three-dimensional digitizer or by direct creation of an appropriately formatted data file. The program will allow screen display of these known points for direct visual comparison with the B-spline surface during modeling. In either case, the user must select the number and initial relative position of the control vertices (by designating the size and shape of a planar matrix of points), as well as the order of the surface in respect to each of the parametric directions. Provision has been made for future addition of an inversion algorithm, which would calculate

initial control vertex locations to approximate known surface data.

The control vertices, the B-spline surface and known surface points may be displayed on the screen in any combination through function key menu selections, and can be rotated, translated, or scaled in real time through use of the control dials. The B-spline surface is formed and modified by moving the control vertices about the screen. Vertex movement is communicated to the FORTRAN program, which recalculates the B-spline surface and sends a new vector list to the PS 300.

Vertices are selected for movement by using the stylus on the data tablet to locate a cross hair on the screen over the desired vertex for picking. Individual vertices or entire rows or columns of vertices may be selected for movement. Movement is defined by manipulation of labeled control dials.

Through the use of z-clipping, narrow sections of the model may be viewed without display of the remainder of the structures. This enables very accurate comparisons of any combination of display elements to be made very easily.

B-spline surfaces are displayed as nets of constant parameter lines. Provision has been made for future implementation of orthogonal plane intersection mapping, which would enable display of more traditional

representations, such as waterlines and transverse section lines, yet based on the B-spline surface.

3.4. Operating System Limitations

At this time the functionality of the program is severely limited by what appears to be a problem in the operating system or FORTRAN compiler of the PDP-11 computer.

During programming, it was noted that previously trouble-free subroutines would often fail after adding seemingly unrelated lines of FORTRAN program. This was found to be caused by variables changing value during program execution, but not under program control. Efforts to eliminate or program around this problem were not successful. Discussion with the Mechanical Engineering Department computer system manager and with Professor Ganter of the thesis committee revealed that the PDP-11 suffers from known operating system and compiler flaws which cause addressing problems resulting in similar symptoms.

While the exact manifestation of the problem varied with the state of program development, the current symptoms include failure of the host-resident input/output library subroutines (PSIO's) after execution of the subroutine which calculates the B-spline surface. This precludes display of the B-spline surface. A slightly earlier version of the program will allow the B-spline

surface to be calculated and displayed, but at an indeterminate time afterward will refuse to answer further calls to the communication subroutines. This usually prevents modification of the B-spline surface once displayed.

Troubleshooting efforts have included careful review of the program for any definition of array elements outside of dimensioned array size, successful operation of each subroutine independently, and attempts to reestablish communication immediately after failure. The problem was further clouded by consultation with the Evans and Sutherland Software Technical Support Department [22], which indicated that use of the PSIO subroutines has been discontinued in favor of more reliable communication packages and that there are documented and undocumented flaws in the preliminary PS 300 operating systems such as the one used for development of SPLINE. Nevertheless, as will be shown in chapter 5, the project results are sufficiently successful to demonstrate the viability of such a computer program.

CHAPTER 4

OPERATING INSTRUCTIONS

4.1. Preparing Known Surface Data

If it is desired to display known points from a surface for purposes of modeling a B-spline representation of the known surface, this data must be properly formatted before entering SPLINE. The data will automatically be correctly formatted by the digitizing program if it is prepared through use of the two-dimensional digitizing program, DIGIT, or the three-dimensional digitizing program, DIGIT3. However, the first parameter in the global header may need to be adjusted (see below).

The PREPS input format is used for the preparation of known surface data. PREPS is a data pre-processor used at the University of Washington for conversion of data describing the geometry of an object into a PS 300 syntax-compatible data file. The input file format for PREPS has been adopted as the standard for data input to SPLINE in order to facilitate the use of SPLINE with other PREPS-compatible programs, such as DIGIT and DIGIT3.

The PREPS input file format is detailed in Calkins' laboratory book [23]. A summary follows.

The data file format prescribes a global header followed by a series of subheaders, each with an associated set of substructure data. SPLINE will allow

the display of individual data points represented in the data file, as well as lines sequentially connecting the data points within a substructure. Division of a structure into substructures will provide the most meaningful display if the substructures correspond to logical divisions in the actual structure. A common choice is the representation of profile lines or section contours as individual substructures.

4.1.1 The global header. The global header is the first group of data in the PREPS-format data file and contains information affecting the display of all substructures. SPLINE reads the data with a list-directed format, so the exact format of the data in the file is not important. However, the correct sequence must be maintained.

The global header contains five parameters:

1. ZNORM the normalization factor for all data point coordinate values. This parameter should be set to the largest absolute value of any x, y, or z coordinate.

2. INSTRT not used by SPLINE. Should be set to 1.

3. XBODY global translation of all data in the input file along the x axis. SPLINE assumes that the

normalized data runs from 0 to 1. If the origin is not actually at one end of the structure, use this to move the structure accordingly for the most convenient display.

4. YBODY the same as XBODY, for the y coordinate.

5. ZBODY the same as XBODY, for the z coordinate.

<u>4.1.2. Substructure headers.</u> Each set of substructure data is comprised of seven subheader parameters immediately followed by the substructure data. The subheader parameters are:

1. NPTS the number of data points in the substructure.

2. NPTS2 the number of repetitions of this substructure to be displayed. Normally set to 1. If several identical substructures are to be displayed, this can be set to the number or repetitions and ICOORD to the coordinate which varies between repetitions. The series of varied coordinate values is then entered for that coordinate

in the data, one per substructure repetition. Examples appear in [23].

3. ICOORD the coordinate which remains constant within a substructure, or varies between repeated substructures.

ICOORD = 1, x coordinate
= 2, y coordinate
= 3, z coordinate
= 0, no repetition.

4. XTRAN translation of the substructure, in non-normalized coordinates, along the x axis.

5. YTRAN the same as XTRAN, for the y axis.

6. ZTRAN the same as XTRAN, for the z axis.

7. IREF1 the plane about which mirror image reflection occurs.

IREF1 = 1, yz plane
= 2, xz plane
= 3, xy plane
= 0, no reflection.

For modeling purposes, it is usually best to select no

reflection, as discussed in chapter 5.

4.1.3. Substructure data. Each subheader is immediately followed by the corresponding substructure data. The data is not paired for each point, but is listed with all x coordinates first, all y coordinates next, and all z coordinates last. If ICOORD is not set to 0, only one value is entered for the constant coordinate, except as noted under NPTS2, above.

A sample data file is presented below, including a global header and subheader with substructure data. Comments contained in brackets are not part of the data file.

```
3.                      [ZNORM]
1     0     0     0     [INSTRT, no x, y, z translation.]
6     1     1           [First subheader; 6 points, one
                        substructure, x coordinate
                        constant for substructure.]
0     0     0     0     [No translation, no reflection.]
0.585E+00               [Constant x coordinate.]
-0.255E+00   -0.202E+00   -0.191E+00
-0.157E+00   -0.861E-01   -0.0        [6 y values.]
-0.962E+00   -0.711E+00   -0.586E+00
-0.482E+00   -0.370E+00   -0.362E+00 [6 z values.]
7     1     1     [Begin second substructure header.]
0     0     0     0
```

0.256E+01 [Begin second substructure data ....]

4.2. Preparing the Computer System

In order to run SPLINE, both the PS 300 computer and the host PDP-11 must be operational. Before logging on to the host, it is suggested that the PS 300 be readied.

SPLINE requires PS Version 8 firmware on the PS 300. The version can be checked by removing and inspecting the disk installed in the lower right-hand corner of the PS 300 cabinet. If it is necessary to reload the firmware, use the following sequence:

1. Turn on the power switch located on the display screen.
2. Turn off the power switch located on the upper right-hand corner of the PS 300 cabinet.
3. Insert the PS Version 8 disk.
4. Turn on the cabinet power switch.
5. Wait for the DUAL LINE READY message on the display screen, which should appear in about five minutes.

It will be necessary to log on to the PDP-11 in the normal manner in order to run SPLINE.

4.3. SPLINE Operating Instructions

After logging on to the PDP-11 computer, execute the program by entering "RUN SPLINE." A series of

"Working..." statements will begin to appear on the screen as the PS 300's program file is read from the disk and passed from the host computer to the PS 300 under FORTRAN program control.

4.3.1. Initial formatting selections. After loading the PS 300 program, the LED indicators on the PS 300 keyboard and control dials will contain illuminated labels and a series of questions will be presented on the PDP-11 terminal screen.

The initial question is, "Do you wish to display a known surface from a PREPS-format file?" Answer "Y" or "N", for yes or no. If a "Y" is entered, the user will be prompted for the name of the data file containing the PREPS-formatted data. See "Preparing Known Surface Data" above.

The next series of questions asks the user to specify the size of the initial control vertex matrix (i.e. 8,5) and the length-to-width ratio of the matrix. The maximum matrix dimensions which can be selected are (10,8). The minimum dimensions are (5,5). The initial control vertices will be positioned in the xy plane at z=0, with the user's first response representing the number of control vertices in the x direction and the second response controlling the number of vertices in the y direction. The selected length-to-width ratio will determine the relative spacing between control vertices in

the x direction (length) and that in the y direction (width). Figure 17 illustrates the initial control vertex matrix resulting from selecting a matrix size of (8,5) with a length-to-width ratio of 3.

The user is finally prompted to select the order of the B-spline in the longitudinal (u) and transverse (w) directions. Third or fourth order B-splines may be selected. After entering the selections, small cubes representing the control vertices will begin to appear on the PS 300 screen.

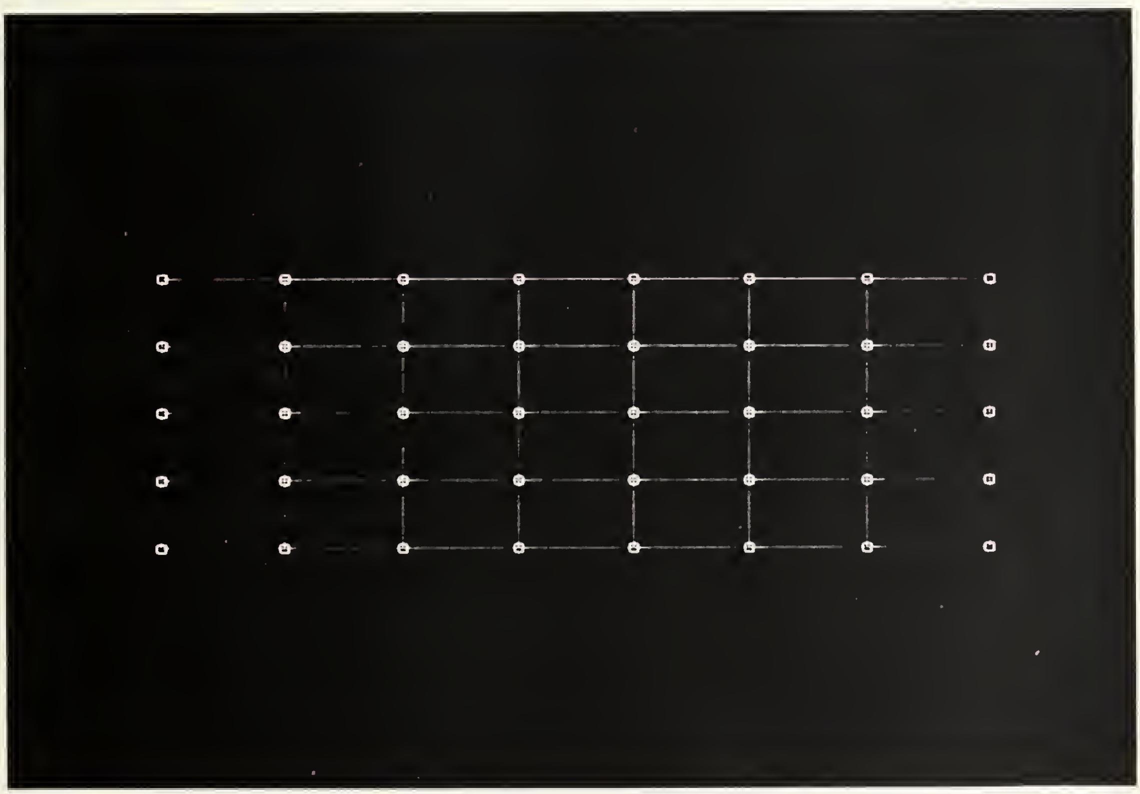

Fig. 17. An 8 X 5 matrix of control vertices with a length/width ratio of 3.

<u>4.3.2. Enabling the PS 300 keyboard.</u> After all initial formatting selections have been made (in response

to display screen prompts), the following message will appear on the screen:

```
****************************************
**     PRESS <SHIFT-LINE/LOCAL>      **
**      ON THE PS-300 KEYBOARD       **
**        FOLLOWED BY <TERM>         **
****************************************.
```

The LINE/LOCAL and TERM keys are located in the keypad at the left side of the PS 300 keyboard. Press the LINE/LOCAL key while holding down the SHIFT key. This enables the twelve function keys at the top of the keyboard. Press the TERM key if it is desired to remove the text display from the screen.

4.3.3. Control dials. The control dials are labeled by LED indicators and have the following functions:

GLOBAL X moves the model in the x direction in the screen coordinate system (left and right).

GLOBAL Y moves the model in the y direction in the screen coordinate system (up and down).

GLOBAL Z moves the model in the z direction in the screen coordinate system (in and out of the screen). As the model moves into the screen (away from the viewer) past the z=0 plane, depth cueing will cause portions of the model to fade, creating the illusion

	of depth. Depth cueing can be disabled by moving the entire model forward of the z=0 plane (toward the viewer) with the GLOBAL Z dial.
SCALING	scales the entire display, allowing the user to examine details of the display more closely.
ROTATE X	rotates the model about the screen's x (horizontal) axis.
ROTATE Y	rotates the model about the screen's y (vertical) axis.
ROTATE Z	rotates the model about the screen's z axis (turning it within the plane of the screen).

<u>4.3.4. Main menu.</u> The LED labels above the keyboard function keys present the main menu, from which SPLINE functions are selected. The following options are available:

F1 VERTICES is a three-way toggle controlling the display of the control vertices. Vertices are displayed on the screen as small cubes. Pressing the F1 key selects one of the following options, in sequence:

(1) vertices displayed,

(2) vertices displayed with straight

lines connecting them in matrix position order,

(3) vertices not displayed.

F2 B-SPLINE toggles display of the B-spline surface on and off. When the F2 key is pressed, the B-SPLINE label above the key will blink while the host computer calculates a vector list for the B-spline surface and sends it to the PS 300. When the surface is displayed, the label will cease blinking.

F3 KNOWN PT is a three-way toggle controlling the display of known surface points contained in a PREPS-format file. Points are displayed as asterisks (*'s). Pressing the F3 key selects one of the following options, in sequence:

(1) known points are displayed,

(2) known points are displayed and points within substructures are connected in sequence by straight lines,

(3) known points are not displayed.

F4 INTERACT toggles SPLINE into and out of the

interactive modeling mode, giving the user the ability to manipulate control point positions. When the F4 key is pressed, several function key and control dial labels change to indicate new assigned functions or availability of previously assigned functions. Interactive modeling is discussed in section 4.3.5 below.

F5 CLIP toggles z-clipping on and off. When z-clipping is enabled, only portions of the model lying within a narrow band parallel to the plane of the screen can be viewed. This allows slices or sections of the model to be viewed without confusion caused by lines in front of and behind the section being examined. When the F5 key is pressed, the screen will be blanked and a CLIPPING label will appear above the eighth control dial. Turning the dial slightly will cause a slice of the model to be displayed. Adjusting the CLIPPING dial will vary the depth of the slice which is displayed. By turning the GLOBAL Z

dial, it is possible to move the model through the clipping window, allowing the viewing of any portion of the model. Note that it is possible to move the model entirely outside of the clipping window, in which case it cannot be viewed.

F6 REFLECT has not been implemented. When implemented, it will toggle on and off the display of a reflection of the model about its XZ plane. This label will not be displayed during interactive modeling, indicating that the reflection function is not available at that time. (This prevents the updating of the reflected surface from slowing down the computer's response to interactive inputs during modeling.)

F7 XY has not been implemented. When implemented, this key will toggle on and off the display of intersections of the XY plane (body-fixed coordinates) with the B-spline surface. On a ship model, this corresponds to the display of

waterlines. This function is not available during interactive modeling.

F8 XZ has not been implemented. When implemented, it will function similarly to the F7 key, described above, and will display lines corresponding to a ship's buttocks lines.

F9 YZ has not been implemented. When implemented, it will function similarly to the F7 key, described above, and will display lines corresponding to transverse sections of a ship hull.

F10 I/O selects the input/output menu. Pressing this key causes the labels above the function keys to display a menu of input and output functions described in section 4.3.6, below.

F11 RESET is a three-way toggle which resets the displayed model to each one of three orthogonal views, in sequence. Pressing the F11 key resets the scale factor and global translations to those of the initial display and orients the XY, XZ, or YZ plane of the

model parallel to the screen. Repeatedly pressing the key will cycle the display through the three orientations. This function key is particularly useful when orienting the model for display of slices with the CLIP function.

F12 QUIT stops execution of the program and initializes the PS 300, clearing the screen. If it is desired to use the keyboard to communicate with the host computer at this point, press the TERM key to return the screen to a text display mode, then press the LINE/LOCAL key, followed by RETURN. The host computer's prompt should now be visible.

4.3.5. Interactive modeling. The interactive modeling mode is entered and exited by pressing the INTERACT (F4) key. This mode is the heart of the computer program, allowing the user to select and relocate control vertices in order to mold a B-spline surface.

When the F4 key is pressed, the labels above the three global translation control dials (GLOBAL X, GLOBAL Y, and GLOBAL Z) change to VERTEX X, VERTEX Y, and VERTEX Z and begin to blink. These dials can now be used

to move the selected vertex in the indicated direction *in the body-fixed coordinate system*. If it is necessary to move the entire body, the global translation functions can be restored by toggling the interactive modeling function off with the F4 key. The blinking labels will remind the user, while in the interactive modeling mode, that dial movement will relocate vertices rather than translate the entire model, in order to prevent inadvertent movement of vertices.

Individual control vertices are selected for movement with the stylus and data tablet. When the tip of the stylus is placed near the surface of the data tablet, a tracking cross will appear on the screen. By moving the stylus lightly over the data tablet surface, the tracking cross can be positioned over the desired control vertex. The vertex is then selected by pressing the stylus momentarily against the data tablet surface while keeping the cursor positioned over the control vertex. This activates a switch in the tip of the stylus, causing the data tablet to report the stylus position, leading to identification of the selected vertex. The vertex can then be moved into the desired position with the vertex translation dials. If it is necessary to reorient the model in order to obtain a better view or reduce ambiguity for the selection or location of control vertices, this may be accomplished with the rotation dials or RESET

function key, which remain operational while in the interactive modeling mode.

When a control vertex is relocated, the movement is reported in increments to the host computer, which recalculates vector lists defining the B-spline surface and the net of lines connecting the vertices. If the vertex-connecting lines or B-spline surface are displayed while in the interactive modeling mode, some delay will be noted between the time of vertex movement completion and the corresponding change in the model. This delay is due to the time required for the host computer to communicate the updated vector lists to the PS 300. The delay will be about two to three seconds for update of the vertex-connecting lines and about ten seconds for update of the B-spline surface.

When the interactive modeling mode is activated, the label above the F7 key will read "PICK ROW." Pressing this key will alter the control vertex selection function such that selections made with the stylus will affect an entire row or column of vertices at a time. The column or row is selected similarly to the selection of an individual vertex, except that the cursor is placed over one of the vertex-connecting lines in the row or column, instead of over the individual vertex. Turning the vertex translation dials will then move the selected row or column of vertices in the desired direction while

retaining the orientation of the individual points within the row or column to each other. The user can return to selection of individual control vertices by pressing the "PICK PT" (F8) key.

After moving a vertex it will occassionally be noticed that the vertex-connecting lines will not quite reflect the new position of the vertex. Since the vertex movement is reported to the host computer incrementally, it is possible that a slight additional movement will trigger the update. At other times, a flaw in the communications between the host computer and the PS 300 causes an incremental change to be reported to the host as a very large or indecipherable movement. In that case, the distorted movement value is disregarded by the host program, which results in loss of the true incremental change. While there is no way to reconcile the difference between the control vertex position as viewed on the screen and that held in the host computer's memory, it should be noted that the intersection of the vertex-connecting lines, and not the vertex cube, will indicate the true position upon which the B-spline surface will be calculated.

4.3.6 Input/output menu. The input/output menu is displayed in the function key labels when the I/O (F10) key is pressed. The menu functions have not yet been implemented (with the exception of the F12 key), but when

implemented will offer the following options:

F1	PRT VERT	prints a list of the current vertex locations and of the B-spline surface order selections.
F4	PLT SCRN	plots a drawing of the model, as oriented and displayed on the screen, on the laser printer.
F8	SAV VERT	saves the current vertex location and surface order information to a disk file.
F9	LD VERT	loads vertex location and surface order information from a disk file created with the SAV VERT function.
F12	EXIT I/O	returns the user to the main menu.

CHAPTER 5

MODELING TECHNIQUES AND CASE STUDIES

The B-spline modeling of sculptured surfaces greatly reduces the requirement for the more traditional techniques of surface development and representation reviewed in chapter 1. As was shown there, use of the traditional techniques requires that engineering and drafting skills be accompanied by an artistic touch in order to visualize and fair the surface in three dimensions. This ability is developed by the naval architect, and others who work often with sculptured surfaces, through years of experience and practice.

On the other hand, while the B-spline techniques deemphasize the artistic requirement, it will still be necessary for the designer to develop an intuition for the placement of control vertices which cannot be entirely reduced to a set of mathematical rules without sacrificing the control and flexibility of interactive modeling. The heuristic observations and case studies which follow are offered in order to facilitate the rapid acquisition of that intuition.

5.1. Modeling Tips and Techniques.

5.1.1. Known data. If the surface design is not to be developed *ab initio*, the preparation of a data file of known surface points will be essential. The selection of

such points for inclusion in this data file requires some consideration and advance planning. Primarily, the goal is to select sufficient points to accurately represent the form and boundaries of the surface, yet not so many as to create unnecessary clutter when the known data is displayed.

Two other considerations should be kept in mind during data point selection. First, the practicalities of displaying and visualizing the two surfaces which are to be matched (the known surface and the B-spline surface or control graph) dictate that most of the modeling be performed while viewing two-dimensional sections (or very thin three-dimensional sections). This generally means locating control vertices nearly in the planes of known substructure data, such as the planes of a ship's transverse sections. Greater flexibility will be available in locating the control vertices for maximum surface control if the vertices' general locations are considered before finalizing the file of known surface data. Sections 5.1.3 and 5.1.4 will address the problem of vertex placement.

Secondly, if a symmetrical body such as a ship hull, automobile body, or airplane fuselage is being modeled, it will generally be easier if only one half of the structure is displayed, both of the known surface data and of the B-spline surface and control graph. Modeling two identical

halves requires twice the manipulation as modeling a single half and results in much more difficulty interpreting the wire frame model due to ambiguities between lines in the foreground and those in the background. It is also virtually impossible to exactly duplicate the control graph for the halves of the body through a purely visual comparison. The only drawback to modeling a single half of a symmetrical body is that it is more difficult to ensure fairness at the juncture of the two halves. This problem can be remedied through use of the reflection display function, when that function is implemented.

5.1.2. Surface order. SPLINE offers the choice of two surface orders: third order or fourth order. The order may be specified independently in the u and w directions. As noted in chapter 2, a fourth order curve produces the smoothest lines, but at the expense of some local control and the ability to easily effect sudden changes in curvature. It should also be remembered that a fourth order curve requires three coincident vertices (or rows of vertices) to create a knuckle or chine, while the third order curve requires only two.

The third order surface will normally produce satisfactory results with the least amount of effort. The fourth order curve should be considered where smoothness of lines is an overriding constraint, such as in the

longitudinal lines of some ships and aircraft.

<u>5.1.3. The control graph.</u> Selection of the dimensions of the control graph (i.e., the number of vertices contained) is best accomplished by considering the locations where control points will be required for sufficient control of the surface. The most obvious considerations include:

(1) Vertices will be required to define the boundaries (ends and edges) of the surface.

(2) A vertex or row of vertices will generally be required at each maximum or minimum point.

(3) Particularly sharp curvatures or sudden changes in curvature will generally require two or more control vertices to achieve. Inflection points will require sufficient control vertices on each side of the point of inflection to establish curvature in opposite directions.

(4) Knuckles or hard chines will require two coincident rows of vertices in a third order surface or three coincident rows of vertices in a fourth order surface.

(5) While there is no constraint imposed by the B-spline algorithm nor by the computer program which would dictate that control

vertices within a row or column of the control graph remain in a plane, the practicalities of modeling usually make such placement the wisest choice, as discussed above. In most cases, comparison with known surface points will be most efficiently accomplished if the rows (or columns) of the control graph are parallel to the known-surface substructures. It is not practical to constrain both the rows and the columns of the control graph to planes, but with one or the other so constrained, some effort should be made to keep the other in as close to a planar placement as possible. This will be emphasized by the case study in section 5.3.

After the number of control vertices has been established, required manipulation of the vertices can be minimized by selecting a length-to-width ratio for the initial control graph which approximates that of the body to be modeled. This will be illustrated in the case studies which follow.

<u>5.1.4. Control vertex placement.</u> The approach to control vertex placement which was found to be most useful is to first locate the rows or columns of the control

graph in the desired plane and then to use the CLIP function to compare an individual row or column of the control graph with the nearest substructure of the known surface data. In this way, most of the interactive modeling is performed in the more readily visualized two-dimensional realm. Placing vertices is primarily a matter of keeping in mind the general properties of the B-spline, as innumerated in chapter 2. Those properties are briefly reviewed below.

(1) The B-spline curve does not interpret the control polygon, but is contained within its convex hull, taken *K* vertices at a time. This dictates locating the control vertices slightly outside of the curvature of the desired surface.

(2) The B-spline surface interpolates the corners of the control graph.

(3) While modeling is most efficiently performed in two dimensions, it must be kept in mind that the influence of each control vertex is three-dimensional. This will occassionally require moving a vertex from its apparently proper position in two dimensions to accommodate some sudden change in the surface in the third dimension, or to avoid distorting the

surface by locating a vertex far from the plane of the remainder of its row or column in the control graph.

(4) The slope of the curve at the end points of the control polygon is determined by a straight line connecting the end point with the next point in the polygon. Again, the effect of vertices in the third dimension may cause unexpected deviations.

(5) A straight line is produced by *K* collinear vertices.

(6) A curve tangent to the control polygon is produce by *K-1* collinear vertices.

(7) A knuckle is produce by *K-1* coincident vertices.

5.1.5. Menu selections. The selection of display options from the menu can have a significant effect on the efficiency with which interactive modeling is performed. The requirement for displaying the control vertices is obvious, but it is useful to display the lines connecting the vertices of the control graph as well. Display of the connecting lines prevents confusion concerning the identity of a vertex in respect to its position in the control graph matrix. If two points in a control polygon are inadvertently interchanged, the result can be dramatic anomalies in the B-spline surface.

The CLIP function should be used as much as possible, since accurate depth perception is difficult with a wire frame model, even with depth cueing. This function is best disabled, however, when rotating the model. The RESET function is particularly useful for orienting the model for viewing sections in orthogonal planes with clipping.

While the object of the modeling is to mold a B-spline surface, most of the initial modeling is best performed without constant display of the surface. Display of the B-spline surface adds a profusion of lines to the already crowded screen display and slows response time considerably, since the host computer must communicate a new vector list for the surface to the PS 300 each time a vertex is moved. Due to the highly intuitive nature of B-spline manipulation, it is possible to achieve surprisingly accurate results by making the initial placement of control vertices without viewing the resulting surface at all. Due to the current limitations of the operating system, the images presented in the three case studies which follow were generated in this manner. Once the B-spline surface was initially displayed, further manipulation was not possible, as discussed in section 3.4. As will be seen, little further manipulation would be required for very accurate surface representation.

5.1.6. General procedure. The basic procedure for performing surface modeling with SPLINE can be summed up in a few steps.

(1) Preplan the initial placement of control graph rows and columns and select corresponding control graph dimensions.

(2) Do as much control vertex placement as possible by moving entire rows or columns of the control graph at a time.

(3) Use the CLIP function while modeling substructures in two dimensions, initially without display of the B-spline surface.

(4) After initial control vertex placement, view the entire control graph in three dimensions. Ensure that the individual control graph rows and columns do not have undesired extremes.

(5) Display the B-spline surface.

(6) Make minor adjustments to the control graph as needed.

5.2. Case Study #1: Modeling the Australia II

The first case study is a model of the hull of the sailing yacht, *Australia II*, excluding the famous winged keel. This case study illustrates the general approach to modeling a B-spline surface to an existing known surface or to preliminary design lines. The known surface data

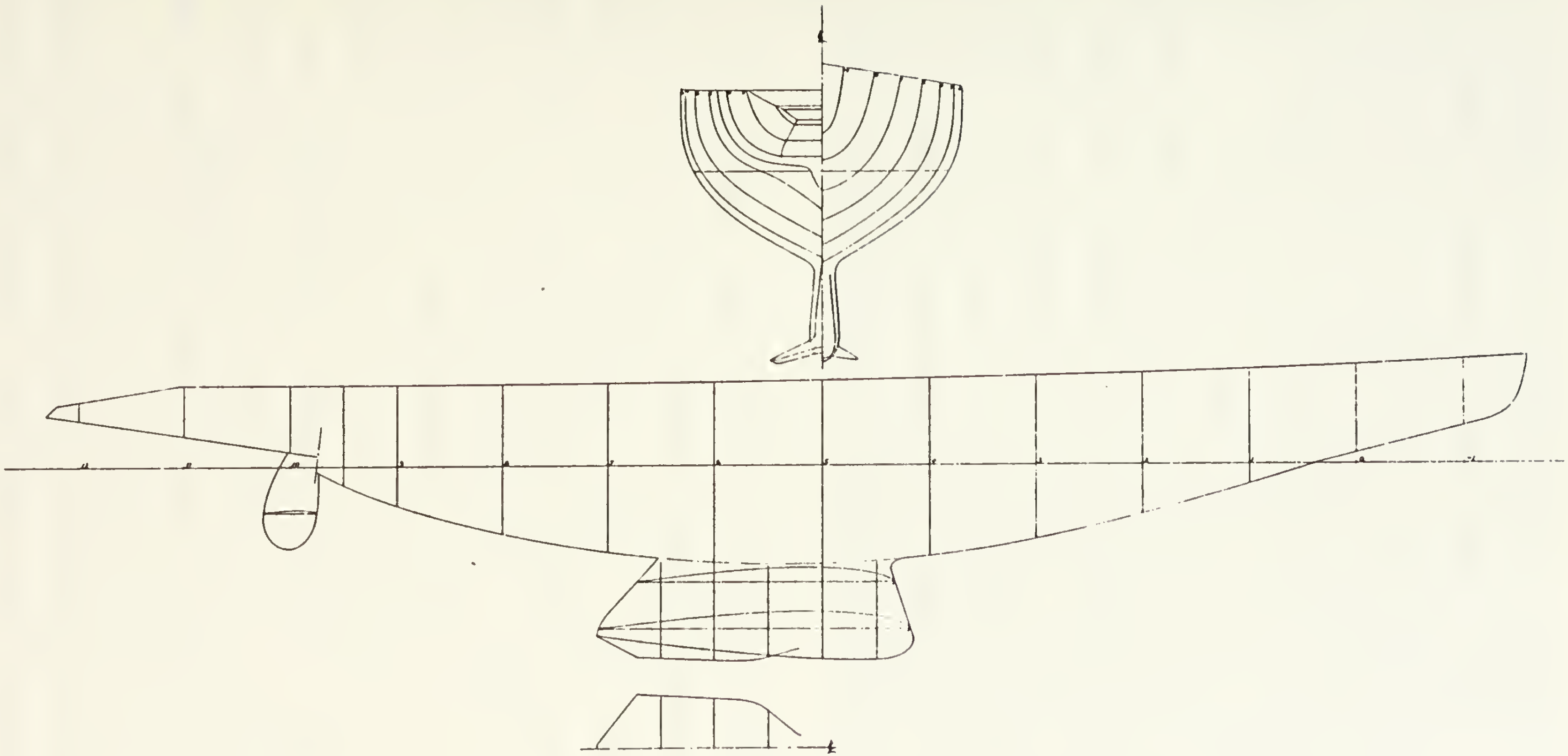

Fig. 18. Lines drawing of the *Australia II*. Source: Stannard [24].

was extracted from a lines drawing (figure 18) contained in Stannard's biography of Ben Lexcen [24].

Before entering SPLINE, the known surface data was put into a PREPS-format file by digitizing it from the lines drawing with the two-dimensional digitizing program, DIGIT. Transverse sections were designated as substructures with about six to ten points each. Since the body plan presents the fore and aft section profiles on opposite sides of the xz plane, it was necessary to manually modify the data file created by DIGIT to place all half-section profiles on the same side of the plane. This was a simple matter of deleting minus signs from the data.

The final step before entering SPLINE was to determine the required control graph dimensions. It was apparent that modeling would be performed most conveniently in two-dimensional sections corresponding to substructures of the digitized data. Since the body shape was relatively uncomplicated in the body plan, it was decided to use only five rows in the control graph.

The longitudinal lines of the yacht required greater consideration. It will be useful to view figure 19, which shows the selected control graph as squares connected by net lines and known surface points as unconnected asterisks, while examining the selection process which led to this control graph. The most easily selected locations

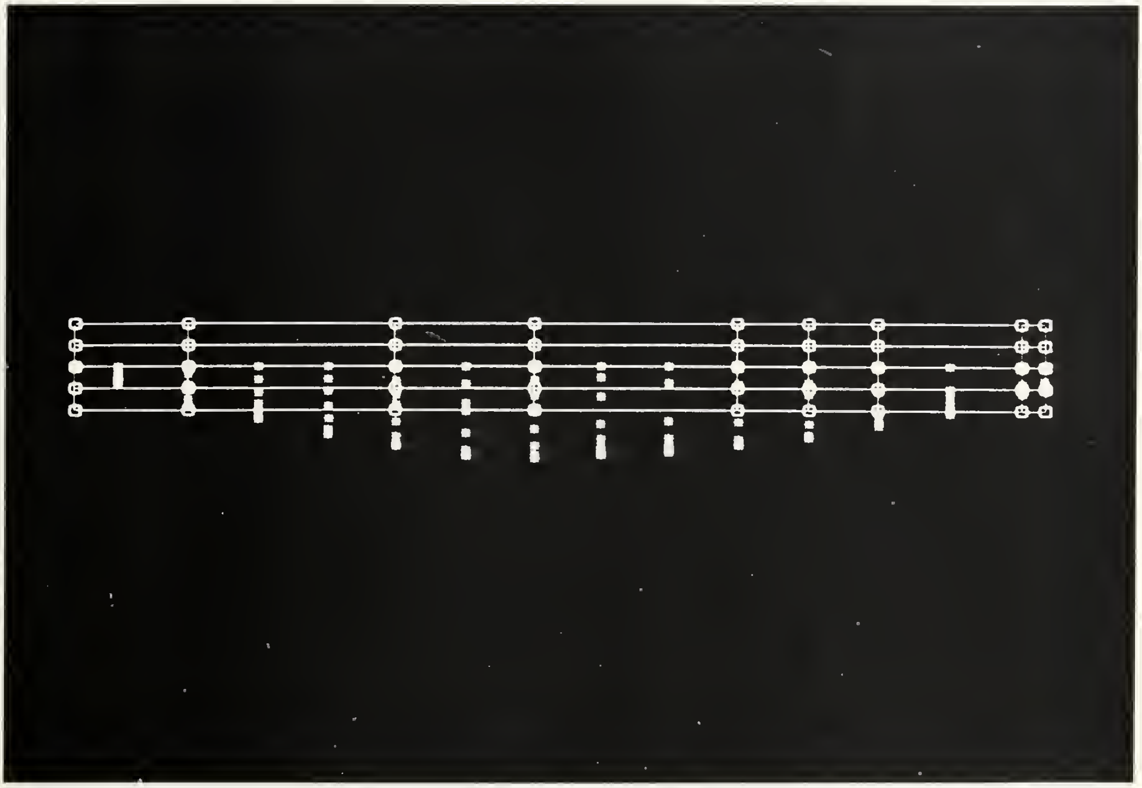

Fig 19. Initial control graph for *Australia II*, after relocating columns to known hull stations.

for columns of the control graph were a station at each end of the hull to, define those boundaries, and a station to define the maximum point of the keel. Since the shape of the forward sections is relatively uniform, two additional stations were expected to be sufficient to define the forebody.

The after half of the hull presented more interesting problems. The stern is not a V-shaped section, as most of the hull, but a U-shaped section. The stern also has a short portion which is somewhat elevated and very narrow, with nearly parallel sides. A second station was selected at the stern to help maintain the shape and size in that

region.

