DAA / LANGLEY

NAG1-391

(NASA-CR-175803) A FAST HIDDEN LINE ALGORITHM WITH CONTOUR OPTION M.S. Thesis (Brigham Young Univ.) 123 p HC A06/MF A01 CSCL 12A

N85-27582

Unclas
G3/64 15091

A FAST HIDDEN LINE ALGORITHM WITH CONTOUR OPTION

Ronald B. Thue

Engineering Research

College of Engineering Sciences and Technology

Brigham Young University

A FAST HIDDEN LINE ALGORITHM
WITH CONTOUR OPTION

A Thesis

Presented to the

Department of Civil Engineering

Brigham Young University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Ronald B. Thue

December 1984

This thesis, by Ronald B. Thue, is accepted in its present form by the Department of Civil Engineering of Brigham Young University as satisfying the thesis requirement for the degree of Master of Science.

Michael B. Stephenson, Committee Chairman

Steven E. Benzley, Committee Member

28 September 84

Date

Henry N. Christiansen, Department Chairman

ACKNOWLEDGEMENTS

The author thanks his wife for her patience and support as they worked together to arrive at this point. She has endured much, for which she deserves more recognition than these few words can provide.

He is indebted to Henry N. Christiansen and Michael B. Stephenson for the opportunity of being a part of their research groups. He also thanks Michael B. Stephenson, Dan S. Compton and Steven C. Macy for their many helps and suggestions through the course of this work.

The work reported in this thesis was made possible, in part, through NASA grant number NAG-1-391.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The Hidden Line Problem

To establish the need for this work, the author feels it is important that the reader obtain a basic understanding of the hidden line problem; this introduction is devoted to providing such an understanding.

Figure 1-1 depicts a simple box prior to hidden line removal. Clearly it is difficult to get an accurate understanding of the actual orientation of the box; is it oriented as shown in Figure 1-1b or as in Figure 1-1c? Hence, one can see the great value in a hidden line representation of a model.

The hidden line problem has been handled by many different individuals [1], each with some uniqueness in their approach. But, aside from the uniqueness of individual approaches there are several considerations that are common to all hidden line removal methods. A number of simple illustrations will aid in a discussion of these similarities, and thereby provide some general insights into the hidden line problem.

In the wire frame model of the box (Figure 1-1a), each of its sides represents a surface; each of the surfaces has the capability of hiding any lines which may lie behind them. So the problem becomes one of determining

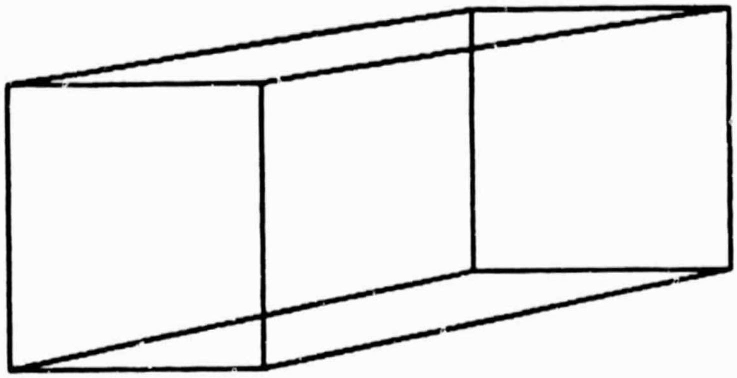


FIGURE 1-1A

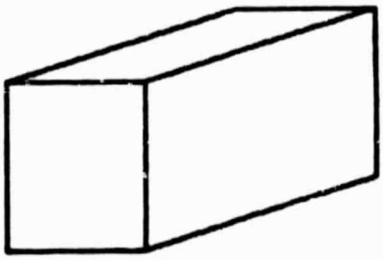


FIGURE 1-1B

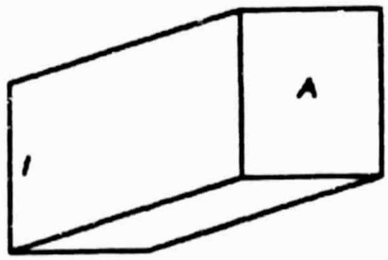


FIGURE 1-1C

which surfaces are in front of which lines. To make this determination, a number of common testing procedures are used.