Between the keel and the stern, the transition from the V-section to the U-section is relatively abrupt, as can be seen in figure 20, which shows the two consecutive transverse sections. (Asterisks represent the known surface points.) A control graph column was selected for

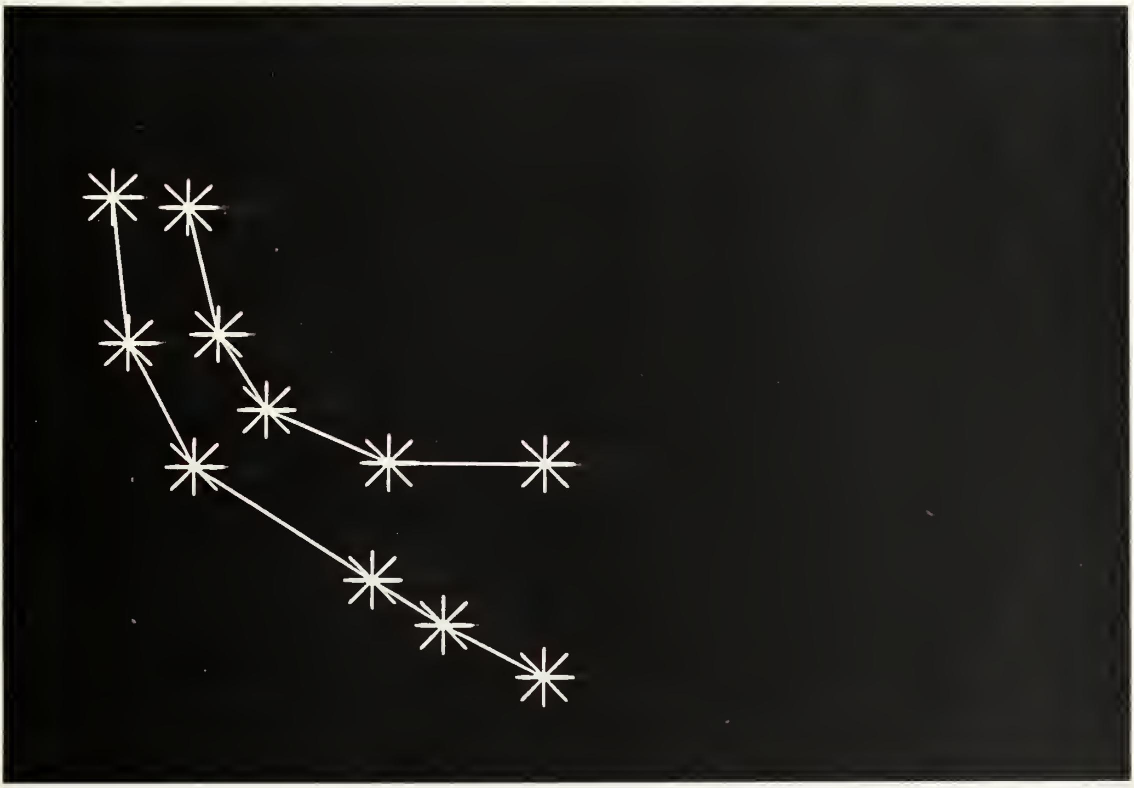

Fig. 20. Abrupt transition in the after hull of *Australia II*.

each of the illustrated sections, to define the approximate beginning and end of the transition region. It was expected that additional control would be required to ensure that the U-shaped station did not unduly influence the hull forward of the transition, nor the V-shaped section influence the shape aft of the transition.

The proximity of the two control graph columns selected for the stern was thought to be sufficient to maintain the shape between the U-shaped station aft of the transition and those at the stern. An additional V-shaped station was selected adjacent to the one located just forward of the transition, to minimize influence of the U-shaped stations forward of the transition. This made a final count of nine columns.

Upon entering SPLINE, a control graph size of (9,5) was selected, based on the above planning. The basis functions were defined as third order in both directions, and an initial control graph length-to-width ratio of eleven was chosen to roughly correspond to that of the

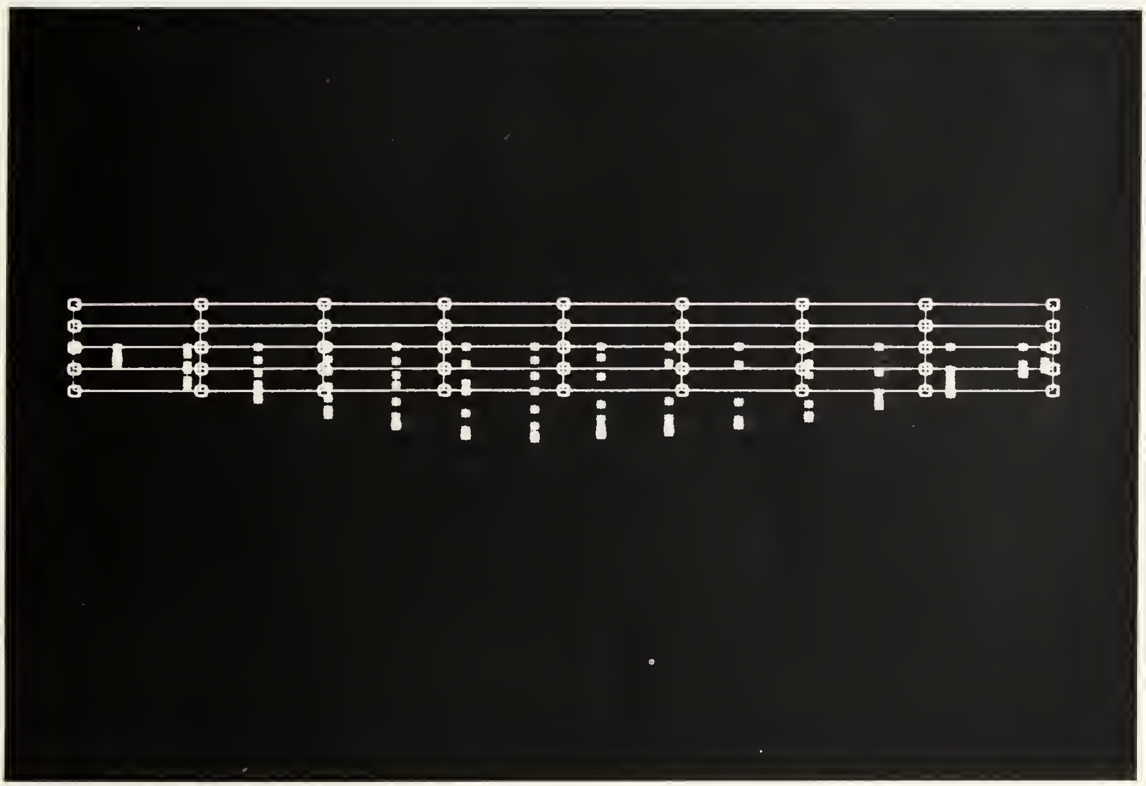

Fig. 21. Initial control graph for *Australia II*.

hull. The resulting initial control graph can be seen in figure 21, along with the digitized known surface data.

Interactive modeling of the yacht hull was begun by selecting the ROW PICK function and moving the control graph columns into the selected positions, as shown in figure 19. The RESET function was then used to orient the hull for a bow view and the display was scaled to fill the screen. The CLIP function was then activated and the clipping window adjusted to a depth which allowed viewing the control vertices and known points of a single transverse station at a time (figure 22).

Individual transverse sections were modeled one at a

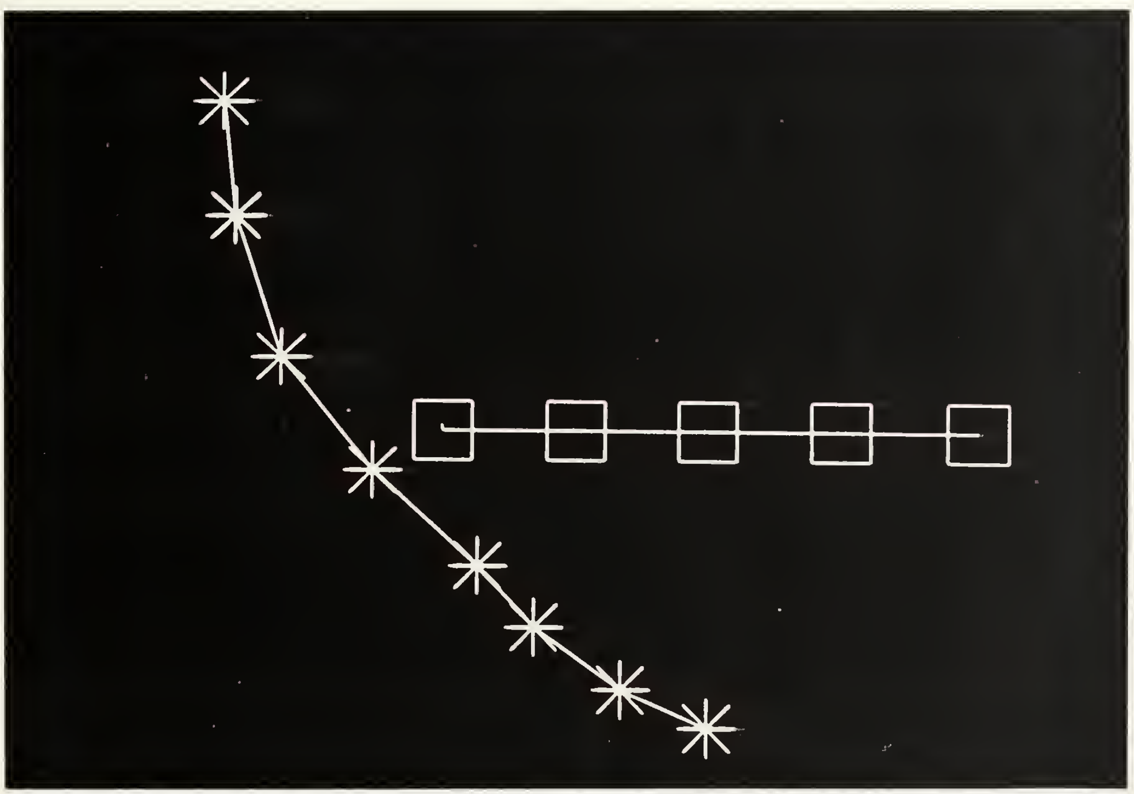

Fig. 22. A station of the *Australia II* before modeling.

time by selecting and moving the control vertices individually and moving the hull through the clipping window with the GLOBAL Z dial to view each new station. All five control vertices were located coincidentally at the bow. Figure 23 shows a more typical station with relocated vertices. Notice that the vertices are located slightly outside of the curvature of the known surface, since the B-spline surface will be contained in the convex hull of the control graph, but will not interpolate it. This offset is exaggerated in the illustration. The small tabs extending from the control vertices are the control graph lines which extend beyond the clipping window to vertices in other control graph columns.

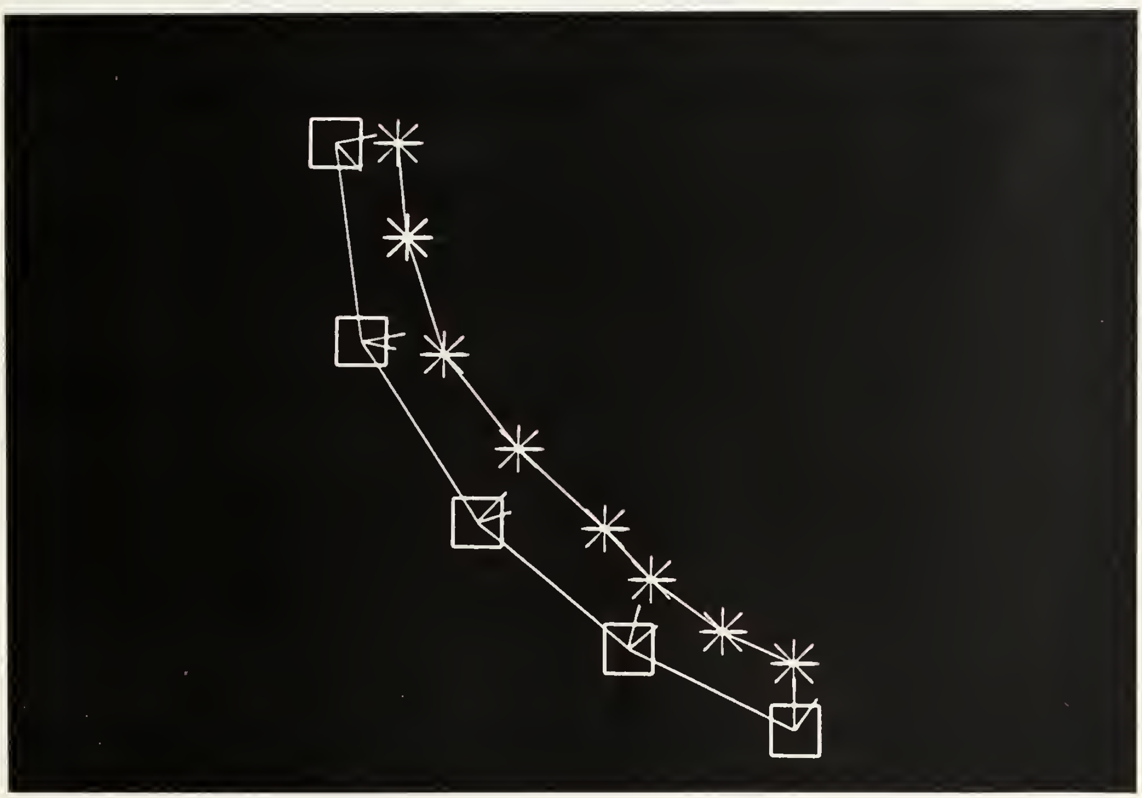

Fig. 23. Control polygon for a V-shaped station.

Figure 24 illustrates the control polygon for a U-shaped section. The control vertices at the right are located above the bottom of the hull in order to remain outside of the curvature which occurs in the longitudinal direction (perpendicular to the screen). Three collinear vertices are used to define the flat bottom between the two right-hand points, as well as the slope of the bottom as it begins the upward curvature between the second and third collinear vertices.

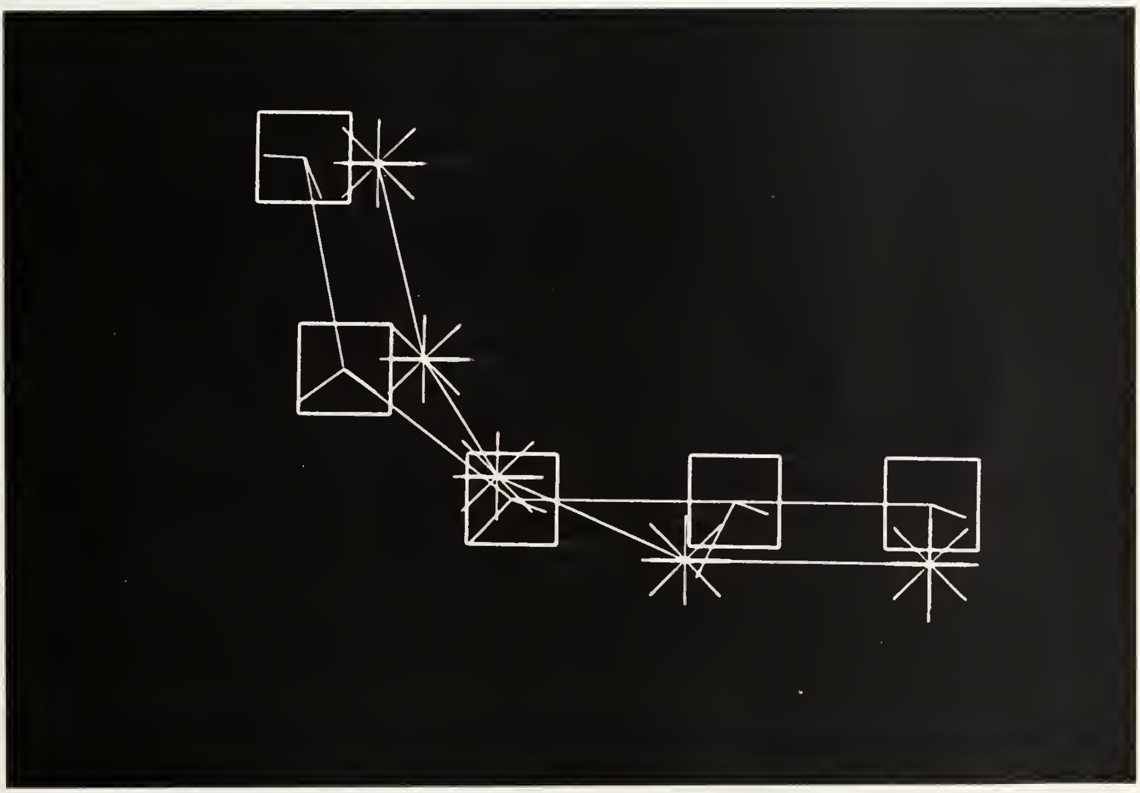

Fig. 24. Control polygon for a U-shaped section.

The final control graph is pictured in figure 25. Notice that the net of lines is uniform in both the transverse and longitudinal directions. The resulting B-spline surface is shown in figure 26 with the known

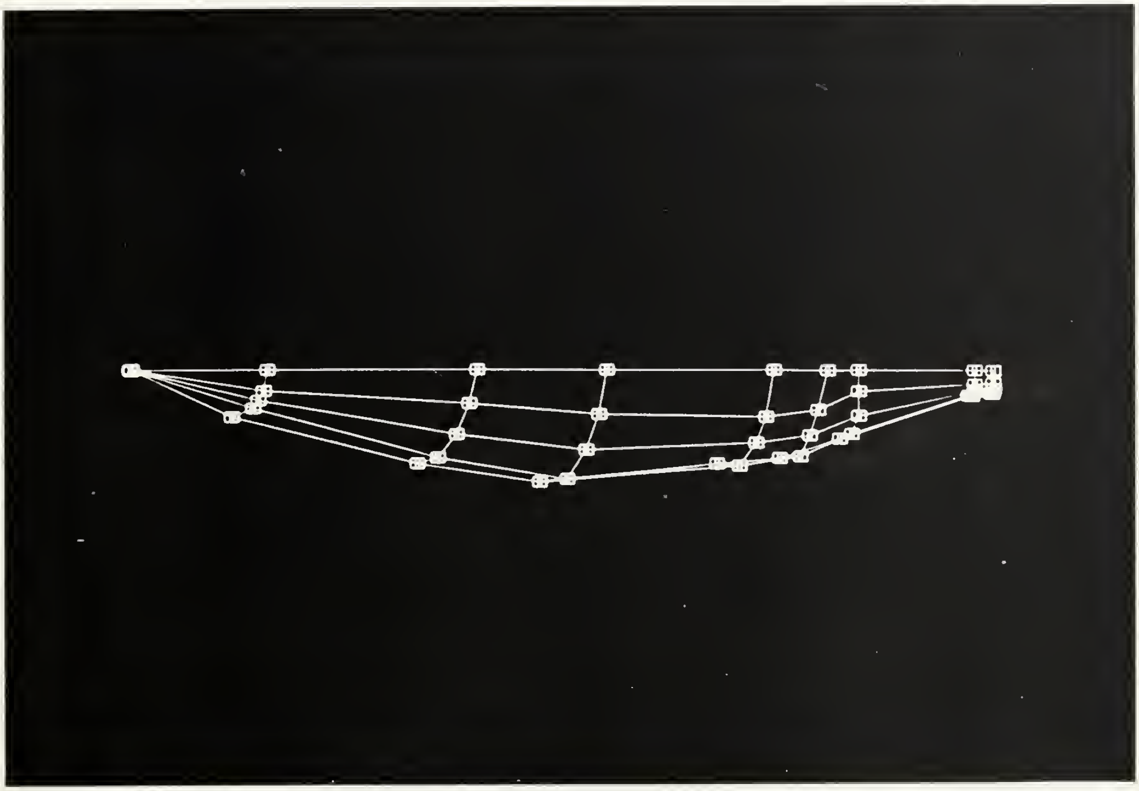

Fig. 25. Final control graph for *Australia II*.

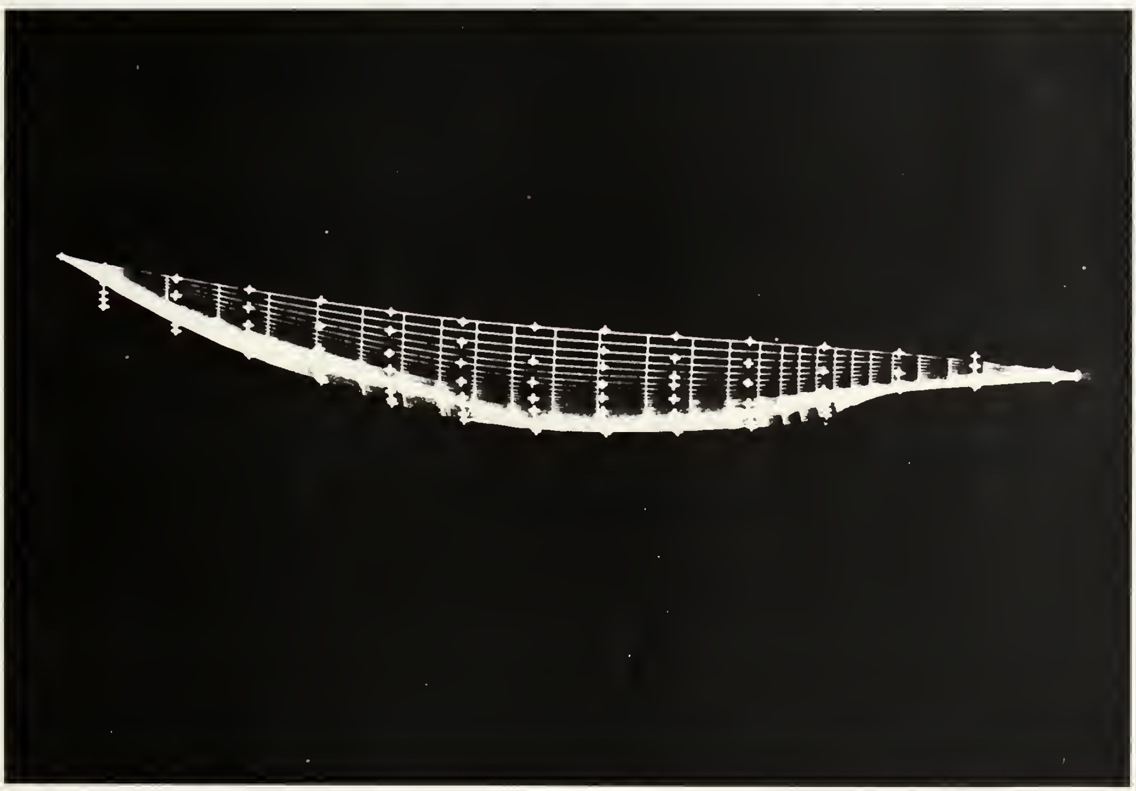

Fig. 26. B-spline model of *Australia II*.

surface points for comparison. The match is generally quite good, but an additional control graph column is needed near the bow to hold the keel line downward. A closer view of the transition from V-shaped to U-shaped sections is shown in figure 27, which depicts the B-spline surface and its associated control graph.

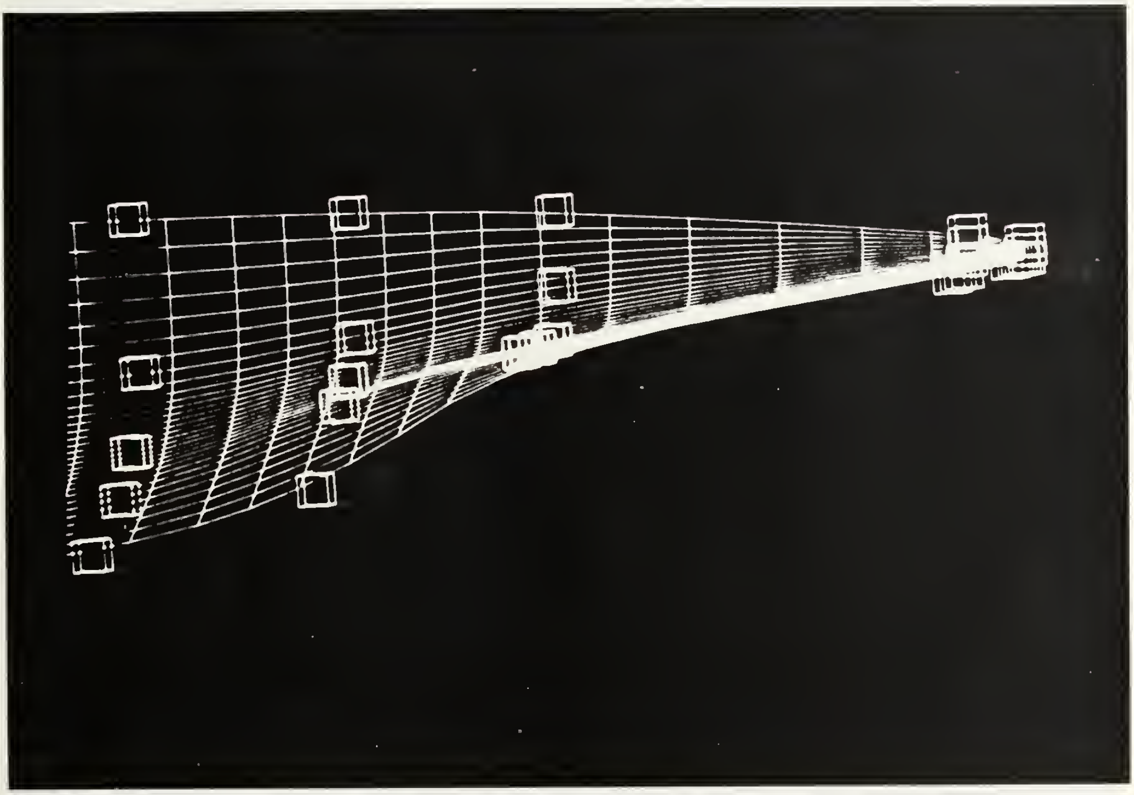

Fig. 27. B-spline model of *Australia II* stern.

The next step in modeling the hull would be to again view the transverse stations one at a time and make minor adjustments in the control graph. Figure 28 compares the control vertices (squares), known surface (asterisks) and B-spline surface (solid line) for a V-shaped station. It can be seen that only very minor adjustments are indicated.

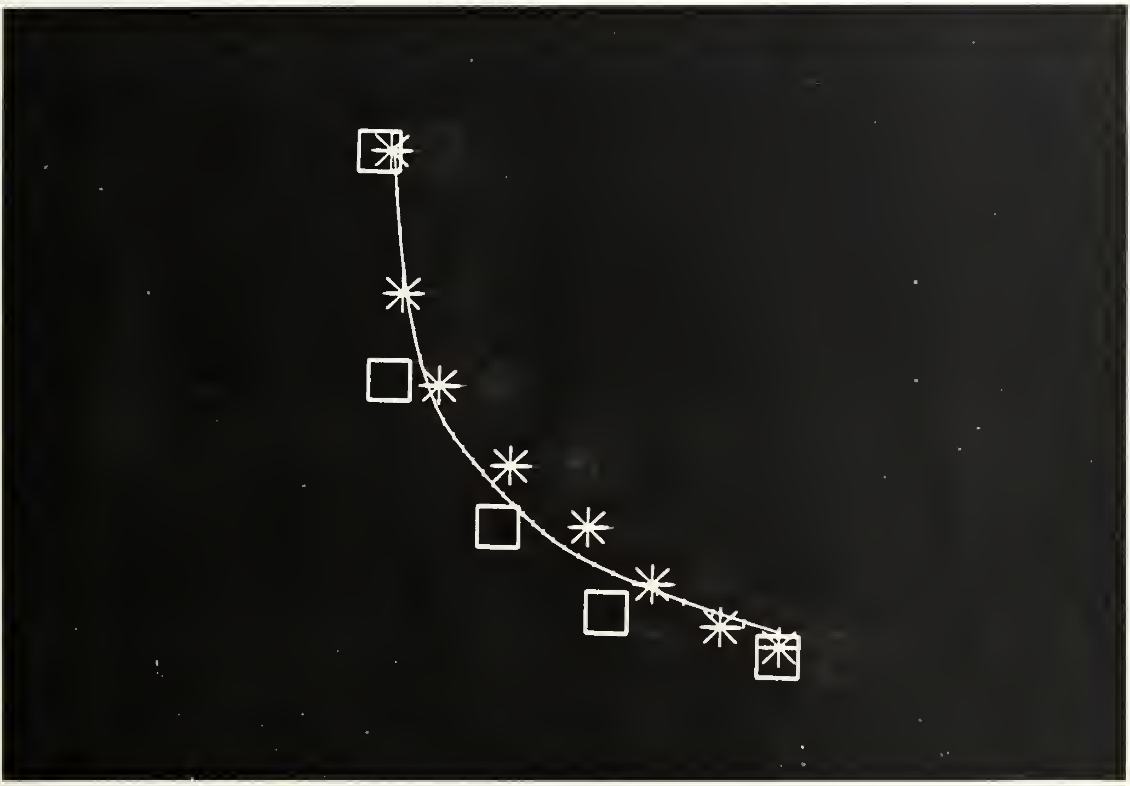

Fig. 28. B-spline surface at a V-shaped station.

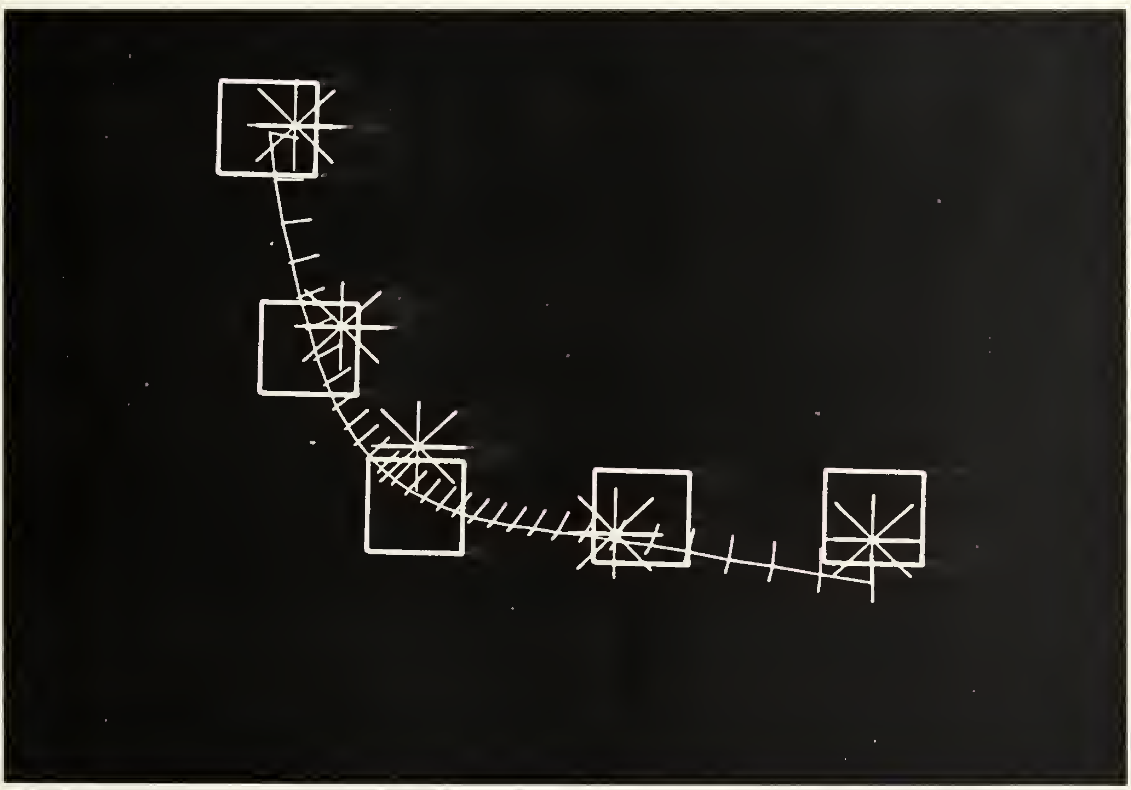

Fig. 29. B-spline surface at a U-shaped station.

Figure 29 shows the B-spline surface achieved at a U-shaped section and illustrates one of the pitfalls of considering primarily two-dimensional sections while modeling a three-dimensional surface. The three collinear vertices at the bottom would dictate a straight line in the right-hand segment of the B-spline curve in two dimensions. However, the surface is pulled downward at the right-hand edge by the effects of the vertices at stations located further forward. A considerable upward adjustment of the right-hand control vertex may be necessary to overcome this pull from other stations.

5.3. Case Study #2: Modeling a Porsche 944

The second case study involves very similar considerations to the first and will be presented more briefly. The body of a Porsche 944 automobile was modeled, reemphasizing the techniques discussed previously and illustrating the versatility of the B-spline and of SPLINE.

The known surface data for the Porsche was approximated by digitizing points from a lines drawing and profile contained in advertising literature. Once again, DIGIT was used to create the data file. The data for half of the symmetrical body was presented to SPLINE without reflection.

The shape of the surface was considered in order to estimate the required dimensions of the control graph.

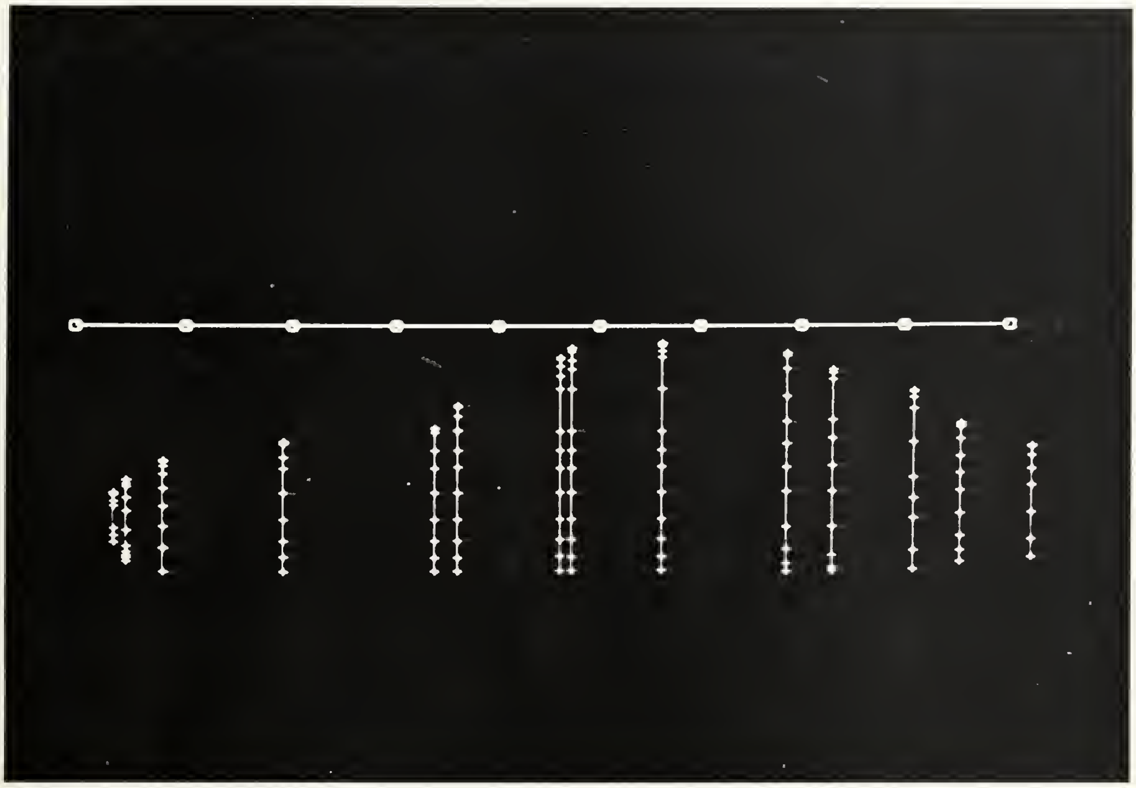

Fig. 30. Initial control graph column locations for Porsche 944 body.