First, and perhaps most simple, is the min-max test. In Figure 1-1c it is apparent that side A can in no way obstruct line 1. The min-max test would determine this by comparing the minimum and maximum coordinate values of surface A to the minimum and maximum coordinate values of line 1. This is analogous to drawing boxes around the coordinate extremes of surface A and line 1 as shown in Figure 1-2a. If the two boxes do not intersect there can be no intersection of the surface and the line, and therefore, the surface cannot hide the line. If the opposite is true, then there is a possibility that the surface and the line intersect, and further testing must be done to make this determination.

This further testing comes in the form of what is called a "surroundedness test". Figure 1-2b demonstrates the need for such a test since the min-max test alone cannot eliminate the possibility that the surface and the line intersect. The surroundedness test takes point "X" on line 1 and determines whether it is surrounded by surface A or outside of surface A; if it is outside of surface A, the line and the surface do not intersect, and if the opposite is true, it has been conclusively determined that the surface and the line do intersect in the X-Y plane.

When it has been determined that the two elements intersect, all that remains is to determine whether the surface is in front of the line or vice versa. This is accomplished by the "depth test". The depth test compares the minimum and maximum depth values between surface A and line 1; if the minimum surface depth is greater than the maximum line depth, then the surface is behind the line and cannot hide it (see Figure 1-2c). On the other hand, if the maximum surface depth is less than the minimum line depth, then the surface will hide all or a part of the line (see Figure 1-2d).

The preceding has been a very simplistic discussion of the basic concepts in hidden line removal. There are many sorting methods employed in the

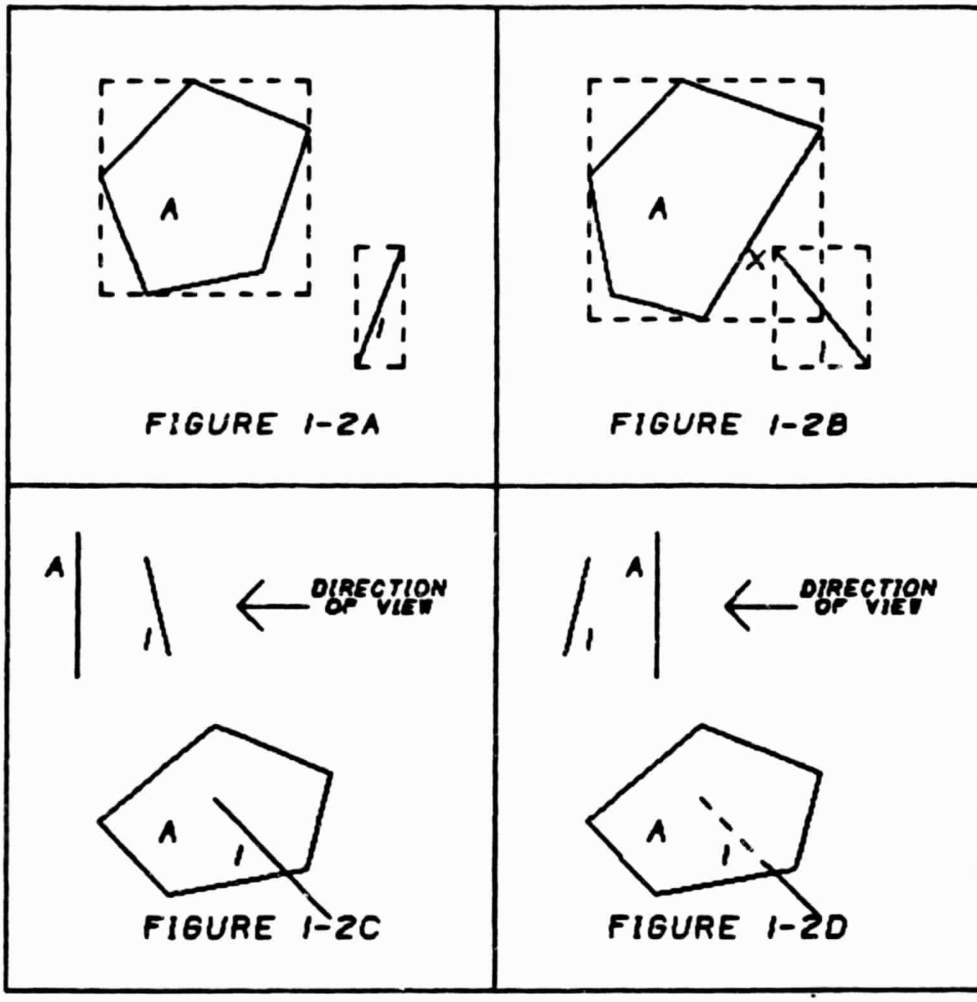


FIGURE 1-2A FIGURE 1-2B FIGURE 1-2C FIGURE 1-2D

different hidden line algorithms which speed up the above discussed processes. Further, the situations where a surface only partly hides a line or where surfaces penetrate into each other have not been discussed, yet must be considered. For a more in depth discussion of the hidden line problem the reader is referred to "A Characterization of 10 Hidden-Surface Algorithms" [1] and Principles of Interactive Computer Graphics [2].

The Need for a Fast Hidden Line Algorithm With Optional Contours

Perhaps the best evidence of the need for a fast hidden line algorithm in the MOVIE.BYU display package is found in the speed comparisons between the modified JonesD algorithm and the MOVIE.BYU modified Watkin's algorithm discussed in Chapter 6. The fact is, the Watkin's Algorithm is a scan conversion algorithm which was written to facilitate the generation of continuous tone images, and as such, carries with it all the overhead required to produce such pictures.[3] This overhead, coupled with the overhead required to allow enhanced continuous tone images (ie. shadows and transparencies) has made the MOVIE.BYU modified version of the Watkin's algorithm quite slow. Further, when contours are desired, the MOVIE.BYU modified Watkin's algorithm goes through a very time consuming Gouraud interpolation scheme to generate contour segments [3]. On the other hand, the vector nature of the JonesD algorithm allows the minimization of interpolation requirements in contour generation and this greatly speeds up the process. In light of the above, the need for a fast hidden line algorithm with accompanying contour option becomes clear.

CHAPTER 2

THE JONESD ALGORITHM

At this point a discussion of the JonesD Algorithm is in order. This discussion will include reasons for the Algorithm's selection, a description of how the algorithm works, and finally a discussion of the algorithm's limitations. To aid the reader in obtaining a general understanding of the algorithm, Figure 2-1 has been provided. The author suggests that the reader take a moment to study the flow chart, as doing so will greatly aid in understanding what follows; in fact, each time the reader encounters a flowchart through the course of this work, this same suggestion is recommended.

Why JonesD was Selected

There are four reasons that the JonesD algorithm was selected. The first and primary reason was the speed its author, Gary Jones, attributed to it. He did some quite elaborate testing of his algorithm against other hidden line algorithms and showed that his algorithm was faster in every case [4]. Second, the JonesD algorithm was written specifically to handle the hidden line problem which aided in making its claimed speed believable. Third, because