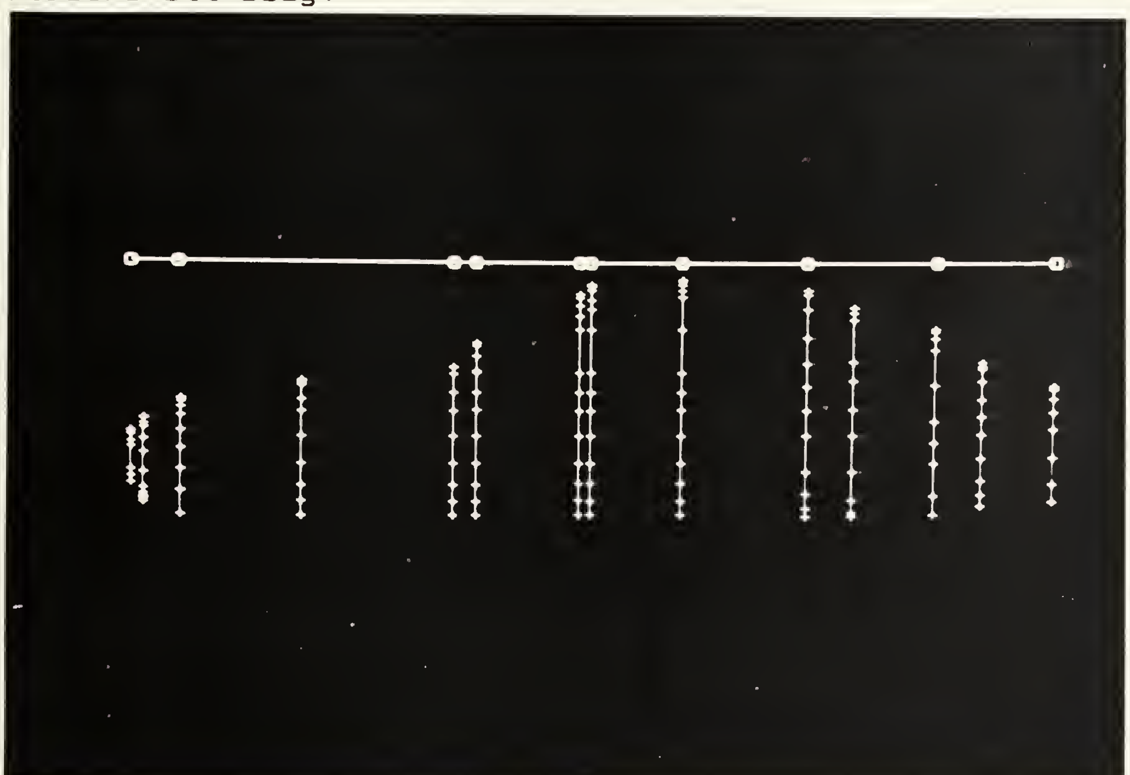

Fig. 31. Relocated control graph columns for Porsche 944.

Figures 30 and 31 illustrate the selection of transverse stations for modeling, similar to those chosen for the previous case study. In this case, the control graph is presented in an edge view. Looking first at the longitudinal lines, it was determined that ten control graph columns would be required. A station was selected at each end, another near the front (to avoid the problem encountered in modeling the bow of the *Australia II*), and a station near the middle at the point of maximum height in the profile. Two stations were determined to be necessary to achieve the small-radius curvature at the base of the windshield, and two more at the top. Finally, two intermediate stations were positioned in the rear half of the body. If an eleventh station had been available, it would have been place midway along the hood of the automobile.

With consideration to the shape of the transverse sections, seven control graph rows were deemed necessary. Figure 32 depicts a section across the hood being modeled and illustrates the reason for selection of seven control graph rows. Since the sharp curvatures and sudden transitions in the body dictated use of a B-spline which was third order in both dimensions, three vertices were required to achieve the nearly flat hood and roof of the automobile, with an additional vertex at the hard chine between the hood and fender. (The brighter vertex seen

there is actually two coincident vertices.) Three vertices mold the curve at the side of the automobile.

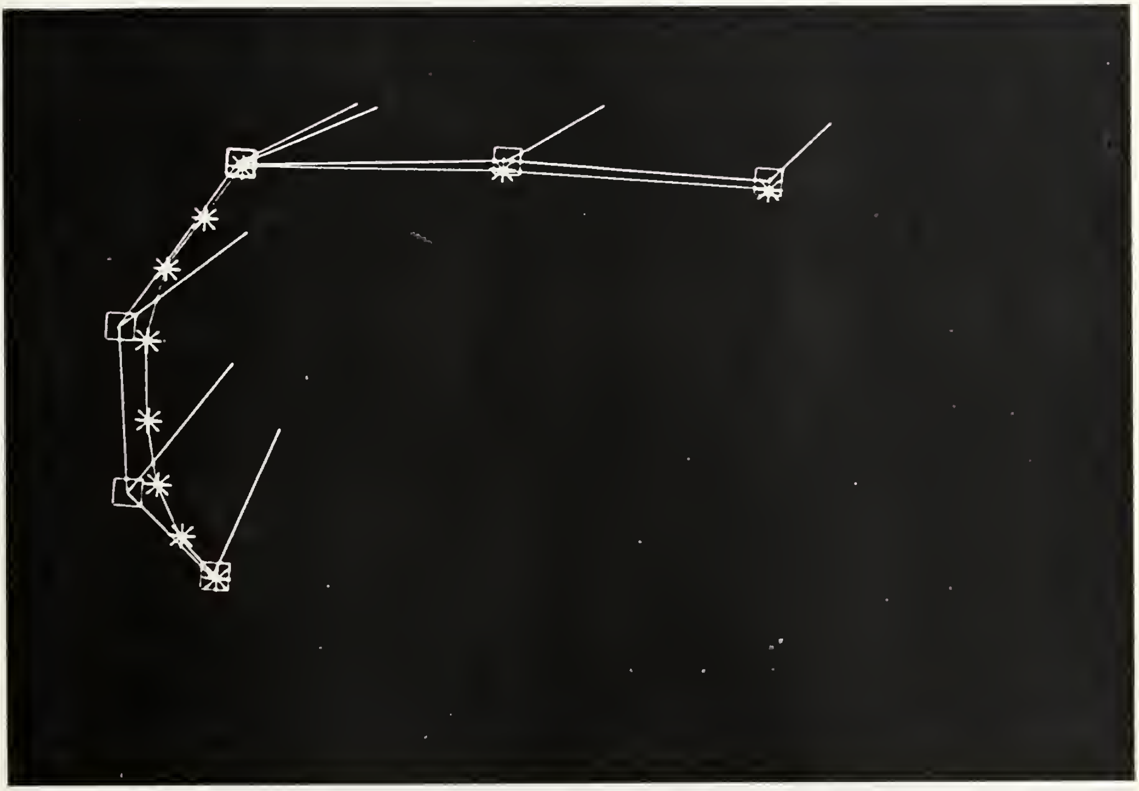

Fig. 32. Modeling a forebody section of a Porsche 944.

As detailed previously in case study #1, interactive modeling was performed by first moving control graph columns to the selected stations, then modeling individual transverse stations with clipping enabled. Examining the control vertices of figure 32 from right to left, the first three vertices are placed nearly linearly to give the nearly flat surface of the hood. The third and fourth vertices are placed at the same point to achieve the knuckle. The fifth vertex serves two functions. The control polygon line connecting this vertex with the vertices at the knuckle defines the slope of the curve as

it flows downward from the knuckle. The position of the fifth and sixth vertices define the maximum for the curve in the fender. Finally, the last vertex defines the lower boundary of the surface.

Fig. 33. Control graph and known surface for a Porsche 944.

Figure 33 presents the final control graph after modeling the individual sections, along with the known surface data. While each individual station appeared to be adequately modeled, displaying the control graph lines (figure 34) revealed a problem. Near the midsection of the automobile, one vertex was placed much lower on the body's contour than the corresponding vertices in adjacent columns. While the placement had appeared adequate when

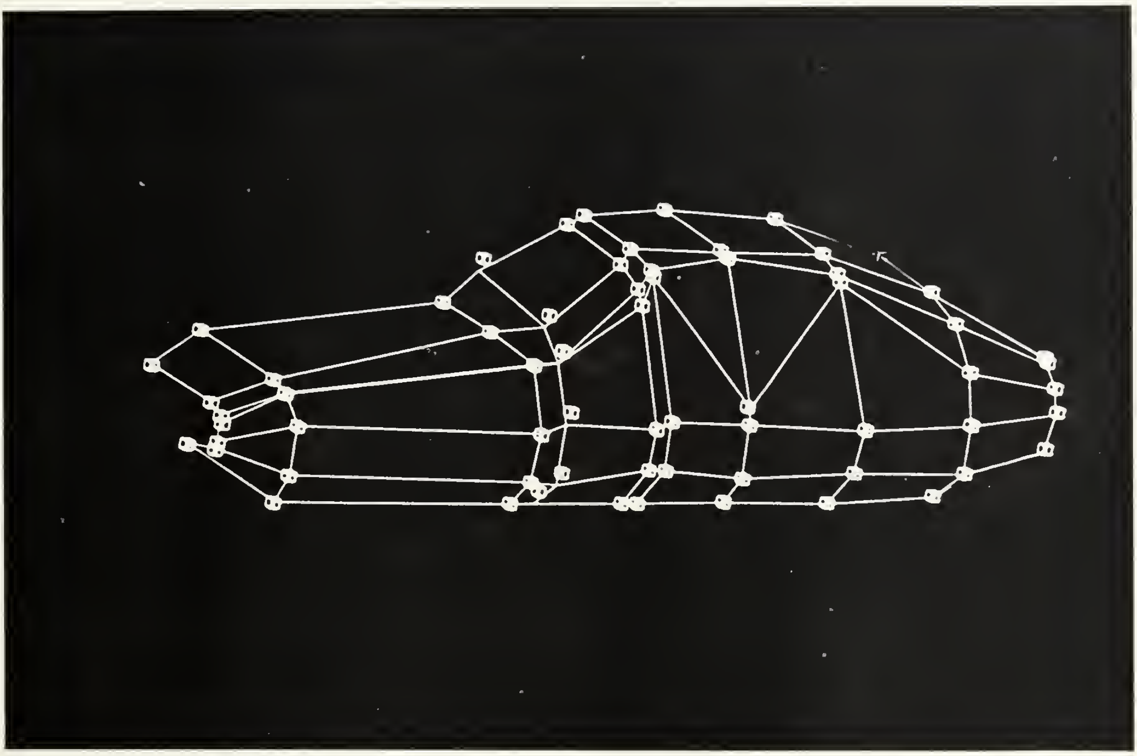

Fig. 34. Control graph for Porsche 944 with an inconsistency in one row at the midsection.

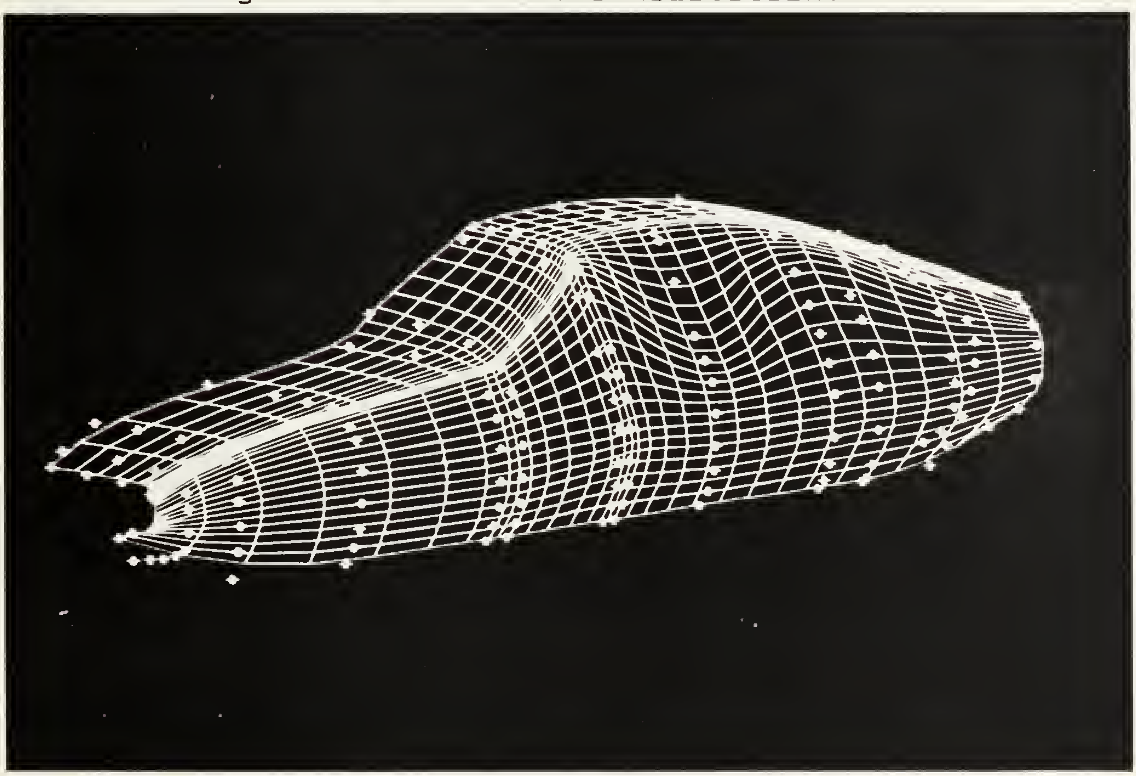

Fig. 35. B-spline representation of a Porsche 944.

viewed in the transverse plane, it was now apparent that this vertex was sharply out of position when viewed as part of its control graph row.

The effect of this placement can be seen in figure 35. Much of the apparent distortion in the body at the midsection is only an illusion caused by the manner in which the net of lines representing the B-spline surface is displayed. Some real distortion does also occur in this region due to the added pull of the misplaced vertex. This case study illustrates well the importance of considering and reviewing the control graph in three dimensions.

Figures 36 and 37 present comparisons of the B-spline

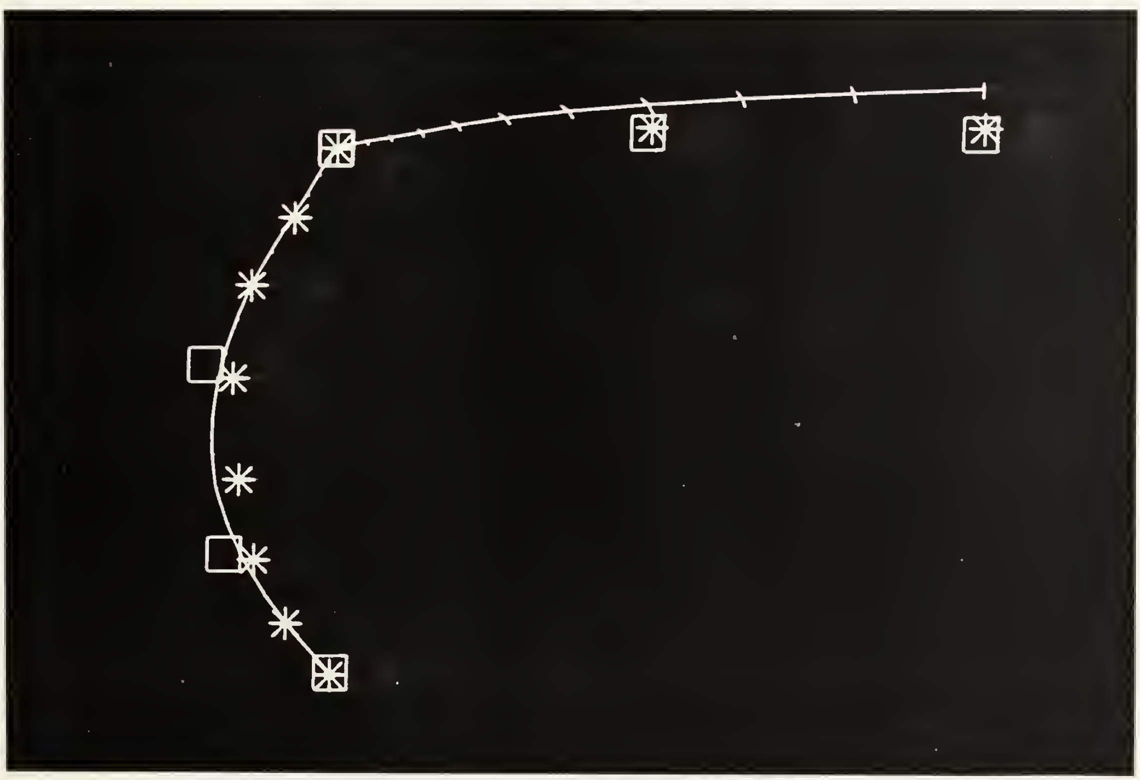

Fig. 36. B-spline surface with a hard chine.

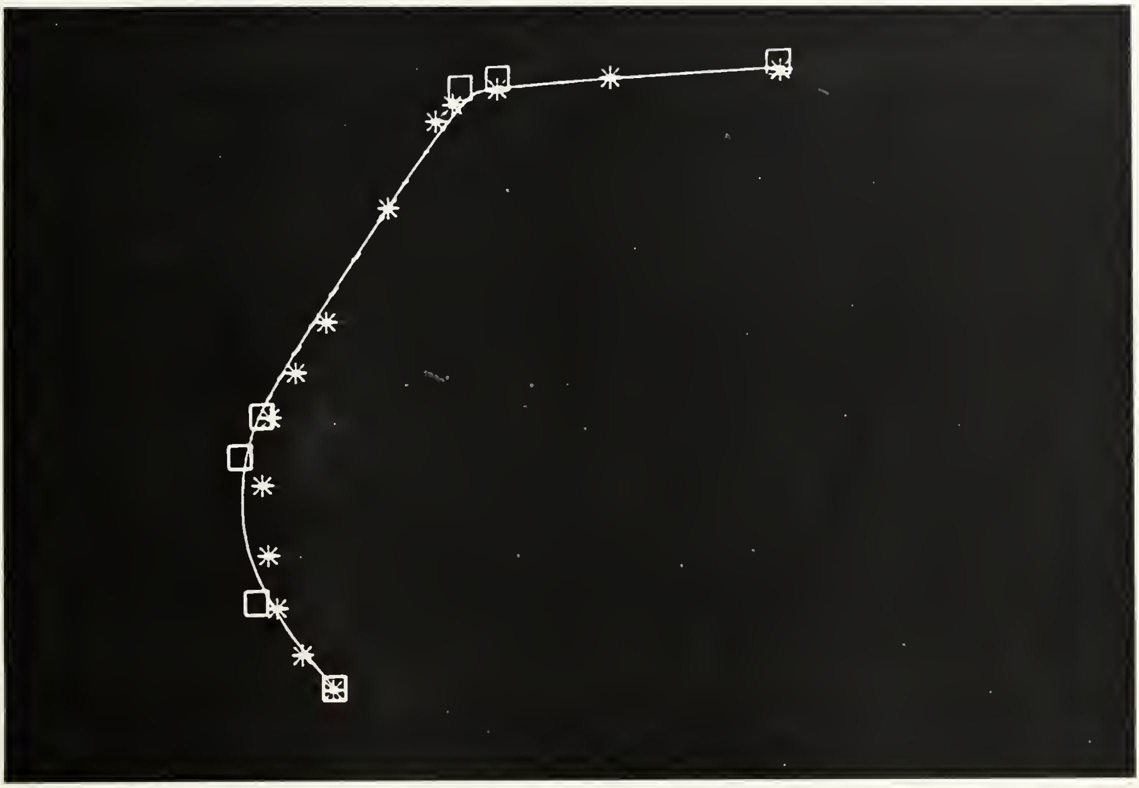

Fig. 37. B-spline representation of a Porsche surface.

surface to the control vertices and known surface points. Note that the two coincident vertices in figure 36 produce a hard chine, while placing the vertices slightly apart, as in figure 37, produces a small-radius curvature.

The effect on this station of control vertices at other stations can again be seen in figure 36, where the surface of the hood is pulled up from the desired form by the control vertices at the next station, which are located higher for the windshield. Figure 37 also illustrates the remarkably accurate surface which can be obtained even without viewing the B-spline surface during initial control graph arrangement.

5.4. Case Study #3: Freeform Modeling

The final case study presents an example of freeform modeling, in this case of a Stetson hat. In *ab initio* design, the engineer often has a general idea of the form which he or she is trying to achieve, but is not constrained by a requirement to exactly match previously existing lines. For this reason, no attempt was made to compare this model to a preexisting design.

In the absence of such previous form definition, it will often be beneficial for the designer to sketch a few basic lines to aid in visualizing the shape and the required control graph dimensions. Because of the difficulty in visualization of the more abstract form, in this particular case it was also found to be useful to model both halves of the object, even though it is symmetrical.

Aside from these initial considerations, the modeling approach is basically the same as that in the first two case studies. A third order B-spline was used with a seven by seven control vertex matrix, based on form considerations which will become apparent as the model development is reviewed. A length to width ratio of 1.3 was selected. Figure 38 presents the initial control graph.

Viewing the planar control graph as being in the horizontal plane of the hat's brim, vertices defining the

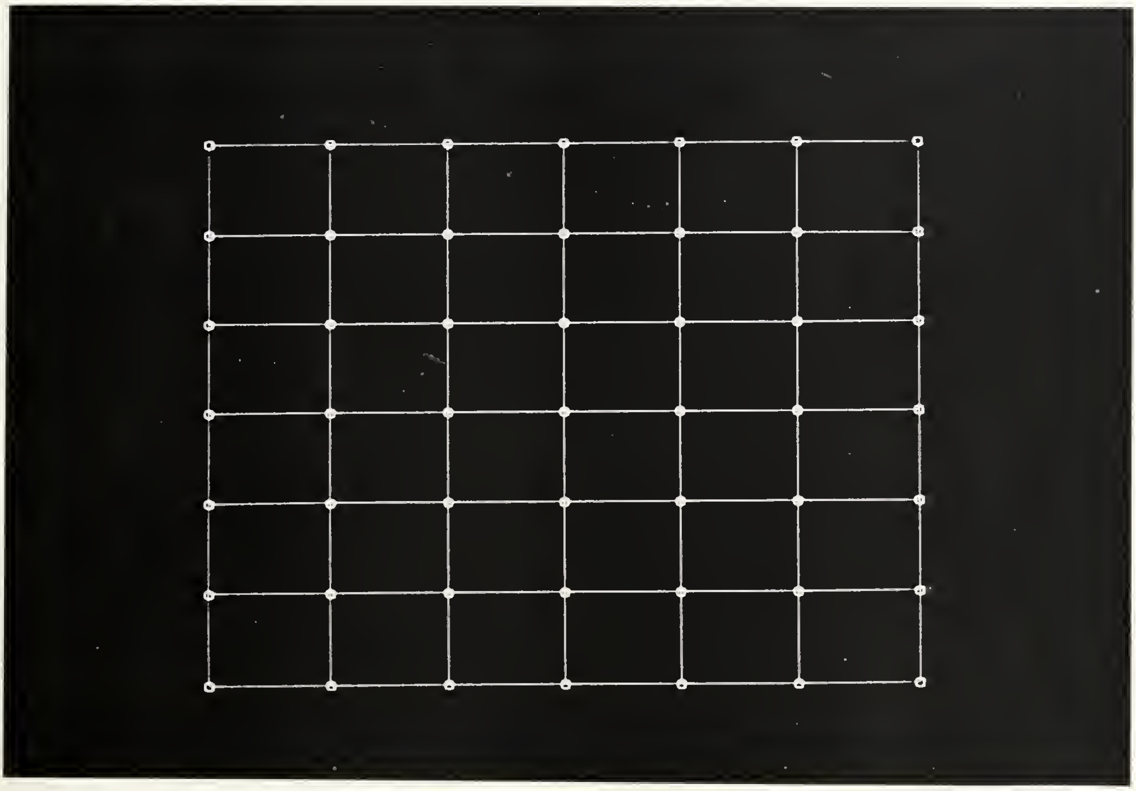

Fig. 38. Initial control graph for a Stetson hat.

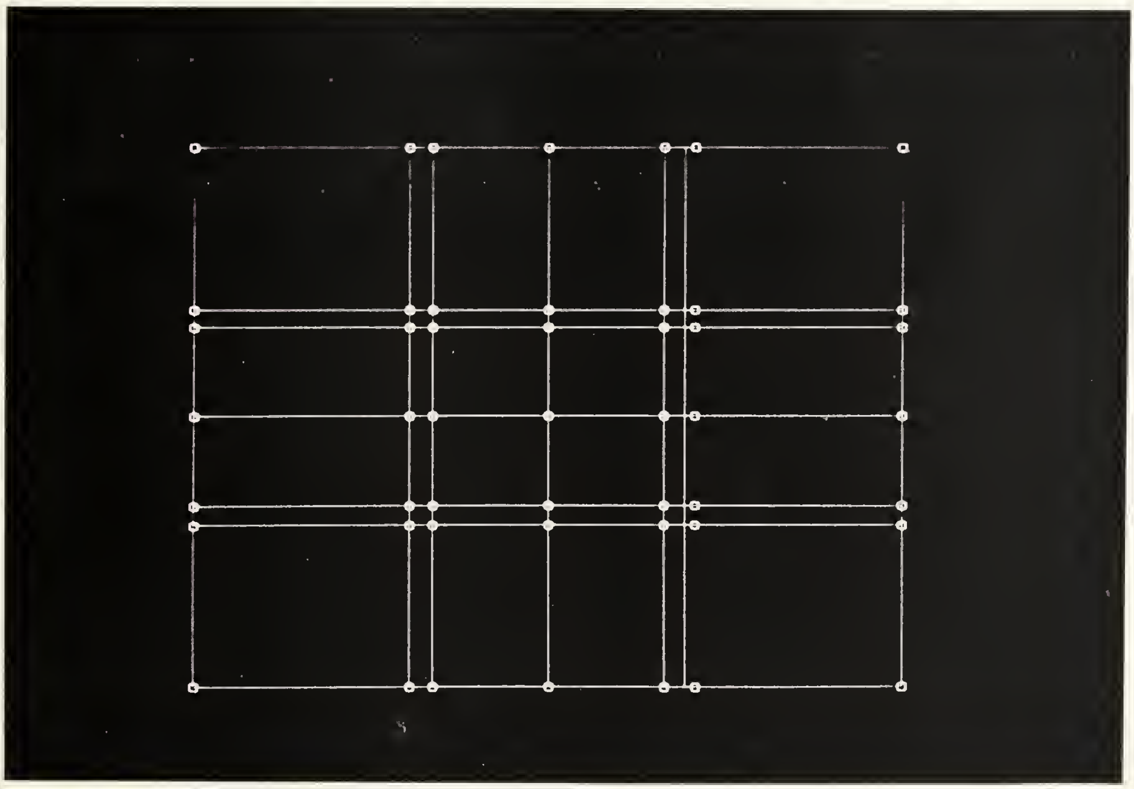

Fig. 39. Control graph rows and columns relocated.

sides of the crown were to require placement nearly vertically above one another. For this reason, the ROW PICK feature was used to relocate rows and columns of the control graph as shown in figure 39. Notice that the second column from the right exhibits the disparity between control vertex position and control graph line position which was discussed at the end of section 4.3.5. When moving the column of control vertices, one increment of the movement was distorted during communication between the two computers and thus ignored by the host computer's program. In all subsequent operations, there will continue to be a disparity between the position of these vertices as displayed by the PS 300 and as used in the preparation of vector lists and calculation of surfaces by the host computer. The intersection of the control graph lines, and not the individual cubes, marks the vertex position upon which the B-spline surface will be based.

After positioning individual vertices to define the rounded shape of the surface edges (figure 40), the vertices controlling the surface of the hat's crown were pulled up from the surface. To achieve uniformity of the similar halves, the points were first selected with the stylus while the control graph was in an orientation similar to that shown in figure 41, then the display was rotated to present an edge view (figure 42) before the selected vertices were moved. Also in the interest of

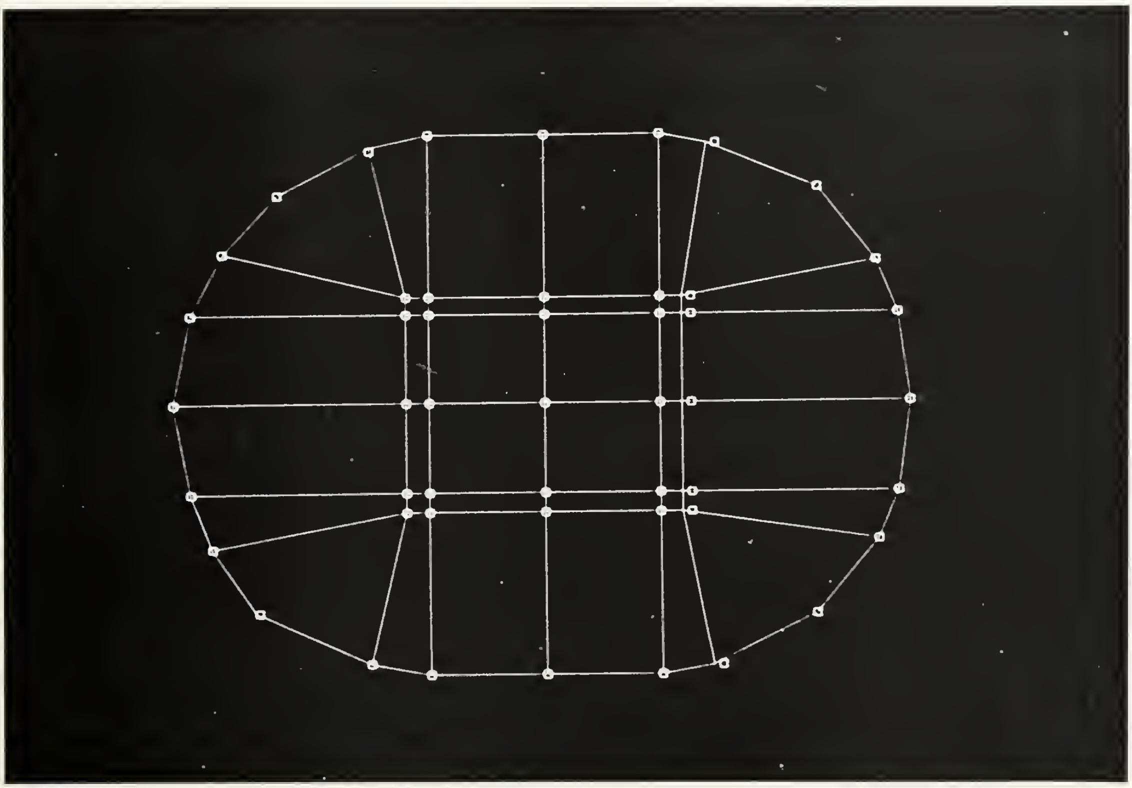

Fig. 40. Surface edge boundaries established.

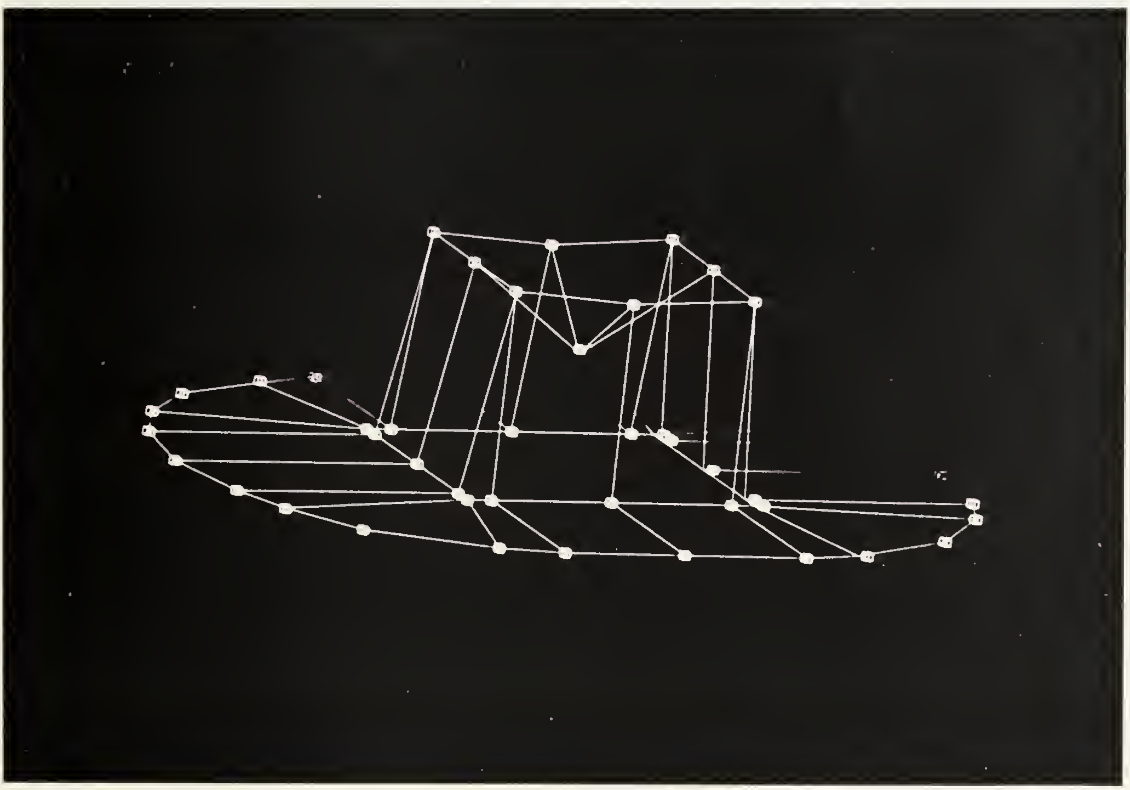

Fig. 41. Elevated vertices define crown.

achieving uniformity, the modeling was performed without use of the CLIP function, allowing direct comparisons to be made between foreground and background lines. Notice the depth cueing in the right rear quadrant of figure 41 which gives the viewer a better perception of the display orientation.

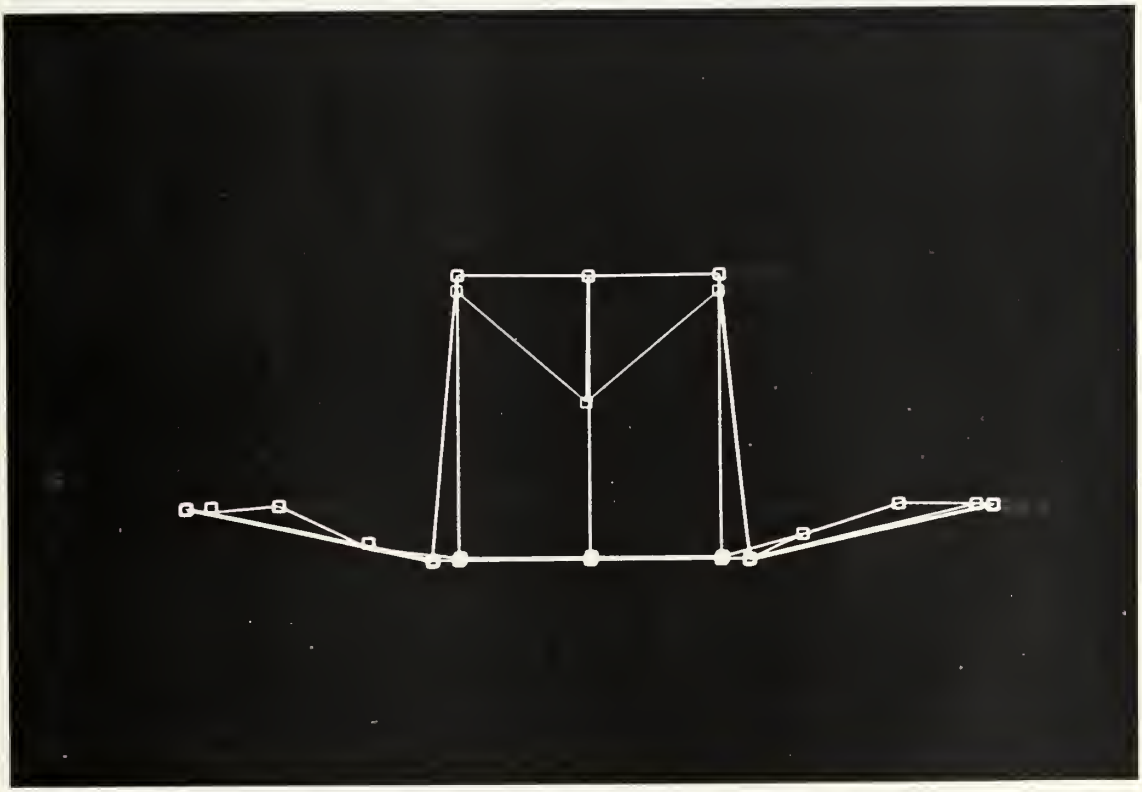

Fig. 42. Elevating the brim.

The final step in forming the control graph was to elevate the first and last rows of the control graph to achieve the raised brim (figure 42). It was possible to do this very easily in the ROW PICK mode, even though points in the rows had been moved around to accomplish the rounding of the edges of the brim. The entire row of vertices could be moved simultaneously without affecting

the relative position of the individual vertices in the row to each other.