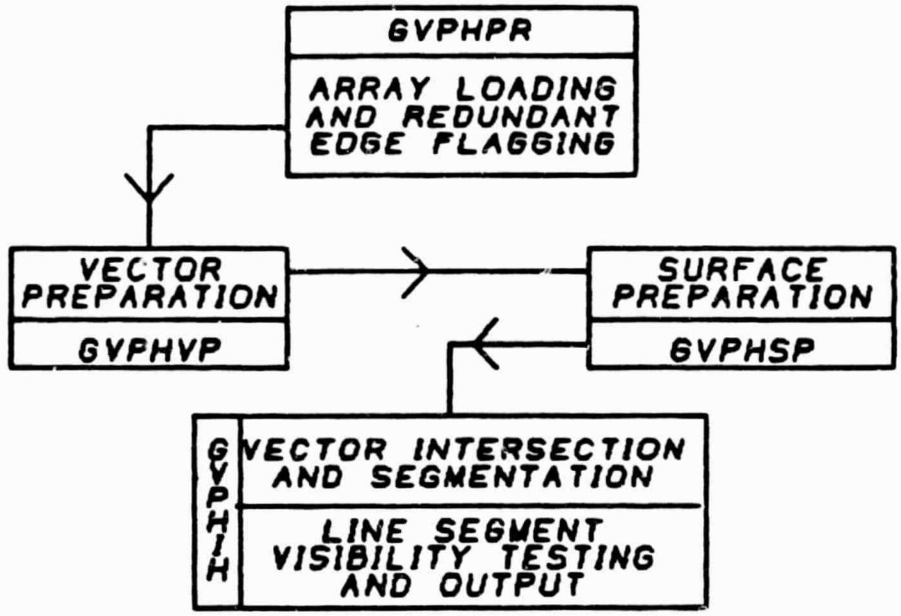


FIGURE 2-1
JONESD FLOWCHART

the algorithm made provision for line elements, it was nicely suited for handling contour vectors. Finally, although the JonesD algorithm had some limitations, which will be discussed later, these limitations were considered either tolerable or surmountable.

How the JonesD Algorithm Works

Preprocessing Requirements

The JonesD algorithm requires significant preprocessing which results in more than worthwhile time savings downstream. The basic components of this preprocessing are (1) the hashing of redundant nodes, (2) the creation of a vector list from the model element connectivity or surface list and (3) the hashing of redundant edges. A more detailed explanation of these preprocessing methods and the reasons for their use follows.

(1) Hashing Redundant Nodes

The hashing algorithm employed in the elimination of redundant nodes takes the X, Y, and Z coordinate values of each node, and from these values generates a key into the hash table. The hash table internal numbers are then used as indices into an array that contains the node numbers for the coordinates. Now, each time that the same key is generated a redundant node will have been found. So rather than enter the redundant coordinate into the coordinate array, it is skipped, and the hashing internal number is then used to retrieve the node number previously assigned to the coordinate values. The node number retrieved is then used in the connectivity list for the element being processed. Once the redundant nodes have been eliminated the next preprocessing step, the creation of a vector list, is possible.

(2) Creating the Vector List

The vector list referred to here is simply an array which contains pointers into the JonesD coordinate array for the endpoints of each of the vectors in the model; this list is required by the JonesD hidden line processor which will be discussed later in this chapter. The vector list is created by looping through the element connectivity array and creating a new two dimensional array containing the endpoints (pointers into PGRID) for each vector in the model. Figure 2-2 illustrates this process. While the vector list is being created it is advantageous to flag redundant vectors, since such flagging dramatically reduces the amount of computation that the JonesD algorithm must do.

(3) Hashing Redundant Edges

The same hashing algorithm used to eliminate redundant nodes is employed in the flagging of redundant edges. The vector endpoints are first arranged in ascending order after which a hashing key is generated from the two endpoint values. For each time the same key is generated, a redundant edge flag (JREDUN) is incremented by 1 and assigned to the vector being worked with. Consequently, the vectors with a redundant edge flag greater than 1 are ignored (Figure 2-2) in the JonesD hidden line processor. It should be pointed out that while the edges are being flagged for redundancy, they are also flagged as being edges of surface elements (JVTYPE). This is done to

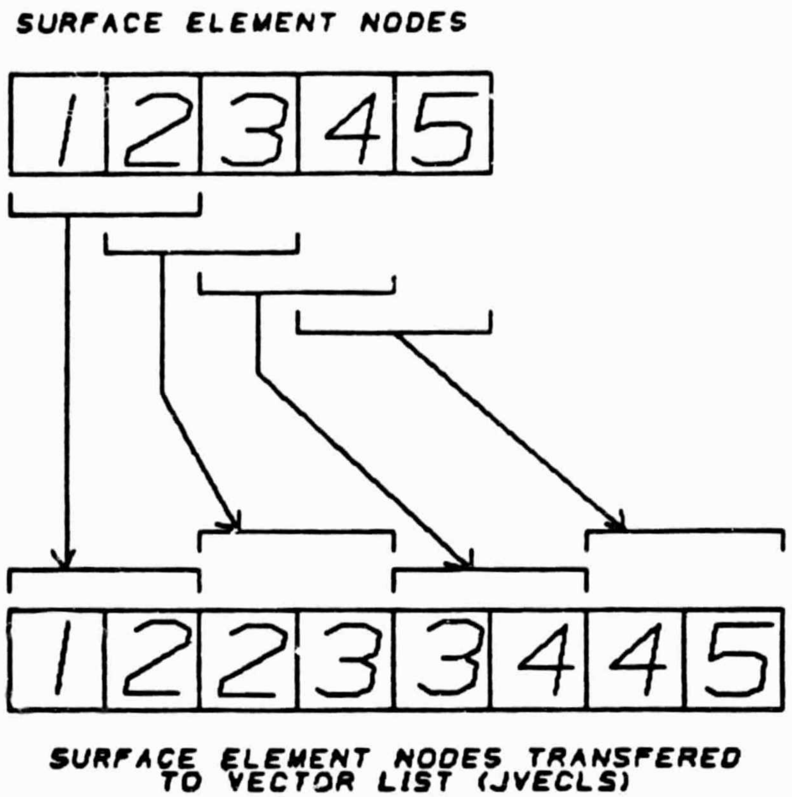


FIGURE 2-2
LOADING THE VECTOR LIST

distinguish surface vectors from strictly line element vectors such as contours; this again speeds the operation of the JonesD processor.

For more information about the hashing algorithm used in this work the reader is referred to [5] and Appendix C.

JonesD Hidden Line Preprocessing

In addition to the preprocessing needed to get the data required by the JonesD algorithm, the actual processor itself does a significant amount of preprocessing. This preprocessing involves bucket sorting, vector preparation, and surface preparation.

Bucket Sorting

The JonesD Algorithm employs a number of processes that increase speed of operation. One of these processes is the sorting of both surface elements and surface vectors into buckets. These buckets are actually cells in an X-Y grid. The number of grids that the screen is divided into depends on the number of surface elements and surface vectors in the model. Further, the bounds for the grid are determined from the minimum and maximum X and Y transformed model coordinates; this results, hopefully, in a fairly even distribution of surface elements and surface vectors among the grid cells (buckets). Figure 2-3 visually depicts the preceding discussion. All of the bucket sorting information required by the JonesD processor is set up in the subroutine GVPHGD.