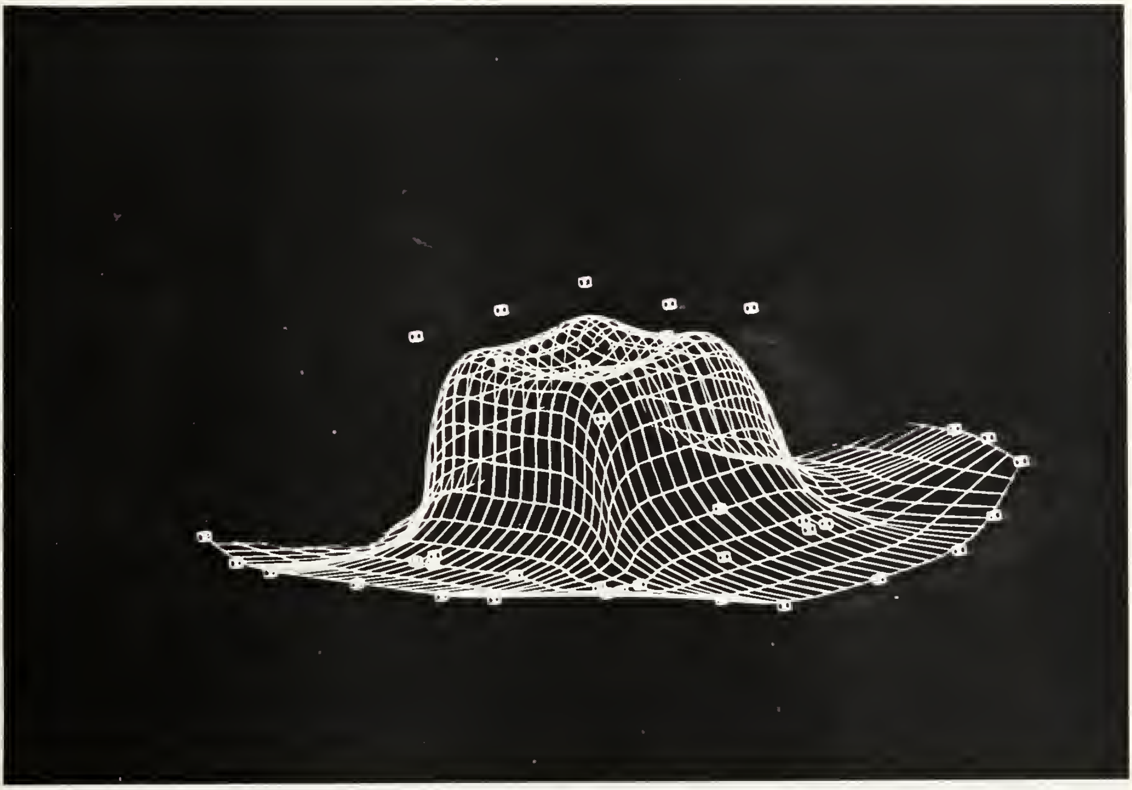

Fig. 43. Stetson hat with control graph.

Figure 43 presents the final control graph of the hat with the resulting B-spline surface. Notice that while the vertices form a relatively rounded brim (figure 40), the resulting surface has distinct corners (figure 43). This effect is due to the fact that only the corner vertices of the control graph are interpolated by the surface. Since the adjacent vertices are not interpolated, the result is a sharp discontinuity at each of the four corners of the control graph. In order to present the desired smooth boundary, these four vertices must be relocated to lie on the intended edge of the

surface, rather than on a smooth curve connecting the adjacent vertices.

While the hat perhaps belongs more correctly on Huckleberry Finn's head than on a cowboy's, it nevertheless illustrates well the great potential for freeform design with surface B-splines. Figure 44 presents a section view of the same hat and control graph.

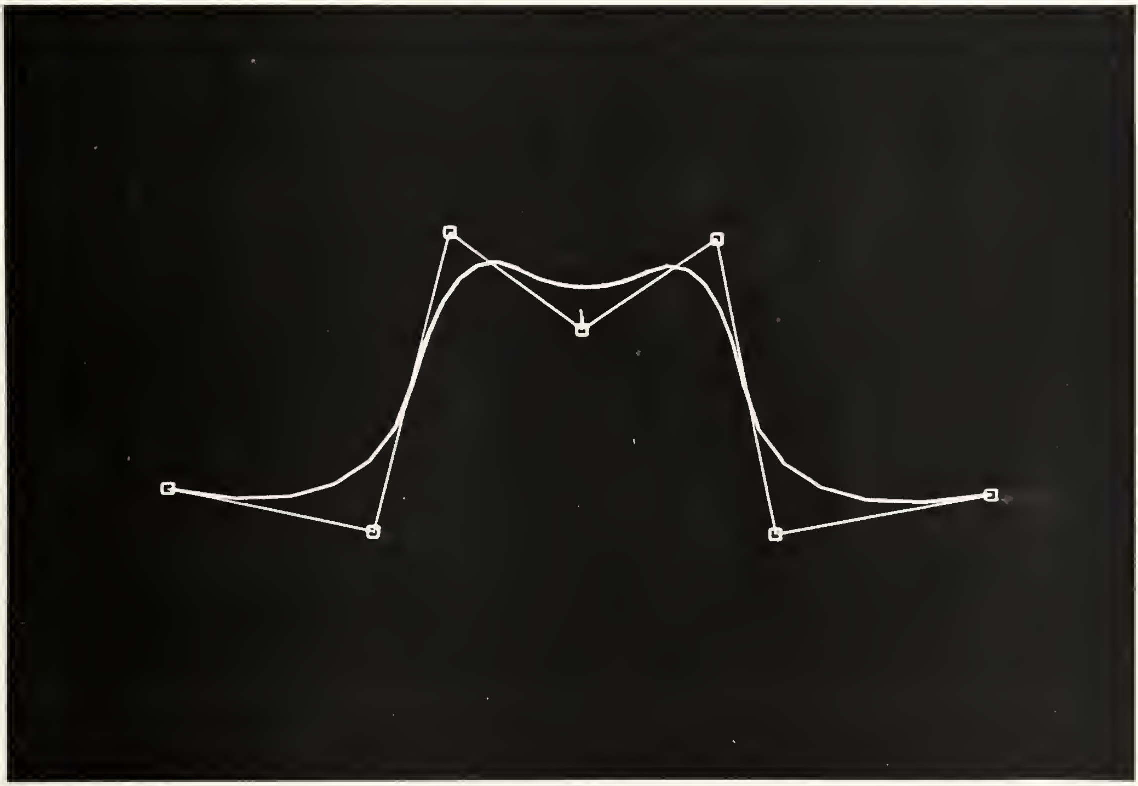

Fig. 44. Section view of hat with control polygon.

CHAPTER 6

RECOMMENDATIONS AND CONCLUSIONS

6.1. Hardware and Firmware Modifications

Several modifications to the hardware and firmware of the system upon which SPLINE was developed would significantly enhance the program. At this time the University of Washington Mechanical Engineering Department's PDP-11 computer is being replaced by a VAX-11/750. This replacement makes possible most of the enhancements discussed below.

6.1.1. High-speed interface.

It was initially planned that SPLINE would make use of rubber sheeting, the real-time modification of the B-spline surface display in response to control vertex manipulations. However, it soon became apparent that communication between the two computers, at 9600 baud, was far too slow to effect real-time response.

Rogers and Satterfield [17] found calculation of the full B-spline algorithm too slow to allow surface dragging or rubber sheeting, so developed an abbreviated algorithm which took into effect the localized effect of a single control vertex movement. Using the present system, even the time for a full B-spline surface calculation was found to be insignificant compared to the time required to communicate a new vector list, however calculated, to the

PS 300. It takes approximately ten seconds for the host computer to send the vector list for the B-spline surface.

The PS 300 manual [20] states, "This interface is best suited for infrequent communication of small amounts of data." The interface simply is not designed to support a program such as SPLINE. Upgrading the system with Evans and Sutherland's 56 kilobaud high-speed interface would provide a distinct improvement in the program operation.

6.1.2. VAX host computer. The greatest potential benefit of interfacing the PS 300 with the newly installed VAX computer is avoidance of the PDP-11 operating system discrepancies. It is expected that SPLINE would require few modifications to compile on the VAX system.

6.1.3. Graphics Support Routines. A totally different host-resident communications software package is available for the PS 300 on VAX systems than is used on the PDP-11. The Graphics Support Routines provide significantly more reliable communications between the host computer and the PS 300. Additionally, these routines perform some preprocessing of data in the host CPU, resulting in substantially faster communications. Upgrading the system with the Graphics Support Routines would require some revision of the manner in which SPLINE presents data for communication to the PS 300, but would provide a worthwhile enhancement.

6.2. Program Expansions

SPLINE was designed to support many features which are not yet fully implemented. For many of these features, the PS 300-resident portion of the program already includes the necessary statements to handle display and selection of the additional options. The programs are carefully documented to facilitate expansion.

6.2.1. Input/output menu selections. The first program expansion which should be investigated is the implementation of all input/output menu selections. At the present time the function key presses are reported to the FORTRAN subroutine, IO. However only a shell of the subroutine exists and is totally nonfunctional.

The most needed output ability is that to print the final control vertex positions on disk or paper. The primary purpose of the program is actually to obtain such a listing, the control vertex positions being an integral part of the B-spline algorithm which defines the modeled surface. Such a listing should include the order of the surface, the homogeneous coordinates, and the knot vectors. These values are readily available and identified within SPLINE.FTN (the FORTRAN portion of SPLINE) and interfacing with the printer should be relatively straightforward.

The variable, KEY, in subroutine IO is equal to the number of the function key which was pressed.

The next input/output menu functions which need to be implemented are the saving to disk and loading from disk of control vertex positions, allowing work in progress to be saved and restarted later at the point from which it was saved. This task will be very similar to that of implementing the print function, except that some effort may be required to avoid having control vertex manipulations for the currently displayed model apply to a new set of vertices when loaded from a disk file.

The most difficult input/output menu option to implement may be the plot function, intended as a screen dump. While the host program is aware of current vertex locations relative to one another, it is not informed of rotations, scaling, global translations, and clipping which are performed by the PS 300. Even if this information were communicated to the FORTRAN program, the programming required to accomplish the matrix transformations in the host-resident program would be sobering. The PS 300 programming language has a function called XFORMDATA, however, which retrieves the transformed data for an object as ASCII data for transmission to the host computer. Using this function, it should be possible to implement a screen dump function without excessive difficulty.

6.2.2. Inversion algorithm. The time required to model a desired surface could be greatly reduce with the

implementation of an inversion algorithm. Such an algorithm would calculate an initial set of control vertex positions, the required surface order, the homogeneous coordinates and the knot vectors which would yield a B-spline surface approximating the surface represented by data in the known-surface data file. The program would branch to such a subroutine almost immediately after program execution, from subroutine INITCP.

Because the B-spline surface does not interpolate the control vertices, developing and implementing a suitable inversion procedure is not a trivial task. Two-dimensional approaches to the problem are presented by Yamaguchi [25] and Wu, Abel, and Greenberg [21]. A more complex (but more immediately applicable) three-dimensional algorithm is presented by Barsky and Greenberg [12,26]. Rogers, Satterfield, and Rodriguez offer yet another three-dimensional approach [6,17], which they describe as "conceptually simpler, but computationally less efficient" than Barsky and Greenberg's. They offer as justification a reminder that the inversion algorithm is used only once during the program, to obtain an initial approximation. Tiller [18] presents some thoughts relevant to approximating surfaces with nonuniform rational B-splines.

It should not be assumed that an inversion algorithm will render interactive modeling obsolete. The algorithm

offered by Rogers, Satterfield, and Rodriguez, for example, does not automatically create hard chines, which must then be added interactively. Even if an inversion algorithm was capable of yielding a B-spline surface which exactly interpolated the known surface data, some interactive manipulation would be required to make the surface suitable. Digitized data from a sculpted surface is almost never fair, having small oscillations due to the digitization process. These oscillations must be removed either by extensive and difficult preprocessing of the data or through interactive manipulation.

6.2.3. Circular arcs. Consideration of Tiller's work with the representation of circular arcs by nonuniform rational B-splines suggests an additional feature useful during interactive modeling. With a suitable inversion algorithm, the user could place several control vertices under program control, identify three points on a desired circular arc, and allow the program to relocate the vertices to yield a circular arc interpolating the three points. Since the manipulation of homogeneous coordinates and of nonuniform knot spacing is not highly intuitive, such an approach is necessary in order to make use of the power of the nonuniform rational B-spline.

6.2.4. Orthogonal plane intersection mapping. The parametric representation of the surface presented by SPLINE is adequate for accurate visualization of the B-

spline surface. For some applications, however, a more traditional representation of waterlines, buttocks lines, and transverse sections is desirable. In order to implement such a display, it is necessary to map the intersection of the B-spline surface with the three basic orthogonal planes.

The PS 300-resident portion of SPLINE, SPLINE.DAT, already has the necessary structure to display the three intersection maps. SPLINE.FTN would have to present the vector lists to the PS 300 with a command of the format:

XY:=VECTOR_LIST ITEMIZED N= . . . .

The vector list would be followed by the sending of an appropriate value to the PS 300 program variable, XYMODE, and the sending of a triggering message to input number one of NEW_MODE. The communication of control graph vector lists in subroutine CPLINE provides a similar example.

Calculation of the vector list should be implemented in subroutine PLANES, which is currently called from subroutine WAIT when the appropriate function key is pressed.

Mapping of the intersection between the planes and the B-spline surface can be difficult, especially when trying to obtain sufficient accuracy for an adequate display. Multiple disconnected contours, such as contours of adjacent peaks, present a particular problem.

Heap [27] presents three algorithms for the production of contour maps of a function defined at the vertices of an irregular triangular mesh. Such a mesh is often used in finite element techniques, particularly for the solution of partial differential equations. Heap's algorithm first creates the triangular mesh, with vertices of the mesh lying on the surface to be mapped. The algorithm then follows the contour from element to element of the mesh. Since straight lines are generated through each mesh element, a very fine mesh is required for an accurate contour, consuming large amounts of memory. SPLINE already approaches the memory limit of the PDP-11.

Satterfield and Rogers [28] suggest using Heap's algorithm to obtain a triangular mesh along the path of the contour, but then present a variation. They use the triangular mesh as a basis for piecewise B-spline calculation of the surface in the region of the mesh. Pairs of surface points separated by a desired tolerance are generated on the B-spline surface and checked for spanning of the contour. When they are found to span the contour, a more exact contour location is determined by interpolating between the two spanning points. This procedure handles properly the problems of correctly ordering the points of a contour and of multiple disconnected contours in a single contour plane.

6.2.5. Reflection about a plane. Like the orthogonal plane intersection mapping, display of a reflection of the model about a plane has been fully implemented in SPLINE.DAT, but remains to be implemented in SPLINE.FTN. Calculation of the vector list would be trivial, requiring only that an offset be added to or deducted from the appropriate vector information already held in memory. Communication of the vector information would be similar to that detailed in section 6.2.4, above. Pressing the appropriate function key already activates a call to subroutine REFLECT.

6.2.6. Calculation of form parameters. The mathematical nature of the surface representation could be made even more useful with automatic calculation of form parameters, such as center of gravity, section areas, or waterplane area. No provision has been made in SPLINE at this time for such a feature. Cruetz and Schubert [29] take this approach one step further yet. They propose allowing the designer to specify initial form parameters, such as a section area curve, block coefficient, and beam to draft ratio, and letting the computer program generate from that an initial body plan.

6.3. Conclusions

SPLINE has demonstrated the tremendous potential which exists for mathematical definition of sculptured surfaces coupled with highly interactive computer

graphics. Dramatic changes in the efficiency with which automobile bodies, aircraft fuselages, and ship hulls are designed and manufactured can be expected in the very near future. Using B-splines, or similar mathematical representations, the definition of a complex sculptured surface will take hours, not days. Modifications to a developed surface will be accomplished in mere minutes, no longer requiring the designer to essentially start over with calculations and fairing of lines.

Not only will surface definitions be achieved more accurately and efficiently, but those definitions will be directly interfaced with computer programs which will evaluate the performance characteristics of the form and with the machinery which will manufacture it. Further effort in the development of accurate inversion algorithms, in user/computer interfacing, and in generation of form parameters from surface definitions will be particularly important.

LIST OF REFERENCES

1. David F. Rogers, and Steven G. Satterfield, "B-Spline Surfaces for Ship Hull Design," Computer Graphics 14 (July 1980):211-217.

2. John S. Letcher, Jr., "Mathematical Hull Design for Sailing Yachts," Chesapeake Sailing Yacht Symposium, n.p.:S.N.A.M.E., 1981.

3. W. Selkirk Owen, "Geometry of the Ship" in Principles of Naval Architecture, pp. 1-53, edited by John P. Comstock, New York:S.N.A.M.E., 1967.

4. David F. Rogers, Francisco Rodriguez, and Steven G. Satterfield, "Computer Aided Ship Design and Numerically Controlled Production of Towing Tank Models," Proceedings, 16th Design Automation Conference, San Diego:n.p.,1979, pp. 25-27.

5. ________, "A Simple CADCAM System for the Design and Construction of Towing Tank Models," Proceedings, 17th Numerical Control Society Conference, n.p., 1980.

6. ________, "Ship Hulls, B-Spline Surfaces, and CAD/CAM," IEEE Computer Graphics and Applications 3 (December 1983):37-45.

7. John S. Letcher, Jr., "Mathematical Hull Design with Fairline/1.5," 13th A.I.A.A. Symposium on Aero/Hydronautics of Sailing, Los Angeles Section Monographs 23, n.p.:A.I.A.A., 1983, pp. 15-27.

8. ________, "Fairline Methods for Computer-Aided Hull Design," Symposium of Southwest Section of S.N.A.M.E., n.p.:S.N.A.M.E., 1984.

9. D. E. Calkins, "Introduction to Computer-Aided Technology," course notes, University of Washington, 1986.

10. I. J. Schoenberg, "Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions," Applied Mathematics 4 (1946):45-99, 112-141.

11. R. F. Riesenfeld, "Applications of B-Spline Approximation to Geometric Problems of Computer-Aided Design," Ph.D. thesis, Syracuse University, 1973.

12. B. A. Barsky and D. P. Greenberg, "Interactive Surface Representation System Using a B-Spline Formulation With Interpolation Capability," Computer-Aided Design 14 (July 1982):187-193.

13. Russel Fish, "Alpha_1: Modeling with Nonuniform Rational B-Splines," IRIS Universe, Spring 1987, pp. 3-6.

14. David F. Rogers and J. Alan Adams, Mathematical Elements for Computer Graphics, New York: McGraw-Hill Book Company, 1976.

15. Richard H. Bartels, John C. Beatty, and Brian A. Barsky, Introduction to the Use of Splines In Freeform Curve and Surface Design: SIGGRAPH 86 Course #4, n.p., 1986.

16. Richard J. Sharpe and Richard W. Thorne, "Numerical Method for Extracting an Arc Length Parameterization from Parametric Curves," Computer Aided Design 14 (March 1982):79-81.

17. David F. Rogers and Steven G. Satterfield, "Dynamic B-Spline Surfaces," Proceedings, Fourth Conference on Computer Applications in the Automation of Shipyard Operation and Ship Design, Annapolis, Maryland: North-Holland Publishing, 1982, pp. 189-196.

18. Wayne Tiller, "Rational B-Splines for Curve and Surface Representation," IEEE Computer Graphics and Applications 3 (September 1983):61-69.

19. National Technical Information Service, Initial Graphics Exchange Specification (IGES), Version 2.0, Document No. PB83-137448, n.d.

20. Evans and Sutherland, PS 300 Computer Graphics System, n.p., n.d.

21. Sheng-Chuan Wu, John F. Abel, and Donald P. Greenberg, "An Interactive Computer Graphics Approach to Surface Representation," Communications of the ACM 20 (October 1977):703-712.

22. Dan Klass, Evans and Sutherland, Software Technical Support Department, Salt Lake City, interview, 7 April 1987.

23. D. E. Calkins, "Introduction to Computer Aided Technology: Lab Book," University of Washington, 1986.

24. Bruce Stannard, Ben Lexcen -- The Man, The Keel and the Cup, London: Faber and Faber, 1984.

25. Fujio Yamaguchi, "A New Curve Fitting Method Using a CRT Computer Display," Computer Graphics and Image Processing 7 (1978):425-437.

26. B. A. Barsky and D. P. Greenberg, "Determining a Set of B-Spline Control Vertices to Generate an Interpolating Surface," Computer Graphics and Image Processing 14 (1980):203-226.

27. B. R. Heap, Algorithms for the Production of Contour Maps over an Irregular Triangular Mesh, National Physical Laboratory Report NAC 10, Teddington, Middlesex: National Physical Laboratory, 1979.

28. Steven G. Satterfield and David F. Rogers, "A Procedure for Generating Contour Lines from a B-Spline Surface," IEEE Computer Graphics and Applications 5 (April 1985):71-75.

29. Gunter Creutz and Christian Schubert, "An Interactive Line Creation Method Using B-Splines," Computers and Graphics 5 (1980):71-78.

APPENDIX A

FLOW CHARTS AND FUNCTION NETWORKS

The figures contained in this appendix present flow charts and function networks for the major program functions of both SPLINE.FTN (the host-resident program) and SPLINE.DAT (the PS 300-resident program). The program listings are given in appendices B and C. The comments contained in those listings will be particularly useful in understanding the function networks. Those who are unfamiliar with the PS 300 programming language will want to consult the computer manual [20] for an explanation of display trees and function networking and for details concerning the individual function blocks.

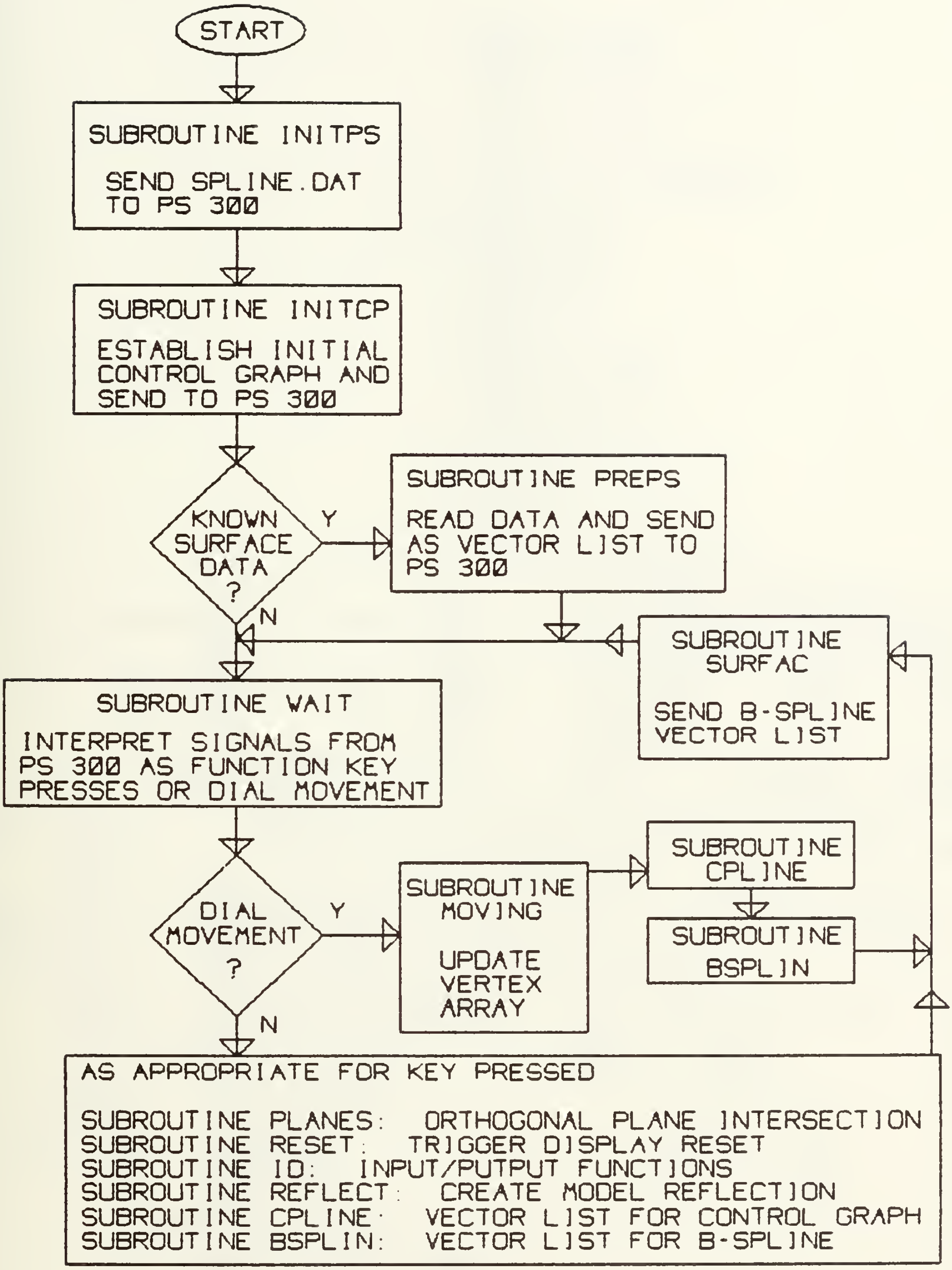

Fig. 45. SPLINE.FTN flow chart.

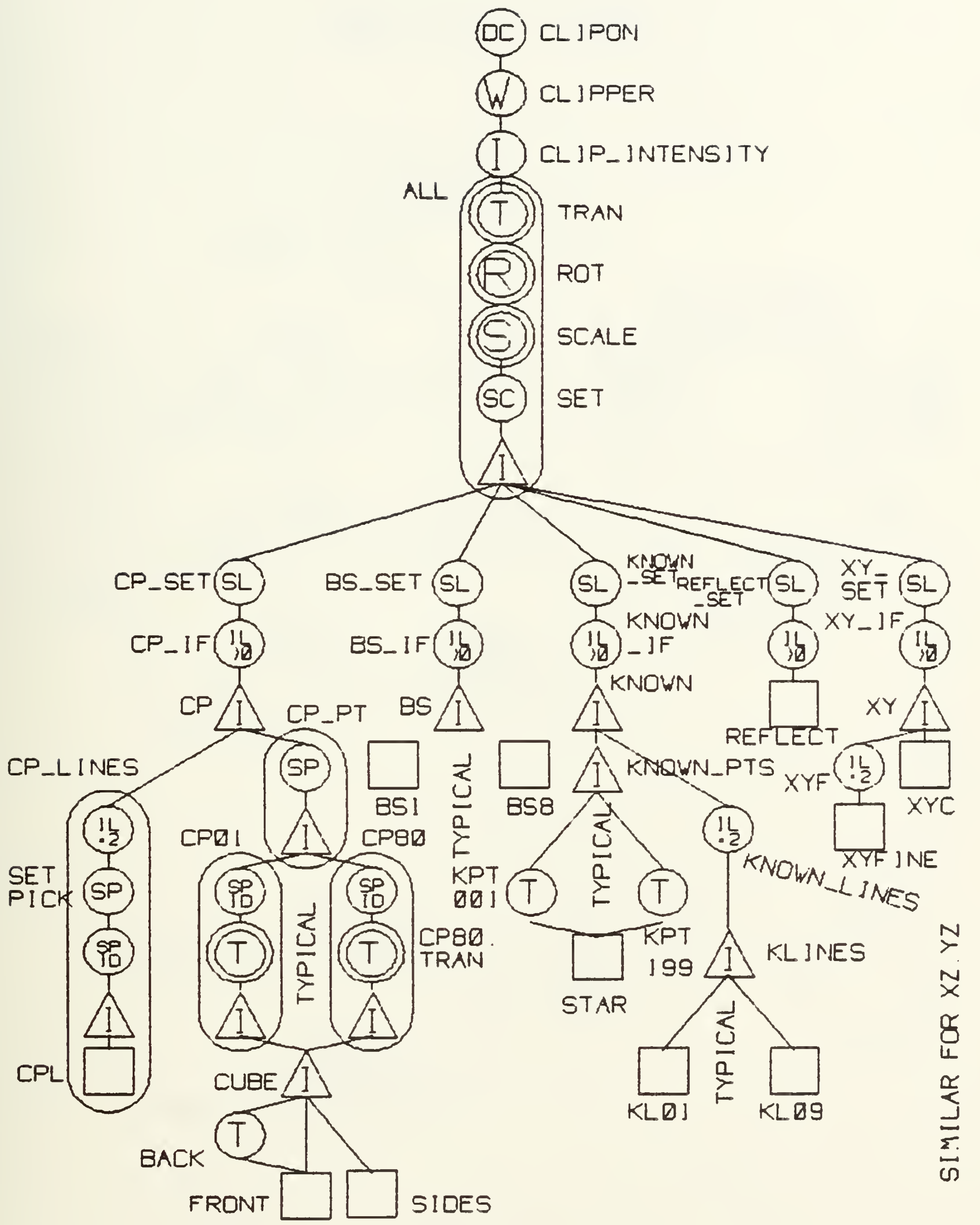

Fig. 46. SPLINE.DAT display tree.

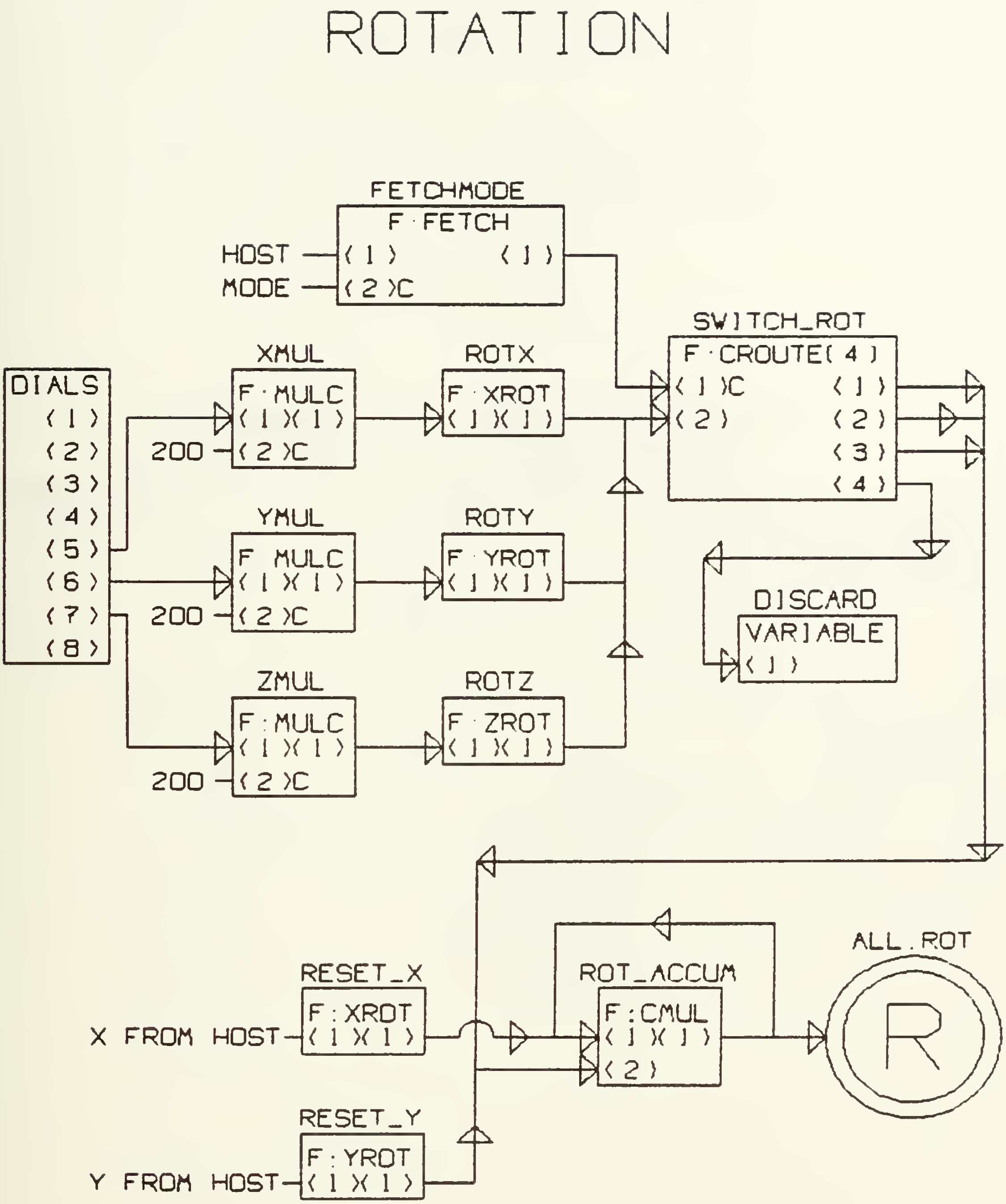

Fig. 47. Rotation function network.

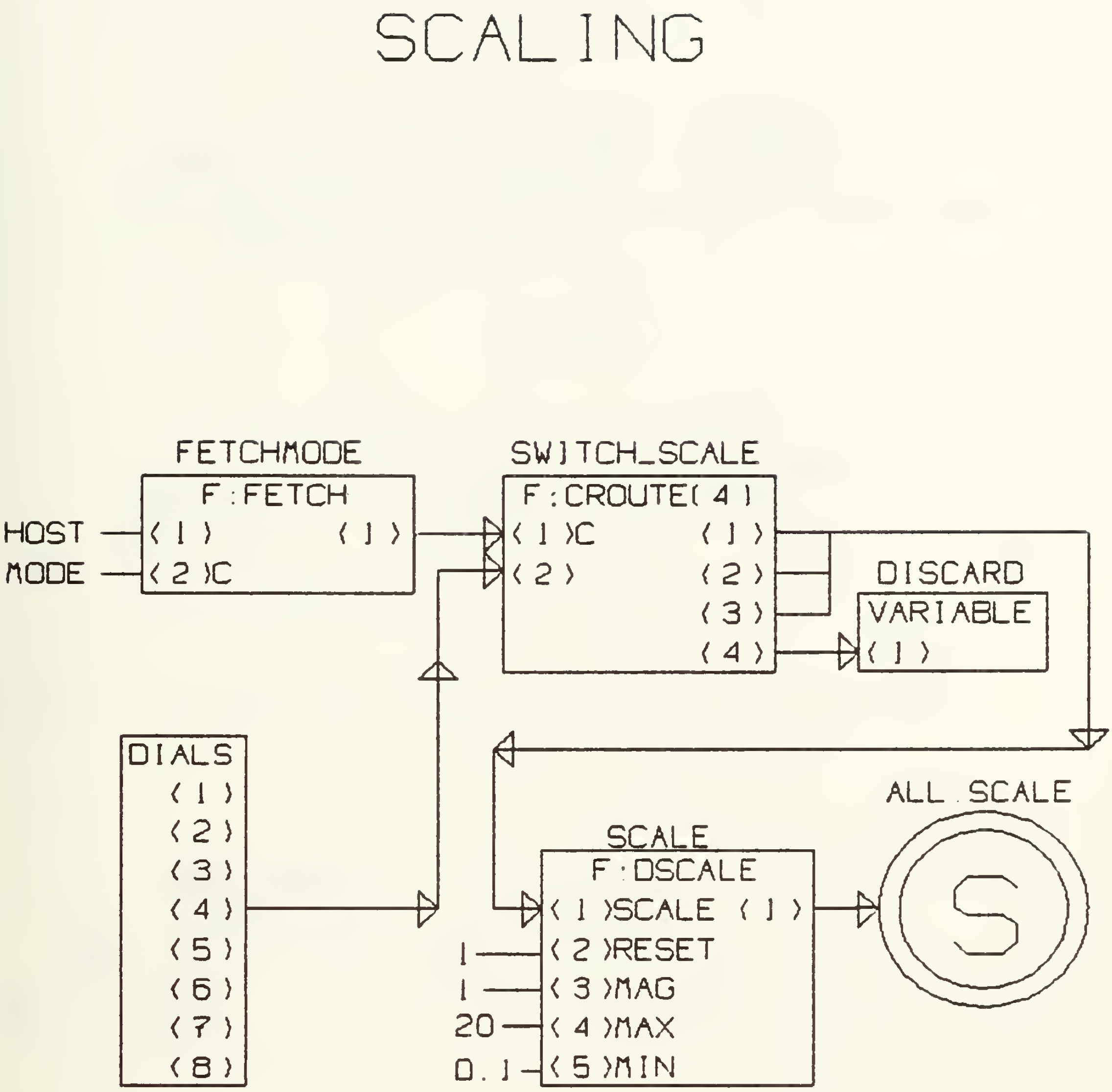

Fig. 48. Scaling function network.

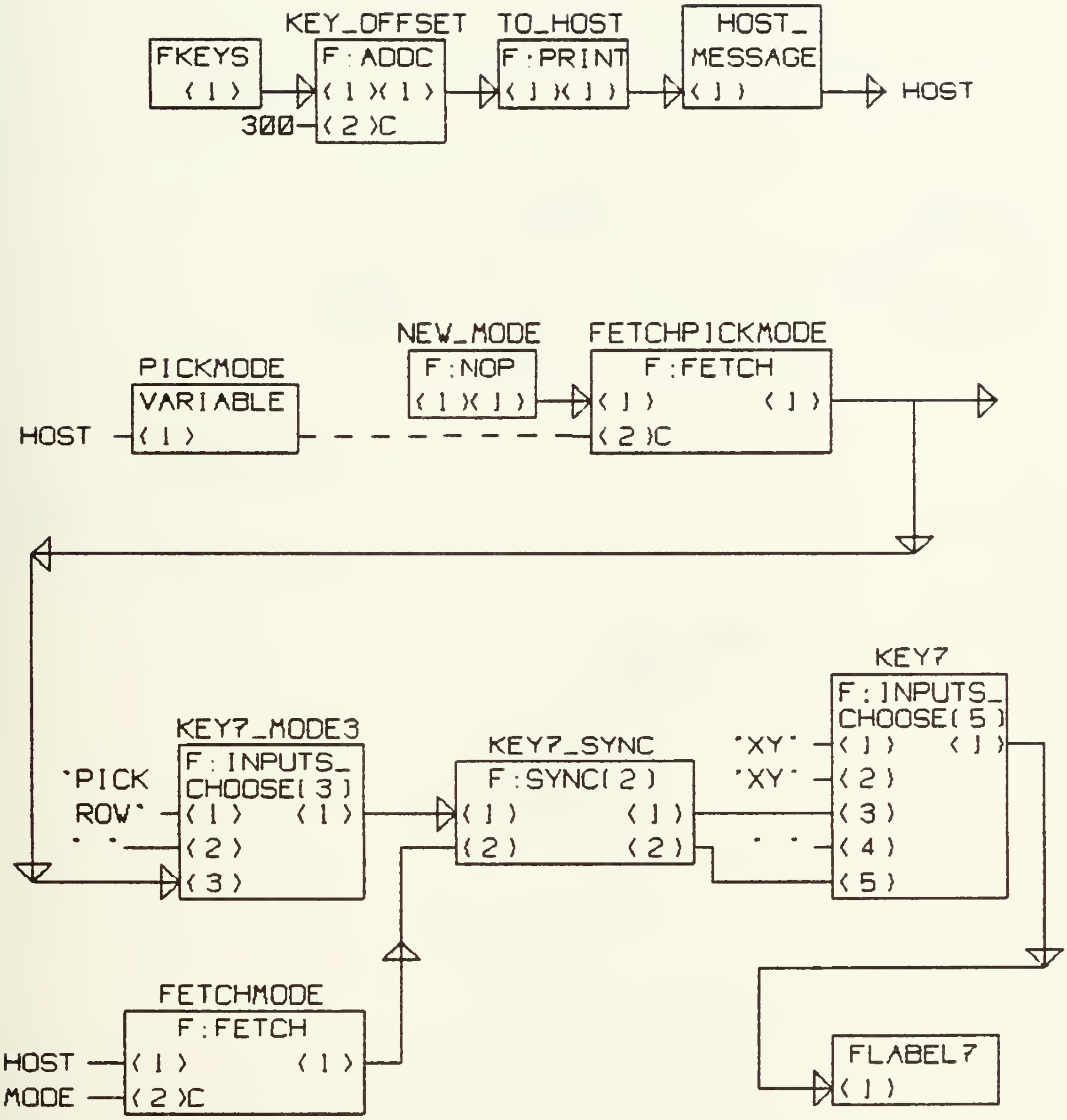

Fig. 49. Typical function network for function key mode selection and labeling.

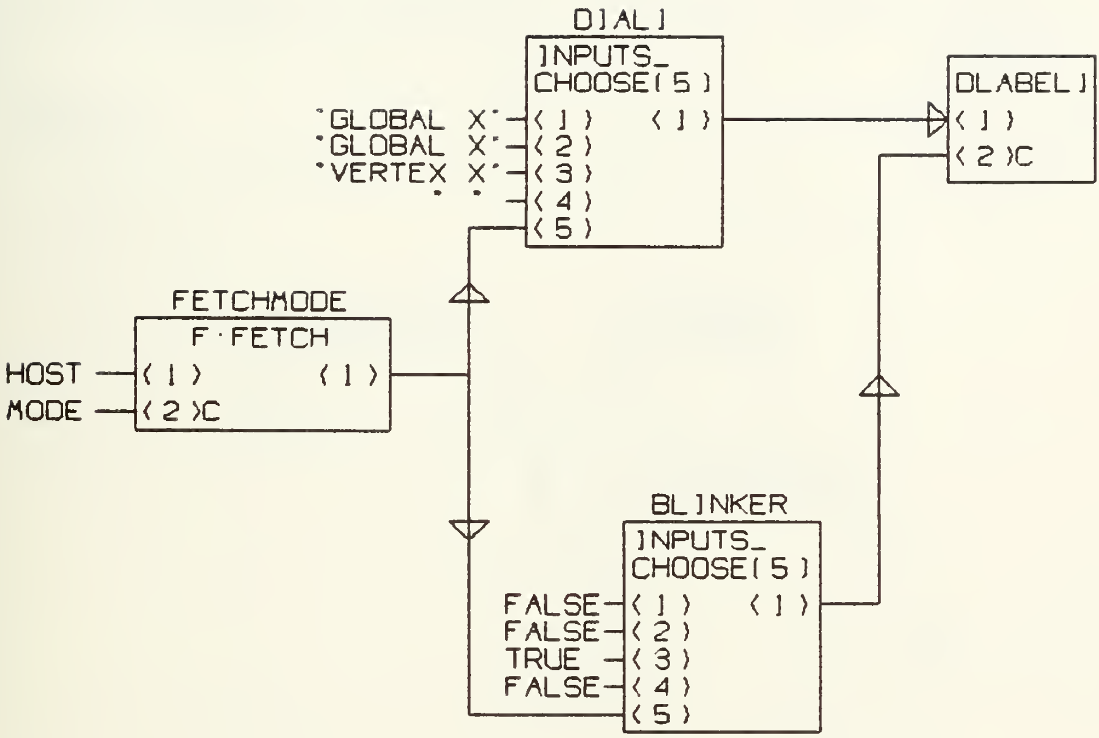

Fig. 50. Function network for blinking of translation dials when in interactive modeling mode.

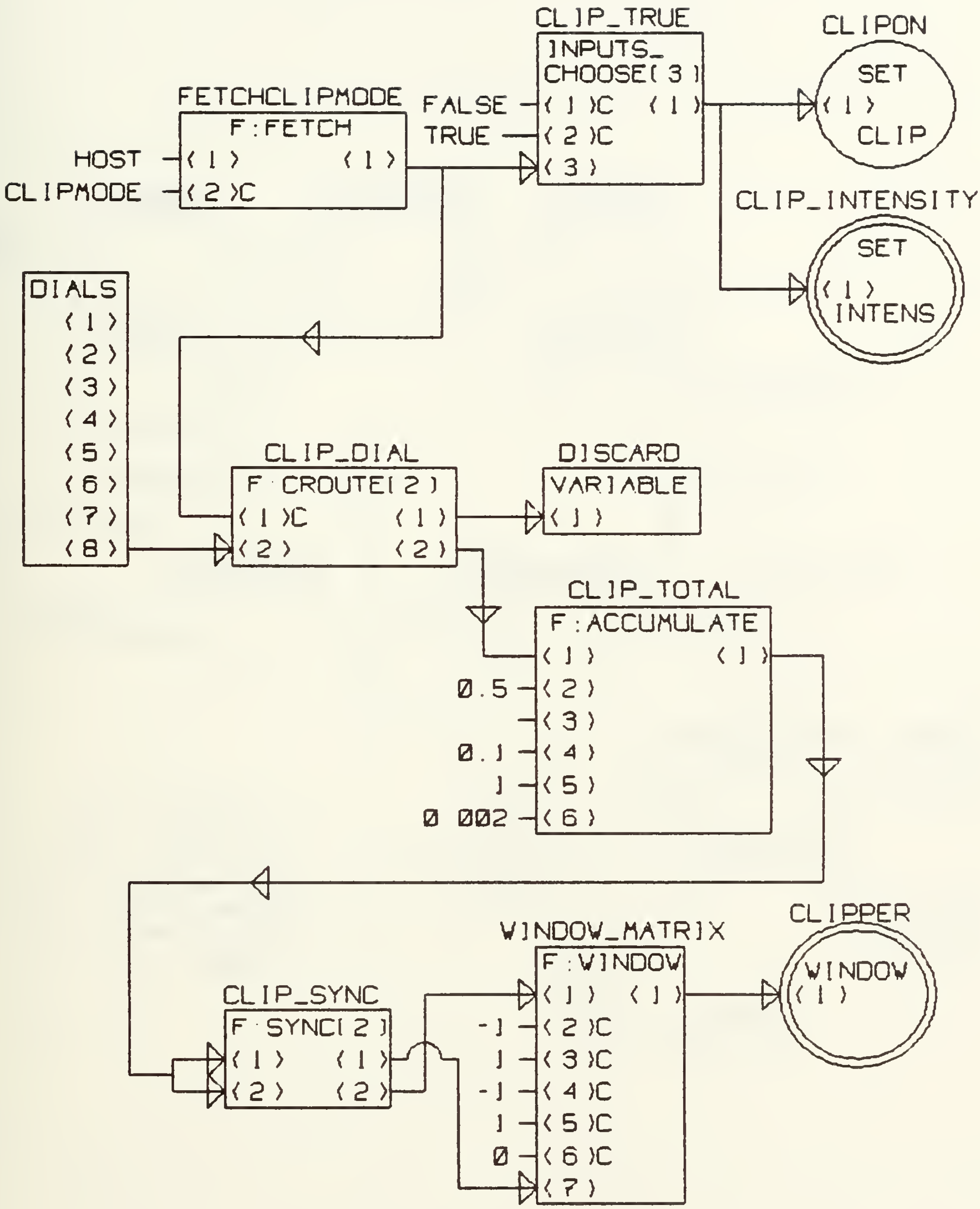

Fig. 51. Clipping function network.

PICKING

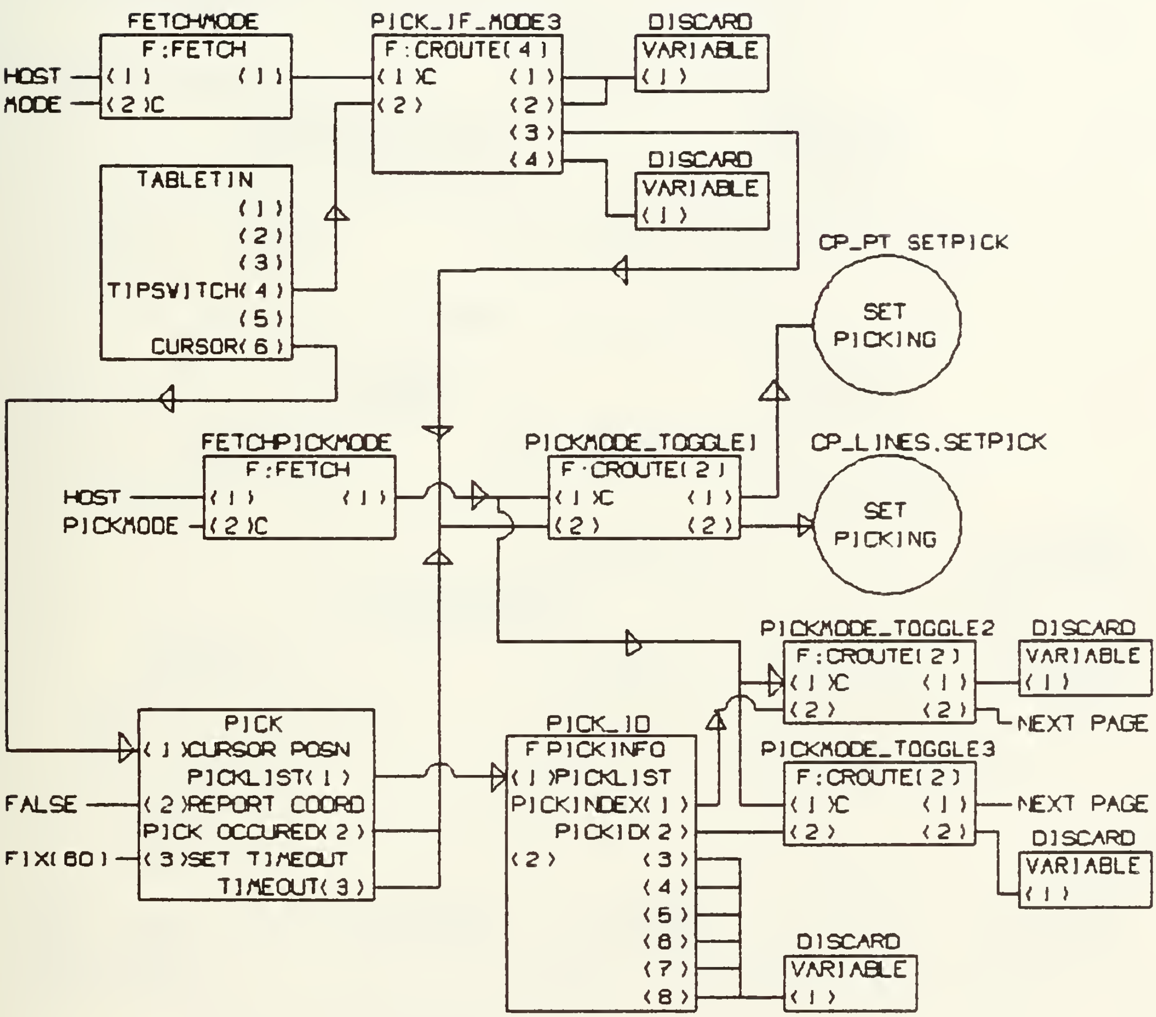

Fig. 52. Picking function network.

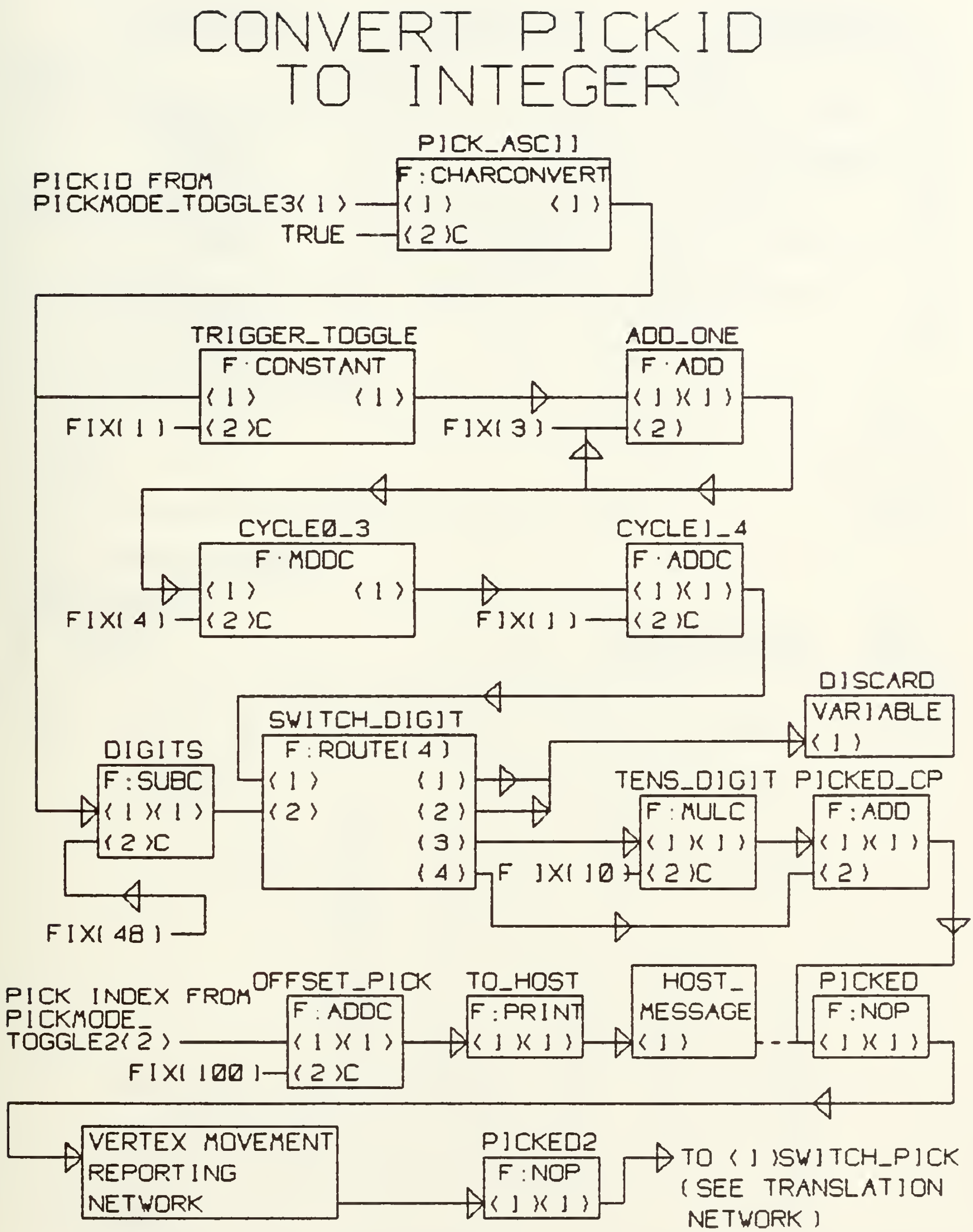

Fig. 53. PICKID conversion function network.

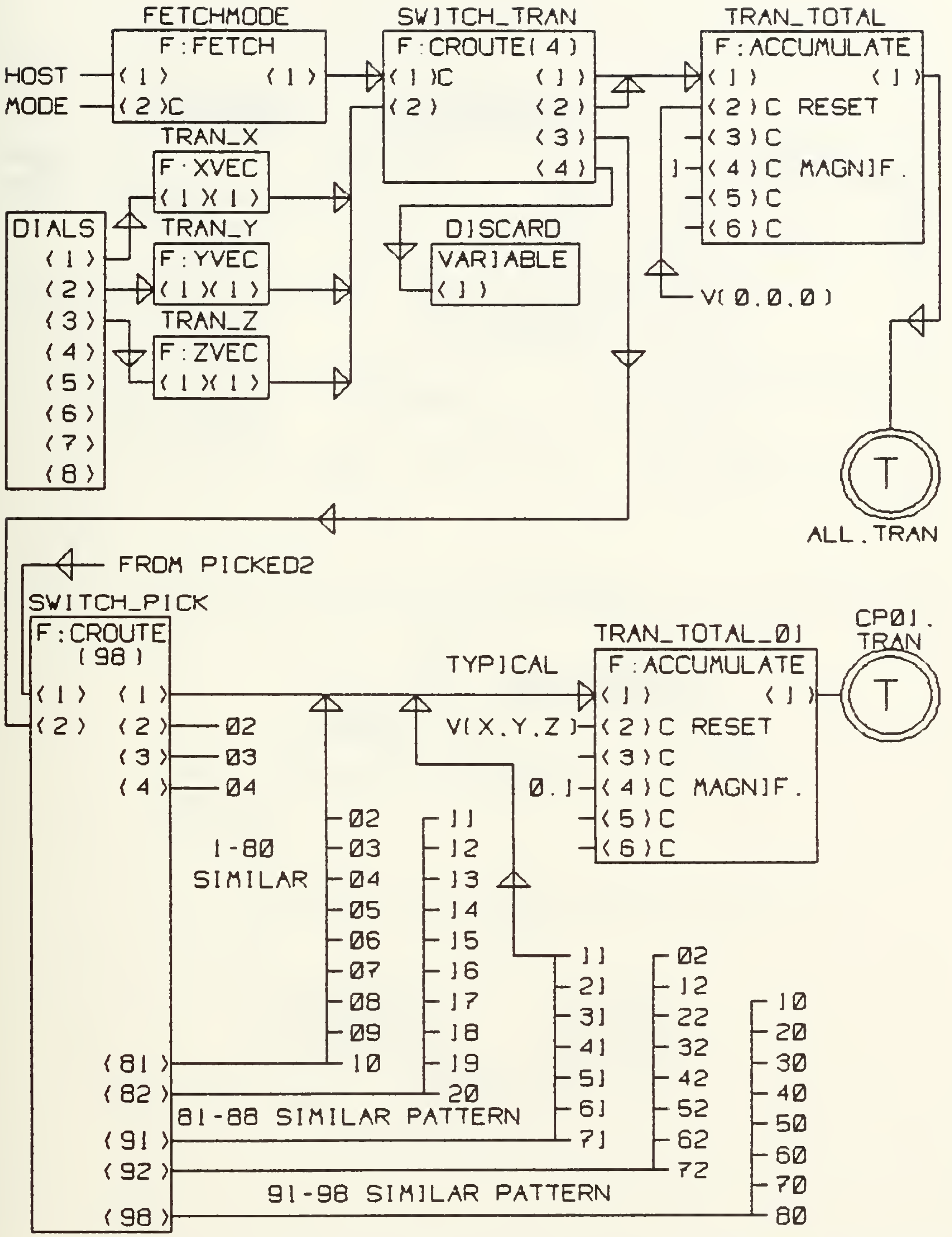

Fig. 54. Function network for global translation and translation of selected vertex during interactive modeling.

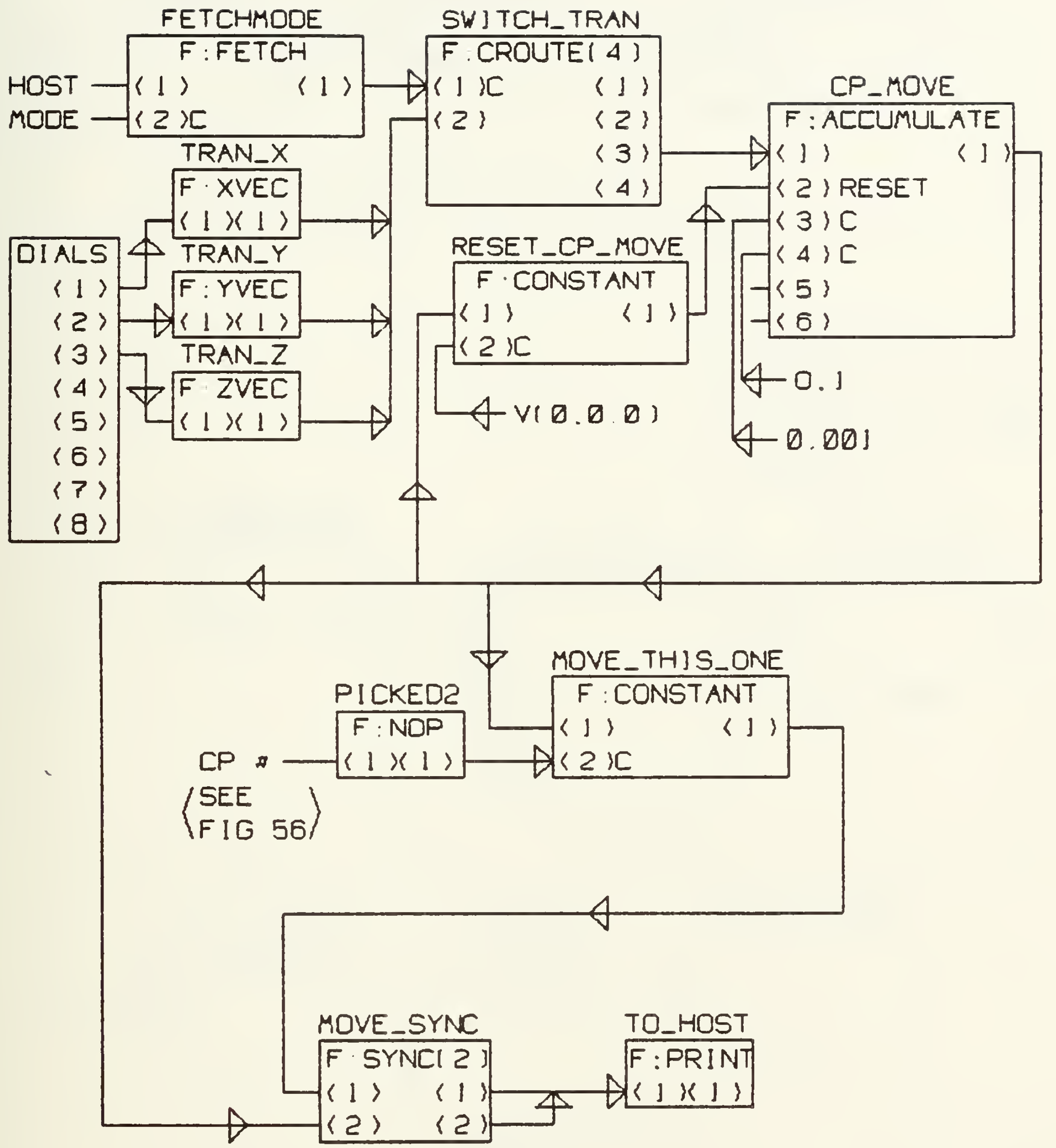

Fig. 55. Function network for reporting of vertex movement to FORTRAN program.

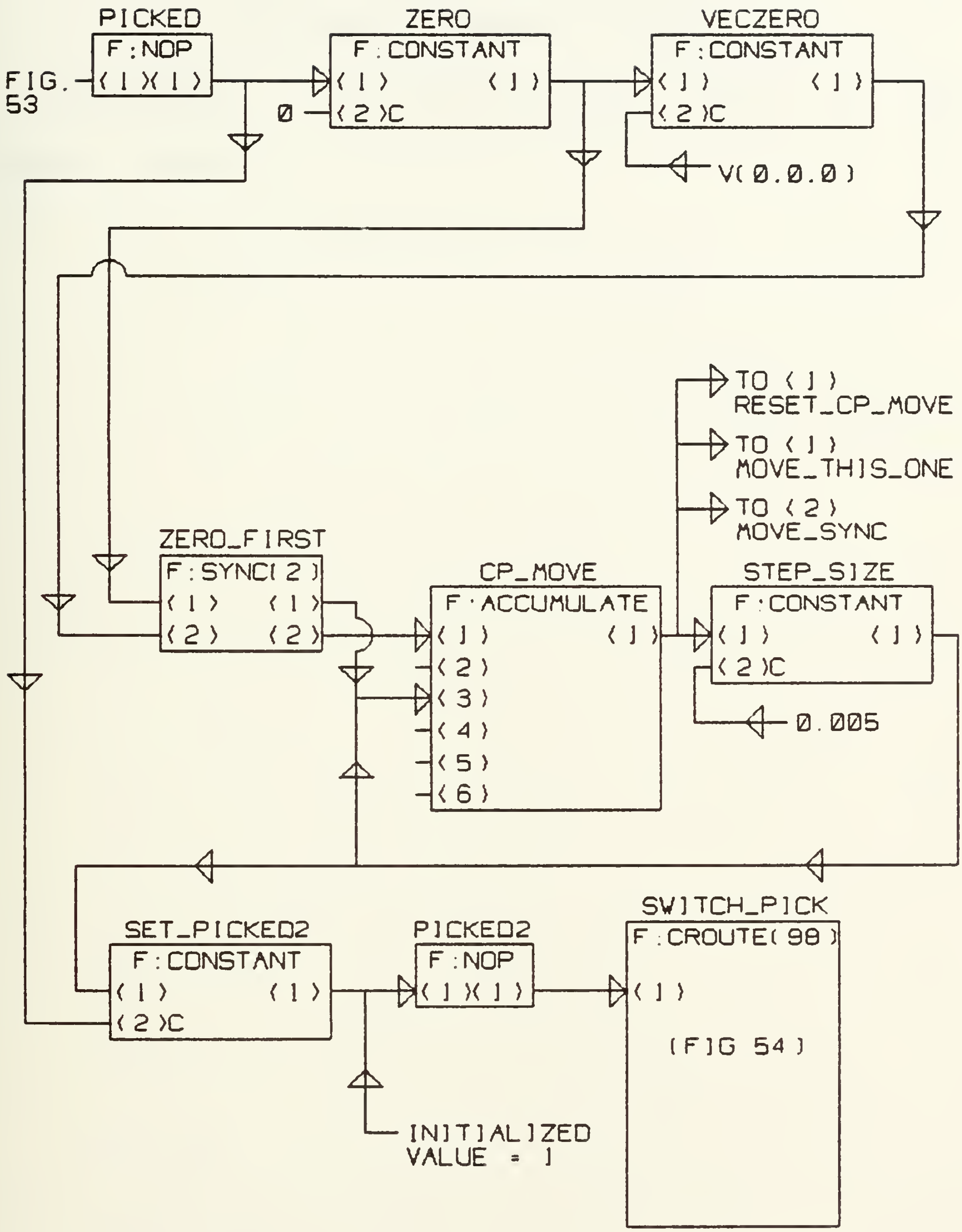

Fig. 56. Function network to reset vertex movement accumulator when new vertex is picked.