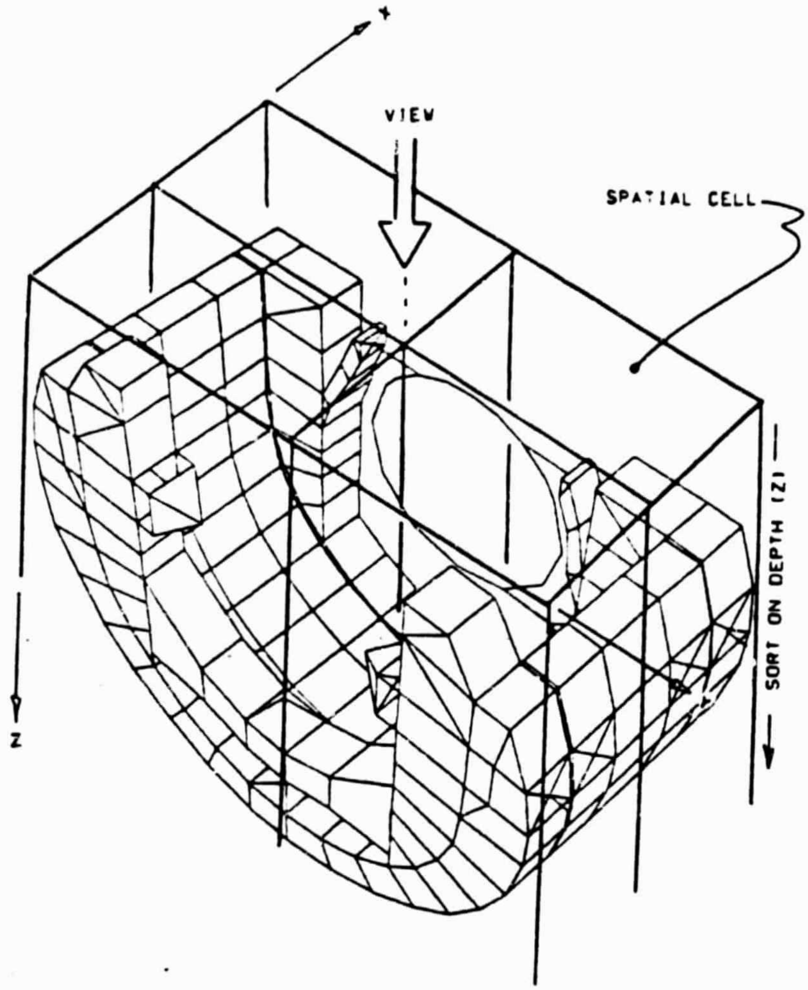


Figure 2-3
Spatial Cells used in X-Y Bucket Sorting
From Reference [4]

Vector Preparation

Following is an explanation of the vector preparation that is performed in subroutine GVPHVP and diagramed in Figure 2-4. The process begins by loading the VECTOR array with information about each of the vectors that

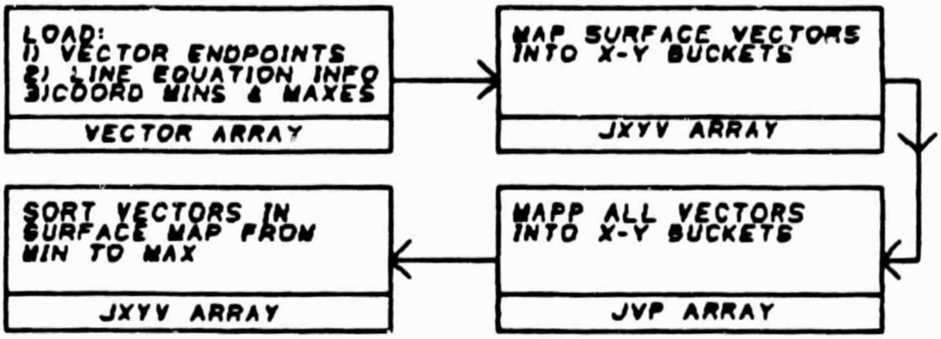


FIGURE 2-4
VECTOR PREPARATION FLOWCHART

has not been flagged as redundant; this information includes endpoint coordinates, line equation information, and minimum and maximum coordinate values for each vector. The surface vectors are then mapped into a vector map (JXYV) according to the cells (buckets) that their endpoints begin and end in, along with any cells that the vector could potentially cross (Figure 2-3). All of the vectors, surface or line elements, are them mapped into an additional array (JVP). Finally, the vectors in the surface vector map are sorted according to depth beginning with the vector closest to the eye of the observer and progressing back from there; a Shell sort is used to accomplish this. The reasons for this vector preparation will become evident in the description of the JonesD hidden line processor.

Surface Preparation

A brief explanation of the surface preparation performed in subroutine GVPHSP and diagramed in Figure 2-5 follows. This process begins by loading the XP, YP, and ZP arrays with the coordinates of the first element being processed. The maximum and minimum X, Y, and Z coordinate values are then determined and loaded into the SURF array. Once this is accomplished, the surface is mapped into the surface map (JXYS) based on its minimum and maximum values in X and Y (Figure 2-3); these minimum and maximum values determine which cells (buckets) the surface might lie in. Next, the area of the surface is computed by taking cross products between nodes, and is then loaded into a location in the SURF array. During the area calculation, the component areas from each cross product are stored in the AR array for later use in the JonesD processor. The above steps are repeated for each element. The final step in surface preparation involves the sorting of the surfaces according to their depth; a Shell sort is used which sorts the surfaces from front to back according to their minimum Z coordinates. As with the vector preparation, the reasons for the surface preparation will become evident in the discussion of the JonesD hidden line processor.

For a more detailed discussion of surface and vector preparation, the reader is referred to the comments in subroutines GVPHSP and GVPHVP in Appendix A.

The JonesD Hidden Line Processor

The JonesD hidden line processor, which can be found in subroutine GVPHID, can be divided into two basic parts. The first part computes vector intersection locations, and the second part computes the visibility of line segments and outputs them to the graphics display device.

Vector Intersections

The flow chart, Figure 2-6, visually illustrates the processes involved in the computation of vector intersection locations, and the segmentation of vectors at such intersection points where required. As can be seen in Figure 2-6, vectors are handled one by one starting at the beginning of the vector list (JVECLS) and proceeding through to the end of it. The main function of this section of the processor is to determine if any surface vectors lie in front of the vector currently being processed; if this is the case, the vector being processed must be divided at all points of X-Y intersection with surface vectors. The reason for this, as shown in Figure 2-7, is that every vector that crosses an element boundary must be divided at that boundary; this insures that all

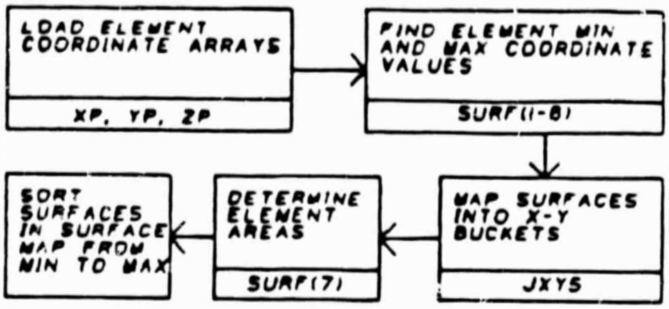


FIGURE 2-5
SURFACE PREPARATION FLOWCHART

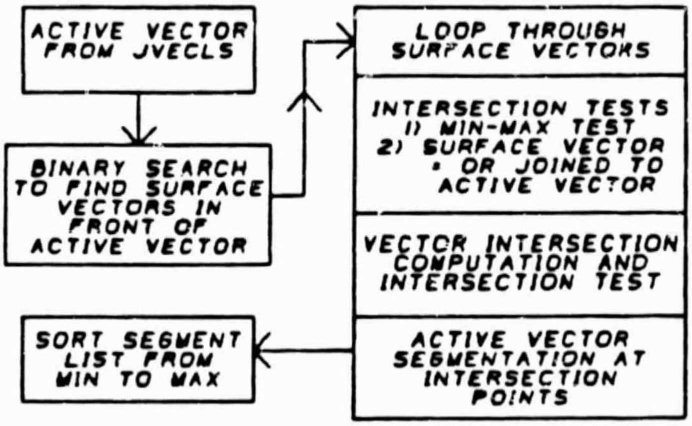


FIGURE 2-6
VECTOR INTERSECTION FLOWCHART

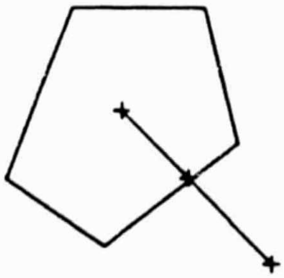


FIGURE 2-7
ACTIVE VECTOR SEGMENTED
AT SURFACE BOUNDARY

vector segments will be completely in front of, or completely behind surfaces.