APPENDIX B

SPLINE.FTN PROGRAM LISTING

SPLINE.FTN is the host-resident portion of the program, SPLINE.

```
*****************************************************************
*****************************************************************
******************** Program SPLINE.FTN *************************
*****************************************************************
*****************************************************************
*                                                               *
*Input       :                                                  *
*Output      :                                                  *
*Calls       : INITPS, PSEXIT,INITCP,WAIT                       *
*Alters      :                                                  *
*Description: This program facilitates three-dimensional        *
*              surface modeling or design using a surface       *
*              B-spline algorithm. The surface is displayed*
*              and interactively manipulated and modified       *
*              in real time on the PS-300 vector refresh        *
*              display.  The file, "SPLINE.DAT," contains       *
*              the  PS-300 command language program which       *
*              produces and controls the graphics display       *
*              using data supplied by SPLINE.FTN.               *
*Author:     : D. Michael Bryant                                *
*Created On : 5/21/87            Modified On:                   *
*                                                               *
*****************************************************************
*Variable    Type      Explanation                              *
*---------   -------   -----------                              *
*ISTAT(1)    Integer   Reports error status from PS-300         *
*****************************************************************

      PROGRAM SPLINE

      COMMON /SPLINE/ MCP, NCP, CP(10,8,4), MORDER, NORDER
      COMMON /BASIS/ R(13,4,31), S(11,4,31), SURF(3,31,31)
      COMMON /VECS/ VECS(1000)
      DIMENSION ISTAT(1)

***** Send PS-300 command language program to PS-300 and
***** execute.
      CALL INITPS

***** Establish initial control point locations.
      CALL INITCP

***** Await and act upon signals from PS-300 program.
      CALL WAIT

***** De-establish link between PS-300 and PDP-11, using
***** library subroutine.
      ISTAT(1)=0
      CALL PSEXIT(1,ISTAT)

      STOP
      END
```

```
*****************************************************************
******** Subroutines Follow in Alphabetical Order ********
*****************************************************************

*****************************************************************
*****************************************************************
******************** Subroutine BLANK ********************
*****************************************************************
*****************************************************************
*                                                               *
*Input       : I, BUFFER                                        *
*Output      : BUFFER                                           *
*Calls       :                                                  *
*Alters      : BUFFER                                           *
*Description: This program makes extensive use of the           *
*              'ENCODE' command to convert integers to          *
*              characters in a LOGICAL*1 array ('BUFFER').      *
*              The buffer is then passes to the PS-300,         *
*              with the encoded integers forming portions       *
*              of the names of nodes, I.E. 'CP23'.  When a      *
*              single-digit integer is encoded with an 'I2'*
*              specification, however, the result is not       *
*              '03', for example, but ' 3'.  The blank          *
*              would not be allowed as part of a node name.*
*              This subroutine replaces the unwanted blanks*
*              with "0's".                                      *
*Author      : D. Michael Bryant                                *
*Created On  : 11/17/86              Modified On:               *
*                                                               *
*****************************************************************
*                                                               *
*Variable    Type      Explanation                              *
*---------   -------   -----------                              *
*BUFFER(80)  Logical   Contains the string to be sent to        *
*                      the PS-300.                              *
*I           Integer   Identifies the array element which       *
*                      contains the blank to be replaced.       *
*                                                               *
*****************************************************************

      SUBROUTINE BLANK(I,BUFFER)

      LOGICAL*1 BUFFER(80)

         IF (BUFFER(I).EQ.' ') BUFFER(I)='0'

      RETURN
      END
```

```
****************************************************************
****************************************************************
****************** Subroutine BSPLIN ****************************
****************************************************************
****************************************************************
*                                                              *
*Input       : MCP,NCP,CP(10,8,4),MORDER,NORDER                *
*Output      : R(13,4,31),S(11,4,31),UKNOT(15),WKNOT(13),      *
*              U(31),W(31)                                      *
*Calls       : PSSEND, SURFAC                                   *
*Alters      :                                                  *
*Description: This subroutine calculates a full B-spline        *
*              surface.  Subroutine SURFAC is then called       *
*              to extract and send a vector list to the         *
*              PS300 for display of the calculated surface.*
*Author:     : D. Michael Bryant                                *
*Created On : 2/22/87              Modified On:                 *
*                                                              *
****************************************************************
*                                                              *
*Variable    Type      Explanation                              *
*--------    -------   -----------                              *
*A           Real      Denominator of the first term of the *
*                      basis function.                          *
*B           Real      First term of the basis function.       *
*C           Real      Denominator of the second term of the*
*                      basis function.                          *
*CP(10,8,4) Real       Holds the X,Y,Z coordinates for each *
*                      of the control points (vertices).        *
*                      Subscripts denote row, column of         *
*                      vertex and X,Y,Z or homogeneous          *
*                      component.                               *
*D           Real      Second term of the basis function.      *
*MCP         Integer   Number of rows of vertices in the        *
*                      B-spline control polygon.                *
*MORDER      Integer   The order of the B-spline surface in *
*                      the longitudinal (u) direction.          *
*NCP         Integer   Number of columns of vertices in the *
*                      B-spline control polygon.                *
*NORDER      Integer   The order of the B-spline surface in *
*                      the transverse (w) direction.            *
*R(13,4,31) Real       The basis functions in the longitud- *
*                      inal direction.  The first subscript *
*                      indicates the associated vertex row, *
*                      the second indicates the order, and  *
*                      the third indicates the number of        *
*                      parametric increments along the curve*
*                      (corresponding to the "u" value).        *
*S(11,4,31) Real        The basis functions in the tranverse*
*                      direction.  The first subscript indi-*
*                      cates the associated vertex column,  *
*                      the second indicates the order, and  *
```

```
*                           the third indicates the number of    *
*                           parametric increments along the curve*
*                           (corresponding to the "w" value).    *
*SURF(3,31,31)              The calculated points on the B-spline*
*             Real          surface, in cartesian coordinates, as*
*                           a function of the parametric         *
*                           coordinates.                         *
*U(31)        Real          The parametric value in the longitud-*
*                           inal (u) direction at each of 31     *
*                           increments along the curve.          *
*UINC         Real          Parametric spacing of the 31 "u"     *
*                           lines.                               *
*UKNOT(15)    Real          The knot vector in the longitudinal  *
*                           (u) direction.                       *
*UMAX         Real          The maximum value in the "u" knot    *
*                           vector.                              *
*W(31)        Real          The parametric value in the trans-   *
*                           verse (w) direction at each of 31    *
*                           increments along the curve.          *
*WINC         Real          Parametric spacing of the 31 "w"     *
*                           lines.                               *
*WKNOT(13)    Real          The knot vector in the transverse (w)*
*                           direction.                           *
*WMAX         Real          The maximum value in the "w" knot    *
*                           vector.                              *
******************************************************************

      SUBROUTINE BSPLIN

      COMMON /SPLINE/ MCP, NCP, CP(10,8,4), MORDER, NORDER
      COMMON /BASIS/ R(13,4,31), S(11,4,31), SURF(3,31,31)
      DIMENSION UKNOT(15), WKNOT(13), U(31), W(31)

***** Blink "B-SPLINE" label on keyboard while calculating.
      CALL PSSEND (24,'SEND TRUE TO <2>FLABEL2;')
      CALL PSSEND (22,'SEND 1 TO <1>NEW_MODE;')

***** Determine knot vectors (default = uniform).

      UMAX = REAL(MCP-MORDER+1)
      WMAX = REAL(NCP-NORDER+1)

      DO 150 K=1,MCP+MORDER
           IF (K.LE.MORDER) UKNOT(K)=0.0
           IF (K.GT.MORDER.AND.K.LE.MCP) UKNOT(K) =
     +                                       UKNOT(K-1)+1.0
           IF (K.GT.MCP) UKNOT(K) = UMAX
150   CONTINUE
```

```
      DO 160 K = 1,NCP+NORDER
           IF (K.LE.NORDER) WKNOT(K) = 0.0
           IF (K.GT.NORDER.AND.K.LE.NCP) WKNOT(K) =
     +                                   WKNOT(K-1)+1.0
           IF (K.GT.NCP) WKNOT(K) = WMAX
160   CONTINUE

***** Determine parametric spacing of displayed B-spline
***** surface net lines.

      UINC = UMAX/30.0
      WINC = WMAX/30.0

      DO 200 K = 1,31
           U(K) = (REAL(K-1))*UINC
           W(K) = (REAL(K-1))*WINC
200   CONTINUE

***** Determine first order basis functions.

      DO 250 M = 1,MCP+MORDER-1
           DO 240 K = 1,31
                IF(UKNOT(M).LE.U(K).AND.U(K).LT.UKNOT(M+1))
     +                     THEN
                     R(M,1,K) = 1.0
                ELSEIF (UKNOT(M).EQ.U(K)) THEN
                     R(M,1,K) = 1.0
                ELSEIF (K.EQ.31.AND.M.EQ.MCP) THEN
                     R(M,1,K) = 1.0
                ELSE
                     R(M,1,K) = 0.0
                ENDIF
240        CONTINUE
250   CONTINUE

      DO 270 N = 1,NCP+NORDER-1
           DO 260 K = 1,31
                IF(WKNOT(N).LE.W(K).AND.W(K).LT.WKNOT(N+1))
     +            THEN
                     S(N,1,K) = 1.0
                  ELSEIF (WKNOT(N).EQ.W(K)) THEN
                     S(N,1,K) = 1.0
                  ELSEIF (K.EQ.31.AND.N.EQ.NCP) THEN
                     S(N,1,K) = 1.0
                  ELSE
                     S(N,1,K) = 0.0
                ENDIF
260        CONTINUE
270   CONTINUE
```

```
***** Calculate higher order basis functions.

      DO 350 K = 2,MORDER
            DO 340 M = 1,MCP+MORDER-K
                  DO 330 I = 1,31

                        A = UKNOT(M+K-1) - UKNOT(M)
                        IF (A.EQ.0.0) THEN
                              B = 0.0
                          ELSE
                              B = (U(I)-UKNOT(M))*R(M,K-1,I)/A
                        ENDIF

                        C = UKNOT(M+K) - UKNOT(M+1)
                        IF (C.EQ.0.0) THEN
                              D = 0.0
                          ELSE
                            D=(UKNOT(M+K)-U(I))*R(M+1,K-1,I)/C
                        ENDIF

290                     R(M,K,I) = B + D
330               CONTINUE
340         CONTINUE
350   CONTINUE

      DO 450 K = 2,NORDER
            DO 440 N = 1,NCP+NORDER-K
                  DO 430 I = 1,31

                        A = WKNOT(N+K-1) - WKNOT(N)
                        IF (A.EQ.0.0) THEN
                              B = 0.0
                          ELSE
                              B = (W(I)-WKNOT(N))*S(N,K-1,I)/A
                        ENDIF

380                     C = WKNOT(N+K) - WKNOT(N+1)
                        IF (C.EQ.0.0) THEN
                              D = 0.0
                          ELSE
                            D=(WKNOT(N+K)-W(I))*S(N+1,K-1,I)/C
                        ENDIF

390                     S(N,K,I) = B + D
430               CONTINUE
440         CONTINUE
450   CONTINUE
```

```
***** Calculate the surface coordinates.

      DO 800 I = 1,MCP
           DO 790 J = 1,NCP
                DO 780 L = 1,31
                     IF (R(I,MORDER,L).EQ.0.0) GOTO 780
                     DO 770 M = 1,31
                          IF(S(J,NORDER,M).EQ.0.0) GOTO 770
                          DO 760 K = 1,3
                               SURF(K,L,M) = SURF(K,L,M) +
     +                                       CP(I,J,K) *
     +                                       CP(I,J,4) *
     +                                       R(I,MORDER,L)*
     +                                       S(J,NORDER,M)
760                       CONTINUE
770                  CONTINUE
780             CONTINUE
790        CONTINUE
800   CONTINUE

***** Send the surface coordinates to the PS-300.

      CALL SURFAC

***** Stop blinking "B-SPLINE" label on keyboard.
      CALL PSSEND (25,'SEND FALSE TO <2>FLABEL2;')
      CALL PSSEND (22,'SEND 1 TO <1>NEW_MODE;')

      RETURN
      END
```

```
*******************************************************************
*******************************************************************
******************** Subroutine CPLINE ****************************
*******************************************************************
*******************************************************************
*                                                                 *
*Input       : MCP,NCP,CP(10,8,4)                                 *
*Output      : VECS(500),LP(333), (vector list defining           *
*              lines connecting vertices)                         *
*Calls       : PSSEND,PSVECS                                      *
*Alters      : SPLINE.DAT structure "CP_LINES"                    *
*Description: This subroutine prepares in the VECS array a*
*              vector list defining the lines connecting          *
*              the B-spline vertices (control points).  The*
*              vector list is then passed to the PS-300.          *
*Created On : 12/09/86            Modified On:                    *
*                                                                 *
*******************************************************************
*Variable    Type       Explanation                               *
*---------   -------    -----------                               *
*BUFFER(80) Logical    Holds a message for transmital to the*
*                       PS-300 program with the PSSEND            *
*                       library subroutine.                       *
*CP(10,8,4) Real       Holds the X,Y,Z coordinates for each *
*                       of the control points (vertices).         *
*                       Subscripts denote row, column of          *
*                       vertex and X,Y,Z or homogeneous           *
*                       component.                                *
*I           Integer    Indicates the X,Y, or Z component of *
*                       the vertex vector is under                *
*                       consideration.                            *
*ILP         Integer    Counter indicates the current array       *
*                       element of LP.                            *
*IOSTAT      Integer    Input/output error status.                *
*IVEC        Integer    Counter indicates the current array       *
*                       element of VECS.                          *
*LP(333)     Logical    .TRUE. indicates corresponding            *
*                       elements of VECS represent a line         *
*                       vector (draw to this point with pen       *
*                       down).  .FALSE. indicates                 *
*                       corresponding elements of VECS            *
*                       represent a point vector (move to         *
*                       this point with pen up).                  *
*M           Integer    Row of vertices under consideration. *
*MCP         Integer    Number of rows of vertices in the         *
*                       B-spline control polygon.                 *
*N           Integer    Column of vertices.                       *
*NCP         Integer    Number of columns of vertices in the *
*                       B-spline control polygon.                 *
*VECS(500)   Real       Vector list defining the lines            *
*                       connecting the B-spline vertices.         *
*******************************************************************
```

```
      SUBROUTINE CPLINE

      COMMON /VECS/ VECS(1000)
      LOGICAL LP(333)
      COMMON /SPLINE/ MCP,NCP,CP(10,8,4),MORDER,NORDER
      LOGICAL*1 BUFFER(80)

      IVEC=0
      ILP=0

***** Vector list for lines along rows of vertices.

      DO 100 M=1,MCP
           DO 90 N=1,NCP
*               Count the vectors created.
                ILP=ILP+1
                DO 80 I=1,3
*                   Count the vector components (X,Y,Z)
*                       created.
                    IVEC = IVEC + 1
                    VECS(IVEC) = CP(M,N,I)
80              CONTINUE

********** Indicate whether vector is point or line vector.

                LP(ILP) = .TRUE.
                IF (N.EQ.1) LP(ILP) = .FALSE.

90         CONTINUE
100   CONTINUE

***** Vector list for lines along columns of vertices.

      DO 200 N=1,NCP
           DO 190 M=1,MCP
                ILP=ILP+1
                DO 180 I=1,3
                    IVEC = IVEC + 1
                    VECS(IVEC) = CP(M,N,I)
180             CONTINUE

********** Indicate whether vector is point or line vector.

                LP(ILP) = .TRUE.
                IF (M.EQ.1) LP(ILP) = .FALSE.

190        CONTINUE
200   CONTINUE
```

```
***** Send vector list to PS-300 program to define
***** substructure CPL of structure CP_LINES.

      ENCODE (31,210,BUFFER) ILP
210        FORMAT ('CPL:=VECTOR_LIST ITEMIZED N=',I3)
      CALL PSSEND (31,BUFFER)
      CALL PSVECS (4,ILP,VECS,LP,.TRUE.,IOSTAT)

      RETURN
      END
```

```
***********************************************************
***********************************************************
******************* Subroutine INITCP *******************
***********************************************************
***********************************************************
*                                                         *
*Input       : RATIO,MCP,NCP                              *
*Output      : CP(10,8,4)                                 *
*Calls       : PSSEND,INVERT,BLANK,PREPS                  *
*Alters      : SPLINE.DAT                                 *
*Description: This subroutine determines an initial set   *
*              of control points (vertices) and passes    *
*              commands to the PS-300 which create the    *
*              associated display tree branches.          *
*Created On  : 11/14/86           Modified On:            *
*                                                         *
***********************************************************
*                                                         *
*Variable    Type     Explanation                         *
*---------   -------  -----------                         *
*ANSWER(1)   Char     Retrieves (Y)es and (N)o answers.   *
*CP(10,8,4)  Real     Holds the X,Y,Z coordinates for each *
*                     of the control points (vertices).   *
*                     Subscripts denote row, column of    *
*                     vertex and X,Y,Z or homogeneous     *
*                     component.                          *
*DM          Real     The X increment between initial     *
*                     control point positions.            *
*DN          Real     The Y increment between initial     *
*                     control point positions.            *
*MCP         Integer  The number of control points in the *
*                     X direction in the initial control  *
*                     point matrix.                       *
*MORDER      Integer  The order of the B-spline surface in *
*                     the longitudinal (u) direction.     *
*NCP         Integer  The number of control points in the *
*                     Y direction in the initial control  *
*                     point matrix.                       *
*NORDER      Integer  The order of the B-spline surface in *
*                     the transverse (w) direction.       *
*NUMCP       Integer  The identifying number of the control*
*                     point under consideration, 1 -- 80, *
*                     where point (1,1) is 1, (1,2) is 2, *
*                     ...,(10,8) is 80.                   *
*RATIO       Real     The length to width ratio of the    *
*                     initial control point matrix.       *
*YMIN        Real     The minimum Y coordinate of the     *
*                     initial control point matrix.       *
*                                                         *
***********************************************************
```

```
      SUBROUTINE INITCP

      COMMON /SPLINE/ MCP, NCP, CP(10,8,4), MORDER, NORDER
      CHARACTER*1 ANSWER
      LOGICAL*1 BUFFER(80)

***** Determine whether it is desired to display a known
***** surface.

50    TYPE 60
60        FORMAT(///,' Do you wish to display a known '
     +            'surface from a PREPS-format file?',/,
     +            '  Please answer (Y) or (N).',/)
      READ(5,70,ERR=50) ANSWER
70            FORMAT (A1)
          IF (ANSWER.EQ.'Y') GOTO 100
          IF (ANSWER.EQ.'N') GOTO 200
          GOTO 50

***** Allow input of coordinates of points on a known
***** surface from a PREPS-format file.

100   CALL PREPS

***** Establish size and length/width ratio of initial
***** control point matrix.

200   TYPE 201
201       FORMAT(//' A matrix of vertices up to (10 X 8)',
     +            ' may be selected.',/,
     +            '  ENTER THE DESIRED MATRIX SIZE.',/,
     +            '  (Number of transverse, longitudinal ',
     +            '  stations.)',/,
     +            '  Example: 7,5',/)

210   READ (5,*,ERR=200) MCP,NCP
          IF (MCP.GT.10.OR.NCP.GT.8) THEN
               TYPE 220
220                FORMAT (/,' The maximum value for the',
     +                      ' matrix is 10,8.'/,
     +                      ' ENTER BOTH VALUES AGAIN.')
               GOTO 210
          ELSEIF (MCP.LT.5.OR.NCP.LT.5) THEN
               TYPE 230
230                FORMAT (/,' The minimum value for the',
     +                      ' matrix is 5,5.'/,
     +                      ' ENTER BOTH VALUES AGAIN.')
               GOTO 210
          ENDIF
```

```
235   TYPE 240
240       FORMAT(///,' Enter the desired length/width ',/,
     +          ' ratio for the vertices matrix.',/)

      READ (5,*,ERR=235) RATIO
          IF (RATIO.LE.0) THEN
              TYPE 250
250               FORMAT(/,' THE LENGTH/WIDTH RATIO MUST',
     +                   ' BE POSITIVE.')
              GOTO 235
          ENDIF

***** Determine the order of the B-spline surface equation.

      TYPE 265
265        FORMAT (//,' Enter the order of the B-spline',/,
     +            ' surface in the longitudinal direction.')
270   TYPE 275
275        FORMAT (' Use integer values of 3 or 4.')
      READ (5,*,ERR=270) MORDER
      IF (MORDER.LT.3.OR.MORDER.GT.4) GOTO 270
      TYPE 280
280        FORMAT (//,' Enter the order of the B-spline',/,
     +            ' surface in the transverse direction.')
285   TYPE 275
      READ (5,*,ERR=285) NORDER
      IF (NORDER.LT.3.OR.NORDER.GT.4) GOTO 285

***** Calculate the coordinates of the initial vertices
***** matrix.

********** Determine the X and Y spacing of the vertices.

      DM=1.0/FLOAT(MCP-1)
      IF (NCP.EQ.1) THEN
          DN=0.0
      ELSE
          DN=1.0/(RATIO*FLOAT(NCP-1))
      ENDIF

********** Calculate the coordinates.

      YMIN = 0.0-1.0/(RATIO*2.0)

      DO 310 M=1,MCP
          DO 300 N=1,NCP
              CP(M,N,1) = -0.5 + FLOAT(M-1) * DM
              CP(M,N,2) = YMIN + FLOAT(N-1) * DN
              CP(M,N,3) = 0.0
              CP(M,N,4) = 1.0
*             Default homogeneous coordinate = 1 for
*             non-rational B-spline.
```

```
300         CONTINUE
310      CONTINUE

***** Send the necessary program lines to the PS-300 to
***** display the vertices correctly positioned.

500      CONTINUE

         DO 700 M = 1,MCP
            DO 690 N = 1,NCP
                NUMCP = M + (N-1)*10

                ENCODE (50,510,BUFFER) NUMCP,NUMCP
510                 FORMAT ('CP',I2,':=BEGIN_STRUCTURE SET',
     +                      ' PICKING IDENTIFIER=CP',I2,';')
                    CALL BLANK(3,BUFFER)
                    CALL BLANK(48,BUFFER)
                CALL PSSEND(50,BUFFER)

                CALL PSSEND (25,'TRAN:=TRANSLATE BY 0,0,0;')
                CALL PSSEND (17,'INSTANCE OF CUBE;')
                CALL PSSEND (14,'END_STRUCTURE;')

                ENCODE (28,520,BUFFER) NUMCP
520                 FORMAT ('TRAN_TOTAL_',I2,':=F:',
     +                      'ACCUMULATE;')
                    CALL BLANK(12,BUFFER)
                CALL PSSEND (28,BUFFER)

                ENCODE (38,530,BUFFER) NUMCP,NUMCP
530                 FORMAT ('CONNECT TRAN_TOTAL_',I2,
     +                      '<1>:<1>CP',I2,'.TRAN;')
                    CALL BLANK(20,BUFFER)
                    CALL BLANK(31,BUFFER)
                CALL PSSEND (38,BUFFER)

                ENCODE (52,540,BUFFER) CP(M,N,1), CP(M,N,2),
     +                                 CP(M,N,3), NUMCP
540                 FORMAT ('SEND V(',F7.4,',',F7.4,',',
     +                  F7.4,') TO <2>TRAN_TOTAL_',I2,';')
                    CALL BLANK(50,BUFFER)
                CALL PSSEND (52,BUFFER)

                ENCODE (34,545,BUFFER) NUMCP
545                 FORMAT ('SEND V(0,0,0) TO',
     +                      ' <1>TRAN_TOTAL_',I2,';')
                    CALL BLANK(32,BUFFER)
                CALL PSSEND(34,BUFFER)
```

```
                ENCODE (28,550,BUFFER) NUMCP
550                 FORMAT ('SEND .1 TO <4>TRAN_TOTAL_',I2,
     +                        ';')
                    CALL BLANK(26,BUFFER)
                CALL PSSEND (28,BUFFER)

                ENCODE (41,560,BUFFER) NUMCP,NUMCP
560                 FORMAT ('CONNECT SWITCH_PICK<',I2,
     +                  '>:<1>TRAN_TOTAL_',I2,';')
                    CALL BLANK (21,BUFFER)
                    CALL BLANK (39,BUFFER)
                CALL PSSEND (41,BUFFER)

***** Connect the dial signal to the switch outputs for row
***** and column movement.

                ENCODE (41,560,BUFFER) M+80,NUMCP
                    CALL BLANK (39,BUFFER)
                CALL PSSEND (41,BUFFER)

                ENCODE (41,560,BUFFER) N+90,NUMCP
                    CALL BLANK (39,BUFFER)
                CALL PSSEND (41,BUFFER)

690         CONTINUE
700     CONTINUE

        TYPE 701
701       FORMAT(/,' ****************************************',
     +          /,' **      PRESS <SHIFT-LINE/LOCAL>       **',
     +          /,' **        ON THE PS-300 KEYBOARD       **',
     +          /,' **          FOLLOWED BY <TERM>.        **',
     +          /,' ****************************************'
     +          //////)

        RETURN
        END
```

```
*****************************************************************
*****************************************************************
******************** Subroutine INITPS **************************
*****************************************************************
*****************************************************************
*                                                               *
*Input       : PS-300 command language program file,            *
*              "SPLINE.DAT"                                     *
*Output      : PS-300 command language file output to           *
*              PS-300.                                          *
*Calls       : PSETUP,PSSEND,PSEXIT                             *
*Alters      :                                                  *
*Description: This subroutine sends the PS-300 command          *
*              language program stored in the PDP-11 disk       *
*              file,"SPLINE.DAT," to the PS-300.                *
*Created On : 11/9/86             Modified On:                  *
*                                                               *
*****************************************************************
*                                                               *
*Variable    Type      Explanation                              *
*----------  --------  -----------                              *
*BUFFER(80) Logical   Used to pass each program line to         *
*                      PS-300.                                  *
*ICOUNT      Integer   Number of characters in program line     *
*                      currently being read and passed.         *
*ISTAT(1)    Integer   Reports error status from PS-300.        *
*                                                               *
*****************************************************************

      SUBROUTINE INITPS

      DIMENSION ISTAT(1)

      LOGICAL*1 BUFFER(80)

      ISTAT(1)=0

***** Initialize PS-300 and set up link with PDP-11 using
***** library subroutine.  The first two parameters set up
***** logical unit numbers for input from and output to the
***** PS-300.  The third parameter indicates whether of not
***** the PS-300 is to issue an "INIT" command.  The last
***** two parameters indicate the length and name of the
***** array used to report error status from the PS-300.

      TYPE 100
100       FORMAT(' *************************************',
     +          /,' ******** SETTING UP PS-300 *********',
     +          /,' *************************************',/)
      CALL PSETUP(7,7,.TRUE.,1,ISTAT)
      CALL PSSEND(11,'INITIALIZE;')
```

```
***** Send file to PS-300 one line at a time, using library
***** subroutine.

      OPEN(UNIT=3,NAME='SPLINE.DAT',TYPE='OLD',ERR=210)

      LINE = 0
120   READ(3,130,END=140,ERR=220) ICOUNT,(BUFFER(I),
     +                                      I=1,ICOUNT)
130       FORMAT(Q,132A1)
      IF(ICOUNT.GT.0)CALL PSSEND(ICOUNT,BUFFER)
      LINE = LINE+1
      MARK = LINE-LINE/50*50
      IF (MARK.EQ.0) TYPE 135
135   FORMAT (' Working.....')
      IF (LINE.EQ.1000) TYPE 136
136   FORMAT (' Don''t give up on me....')
      GOTO 120

140   CLOSE(3)

      RETURN

***** Error Routines.

210   TYPE 211
211            FORMAT(' ERROR WHILE OPENING DISK FILE ',
     +                 'SPLINE.DAT.')
          CALL PSEXIT(1,ISTAT)
          GOTO 140

220     TYPE 221
221            FORMAT(' READ ERROR WHILE TRANSFERRING ',
     +                 'DATA FILE.')
          CALL PSEXIT(1,ISTAT)
          GOTO 140

      END

************************************************************
************************************************************
********************** Subroutine IO ***********************
************************************************************
************************************************************

      SUBROUTINE IO(KEY)

***** This subroutine directs all input/output functions,
***** based on the function key press reported in the
***** variable "KEY".

      RETURN
      END
```

```
*****************************************************************
*****************************************************************
******************** Subroutine MOVING **************************
*****************************************************************
*****************************************************************
*                                                               *
*Input       : ICP,IBS,MSG,MCP,NCP,CP(10,8,4),vector from      *
*               PS-300 program indicating vertex movement.      *
*Output      : Updated CP(10,8,4)                              *
*Calls       : PSSEND,PSREAD,CPLINE,PSPOLL,BSPLIN              *
*Alters      : CP(10,8,4)                                      *
*Description: This subroutine receives vectors from the        *
*               PS-300 program which define vertex movement.*
*               The array, CP, is updated to reflect that       *
*               movement.  CPLINE is called to recalculate      *
*               the vector list for the lines connecting the*
*               vertices if the lines are being displayed       *
*               and the B-spline surface is recalculated        *
*               using an abbreviated algorithm which takes      *
*               into account the localized influence of the     *
*               relocated vertex.                               *
*Created On : 12/17/86           Modified On:                  *
*                                                               *
*****************************************************************
*                                                               *
*Variable    Type     Explanation                               *
*---------   -------  -----------                               *
*BUFFER(80) Logical  Used to pass text messages to and          *
*                     from the PS-300 program.                  *
*CP(10,8,4) Real     The coordinates of the vertices, with*
*                     the subscripts indicating row,            *
*                     column, and X,Y,Z or homogeneous          *
*                     component.                                *
*DX          Real     The X movement of a selected vertex.      *
*DY          Real     The Y movement of a selected vertex.      *
*DZ          Real     The Z movement of a selected vertex.      *
*IBS         Integer  Indicates the display status of the       *
*                     B-spline surface. 0 = not displayed,      *
*                     1-4=varying combinations of spacing.      *
*ICP         Integer  Indicates the display status of the       *
*                     vertices.  ICP=2 indicates the lines      *
*                     connecting the vertices are               *
*                     displayed.                                *
*IVEC        Integer  The position of the picked vector in      *
*                     the vector list.                          *
*LENGTH      Integer  Byte length of a message returned         *
*                     through a call to PSPOLL or PSREAD.       *
*M           Integer  The row of vertices under                 *
*                     consideration.                            *
*MCP         Integer  The number of rows of vertices.           *
```

```
*MSG         Integer   This variable is passed from the      *
*                      PS-300 program.  A value of 1-80      *
*                      indicates that the vector to be sent  *
*                      next represents movement of the       *
*                      correspondingly numbered vertex.      *
*                      Values of 81-90 indicate the vector   *
*                      represents movement of a row of       *
*                      vertices, and values of 91-98         *
*                      indicate movement of a column of      *
*                      vertices.  Values greater than 100    *
*                      are used to report the picking of a   *
*                      row or column of vertices, with the   *
*                      value being the position of the       *
*                      vector in the vector list, offset by  *
*                      100.                                  *
*N           Integer   The column of vertices under          *
*                      consideration.                        *
*NCP         Integer   The number of columns of vertices.    *
*NUM         Integer   Indicates which switch position the   *
*                      PS-300 program should send the        *
*                      translation dial signals to when a    *
*                      row or column of vertices has been    *
*                      picked for movement.                  *
*                                                            *
**************************************************************

      SUBROUTINE MOVING(ICP,IBS,MSG,LENGTH,BUFFER)

      COMMON /SPLINE/ MCP,NCP,CP(10,8,4),MORDER,NORDER
      LOGICAL*1 BUFFER(80)

***** If MSG>100, row or column of vertices has been picked
***** for movement.  Compute the appropriate identifying
***** number (81-90 = rows 1-10; 91-98 = columns 1-8) and
***** send it to the PS-300 program to use in routing the
***** dial signals.

      IF (MSG.GT.100) THEN
           IVEC = MSG - 100
           IF (IVEC.LE.MCP*NCP) THEN
*                a vector in a row of vertices was picked
                 NUM = (IVEC+NCP-1)/NCP+80
             ELSE
*                a vector in a column of vertices was picked
                 NUM = (IVEC+MCP-1-MCP*NCP)/MCP+90
           ENDIF
           ENCODE (26,100,BUFFER) NUM
100              FORMAT ('SEND FIX(',I2,') TO <1>PICKED;')
           CALL PSSEND (26,BUFFER)
           RETURN
      ENDIF
```

```
***** A vertex has been moved.

      IF (MSG.LT.100) THEN
*           wait for the vector defining the movement
            CALL PSREAD (80,BUFFER,LENGTH)

********** Decode the message, which is in the form of
********** "L X,Y,Z", where X, Y, and Z are in an E10
********** format if positive, and an E11 format if
********** negative.

            IF (LENGTH.EQ.34) THEN
                 DECODE (34,200,BUFFER) DX,DY,DZ
200                   FORMAT (1X,3(1X,E10.4))

            ELSEIF (LENGTH.EQ.37) THEN
                 DECODE (37,201,BUFFER) DX,DY,DZ
201                   FORMAT (1X,3(1X,E11.4))

            ELSEIF (LENGTH.EQ.35) THEN
                 IF (BUFFER(3).EQ.'-') THEN
                      DECODE (35,202,BUFFER) DX,DY,DZ
202                       FORMAT (2X,E11.4,2(1X,E10.4))

                 ELSEIF (BUFFER(14).EQ.'-') THEN
                      DECODE (35,203,BUFFER) DX,DY,DZ
203                   FORMAT (2X,E10.4,1X,E11.4,1X,E10.4)

                 ELSE
                      DECODE (35,204,BUFFER) DX,DY,DZ
204                   FORMAT (1X,2(1X,E10.4),1X,E11.4)
                 ENDIF

            ELSEIF (LENGTH.EQ.36) THEN
                 IF(BUFFER(3).EQ.'-'.AND.BUFFER(15).EQ.'-')
     +                                                     THEN
                      DECODE (36,205,BUFFER) DX,DY,DZ
205                   FORMAT (1X,2(1X,E11.4),1X,E10.4)

                 ELSEIF(BUFFER(3).EQ.'-') THEN
                      DECODE (36,206,BUFFER) DX,DY,DZ
206                   FORMAT (2X,E11.4,1X,E10.4,1X,E11.4)

                 ELSE
                      DECODE (36,207,BUFFER) DX,DY,DZ
207                   FORMAT (2X,E10.4,2(1X,E11.4))
                 ENDIF

            ELSE
                 TYPE 208
208              FORMAT (//,1X,'VECTOR LENGTH OUT OF RANGE')
```

```
                RETURN
          ENDIF

********** An individual vertex has been moved.

************ Filter out occasional erroneous transmissions.
      IF (ABS(DX).GE.0.1) DX=0.0
      IF (ABS(DY).GE.0.1) DY=0.0
      IF (ABS(DZ).GE.0.1) DZ=0.0

          IF (MSG.LE.80) THEN
               N = (MSG+9)/10
               M = MSG-((MSG-1)/10)*10

               CP(M,N,1) = CP(M,N,1) + DX
               CP(M,N,2) = CP(M,N,2) + DY
               CP(M,N,3) = CP(M,N,3) + DZ
          ENDIF

********** A row or column of vertices has been moved.

          IF (MSG.GT.80) THEN

               IF (MSG.LE.90) THEN
*                   a row has been moved
                    M = MSG - 80
                    DO 210 N = 1,NCP
                         CP(M,N,1) = CP(M,N,1) + DX
                         CP(M,N,2) = CP(M,N,2) + DY
                         CP(M,N,3) = CP(M,N,3) + DZ
210                 CONTINUE
               ENDIF

               IF (MSG.GT.90) THEN
*                   a column has been moved
                    N = MSG - 90
                    DO 220 M = 1,MCP
                         CP(M,N,1) = CP(M,N,1) + DX
                         CP(M,N,2) = CP(M,N,2) + DY
                         CP(M,N,3) = CP(M,N,3) + DZ
220                 CONTINUE
               ENDIF
          ENDIF

***** Report the new B-spline surface and vertices'
***** vectors to the PS-300 only if vertex movement
***** is completed, to minimize transmission time.

          CALL PSPOLL(80,BUFFER,LENGTH)
          IF (LENGTH.EQ.0) THEN
               IF (ICP.EQ.2) CALL CPLINE
```

```
            IF (IBS.GT.0) CALL BSPLIN
         ENDIF

         RETURN
      ENDIF

      STOP

      END

************************************************************
************************************************************
******************** Subroutine PLANES *******************
************************************************************
************************************************************

      SUBROUTINE PLANES(IPLANE)

***** When implemented, this subroutine will calulate and
***** transmit to the PS-300 the intersection of the
***** B-spline surface with the orthogonal plane specified
***** by IPLANE.

      RETURN
      END
```

```
*****************************************************************
*****************************************************************
******************** SUBROUTINE PREPS ***************************
*****************************************************************
*****************************************************************
*                                                               *
*Input       : PREPS-format data file defining                  *
*              structure to be displayed.                       *
*Output      : PS-300 vector list for display of given          *
*              structure.                                       *
*Calls       : PSSEND,PSVECS,BLANK                              *
*Alters      : PS-300 program structure KLINES.                 *
*Description: This subroutine reads a data file of points       *
*              known to be on the surface or limiting the       *
*              surface to be modeled with the B-spline.         *
*              The data is interpreted in accordance with       *
*              the format of the program, PREPS.  This          *
*              format is utilized at the University of          *
*              Washington as a standard format for all          *
*              digitized data to be input to the PS-300.        *
*              The data is then passed to the PS-300 in the*
*              form of a vector list for display.               *
*Created By : D. Michael Bryant                                 *
*Created On : 4/20/87              Modified On:                 *
*                                                               *
*****************************************************************
*                                                               *
*Variable    Type      Explanation                              *
*---------   -------   -----------                              *
*BUFFER(80) Logical   Contains strings being received from      *
*                      or sent to the PS-300.                   *
*FILNAM      Char      Name of the PREPS-format data file       *
*                      containing the data for the known        *
*                      points.                                  *
*ICOORD      Integer   Coordinate to be held constant for       *
*                      each repetition of the substructure      *
*                      directed by NPTS2.  0=none; 1=X; 2=Y;*
*                      3=Z.                                     *
*IGNORE      Integer   Dummy variable used to read data from*
*                      PREPS-format file which is not needed*
*                      by this program.                         *
*IOSTAT      Integer   Reports input/output error status.       *
*IREF        Integer   Plane about which substructure is to     *
*                      be reflected.  0 = none; 1 = YZ;         *
*                      2 = XZ; 3 = XY.                          *
*ISTAR       Integer   Number of vector components required     *
*                      to define the position of stars used     *
*                      to represent data points from data       *
*                      file on the PS-300 screen.  A            *
*                      maximum of 199 points can be so          *
*                      labeled, although the structure can      *
```

```
*                           contain many more points.              *
*IVEC         Integer      Identifies the element of VECS which   *
*                           contains the Z vector component of     *
*                           the final vector in the previously     *
*                           processed substructure.                *
*LP(1000)     Logical      Contains "point" (.FALSE.) or "line"   *
*                           (.TRUE.) designations for the          *
*                           itemized vector list defined by VECS.*
*MAXVEC       Integer      Number of vector components which the*
*                           current substructure adds to VECS.    *
*NDONE        Integer      Value greater than zero indicates end*
*                           of data file has been reached.        *
*NPTS         Integer      Number of points defining the current*
*                           substructure.                          *
*NPTS2        Integer      Number of repetitions of the sub-      *
*                           structure to be constructed.           *
*NSTRUC       Integer      Number identifying the current         *
*                           portion of the known data being        *
*                           processed.  To minimize memory         *
*                           consumption, the known data is split  *
*                           into a number of vector lists rather  *
*                           than a single long one.                *
*REPEAT(10)   Real         Contains the values of the coordinate*
*                           specified by ICOORD to be used with   *
*                           repetitions of a substructure.         *
*VECS(3000)   Real         Contains the vectors for the vector   *
*                           list to be transmitted to the PS-300.*
*XBODY        Real         Global translation along the X axis   *
*                           to be applied to the structure.        *
*                           Applied before normalization.          *
*XNORM        Real         Normalization factor by which all      *
*                           vectors in data file are to be         *
*                           divided.                               *
*XTRAN        Real         Translation of substructure along the*
*                           X axis. Applied before normalization.*
*YBODY        Real         Global translation along the Y axis   *
*                           to be applied to the structure.        *
*                           Applied before normalization.          *
*YTRAN        Real         Translation of substructure along the*
*                           Y axis. Applied before normalization.*
*ZBODY        Real         Global translation along the Z axis   *
*                           to be applied to the structure.        *
*                           Applied before normalization.          *
*ZTRAN        Real         Translation of substructure along the*
*                           Z axis. Applied before normalization.*
**********************************************************************
      SUBROUTINE PREPS

      COMMON /VECS/ VECS(1000)
      DIMENSION REPEAT(10)
      LOGICAL LP(333)
```

```
      LOGICAL*1 BUFFER(80)
      CHARACTER*80 FILNAM
      NSTRUC = 1
      NDONE = 0
      IVEC = 0

      TYPE *,' ENTER FILE NAME.'
      READ (5,5) FILNAM
5           FORMAT (A80)

      OPEN (UNIT=3,NAME=FILNAM,TYPE='OLD',ERR=200)

***** Read global header.

      READ (3,*,ERR=300) XNORM
      READ (3,*,ERR=300) IGNORE
      READ (3,*,ERR=300) XBODY,YBODY,ZBODY

***** Read substructure header and data.

10    READ (3,*,END=500,ERR=300) NPTS,NPTS2,ICOORD
      READ (3,*,ERR=300) XTRAN,YTRAN,ZTRAN,IREF

********** Ensure the VECS array dimension will not be
********** exceeded by reading the substructure data.

      MAXVEC = NPTS*NPTS2*3
      IF (IREF.NE.0) MAXVEC = MAXVEC*2
      IF (IVEC+MAXVEC.GT.1000) THEN
          GOTO 400
      ENDIF

********** Read substructure data.

15    CONTINUE
      IF (ICOORD.NE.1) THEN
          READ (3,*,ERR=300) (VECS(IVEC+I*3-2),I=1,NPTS)
        ELSE
          READ (3,*,ERR=300) (REPEAT(I),I=1,NPTS2)
      ENDIF

      IF (ICOORD.NE.2) THEN
          READ (3,*,ERR=300) (VECS(IVEC+I*3-1),I=1,NPTS)
        ELSE
          READ (3,*,ERR=300) (REPEAT(I),I=1,NPTS2)
      ENDIF

      IF (ICOORD.NE.3) THEN
          READ (3,*,ERR=300) (VECS(IVEC+I*3),I=1,NPTS)
        ELSE
          READ (3,*,ERR=300) (REPEAT(I),I=1,NPTS2)
```

```
      ENDIF

      TYPE *,' Working . . . .'
********** Construct substructure repetitions and perform
********** translations and normalization.

      DO 30 I = 1,NPTS2
           DO 20 J = 1,NPTS
                L = (I-1)*NPTS*3

                IF (ICOORD.EQ.1) THEN
                     VECS(IVEC+J*3-2+L) =
     +               (REPEAT(I)+XTRAN+XBODY)/XNORM-0.5
                  ELSE
                     VECS(IVEC+J*3-2+L) =
     +               (VECS(IVEC+J*3-2)+XTRAN+XBODY)/
     +               XNORM-0.5
                ENDIF

                IF (ICOORD.EQ.2) THEN
                     VECS(IVEC+J*3-1+L) =
     +               (REPEAT(I)+YTRAN+YBODY)/XNORM
                  ELSE
                     VECS(IVEC+J*3-1+L) =
     +               (VECS(IVEC+J*3-1)+YTRAN+YBODY)/XNORM
                ENDIF

                IF (ICOORD.EQ.3) THEN
                     VECS(IVEC+J*3+L) =
     +               (REPEAT(I)+ZTRAN+ZBODY)/XNORM
                  ELSE
                     VECS(IVEC+J*3+L) =
     +               (VECS(IVEC+J*3)+ZTRAN+ZBODY)/XNORM
                ENDIF

                LP((I-1)*NPTS+J+IVEC/3) = .TRUE.
                IF(J.EQ.1)  LP((I-1)*NPTS+J+IVEC/3)=.FALSE.

20         CONTINUE
30    CONTINUE

********** Construct substructure reflections.

      IF (IREF.GT.0) THEN
           DO 40 I = IVEC,IVEC+MAXVEC/2-3,3
                DO 35 J = 1,3
                     VECS(I+J+MAXVEC/2)=VECS(I+J)
35              CONTINUE
                VECS(I+IREF+MAXVEC/2) = VECS(I+IREF)*(-1.0)
                LP((I+MAXVEC/2)/3+1) = LP(I/3+1)
40         CONTINUE
```

```
      ENDIF

      IVEC = IVEC + MAXVEC
      GOTO 10
***** Error statements.

200   WRITE (5,210) FILNAM
210        FORMAT (//,' Error opening disk file',/,
     +            1X,A20)
      STOP

300   WRITE (5,310) FILNAM
310        FORMAT (//,' Error reading data from ',A20,/,
     +             ' Ensure file is formatted correctly.',//)
      CLOSE(3)
      STOP

***** Define the position of stars to represent up to 199
***** data points.

400   IF (NSTRUC.EQ.1) THEN
           ISTAR = 199*3
           IF (IVEC.LT.ISTAR) ISTAR = IVEC
           DO 450 I = 3,ISTAR,3
                IF (I-I/10*10.EQ.0) TYPE *,' Working....'
                ENCODE (61,410,BUFFER) I/3,VECS(I-2),
     +                                 VECS(I-1),VECS(I)
410                 FORMAT ('KPT',I3,':=TRANSLATE BY ',
     +                     F7.4,',',F7.4,',',F7.4,
     +                     ' APPLIED TO STAR;')
                CALL BLANK(4,BUFFER)
                CALL BLANK(5,BUFFER)
                CALL PSSEND(61,BUFFER)
450        CONTINUE
      ENDIF

***** Send vector list for partial structure to PS-300.

      ENCODE (33,460,BUFFER) NSTRUC, IVEC/3
460        FORMAT ('KL',I2,':=VECTOR_LIST ITEMIZED N=',I4)
      CALL BLANK (3,BUFFER)
      CALL PSSEND (33,BUFFER)
      CALL PSVECS (4,IVEC/3,VECS,LP,.TRUE.,IOSTAT)

      NSTRUC = NSTRUC + 1
      IVEC = 0
      IF (NDONE.EQ.1) GOTO 600
      GOTO 15
```

```
***** The end of the data file has been reached.

500   CLOSE (3)
      IF (IVEC.GT.0) THEN
          NDONE = 1
          GOTO 400
      ENDIF
600   CONTINUE
      RETURN
      END

************************************************************
************************************************************
******************** Subroutine REFLCT *********************
************************************************************
************************************************************

      SUBROUTINE REFLCT

***** This subroutine will calculate and display a
***** reflection of the displayed B-spline surface about
***** the body's XZ plane.

      RETURN
      END

************************************************************
************************************************************
******************* SUBROUTINE RESET ***********************
************************************************************
************************************************************
*                                                          *
*Input       : IRESET                                      *
*Output      : PS-300 program lines resetting rotation,    *
*              translation, and scaling nodes.             *
*Calls       : PSSEND                                      *
*Alters      : PS-300 program translation, rotation, and   *
*              scaling accumulators.                       *
*Description: When triggered by the appropriate function   *
*              key, this subroutine resets the translation *
*              accumulator to (0,0,0), the scaling         *
*              accumulator to 1, and the rotation          *
*              accumulator to the value indicated by       *
*              IRESET:  1=(0,0,0), 2=(90,0,0), 3=(90,90,0).*
*Created By : D. Michael Bryant                            *
*Created On : 12/17/86             Modified On:            *
*                                                          *
************************************************************
*                                                          *
*Variable    Type      Explanation                         *
*---------   -------   -----------                         *
```

```
*BUFFER(80) Logical  Contains strings being received from *
*                    or sent to the PS-300.               *
*IRESET     Integer  Indicates the current value of the   *
*                    three-way rotation resetting toggle. *
*                    1=(0,0,0), 2=(90,0,0), 3=(90,90,0).  *
*                                                         *
***********************************************************

      SUBROUTINE RESET(IRESET)

      LOGICAL*1 BUFFER(80)

      CALL PSSEND(31,'SEND V(0,0,0) TO <2>TRAN_TOTAL;')
      CALL PSSEND(29,'SEND V(0,0,0) TO <1>ALL.TRAN;')

      CALL PSSEND(19,'SEND 1 TO <2>SCALE;')
      CALL PSSEND(19,'SEND 0 TO <1>SCALE;')

      IF (IRESET.GT.1) THEN
          CALL PSSEND(22,'SEND 90 TO <1>RESET_X;')
        ELSE
          CALL PSSEND(21,'SEND 0 TO <1>RESET_X;')
      ENDIF

      IF (IRESET.EQ.3) THEN
          CALL PSSEND(22,'SEND 90 TO <1>RESET_Y;')
        ELSE
          CALL PSSEND(21,'SEND 0 TO <1>RESET_Y;')
      ENDIF

      RETURN
      END
```

```
*****************************************************************
*****************************************************************
******************** Subroutine SURFAC *******************
*****************************************************************
*****************************************************************
*                                                               *
*Input        : SURF(3,31,31)                                   *
*Output       : VECS(500),LP(333)                               *
*Calls        : PSSEND, PSVECS                                  *
*Alters       : Structure "BS" in PS-300 program.               *
*Description: This subroutine extracts a vector list from *
*              the array variable SURF and passes it to        *
*              the PS-300 for display of the calculated        *
*              B-spline surface.                                *
*Author:      : D. Michael Bryant                               *
*Created On : 2/22/87              Modified On:                 *
*                                                               *
*****************************************************************
*                                                               *
*Variable    Type      Explanation                              *
*---------   -------   -----------                              *
*BUFFER(80) Logical   Holds a message for transmital to the*
*                      PS-300 program with the PSSEND           *
*                      library subroutine.                      *
*IBS         Integer   Indicates status of display of           *
*                      B-spline surface:  0 = not displayed,*
*                      1 = displayed.                           *
*ILP         Integer   Counter indicates the current array      *
*                      element of LP.                           *
*IOSTAT      Integer   Input/output error status.               *
*IVECS       Integer   Counter indicates the current array      *
*                      element of VECS.                         *
*LP(170)     Logical   .TRUE. indicates corresponding           *
*                      elements of VECS represent a line        *
*                      vector (draw to this point with pen      *
*                      down).  .FALSE. indicates                *
*                      corresponding elements of VECS           *
*                      represent a point vector (move to        *
*                      this point with pen up).                 *
*NV          Integer   Total number of vectors in the vector*
*                      list for the B-spline surface.           *
*SURF(3,31,31)         The calculated points on the B-spline*
*            Real      surface, in cartesian coordinates, as*
*                      a function of the parametric             *
*                      coordinates.                             *
*VECS(500)   Real      Vector list defining the parametric      *
*                      lines representing the B-spline          *
*                      surface.                                 *
*                                                               *
*****************************************************************
```

```
      SUBROUTINE SURFAC

      LOGICAL LP(333)
      COMMON /VECS/ VECS(1000)
      COMMON /BASIS/ R(13,4,31), S(11,4,31), SURF(3,31,31)
      LOGICAL*1 BUFFER(80)

      NSTRUC = 0

***** Parametric lines in u (longitudinal) direction.

10    CONTINUE
      NSTRUC = NSTRUC + 1
      NV = 310
      IVECS = 0
      ILP = 0
      NBEGIN = (NSTRUC-1)*10+1
      NEND = NSTRUC*10
      IF (NSTRUC.EQ.4) THEN
           NEND = NBEGIN
           NV = 31
      ENDIF

      DO 100 I = NBEGIN,NEND
           DO 90 J = 1,31
*               Count the vectors created.
                ILP = ILP + 1
                DO 80 K = 1,3
*                    Count the vector components (x,y,z).
                     IVECS = IVECS + 1
                     VECS(IVECS) = SURF(K,I,J)
80              CONTINUE
*               Indicate whether vector is point or line
*               vector.
                LP(ILP) = .TRUE.
                IF (J.EQ.1) LP(ILP) = .FALSE.
90         CONTINUE
100   CONTINUE

      GOTO 300

***** Parametric lines in w (transverse) direction.

110   CONTINUE
      NSTRUC = NSTRUC + 1
      NV = 310
      IVECS = 0
      ILP = 0
      NBEGIN = (NSTRUC-5)*10+1
      NEND = (NSTRUC-4)*10
```

```
      IF (NSTRUC.EQ.8) THEN
           NEND = NBEGIN
           NV = 31
      ENDIF

      DO 200 J = NBEGIN,NEND
           DO 190 I = 1,31
                ILP = ILP + 1
                DO 180 K = 1,3
                    IVECS = IVECS + 1
                    VECS(IVECS) = SURF(K,I,J)
180             CONTINUE
                LP(ILP) = .TRUE.
                IF (I.EQ.1) LP(ILP) = .FALSE.
190        CONTINUE
200   CONTINUE

300   CONTINUE
      ENCODE (31,310,BUFFER) NSTRUC,NV
310        FORMAT('BS',I1,':=VECTOR_LIST ITEMIZED N=',I3)
      CALL PSSEND (31,BUFFER)
      CALL PSVECS (4,ILP,VECS,LP,.TRUE.,IOSTAT)

      TYPE *,' Working....'
      IF (NSTRUC.LT.4) GOTO 10
      IF (NSTRUC.LT.8) GOTO 110

      RETURN
      END

************************************************************
************************************************************
********************** Subroutine WAIT *********************
************************************************************
************************************************************
*                                                          *
*Input       : BUFFER, LENGTH                              *
*Output      : ICLIP,ICP,IKNOWN,IPICK,IRFLCT,IRESET,       *
*              IXY,IXZ,IYZ, and PS-300 equivalents         *
*Calls       : BSPLIN,CPLINE,IO,MOVING,PLANES,PSREAD,      *
*              PSSEND,REFLCT,RESET                         *
*Alters      :                                             *
*Description: Awaits signals under the control of the      *
*              PS-300 program which report function key    *
*              presses or the picking of vertices for      *
*              movement.  Determines the mode of operation *
*              dictated by the key presses, updates the    *
*              related variables and informs the PS-300    *
*              program of the changes, and invokes the     *
*              appropriate FORTRAN subroutines for any     *
*              resultant actions.                          *
```

```
*Created On : 11/25/86            Modified On:                 *
*                                                              *
****************************************************************
*                                                              *
*Variable   Type      Explanation                              *
*---------- -------   -----------                              *
*BUFFER(80) Logical   Contains strings being received from     *
*                     or sent to the PS-300.                   *
*IBS        Integer   Status of B-spline surface display:      *
*                     0 = not displayed, 1 = displayed.        *
*ICLIP      Integer   Z-clipping status: 1=off, 2=on.          *
*ICP        Integer   Control polygon display status: 0=not*
*                     displayed, 1=points displayed,           *
*                     2=connecting lines displayed.            *
*IKNOWN     Integer   Display status of known surface:         *
*                     0=not displayed, 1=points displayed,     *
*                     2=connecting lines displayed.            *
*IPICK      Integer   Picking mode indicator: 1=picking        *
*                     individual vertices, 2=picking rows      *
*                     or columns of vertices.                  *
*IRESET     Integer   Reset toggle indicator: 1=(X0,Y0,Z0),*
*                     2=(X90,Y0,Z0), 3=(X90,Y90,Z0).           *
*IRFLCT     Integer   Status of display of reflection of       *
*                     B-spline surface about an XZ plane:      *
*                     0=not displayed, 1=displayed.            *
*IXY        INTEGER   Status of display of intersection of     *
*                     B-spline surface with XY orthogonal      *
*                     plane (waterlines): 0=not displayed,     *
*                     1=coarse spacing, 2=fine spacing.        *
*IXZ        Integer   See IXY.                                 *
*IYZ        Integer   See IXY.                                 *
*LENGTH     Integer   Length of incoming message from          *
*                     PS-300, in bytes.                        *
*MODE       Integer   Function key availability status:        *
*                     1=standard mode, most functions          *
*                     available, 2=display of reflection or*
*                     intersection with planes selected,       *
*                     interactive mode not available,          *
*                     3=interactive mode selected,             *
*                     4=input/output menu selected.            *
*MSG        Integer   Contains a numeric message from the      *
*                     PS-300, as decoded from BUFFER.          *
*                     1-80 indicate next message will be a     *
*                     new vector for the vertex of the same*
*                     number.  101-242 indicate next           *
*                     message will be new vector of the        *
*                     vertex in the row or column of the       *
*                     control polygon line which has the       *
*                     position in the vector list (N-100).     *
*                     301-312 indicate a keypress of           *
*                     function keys 1-12.                      *
```

```
*************************************************************
      SUBROUTINE WAIT

      LOGICAL*1 BUFFER(80)

***** Initialize variables for default operating modes.

      LENGTH = 0
      MODE=1
      ICP=1
      IBS=0
      IKNOWN=0
      IPICK=1
      ICLIP=1
      IRFLCT=0
      IXY=0
      IXZ=0
      IYZ=0
      IRESET=3

***** Receive and decode integer message from PS-300
***** program.
100   IF (LENGTH.GT.0) GOTO 190
*          LENGTH > 0 indicates message already received in
*                     another subroutine.

110   CALL PSPOLL(80,BUFFER,LENGTH)
      IF (LENGTH.EQ.0) GOTO 110
190   DECODE(LENGTH,200,BUFFER) MSG
200             FORMAT (I3)
      LENGTH = 0

***** If message indicates new vertex vector is coming,
***** go to subroutine MOVING to receive and act upon the
***** vector.
      IF (MODE.EQ.3.AND.MSG.LT.300) THEN
           CALL MOVING (ICP,IBS,MSG,LENGTH,BUFFER)
      ENDIF
      IF (MODE.EQ.3.AND.MSG.LT.300) GOTO 100

***** Vertex display toggle.

      IF (MODE.LE.3) THEN
           IF (MSG.EQ.301) THEN
                ICP=ICP+1
                IF (ICP.GT.2) ICP=0
                IF (ICP.EQ.2) CALL CPLINE
                ENCODE (25,300,BUFFER) ICP
300                 FORMAT ('SEND FIX('I1') TO <1>CPMODE;')
                CALL PSSEND (25,BUFFER)
           ENDIF
```

```
***** B-spline surface display toggle.

          IF (MSG.EQ.302) THEN
               IBS=IBS+1
               IF (IBS.GT.1) IBS=0
               IF (IBS.GT.0) CALL BSPLIN
               ENCODE (25,310,BUFFER) IBS
310                FORMAT ('SEND FIX('I1') TO <1>BSMODE;')
               CALL PSSEND (25,BUFFER)
          ENDIF

***** Known surface display toggle.

          IF (MSG.EQ.303) THEN
               IKNOWN=IKNOWN+1
               IF (IKNOWN.GT.2) IKNOWN=0
               ENCODE (28,320,BUFFER) IKNOWN
320              FORMAT('SEND FIX('I1') TO <1>KNOWNMODE;')
               CALL PSSEND (28,BUFFER)
          ENDIF

***** Interactive mode toggle.

          IF (MSG.EQ.304) THEN
               IF (MODE.EQ.1) THEN
                    MODE=3
                 ELSEIF (MODE.EQ.3) THEN
                    MODE=1
               ENDIF
               ENCODE (23,330,BUFFER) MODE
330                 FORMAT ('SEND FIX('I1') TO <1>MODE;')
               CALL PSSEND (23,BUFFER)
          ENDIF

***** Z-clipping toggle.

          IF (MSG.EQ.305) THEN
               ICLIP=ICLIP+1
               IF (ICLIP.GT.2) ICLIP=1
               ENCODE (27,340,BUFFER) ICLIP
340              FORMAT ('SEND FIX('I1') TO <1>CLIPMODE;')
               CALL PSSEND (27,BUFFER)
          ENDIF

***** Reflection display toggle.

C         IF (MSG.EQ.306) THEN
C              IF (MODE.EQ.1) THEN
C                   MODE=2
C                   IRFLCT=1
```

```
C                       CALL PSSEND(23,
C     +                    'SEND FIX(2) TO <1>MODE;')
C                       CALL PSSEND(22,
C     +                    'SEND 1 TO <1>NEW_MODE;')
C                       CALL REFLCT
C                       CALL PSSEND(30,
C     +                    'SEND FIX(1) TO <1>REFLECTMODE;')
C                    ELSEIF (MODE.EQ.2.AND.IRFLCT.EQ.0) THEN
C                       IRFLCT=1
C                       CALL REFLCT
C                       CALL PSSEND (30,
C     +                    'SEND FIX(1) TO <1>REFLECTMODE;')
C                    ELSEIF (MODE.EQ.2.AND.IRFLCT.EQ.1) THEN
C                       IRFLCT=0
C                       CALL PSSEND (30,
C     +                    'SEND FIX(0) TO <1>REFLECTMODE;')
C                       GOTO 400
C                 ENDIF
C            ENDIF

***** XY intersection display toggle.

             IF (MSG.EQ.307) THEN
C                 IF (MODE.LT.3) THEN
C                       IXY=IXY+1
C                       IF (IXY.GT.2) IXY=0
C                       IF (IXY.EQ.1) CALL PLANES(1)
C                       ENCODE (25,360,BUFFER) IXY
C360                          FORMAT ('SEND FIX(',I1,
C     +                          ') TO <1>XYMODE;')
C                       CALL PSSEND (25,BUFFER)
C                       IF (IXY.EQ.0) GOTO 400
C                 ENDIF

***** Point-to-row picking toggle.
                  IF (MODE.EQ.3.AND.IPICK.EQ.1) THEN
                        IPICK=2
                        CALL PSSEND (27,
      +                    'SEND FIX(2) TO <1>PICKMODE;')
                  ENDIF
             ENDIF

***** XZ intersection display toggle.
             IF (MSG.EQ.308) THEN
C                 IF (MODE.LT.3) THEN
C                       IXZ=IXZ+1
C                       IF (IXZ.GT.2) IXY=0
C                       IF (IXZ.EQ.1) CALL PLANES(2)
C                       ENCODE (25,370,BUFFER) IXZ
C370                          FORMAT('SEND FIX(',I1,
C     +                          ') TO <1>XZMODE;')
```

```
C                   CALL PSSEND (25,BUFFER)
C                   IF (IXY.EQ.0) GOTO 400
C              ENDIF
***** Row-to-point picking toggle.

               IF (MODE.EQ.3.AND.IPICK.EQ.2) THEN
                    IPICK=1
                    CALL PSSEND (27,
     +                'SEND FIX(1) TO <1>PICKMODE;')
               ENDIF
          ENDIF

***** YZ intersection display toggle.

C         IF (MSG.EQ.309.AND.MODE.LT.3) THEN
C              IYZ=IYZ+1
C              IF (IYZ.GT.2) IYZ=0
C              IF (IYZ.EQ.1) CALL PLANES(3)
C              ENCODE (25,380,BUFFER) IYZ
C380               FORMAT ('SEND FIX('I1') TO <1>YZMODE;')
C              CALL PSSEND (25,BUFFER)
C              IF (IYZ.EQ.0) GOTO 400
C         ENDIF

***** Select input/output menu.

          IF (MSG.EQ.310) THEN
               LAST = MODE
               MODE = 4
               CALL PSSEND (23,'SEND FIX(4) TO <1>MODE;')
          ENDIF

***** Reset function toggle.
          IF (MSG.EQ.311) THEN
               IRESET=IRESET+1
               IF (IRESET.GT.3) IRESET=1
               CALL RESET (IRESET)
          ENDIF

***** Quit.
          IF (MSG.EQ.312) THEN
              CALL PSSEND (11,'INITIALIZE;')
              RETURN
          ENDIF
     ENDIF

***** I/O menu selections.
     IF (MODE.EQ.4) THEN
          IF (MSG.LT.312) CALL IO(MSG-300)
```

```
          IF (MSG.EQ.312) THEN
               MODE = LAST
               ENCODE (23,330,BUFFER) MODE
               CALL PSSEND (23,BUFFER)
          ENDIF
     ENDIF
***** Trigger PS-300 program to send new variable values to
***** all 'FETCH' nodes.

      CALL PSSEND (22,'SEND 1 TO <1>NEW_MODE;')
      GOTO 100

***** If toggling a function 'off' while in mode 2, ensure
***** all mode-2 functions are off before returning to
***** mode 1.

400   IF(IRFLCT.EQ.0.AND.IXY.EQ.0.AND.IXZ.EQ.0.AND.IYZ.EQ.0
     + ) THEN
           MODE = 1
           CALL PSSEND (23,'SEND FIX(1) TO <1>MODE;')
           CALL PSSEND (22,'SEND 1 TO <1>NEW_MODE;')
      ENDIF
      GOTO 100

      END
```

APPENDIX C

SPLINE.DAT PROGRAM LISTING

SPLINE.DAT is the PS 300-resident portion of the program, SPLINE.

```
{
**************************************************************
*********************** SPLINE.DAT ***************************
**************************************************************
****************** Program Organization *********************
**************************************************************

  I. Variables.

 II. Main display tree.
     A.   Display vertices.
     B.   Display B-spline surface.
     C.   Display points on known surface.
     D.   Display reflection of all displayed structures
          about a plane.
     E.   Display intersection of XY planes with B-spline
          surface.
     F.   Display intersection of XZ planes with B-spline
          surface.
     G.   Display intersection of YZ planes with B-spline
          surface.

III. Connect dials.
     A.   Translation.
     B.   Rotation.
     C.   Scaling.

 IV. Vertex picking network.
     A.   Enable picking.
     B.   Convert cursor position to picklist.
     C.   Convert picklist to integer identifying vertex.
     D.   Report pick of row or column to host computer.
     E.   Route translation dial signals to picked vertices

  V. Report vertex movement to host.

 VI. Clipping.

VII. Function keys and labels.
     A.   Report key press to host computer.
     B.   Label function keys and dials.
```

```
*****************************************************************
*****************************************************************
************************* Variables *****************************
*****************************************************************
*****************************************************************
}
SEND
  'SPLINE, A B-SPLINE SURFACE MODELER BY D. MICHAEL BRYANT'
   TO <1>FLABELO;                        {Display title message.}
{}
TO_HOST:=F:PRINT;                        {Convert messages going
                                         to host to strings}
     CONNECT TO_HOST<1>:<1>HOST_MESSAGE;
     SEND TRUE TO <2>TO_HOST;            {Convert floating-point
                                         numbers to exponential
                                         format.}
{}
NEW_MODE:=F:NOP;                         {No-operation function
                                         used as a tie point for
                                         the triggers of FETCH
                                         commands for all
                                         variables, which are
                                         triggered by the host
                                         computer sending any
                                         value to NEW_MODE}
{}
VARIABLE BSMODE;                         {B-spline surface
                                         display mode: 0=not
                                         displayed, 1=displayed.}
     SEND FIX(1) TO <1>BSMODE;           {Initial default value}
     FETCHBSMODE:=F:FETCH;
          SEND 'BSMODE' TO <2>FETCHBSMODE;
          CONNECT NEW_MODE<1>:<1>FETCHBSMODE;
{}
VARIABLE CLIPMODE;                       {Z-clipping mode: 1=off,
                                         2=on}
     SEND FIX(1) TO <1>CLIPMODE;         {Initial default value}
     FETCHCLIPMODE:=F:FETCH;
          SEND 'CLIPMODE' TO <2>FETCHCLIPMODE;
          CONNECT NEW_MODE<1>:<1>FETCHCLIPMODE;
{}
VARIABLE CPMODE;                         {Control point display
                                         mode: 0=not displayed,
                                         1=points displayed,
                                         2=connecting lines}
     SEND FIX(2) TO <1>CPMODE;           {Initial default value}
     FETCHCPMODE:=F:FETCH;
          SEND 'CPMODE' TO <2>FETCHCPMODE;
          CONNECT NEW_MODE<1>:<1>FETCHCPMODE;
{}
```

```
VARIABLE DISCARD;                            {Provides connection for
                                              function outputs left
                                              intentionally unused,
                                              without receiving a "no
                                              connection made to ...."
                                              error message on screen}
{}
VARIABLE KNOWNMODE;                          {Known surface display
                                              mode: 0=not displayed,
                                              1=points displayed,
                                              2=connecting lines}
     SEND FIX(0) TO <1>KNOWNMODE;            {Initial default value}
     FETCHKNOWNMODE:=F:FETCH;
          SEND 'KNOWNMODE' TO <2>FETCHKNOWNMODE;
          CONNECT NEW_MODE<1>:<1>FETCHKNOWNMODE;
{}
VARIABLE MODE;                               {Function key menu mode:
                                              1=main menu, most
                                              functions available
                                              2=display of reflection
                                              or intersecting planes
                                              selected, interactive
                                              mode not available
                                              3=interactive mode
                                              selected, mode 2
                                              displays not available
                                              4=input/output menu
                                              selected}
     SEND FIX(1) TO <1>MODE;                 {Initial default value}
     FETCHMODE:=F:FETCH;
          SEND 'MODE' TO <2>FETCHMODE;
          CONNECT NEW_MODE<1>:<1>FETCHMODE;
{}
VARIABLE PICKMODE;                           {Vertex picking mode:
                                              1=individual points,
                                              2=rows/columns of
                                              points}
     SEND FIX(1) TO <1>PICKMODE;             {Initial default value}
     FETCHPICKMODE:=F:FETCH;
          SEND 'PICKMODE' TO <2>FETCHPICKMODE;
          CONNECT NEW_MODE<1>:<1>FETCHPICKMODE;
{}
VARIABLE REFLECTMODE;                        {Reflection-about-plane
                                              display status: 0=off,
                                              1=on}
     SEND FIX(0) TO <1>REFLECTMODE;{Initial default value}
     FETCHREFLECTMODE:=F:FETCH;
          SEND 'REFLECTMODE' TO <2>FETCHREFLECTMODE;
          CONNECT NEW_MODE<1>:<1>FETCHREFLECTMODE;
{}
```

```
VARIABLE XYMODE, XZMODE, YZMODE;     {Display of inter-
                                      section of B-spline
                                      surface with orthogonal
                                      planes (i.e.
                                      waterlines): 0=not
                                      displayed, 1=coarsely
                                      spaced, 2=finely spaced}
     SEND FIX(0) TO <1>XYMODE;        {Initial default value}
     FETCHXYMODE:=F:FETCH;
          SEND 'XYMODE' TO <2>FETCHXYMODE;
          CONNECT NEW_MODE<1>:<1>FETCHXYMODE;
     SEND FIX(0) TO <1>XZMODE;        {Initial default value}
     FETCHXZMODE:=F:FETCH;
          SEND 'XZMODE' TO <2>FETCHXZMODE;
          CONNECT NEW_MODE<1>:<1>FETCHXZMODE;
     SEND FIX(0) TO <1>YZMODE;        {Initial default value}
     FETCHYZMODE:=F:FETCH;
          SEND 'YZMODE' TO <2>FETCHYZMODE;
          CONNECT NEW_MODE<1>:<1>FETCHYZMODE;
{




************************************************************
************************************************************
******************** Main Display Tree *********************
************************************************************
************************************************************
}
CLIPON:= SET DEPTH_CLIPPING OFF APPLIED TO CLIPPER;
CLIPPER:= WINDOW X=-1:1 Y=-1:1 FRONT=0
     BACK=0.1 APPLIED TO CLIP_INTENSITY;
CLIP_INTENSITY:=SET INTENSITY OFF 1:1 APPLIED TO ALL;
                                      {Disables depth cueing
                                      when clipping is on, to
                                      allow viewing within
                                      narrow clipping windows}
DISPLAY CLIPON;
ALL:=BEGIN_STRUCTURE
          VIEWPORT HOR=-1:1 VERT=-1:1 INTENSITY=.75:1;
          TRAN:=TRANSLATE BY 0,0,0;
          ROT:=ROTATE 0;
          SCALE:=SCALE BY 1;
          SET:=SET CONDITIONAL_BIT 1 ON;
          INSTANCE OF CP_SET,BS_SET,KNOWN_SET,PLANE_SET;
      END_STRUCTURE;
{
```

```
****************************************************************
************** Display Control Points (Vertices)**********
****************************************************************
}
CP_SET:=SET LEVEL_OF_DETAIL TO 1 THEN CP_IF;
                                        {Node to connect toggle
                                        to}
     CONNECT FETCHCPMODE<1>:<1>CP_SET;
CP_IF:=IF LEVEL_OF_DETAIL >0 THEN CP;
                                        {Display vertices unless
                                        toggled off.}
CP:=INSTANCE OF CP_PT,CP_LINES;         {Structure consists of
                                        two branches: points and
                                        connecting lines.}
CP_LINES:=BEGIN_STRUCTURE
               IF LEVEL_OF_DETAIL =2;
                                        {Display if toggled on.}
               SETPICK:=SET PICKING OFF;
                                        {Create picking node to
                                        allow vertices to be
                                        moved a row or column at
                                        a time by picking the
                                        connecting line}
               SET PICKING IDENTIFIER=CP99;
               INSTANCE OF CPL;         {CPL is a vector list
                                        which will be provided
                                        dynamically by the host
                                        computer.}
               CPL:=VECTOR_LIST ITEMIZED N=200
                    0.0,0.0,0.0;        {Reserve memory for
                                        vector list.}
          END_STRUCTURE;
{}
CP_PT:=BEGIN_STRUCTURE
          SETPICK:=SET PICKING OFF;{Create picking node to
                                        allow vertices to be
                                        selected in interactive
                                        mode by picking with
                                        tablet and stylus}
          INSTANCE OF
               CP01,CP02,CP03,CP04,CP05,CP06,CP07,CP08,
               CP09,CP10,CP11,CP12,CP13,CP14,CP15,CP16,
               CP17,CP18,CP19,CP20,CP21,CP22,CP23,CP24,
               CP25,CP26,CP27,CP28,CP29,CP30,CP31,CP32,
               CP33,CP34,CP35,CP36,CP37,CP38,CP39,CP40,
               CP41,CP42,CP43,CP44,CP45,CP46,CP47,CP48,
               CP49,CP50,CP51,CP52,CP53,CP54,CP55,CP56,
               CP57,CP58,CP59,CP60,CP61,CP62,CP63,CP64,
               CP65,CP66,CP67,CP68,CP69,CP70,CP71,CP72,
               CP73,CP74,CP75,CP76,CP77,CP78,CP79,CP80;
     END_STRUCTURE;
```

```
{
Definitions of CP01 through CP80 will be supplied by the
host computer, with the necessary nodes and connections to
allow the vertices to be individually manipulated in the
interactive mode.  The program lines supplied dynamically
from the host computer will have the following form (with
the 01 replaced with the appropriate number for the nodes
associated with each vertex):

*CP01:=BEGIN_STRUCTURE                                        *
*    SET PICKING IDENTIFIER=CP01    (Allows identificaiton *
*                                   of which node was       *
*                                   picked for interaction.*
*    TRAN:=TRANSLATE BY 0,0,0       (This node will connect*
*                                   through an accumulator *
*                                   and switching network  *
*                                   to the dials for       *
*                                   manipulation of the    *
*                                   picked vertex.  Initial*
*                                   placement of the vertex*
*                                   will also be specified *
*                                   through the network    *
*                                   connected to this      *
*                                   node.)                 *
*    INSTANCE OF CUBE                                       *
* END_STRUCTURE                                             *
*                                                           *
*TRAN_TOTAL_01:=F:ACCUMULATE        (Create translation    *
*                                   accumulator for each   *
*                                   vertex.)               *
*    CONNECT TRAN_TOTAL_01<1>:<1>CP01.TRAN                  *
*                                                           *
*    SEND V(X,Y,Z) TO <2>TRAN_TOTAL_01                      *
*                                   (Initial vertex        *
*                                   position.)             *
*    SEND V(0,0,0) TO <1>TRAN_TOTAL_01                      *
*                                   (Give accumulator an   *
*                                   initial value to act   *
*                                   upon before input from *
*                                   dial is received).     *
*    SEND 0.1 TO <4>TRAN_TOTAL_01   (Multiplier for dial   *
*                                   sensitivity.)          *
}
CUBE:=INSTANCE OF FRONT,BACK,SIDES;{Each vertex is
                                   represented on the
                                   screen by a cube.}
    BACK: =TRANSLATE BY 0,0,0.01 APPLIED TO FRONT;
    FRONT:=VECTOR_LIST Z=-0.005 N=5
            -0.005,-0.005   0.005,-0.005  0.005,0.005
            -0.005, 0.005  -0.005,-0.005;
```

```
     SIDES:=VECTOR_LIST SEP N=8
                    -0.005, 0.005,0.005  -0.005, 0.005,-0.005
                    -0.005,-0.005,0.005  -0.005,-0.005,-0.005
                     0.005, 0.005,0.005   0.005, 0.005,-0.005
                     0.005,-0.005,0.005   0.005,-0.005,-0.005;
{
***********************************************************
**************** Display B-Spline Surface ****************
***********************************************************
}
BS_SET:=SET LEVEL_OF_DETAIL TO 1 THEN BS_IF;
                                  {Display B-spline
                                  surface if toggled on.}
     CONNECT FETCHBSMODE<1>:<1>BS_SET;
BS_IF:=IF LEVEL_OF_DETAIL >0 THEN BS;
BS:=INSTANCE OF BS1,BS2,BS3,BS4,BS5,BS6,BS7,BS8;
{
The host computer will dynamically provide program lines
defining BS.  These commands will be of the form:

*    BS1:=VECTOR_LIST ITEMIZED N=XXX                       *

}
{
***********************************************************
************ Display Points on Known Surface ************
***********************************************************
}
KNOWN_SET:=SET LEVEL_OF_DETAIL TO 1 THEN KNOWN_IF;
                                  {Display known points
                                  if toggled on.}
     CONNECT FETCHKNOWNMODE<1>:<1>KNOWN_SET;
KNOWN_IF:=IF LEVEL_OF_DETAIL >0 THEN KNOWN;
KNOWN:=INSTANCE OF KNOWN_PTS, KNOWN_LINES;
{}
KNOWN_PTS:=INSTANCE OF
     KPT001, KPT002, KPT003, KPT004, KPT005, KPT006,
     KPT007, KPT008, KPT009, KPT010, KPT011, KPT012,
     KPT013, KPT014, KPT015, KPT016, KPT017, KPT018,
     KPT019, KPT020, KPT021, KPT022, KPT023, KPT024,
     KPT025, KPT026, KPT027, KPT028, KPT029, KPT029,
     KPT030, KPT031, KPT032, KPT033, KPT034, KPT035,
     KPT036, KPT037, KPT038, KPT039, KPT040, KPT041,
     KPT042, KPT043, KPT044, KPT045, KPT046, KPT047,
     KPT048, KPT049, KPT050, KPT051, KPT052, KPT053,
     KPT054, KPT055, KPT056, KPT057, KPT058, KPT059,
     KPT060, KPT061, KPT062, KPT063, KPT064, KPT065,
     KPT066, KPT067, KPT068, KPT069, KPT070, KPT071,
     KPT072, KPT073, KPT074, KPT075, KPT076, KPT077,
     KPT078, KPT079, KPT080, KPT081, KPT082, KPT083,
     KPT084, KPT085, KPT086, KPT087, KPT088, KPT089,
```

```
     KPT090, KPT091, KPT092, KPT093, KPT094, KPT095,
     KPT096, KPT097, KPT098, KPT099, KPT100, KPT101,
     KPT102, KPT103, KPT104, KPT105, KPT106, KPT107,
     KPT108, KPT109, KPT110, KPT111, KPT112, KPT113,
     KPT114, KPT115, KPT116, KPT117, KPT118, KPT119,
     KPT120, KPT121, KPT122, KPT123, KPT124, KPT125,
     KPT126, KPT127, KPT128, KPT129, KPT130, KPT131,
     KPT132, KPT133, KPT134, KPT135, KPT136, KPT137,
     KPT138, KPT139, KPT140, KPT141, KPT142, KPT143,
     KPT144, KPT145, KPT146, KPT147, KPT148, KPT149,
     KPT150, KPT151, KPT152, KPT153, KPT154, KPT155,
     KPT156, KPT157, KPT158, KPT159, KPT160, KPT161,
     KPT162, KPT163, KPT164, KPT165, KPT166, KPT167,
     KPT168, KPT169, KPT170, KPT171, KPT172, KPT173,
     KPT174, KPT175, KPT176, KPT177, KPT178, KPT179,
     KPT180, KPT181, KPT182, KPT183, KPT184, KPT185,
     KPT186, KPT187, KPT188, KPT189, KPT190, KPT191,
     KPT192, KPT193, KPT194, KPT195, KPT196, KPT197,
     KPT198, KPT199;
{
Structures KPT001 through KPT199 will be provided
dynamically by the host computer.  The commands provided
will be of the form:

*    KPT037:=TRANSLATE BY x,y,z APPLIED TO STAR             *
}
STAR:=VECTOR_LIST SEP N=14
      0.000, 0.005, 0.000     0.000,-0.005, 0.000
     -0.005, 0.000, 0.000     0.005, 0.000, 0.000
     -0.004, 0.004, 0.000     0.004,-0.004, 0.000
     -0.004,-0.004, 0.000     0.004, 0.004, 0.000
      0.000,-0.004,-0.004     0.000, 0.004, 0.004
      0.000, 0.004,-0.004     0.000,-0.004, 0.004
      0.000, 0.000,-0.005     0.000, 0.000, 0.005;
{}
KNOWN_LINES:=IF LEVEL_OF_DETAIL =2 THEN KLINES;
                                        {Display lines
                                        connecting known points
                                        if toggled on.}
KLINES:=INSTANCE OF KL01,KL02,KL03,KL04,KL05,KL06,KL07,
                    KL08,KL09,KL10,KL11,KL12,KL13,KL14;
{
KLINES, the vector list defining the lines connecting the
known points, will be provided dynamically by the host
computer.  The command provided by the host will be of the
form:

*    KL01:=VECTOR_LIST ITEMIZED N=xxx                       *
}
{
```

```
****************************************************************
************ Display Reflection About a Plane ************
****************************************************************
}
REFLECT_SET:=SET LEVEL_OF_DETAIL TO 0 THEN REFLECT_IF;
                                   {Display reflection
                                   if toggled on.}
     CONNECT FETCHREFLECTMODE<1>:<1>REFLECT_SET;
REFLECT_IF:=IF LEVEL_OF_DETAIL >0 THEN REFLECT;
{
REFLECT, the vector list defining the reflection of all
displayed structures about a plane, will be provided
dynamically by the host computer.  The command provided by
the host will be of the form:

*    REFLECT:=VECTOR_LIST ITEMIZED N=xxx                      *
}
{
****************************************************************
*********** Display Intersection With XY Planes ***********
****************************************************************
}
XY_SET:=SET LEVEL_OF_DETAIL TO 0 THEN XY_IF;
                                   {Display intersection
                                   if toggled on.}
     CONNECT FETCHXYMODE<1>:<1>XY_SET;
XY_IF:=IF LEVEL_OF_DETAIL >0 THEN XY;
{}
XY:=INSTANCE OF XYC,XYF;           {The sturucture has two
                                   levels of display
                                   detail: coarse and fine}
     XYF:=IF LEVEL_OF_DETAIL =2 THEN XYFINE;
{
The structures XYC and XYFINE will be provided dynamically
by the host computer.  The commands provided will be of the
form:

*    XYC:=VECTOR_LIST ITEMIZED N=xxx                          *
*    XYFINE:= VECTOR_LIST ITEMIZED N=xxx                      *
}
{
****************************************************************
********** Display Intersection With XZ Planes **********
****************************************************************
}
XZ_SET:=SET LEVEL_OF_DETAIL TO 0 THEN XZ_IF;
                                   {Display intersection
                                   if toggled on.}
     CONNECT FETCHXZMODE<1>:<1>XZ_SET;
XZ_IF:=IF LEVEL_OF_DETAIL >0 THEN XZ;
{}
```

```
XZ:=INSTANCE OF XZC,XZF;              {The sturucture has two
                                      levels of display
                                      detail: coarse and fine}
     XZF:=IF LEVEL_OF_DETAIL =2 THEN XZFINE;
{
The structures XZC and XZFINE will be provided dynamically
by the host computer.  The commands provided will be of the
form:

*    XZC:=VECTOR_LIST ITEMIZED N=xxx                          *
*    XZFINE:= VECTOR_LIST ITEMIZED N=xxx                      *
}
{
***************************************************************
*********** Display Intersection With YZ Planes ***********
***************************************************************
}
YZ_SET:=SET LEVEL_OF_DETAIL TO 0 THEN YZ_IF;
                                      {Display intersection
                                      if toggled on.}
     CONNECT FETCHYZMODE<1>:<1>YZ_SET;
YZ_IF:=IF LEVEL_OF_DETAIL >0 THEN YZ;
{}
YZ:=INSTANCE OF YZC,YZF;              {The sturucture has two
                                      levels of display
                                      detail: coarse and fine}
     YZF:=IF LEVEL_OF_DETAIL =2 THEN YZFINE;
{    .
The structures YZC and YZFINE will be provided dynamically
by the host computer.  The commands provided will be of the
form:

*    YZC:=VECTOR_LIST ITEMIZED N=xxx                          *
*    YZFINE:= VECTOR_LIST ITEMIZED N=xxx                      *
}
{
```

```
*****************************************************************
*****************************************************************
********************** Connect Dials ****************************
*****************************************************************
*****************************************************************
}
{
*****************************************************************
******************** Translation Dials **************************
*****************************************************************

************ Convert dial signals to vectors. *************
}
TRAN_X:=F:XVEC;
TRAN_Y:=F:YVEC;
TRAN_Z:=F:ZVEC;
CONNECT DIALS<1>:<1>TRAN_X;
CONNECT DIALS<2>:<1>TRAN_Y;
CONNECT DIALS<3>:<1>TRAN_Z;
{
***** Create switching network which applies                 *****
***** translation dials to global translation (if in         *****
***** modes 1 or 2), vertex translation (if in mode 3,*****
***** interactive mode), or no translation (mode 4,          *****
***** I/O menu)                                               *****
}
SWITCH_TRAN:=F:CROUTE(4);
     CONNECT TRAN_X<1>:<2>SWITCH_TRAN;
     CONNECT TRAN_Y<1>:<2>SWITCH_TRAN;
     CONNECT TRAN_Z<1>:<2>SWITCH_TRAN;
     CONNECT FETCHMODE<1>:<1>SWITCH_TRAN;
{
***** Connect switch to global translation node if in *****
*****                   mode 1 or 2.                      *****
}
TRAN_TOTAL:=F:ACCUMULATE;           {Accumulator for global
                                     translation.}
     CONNECT SWITCH_TRAN<1>:<1>TRAN_TOTAL;
     CONNECT SWITCH_TRAN<2>:<1>TRAN_TOTAL;
     {SWITCH_TRAN<3> will be connected later}
     CONNECT SWITCH_TRAN<4>:<1>DISCARD;
     CONNECT TRAN_TOTAL<1>:<1>ALL.TRAN;
{}
     SEND V(0,0,0) TO <2>TRAN_TOTAL;
                                     {Reset value for
                                     translation.}
     SEND 1 TO <4>TRAN_TOTAL;        {Signal multiplier for
                                     dial sensitivity}
SEND FIX(1) TO <1>SWITCH_TRAN;      {Default initially to
                                     mode 1}
```

```
{    Switch will be connected to control point translation
     nodes at a later point in the program.}
{
***************************************************************
********************** Rotation Dials *************************
***************************************************************
}
XMUL:=F:MULC;                          {Multipliers for dial
                                        sensitivity}
YMUL:=F:MULC;
ZMUL:=F:MULC;
CONNECT DIALS<5>:<1>XMUL;
CONNECT DIALS<6>:<1>YMUL;
CONNECT DIALS<7>:<1>ZMUL;
{}
ROTX:=F:XROTATE;                       {Convert dial signal to
                                        rotation}
ROTY:=F:YROTATE;
ROTZ:=F:ZROTATE;
     CONNECT XMUL<1>:<1>ROTX;
     CONNECT YMUL<1>:<1>ROTY;
     CONNECT ZMUL<1>:<1>ROTZ;
{}
ROT_ACCUM:=F:CMUL;                     {Accumulator for
                                        rotations}
     CONNECT ROT_ACCUM<1>:<1>ALL.ROT;
     CONNECT ROT_ACCUM<1>:<1>ROT_ACCUM;
{}
RESET_X:=F:XROTATE;                    {Upon signal from host
                                        computer, resets
                                        accumulator with given X
                                        rotation and 0 for Y and
                                        Z.}
     CONNECT RESET_X<1>:<1>ROT_ACCUM;
RESET_Y:=F:YROTATE;                    {Upon signal from host
                                        computer, adds Y
                                        rotation reset value to
                                        accumulator and triggers
                                        accumulator to fire.}
     CONNECT RESET_Y<1>:<2>ROT_ACCUM;
{
***** Create switching network for ROTATE function   *****
***** which will route dials to global rotation node *****
***** in all modes (1-3), except when in interactive *****
***** mode (4), in which case no transformations are *****
***** allowed.                                       *****
}
```

```
SWITCH_ROT:=F:CROUTE(4);
      CONNECT ROTX<1>:<2>SWITCH_ROT;
      CONNECT ROTY<1>:<2>SWITCH_ROT;
      CONNECT ROTZ<1>:<2>SWITCH_ROT;
      CONNECT SWITCH_ROT<1>:<2>ROT_ACCUM;
      CONNECT SWITCH_ROT<2>:<2>ROT_ACCUM;
      CONNECT SWITCH_ROT<3>:<2>ROT_ACCUM;
      CONNECT SWITCH_ROT<4>:<1>DISCARD;
      CONNECT FETCHMODE<1>:<1>SWITCH_ROT;
{
********** Send initial values to rotation network. *******
}
SEND 200 TO <2>XMUL;                    {Multiplication factor
                                        for dial sensitivity}
SEND 200 TO <2>YMUL;
SEND 200 TO <2>ZMUL;
SEND FIX(1) TO <1>SWITCH_ROT;           {Default initially to
                                        mode 1}
SEND M3D(1,0,0 0,1,0 0,0,1) TO <1>ROT_ACCUM;
                                        {Prime accumulator}
{
***********************************************************
*********************** Scaling Dial **********************
***********************************************************
}
SWITCH_SCALE:=F:CROUTE(4);              {See SWITCH_ROT above
                                        for description}
SCALE:=F:DSCALE;
      CONNECT DIALS<4>:<2>SWITCH_SCALE;
      CONNECT SWITCH_SCALE<1>:<1>SCALE;
      CONNECT SWITCH_SCALE<2>:<1>SCALE;
      CONNECT SWITCH_SCALE<3>:<1>SCALE;
      CONNECT SWITCH_SCALE<4>:<1>DISCARD;
      CONNECT SCALE<1>:<1>ALL.SCALE;
{}
      SEND FIX(1) TO <1>SWITCH_SCALE;
                                        {Default initially to
                                        mode 1}
      SEND 1 TO <2>SCALE;               {Reset value}
      SEND 1 TO <3>SCALE;               {Multiplication factor
                                        for dial sensitivity}
      SEND 20 TO <4>SCALE;              {Maximum scaling
                                        allowed}
      SEND 0.1 TO <5>SCALE;             {Minimum scaling
                                        allowed}
{}
CONNECT FETCHMODE<1>:<1>SWITCH_SCALE;

{
```

```
************************************************************
************************************************************
**************** Vertex Picking Network ********************
************************************************************
************************************************************
}
{
***** Enable picking when stylus tipswitch is pressed.*****
}
PICK_IF_MODE3:=F:CROUTE(4);
     CONNECT FETCHMODE<1>:<1>PICK_IF_MODE3;
     CONNECT TABLETIN<4>:<2>PICK_IF_MODE3;
                                   {Sends 'TRUE' when
                                   tipswitch pressed}
     CONNECT PICK_IF_MODE3<1>:<1>DISCARD;
                                   {Only regard stylus if
                                   in interactive mode (3)}
     CONNECT PICK_IF_MODE3<2>:<1>DISCARD;
     CONNECT PICK_IF_MODE3<4>:<1>DISCARD;
{}
PICKMODE_TOGGLE1:=F:CROUTE(2);     {Directs 'true' to
                                   proper set_picking node,
                                   depending on whether in
                                   single point or row pick
                                   modes}
     CONNECT FETCHPICKMODE<1>:<1>PICKMODE_TOGGLE1;
                                   {See variables}
     CONNECT PICK_IF_MODE3<3>:<2>PICKMODE_TOGGLE1;
                                   {Passes 'TRUE' to SET
                                   PICKING node only if in
                                   mode 3}
     CONNECT PICKMODE_TOGGLE1<1>:<1>CP_PT.SETPICK;
     CONNECT PICKMODE_TOGGLE1<2>:<1>CP_LINES.SETPICK;
{
***** Convert cursor position to picklist identifying *****
*****                the component picked.            *****
}
     CONNECT TABLETIN<6>:<1>PICK;  {Send cursor coordinates
                                   to 'PICK' for conversion
                                   to picklist}
     CONNECT PICK<2>:<2>PICKMODE_TOGGLE1;
                                   {Send 'FALSE' to
                                   SET_PICKING node to
                                   disable picking once
                                   pick occurs}
     CONNECT PICK<3>:<2>PICKMODE_TOGGLE1;
                                   {Disable picking if
                                   timeout occurs without a
                                   pick}
```

```
     SEND FALSE TO <2>PICK;          {Coordinates of picked
                                     point not required to be
                                     reported}
     SEND FIX(60) TO <3>PICK;        {Set timeout limit for
                                     when no pick occurs}
{
*****   Convert picklist info to integer identifying   *****
*****             the vertex picked.                    *****
}
PICK_ID:=F:PICKINFO;                 {Extracts pick ID (i.e.
                                     'CP32') from picklist}
     CONNECT PICK<1>:<1>PICK_ID;     {Send picklist}
{}
PICKMODE_TOGGLE2:=F:CROUTE(2);       {Sends pick index if
                                     picking a row of
                                     vertices, discards it if
                                     picking individual
                                     vertex.}
     CONNECT FETCHPICKMODE<1>:<1>PICKMODE_TOGGLE2;
     CONNECT PICK_ID<1>:<2>PICKMODE_TOGGLE2;
     CONNECT PICKMODE_TOGGLE2<1>:<1>DISCARD;
{}
PICKMODE_TOGGLE3:=F:CROUTE(2);       {Sends pick ID if
                                     picking individual
                                     vertex, discards it if
                                     picking a row of
                                     vertices.}
     CONNECT FETCHPICKMODE<1>:<1>PICKMODE_TOGGLE3;
     CONNECT PICK_ID<2>:<2>PICKMODE_TOGGLE3;
     CONNECT PICKMODE_TOGGLE3<2>:<1>DISCARD;
     CONNECT PICK_ID<3>:<1>DISCARD;
     CONNECT PICK_ID<4>:<1>DISCARD;
     CONNECT PICK_ID<5>:<1>DISCARD;
     CONNECT PICK_ID<6>:<1>DISCARD;
     CONNECT PICK_ID<7>:<1>DISCARD;
     CONNECT PICK_ID<8>:<1>DISCARD;
{}
PICK_ASCII:=F:CHARCONVERT;           {Converts characters of
                                     pick ID to stream of
                                     integers representing
                                     ASCII equivalents}
     CONNECT PICKMODE_TOGGLE3<1>:<1>PICK_ASCII;
                                     {Provides pick ID for
                                     conversion to ASCII}
     SEND TRUE TO <2>PICK_ASCII;
{}
DIGITS:=F:SUBC;                      {Converts ASCII codes of
                                     pick ID characters to
                                     numeric digits.}
```

```
     CONNECT PICK_ASCII<1>:<1>DIGITS;
                                        {Provides ASCII-coded
                                        pick ID for conversion}
     SEND FIX(48) TO <2>DIGITS;         {Subtract 48 from ASCII
                                        code}
{}
SWITCH_DIGIT:=F:ROUTE(4);               {This switch will send
                                        each of the four
                                        numerals from the
                                        converted pick ID for
                                        separate processing}
     CONNECT DIGITS<1>:<2>SWITCH_DIGIT;
                                        {Provides the pick ID
                                        digits to the switch}
{
The following is a toggling network which provides an input
of integers which progress from 1 to 4, then start again,
to the switch.  This causes each of the four successive
digits from the converted pick ID to be routed to the
corresponding output port for processing.
}
          TRIGGER_TOGGLE:=F:CONSTANT;
                                        {For each digit
                                        received of the pick ID,
                                        sends a 1 to ADD_ONE}
               SEND FIX(1) TO <2>TRIGGER_TOGGLE;
               CONNECT PICK_ASCII<1>:<1>TRIGGER_TOGGLE;
                                        {Provide a signal to
                                        trigger the toggle}
          ADD_ONE:=F:ADD;               {Provides an integer
                                        which increases by 1
                                        each time a digit is
                                        sent of the pick ID}
               CONNECT TRIGGER_TOGGLE<1>:<1>ADD_ONE;
                                        {Supply the 1 to be
                                        added each time}
               CONNECT ADD_ONE<1>:<2>ADD_ONE;
                                        {Supply the sum back to
                                        the input for
                                        incrementing next time}
               SEND FIX(3) TO <2>ADD_ONE;
                                        {Prime the summing
                                        function}
{}
          CYCLE0_3:=F:MODC;             {Converts the constantly
                                        increasing output from
                                        ADD_ONE to an integer
                                        cycling from 0 through
                                        3}
               CONNECT ADD_ONE<1>:<1>CYCLE0_3;
               SEND FIX(4) TO <2>CYCLE0_3;
```

```
{}
        CYCLE1_4:=F:ADDC;           {Converts 0-3 cyclic
                                    output from previous
                                    function to 1-4 cyclic
                                    output}
            CONNECT CYCLE0_3<1>:<1>CYCLE1_4;
            SEND FIX(1) TO <2>CYCLE1_4;
{}
        CONNECT CYCLE1_4<1>:<1>SWITCH_DIGIT;
                                    {Provide integers 1-4 to
                                    control switch}
        CONNECT SWITCH_DIGIT<1>:<1>DISCARD;
                                    {Discard the "CP"
                                    prefix}
        CONNECT SWITCH_DIGIT<2>:<1>DISCARD;
{}
        TENS_DIGIT:=F:MULC;         {Multiplies tens digit
                                    of ID number by 10}
            SEND FIX(10) TO <2>TENS_DIGIT;
            CONNECT SWITCH_DIGIT<3>:<1>TENS_DIGIT;
                                    {Provides first of two
                                    digits of ID number for
                                    conversion to tens.}
{}
        PICKED_CP:=F:ADD;           {Reconstructs the ID
                                    number of the picked
                                    vertex from the separate
                                    ones and tens digits}
            CONNECT TENS_DIGIT<1>:<1>PICKED_CP;
            CONNECT SWITCH_DIGIT<4>:<2>PICKED_CP;
{}
        PICKED:=F:NOP;              {Provides a node for
                                    other other portions of
                                    the program to obtain
                                    the picking ID from, and
                                    for the host computer to
                                    send it to in some
                                    cases.}
            CONNECT PICKED_CP<1>:<1>PICKED;
{
```

```
***** If picking rows and columns of vertices, pass    *****
*****     the pick index to the host computer for      *****
*****  conversion to a number identifying the row or   *****
*****    column to be connected to the translation     *****
*****                      dials.                      *****
}
OFFSET_PICK:=F:ADDC;                    {Adds offset to pick
                                         index to put it into
                                         range which the host
                                         program recognizes as
                                         identifying a pick
                                         index.}
     CONNECT PICKMODE_TOGGLE2<2>:<1>OFFSET_PICK;
     SEND FIX(100) TO <2>OFFSET_PICK;
     CONNECT OFFSET_PICK<1>:<1>TO_HOST;
{
After conversion by the host to a number identifying
the row or column picked, the identifier is returned
from the host in the form:

*               SEND FIX(   ) TO <1>PICKED                 *
}
{
*****   Establish a switch which directs the output    *****
*****    of the translation dials, when in mode 3,     *****
*****     to the translation node of the selected      *****
*****                    vertex.                       *****
}
SWITCH_PICK:=F:CROUTE(98);              {A switch with 98
                                         positions.  The first 80
                                         are for the 80 vertices,
                                         the next ten for the ten
                                         rows of vertices, and
                                         the last eight are for
                                         the columns of vertices}
     CONNECT SWITCH_TRAN<3>:<2>SWITCH_PICK;
                                        {Provide the dial output
                                         to the switch if in
                                         interactive mode (3).}
     {In the program section which reports vertex movement
     to the host computer, the value in PICKED will be
     reported to <1>SWITCHPICK to provide the identifying
     number of the selected vertex to the switch.}
{
The FORTRAN program on the host computer will provide
PS-300 commands which will connect the outputs of the
switch to the inputs of the translation accumulators
of the vertices.  These commands will be of the form:

*          CONNECT SWITCH_PICK<32>:<1>TRAN_TOTAL_32        *
}
```

```
{
***********************************************************
***********************************************************
********* Report Vertex Movement to Host Computer *********
***********************************************************
***********************************************************
}
CP_MOVE:=F:ACCUMULATE;                 {Accumulates translation
                                        dial signals if in mode
                                        3 (interactive mode).}
     CONNECT SWITCH_TRAN<3>:<1>CP_MOVE;
     SEND V(0,0,0) TO       <2>CP_MOVE;
     SEND 0.001    TO       <3>CP_MOVE;
                                       {Report accumulation at
                                        this interval.}
     SEND 0.1      TO       <4>CP_MOVE;
                                       {Scale signal to match
                                        scaling at individual
                                        vertex accumulators.}
     RESET_CP_MOVE:=F:CONSTANT;        {Resets accumulator to 0
                                        each time the
                                        accumulator fires, so
                                        only delta values are
                                        sent to host computer.}
          CONNECT CP_MOVE<1>:<1>RESET_CP_MOVE;
          SEND V(0,0,0) TO   <2>RESET_CP_MOVE;
          CONNECT RESET_CP_MOVE<1>:<2>CP_MOVE;
{}
MOVE_SYNC:=F:SYNC(2);                  {Ensures that currently
                                        picked vertex identifier
                                        is reported to the host
                                        computer immediately
                                        before the vertex
                                        position delta is
                                        reported.}
          MOVE_THIS_ONE:=F:CONSTANT;
                                       {Sends the current
                                        picked vertex identifier
                                        to MOVE_SYNC each time
                                        the CP_MOVE accumulator
                                        fires.}
               CONNECT CP_MOVE<1>:<1>MOVE_THIS_ONE;
               {PICKED2<1> will be connected to
               <2>MOVE_THIS_ONE below.}
     CONNECT MOVE_THIS_ONE<1>:<1>MOVE_SYNC;
     CONNECT CP_MOVE<1>:<2>MOVE_SYNC;
{}
     CONNECT MOVE_SYNC<1>:<1>TO_HOST;
     CONNECT MOVE_SYNC<2>:<1>TO_HOST;
{
```

```
***** Before reusing the accumulator, CP_MOVE, for     *****
***** another vertex, report any remnant in the        *****
***** accumulator to the host computer.                *****
}
ZERO:=F:CONSTANT;                          {Provides a zero to be
                                           sent to <3>CP_MOVE,
                                           allowing accumulator to
                                           fire without waiting to
                                           reach normal firing
                                           increment.}
     CONNECT PICKED<1>:<1>ZERO;            {Trigger the accumulator
                                           firing increment to be
                                           reset to zero whenever a
                                           new vertex is picked for
                                           movement.}
     SEND 0 TO <2>ZERO;
VECZERO:=F:CONSTANT;                       {Sends a zero vector to
                                           <1>CP_MOVE, causing the
                                           accumulator to fire.}
     CONNECT ZERO<1>:<1>VECZERO;           {Send the zero vector
                                           when triggered by output
                                           from ZERO.}
     SEND V(0,0,0) TO <2>VECZERO;
ZERO_FIRST:=F:SYNC(2);                     {Ensures <3>CP_MOVE is
                                           set to zero before
                                           <1>CP_MOVE is provided
                                           with signal to fire
                                           accumulator.}
     CONNECT ZERO<1>:<1>ZERO_FIRST;
     CONNECT VECZERO<1>:<2>ZERO_FIRST;
     CONNECT ZERO_FIRST<1>:<3>CP_MOVE;
     CONNECT ZERO_FIRST<2>:<1>CP_MOVE;
STEP_SIZE:=F:CONSTANT;                     {After CP_MOVE fires,
                                           resets CP_MOVE firing
                                           increment.}
     CONNECT CP_MOVE<1>:<1>STEP_SIZE;
     SEND 0.005 TO      <2>STEP_SIZE;
     CONNECT STEP_SIZE<1>:<3>CP_MOVE;
SET_PICKED2:=F:CONSTANT;                   {Sends new value of
                                           PICKED to PICKED2, after
                                           any remnant in the
                                           CP_MOVE accumulator has
                                           been dumped to the host
                                           for addition to the
                                           position of the
                                           previously picked
                                           vertex.}
     CONNECT PICKED<1>:<2>SET_PICKED2;
     CONNECT STEP_SIZE<1>:<1>SET_PICKED2;
```

```
PICKED2:=F:NOP;                          {Contains identifier of
                                         currently selected
                                         vertex, but is not
                                         updated until CP_MOVE
                                         accumulator remnant has
                                         been reported in
                                         accordance with previous
                                         value of PICKED2.}
     CONNECT SET_PICKED2<1>:<1>PICKED2;
     CONNECT PICKED2<1>:<1>SWITCH_PICK;
                                         {See SWITCH_PICK near
                                         end of Vertex Picking
                                         Network.}
     CONNECT PICKED2<1>:<2>MOVE_THIS_ONE;
                                         {See MOVE_SYNC above.}
     SEND FIX(1) TO <1>PICKED2;
{
************************************************************
************************************************************
************************ Clipping **************************
************************************************************
************************************************************
}
CLIP_TRUE:=F:INPUTS_CHOOSE(3);           {Send a FALSE if clip-
                                         ping is off, TRUE if
                                         clipping is on.}
     SEND FALSE TO                  <1>CLIP_TRUE;
     SEND TRUE  TO                  <2>CLIP_TRUE;
     CONNECT FETCHCLIPMODE<1>:<3>CLIP_TRUE;
     CONNECT CLIP_TRUE<1>:<1>CLIPON;{Enable clipping.}
     CONNECT CLIP_TRUE<1>:<1>CLIP_INTENSITY;
                                         {Disable depth cueing}
CLIP_DIAL:=F:CROUTE(2);                  {Routes signal from dial
                                         8 to control clipping
                                         window width if clipping
                                         enabled.}
     CONNECT FETCHCLIPMODE<1>:<1>CLIP_DIAL;
     CONNECT        DIALS<8>:<2>CLIP_DIAL;
     CONNECT CLIP_DIAL<1>:<1>DISCARD;
                                         {Discard signal from
                                         dial 8 if clipping not
                                         enabled.}
CLIP_TOTAL:=F:ACCUMULATE;                {Accumulate dial signal}
     CONNECT CLIP_DIAL<2>:<1>CLIP_TOTAL;
                                         {Routes signal from dial
                                         to accumulator only if
                                         clipping is on (CLIPMODE
                                         =2}
     SEND 0.5    TO <2>CLIP_TOTAL;       {Reset value.}
     SEND 0.1    TO <4>CLIP_TOTAL;       {Scale dial signal and
                                         convert to real number.}
```

```
     SEND 1      TO <5>CLIP_TOTAL; {Maximum clipping width}
     SEND 0.002 TO <6>CLIP_TOTAL;  {Minimum clipping width}
CLIP_SYNC:=F:SYNC(2);              {Ensures clipping window
                                   limit is reset before
                                   the new window matrix is
                                   sent to CLIPPER.}
     CONNECT CLIP_TOTAL<1>:<1>CLIP_SYNC;
     CONNECT CLIP_TOTAL<1>:<2>CLIP_SYNC;
WINDOW_MATRIX:=F:WINDOW;           {Provides a new matrix
                                   defining the window with
                                   the new clipping width.}
     SEND -1 TO <2>WINDOW_MATRIX;  {Window X minimum.}
     SEND  1 TO <3>WINDOW_MATRIX;  {Window X maximum.}
     SEND -1 TO <4>WINDOW_MATRIX;  {Window Y minimum.}
     SEND  1 TO <5>WINDOW_MATRIX;  {Window Y maximum.}
     SEND  0 TO <6>WINDOW_MATRIX;  {Front Z-clipping plane}
     CONNECT CLIP_SYNC<1>:<7>WINDOW_MATRIX;
                                   {Back Z-clipping plane
                                   defined by dial signal}
     CONNECT CLIP_SYNC<2>:<1>WINDOW_MATRIX;
                                   {Trigger WINDOW_MATRIX
                                   to send new matrix.}
CONNECT WINDOW_MATRIX<1>:<1>CLIPPER;
{
***********************************************************
***********************************************************
************* Make Function Keys Operational ************
***********************************************************
***********************************************************
}
KEY_OFFSET:=F:ADDC;                {Add offset to function
                                   key signals to put them
                                   into range which host
                                   computer program
                                   recognizes as indicating
                                   a pressed function key}
     SEND FIX(300) TO <2>KEY_OFFSET;
     CONNECT FKEYS<1>:<1>KEY_OFFSET;
     CONNECT KEY_OFFSET<1>:<1>TO_HOST;
{
```

```
***************Label the Function Keys and Dials **********
}
KEY1:=F:INPUTS_CHOOSE(5);             {Stores labels for
                                      FKEY 1}
      SEND 'VERTICES' TO <1>KEY1;
      SEND 'VERTICES' TO <2>KEY1;
      SEND 'VERTICES' TO <3>KEY1;
      SEND 'PRT VERT' TO <4>KEY1;
      CONNECT FETCHMODE<1>:<5>KEY1;  {Notifies KEY1 of which
                                      label to use, depending
                                      on mode.}

      CONNECT KEY1<1>:<1>FLABEL1;    {Sends label to
                                      keyboard.}
KEY2:=F:INPUTS_CHOOSE(5);
      SEND 'B-SPLINE' TO <1>KEY2;
      SEND 'B-SPLINE' TO <2>KEY2;
      SEND 'B-SPLINE' TO <3>KEY2;
      SEND ' ' TO <4>KEY2;
      CONNECT FETCHMODE<1>:<5>KEY2;
      CONNECT KEY2<1>:<1>FLABEL2;
KEY3:=F:INPUTS_CHOOSE(5);
      SEND 'KNOWN PT' TO <1>KEY3;
      SEND 'KNOWN PT' TO <2>KEY3;
      SEND 'KNOWN PT' TO <3>KEY3;
      SEND ' ' TO <4>KEY3;
      CONNECT FETCHMODE<1>:<5>KEY3;
      CONNECT KEY3<1>:<1>FLABEL3;
KEY4:=F:INPUTS_CHOOSE(5);
      SEND 'INTERACT' TO <1>KEY4;
      SEND ' ' TO <2>KEY4;
      SEND 'INTERACT' TO <3>KEY4;
      SEND 'PLT SCRN' TO <4>KEY4;
      CONNECT FETCHMODE<1>:<5>KEY4;
      CONNECT KEY4<1>:<1>FLABEL4;
{}
KEY5:=F:INPUTS_CHOOSE(5);
      SEND 'CLIP' TO <1>KEY5;
      SEND 'CLIP' TO <2>KEY5;
      SEND 'CLIP' TO <3>KEY5;
      SEND ' ' TO <4>KEY5;
      CONNECT FETCHMODE<1>:<5>KEY5;
      CONNECT KEY5<1>:<1>FLABEL5;
{}
KEY6:=F:INPUTS_CHOOSE(5);
      SEND 'REFLECT' TO <1>KEY6;
      SEND 'REFLECT' TO <2>KEY6;
      SEND ' ' TO <3>KEY6;
      SEND ' ' TO <4>KEY6;
      CONNECT FETCHMODE<1>:<5>KEY6;
      CONNECT KEY6<1>:<1>FLABEL6;
```

```
{}
KEY7:=F:INPUTS_CHOOSE(5);
     SEND 'XY' TO <1>KEY7;
     SEND 'XY' TO <2>KEY7;
     KEY7_MODE3:=F:INPUTS_CHOOSE(3);
          SEND 'PICK ROW' TO <1>KEY7_MODE3;
          SEND ' '        TO <2>KEY7_MODE3;
          CONNECT FETCHPICKMODE<1>:<3>KEY7_MODE3;
               KEY7_SYNC:=F:SYNC(2);{Ensures new PICKMODE
                                    value is in place on
                                    KEY7_MODE3 before new
                                    MODE value is sent to
                                    KEY7, triggering that
                                    function.}
                    CONNECT KEY7_MODE3<1>:<1>KEY7_SYNC;
                    CONNECT  FETCHMODE<1>:<2>KEY7_SYNC;
     CONNECT KEY7_SYNC<1>:<3>KEY7;
     SEND ' ' TO           <4>KEY7;
     CONNECT  KEY7_SYNC<2>:<5>KEY7;
     CONNECT KEY7<1>:<1>FLABEL7;
{}
KEY8:=F:INPUTS_CHOOSE(5);
     SEND 'XZ' TO <1>KEY8;
     SEND 'XZ' TO <2>KEY8;
     KEY8_MODE3:=F:INPUTS_CHOOSE(3);
          SEND ' '       TO <1>KEY8_MODE3;
          SEND 'PICK PT' TO <2>KEY8_MODE3;
          CONNECT FETCHPICKMODE<1>:<3>KEY8_MODE3;
               KEY8_SYNC:=F:SYNC(2);
                    CONNECT KEY8_MODE3<1>:<1>KEY8_SYNC;
                    CONNECT  FETCHMODE<1>:<2>KEY8_SYNC;
     CONNECT KEY8_SYNC<1>:<3>KEY8;
     SEND 'SAV VERT'    TO <4>KEY8;
     CONNECT  KEY8_SYNC<2>:<5>KEY8;
     CONNECT KEY8<1>:<1>FLABEL8;
{}
KEY9:=F:INPUTS_CHOOSE(5);
     SEND 'YZ' TO <1>KEY9;
     SEND 'YZ' TO <2>KEY9;
     SEND ' ' TO <3>KEY9;
     SEND 'LD VERT' TO <4>KEY9;
     CONNECT FETCHMODE<1>:<5>KEY9;
     CONNECT KEY9<1>:<1>FLABEL9;
{}
KEY10:=F:INPUTS_CHOOSE(5);
     SEND 'I/O' TO <1>KEY10;
     SEND 'I/O' TO <2>KEY10;
     SEND 'I/O' TO <3>KEY10;
     SEND ' ' TO <4>KEY10;
     CONNECT FETCHMODE<1>:<5>KEY10;
     CONNECT KEY10<1>:<1>FLABEL10;
```

```
KEY11:=F:INPUTS_CHOOSE(5);
      SEND 'RESET' TO <1>KEY11;
      SEND 'RESET' TO <2>KEY11;
      SEND 'RESET' TO <3>KEY11;
      SEND ' ' TO <4>KEY11;
      CONNECT FETCHMODE<1>:<5>KEY11;
      CONNECT KEY11<1>:<1>FLABEL11;
{}
KEY12:=F:INPUTS_CHOOSE(5);
      SEND 'QUIT' TO <1>KEY12;
      SEND 'QUIT' TO <2>KEY12;
      SEND 'QUIT' TO <3>KEY12;
      SEND 'EXIT I/O' TO <4>KEY12;
      CONNECT FETCHMODE<1>:<5>KEY12;
      CONNECT KEY12<1>:<1>FLABEL12;
{}
DIAL1:=F:INPUTS_CHOOSE(5);
      SEND 'GLOBAL X' TO <1>DIAL1;
      SEND 'GLOBAL X' TO <2>DIAL1;
      SEND 'VERTEX X' TO <3>DIAL1;
      SEND ' ' TO <4>DIAL1;
      CONNECT FETCHMODE<1>:<5>DIAL1;
      CONNECT DIAL1<1>:<1>DLABEL1;
{}
DIAL2:=F:INPUTS_CHOOSE(5);
      SEND 'GLOBAL Y' TO <1>DIAL2;
      SEND 'GLOBAL Y' TO <2>DIAL2;
      SEND 'VERTEX Y' TO <3>DIAL2;
      SEND ' ' TO <4>DIAL2;
      CONNECT FETCHMODE<1>:<5>DIAL2;
      CONNECT DIAL2<1>:<1>DLABEL2;
{}
DIAL3:=F:INPUTS_CHOOSE(5);
      SEND 'GLOBAL Z' TO <1>DIAL3;
      SEND 'GLOBAL Z' TO <2>DIAL3;
      SEND 'VERTEX Z' TO <3>DIAL3;
      SEND ' ' TO <4>DIAL3;
      CONNECT FETCHMODE<1>:<5>DIAL3;
      CONNECT DIAL3<1>:<1>DLABEL3;
{}
      BLINKER:=F:INPUTS_CHOOSE(5);   {Cause translation dials
                                      to blink when in inter-
                                      active mode.}
            SEND FALSE TO        <1>BLINKER;
            SEND FALSE TO        <2>BLINKER;
            SEND TRUE  TO        <3>BLINKER;
            SEND FALSE TO        <4>BLINKER;
            CONNECT FETCHMODE<1>:<5>BLINKER;
            CONNECT BLINKER<1>:<2>DLABEL1;
            CONNECT BLINKER<1>:<2>DLABEL2;
            CONNECT BLINKER<1>:<2>DLABEL3;
```

```
{}
DIAL4:=F:INPUTS_CHOOSE(5);
     SEND 'SCALING' TO <1>DIAL4;
     SEND 'SCALING' TO <2>DIAL4;
     SEND 'SCALING' TO <3>DIAL4;
     SEND ' ' TO <4>DIAL4;
     CONNECT FETCHMODE<1>:<5>DIAL4;

     CONNECT DIAL4<1>:<1>DLABEL4;
{}
DIAL5:=F:INPUTS_CHOOSE(5);
     SEND 'ROTATE X' TO <1>DIAL5;
     SEND 'ROTATE X' TO <2>DIAL5;
     SEND 'ROTATE X' TO <3>DIAL5;
     SEND ' ' TO <4>DIAL5;
     CONNECT FETCHMODE<1>:<5>DIAL5;
     CONNECT DIAL5<1>:<1>DLABEL5;
{}
DIAL6:=F:INPUTS_CHOOSE(5);
     SEND 'ROTATE Y' TO <1>DIAL6;
     SEND 'ROTATE Y' TO <2>DIAL6;
     SEND 'ROTATE Y' TO <3>DIAL6;
     SEND ' ' TO <4>DIAL6;
     CONNECT FETCHMODE<1>:<5>DIAL6;
     CONNECT DIAL6<1>:<1>DLABEL6;
{}
DIAL7:=F:INPUTS_CHOOSE(5);
     SEND 'ROTATE Z' TO <1>DIAL7;
     SEND 'ROTATE Z' TO <2>DIAL7;
     SEND 'ROTATE Z' TO <3>DIAL7;
     SEND ' ' TO <4>DIAL7;
     CONNECT FETCHMODE<1>:<5>DIAL7;
     CONNECT DIAL7<1>:<1>DLABEL7;
{}
DIAL8:=F:INPUTS_CHOOSE(3);
     SEND ' ' TO <1>DIAL8;
     SEND 'CLIPPING' TO <2>DIAL8;
     CONNECT FETCHCLIPMODE<1>:<3>DIAL8;
     CONNECT DIAL8<1>:<1>DLABEL8;
{}
SEND 1 TO <1>NEW_MODE;                    {Trigger initial sending
                                           of default operating
                                           mode indicators.
                                           Whenever an operating
                                           mode indicator
                                           (variable) changes, the
                                           host computer will send
                                           a similar triggering
                                           message.}
```

APPENDIX D

LINK COMMAND FILE

The following is the command file used to link SPLINE.FTN with the various host-resident PS 300 library subroutines.

```
SPLINE/FP/CP/ID,=spline,bd
[1,1]MELIB/LB
[5,200]ENSNEW/LB
/
LIBR=F4PRES:RO
ACTFIL=4
UNITS=8
ASG=TI46:7
ASG=SY0:6
PRI=50
//
```

Thesis
B82912 Bryant
c.1 A high-level computer
graphics implementation
of three-dimensional B-
spline surface generation.