The vector intersection calculation begins by locating all surface vectors that lie in the same bucket or buckets as the vector being processed (active vector), and which are in front of the active vector. This is easily accomplished because of the vector mapping and sorting, along with the Z coordinate minimum and maximum determination, that was done for each vector in subroutine GVPHVP. A binary search locates, from the sorted surface vector map (JXYV), the dividing point between surfaces that lie in front of, and behind, the minimum Z coordinate of the active vector. This idea is shown graphically in Figure 2-8a. Once the appropriate surface vectors have been located, each one must fail two trivial rejection tests before an intersection location will be calculated. The first of these two tests is the min-max test that was discussed in the introduction to this work. If the bounding box around the surface vector being looked at, and the bounding box around the active vector do not overlap, there is no need for further testing of the current surface vector and the algorithm goes immediately to the next one. If, on the other hand, the min-max test reveals a possible intersection between the two vectors, a second rejection test is employed. This test compares the endpoints of the two vectors since the surface vector and the active vector could either be one and the same, or could be connected to each other. In either case there would be no point in computing the intersection between the two. If the surface vector is not rejected by either of these two tests, calculations to determine a possible point of intersection between the two are begun; these calculations are accomplished via the line equation data for each of the vectors that was created in subroutine GVPHVP. As possible coordinate intersections are determined, they are checked against the minimum and maximum values belonging to the active vector. If these minimums and maximums lie outside the computed coordinate intersection location, the surface vector can be rejected (Figure 2-8b). Finally, if a point of intersection is located, the active vector is divided into two line segments at that point of intersection. Once the active vector has been divided into

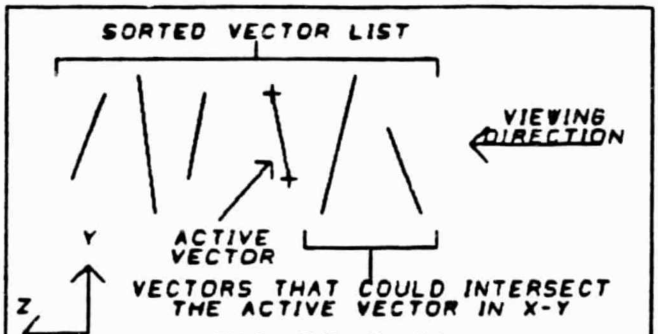


FIGURE 2-8A

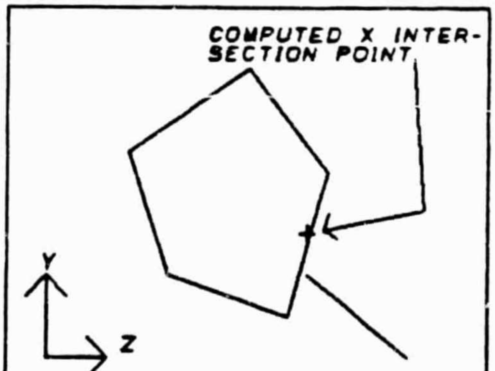


FIGURE 2-8B
COMPUTED X INTERSECTION OUTSIDE OF VECTOR

segments, a Shell sort is once again employed to order the segment list from minimum to maximum, and the second logical division of the algorithm, segment visibility testing, begins.

Segment Visibility Testing

At the commencement of this discussion it is important to realize that the segment visibility testing process is handling the segments that were created by dividing up one vector in the vector intersection portion of the code. In other words, one vector at a time comes down the hidden computation pipeline, in which it is first divided into segments; each segment is then visibility tested and flagged as invisible if such is the case. If a segment is still visible it is immediately drawn on the graphics device, after which the hidden process is started over on the next vector in the vector list.

At this point it will prove informative to follow one segment through the visibility testing process. First, the midpoint coordinates of the segment are computed from its endpoints; these midpoint values will be used for comparison purposes. This can safely be done because all vectors have been divided at locations where they intersect surface edges (Figure 2-7). This division insures that if the segment midpoint is behind a surface, the whole segment must be behind the surface and vice versa. Now, knowing the midpoint location, the surface bucket in which it resides is determined. A binary search is then used to find the point in the sorted surface list that divides the surfaces that fall in front of the segment from those that lie behind it (Figure 2-9). The segment is then checked against all of the surfaces that could lie in front of it, and therefore hide it. The following tests are used to accomplish this visibility check. First, as in the vector intersection portion of the processor, the min-max test is used. Once again, if the minimum and maximum X and Y coordinate values of the surface and the vector segment do not overlap, the surface is trivially rejected since it could not possibly hide the segment. If the surface cannot be rejected solely on the basis of the min-max test, a more refined containment test is done. Figure 2-10 illustrates the basics underlying this test; it is accomplished by computing the areas of all the triangles, created by joining each of the element vertices to the midpoint of

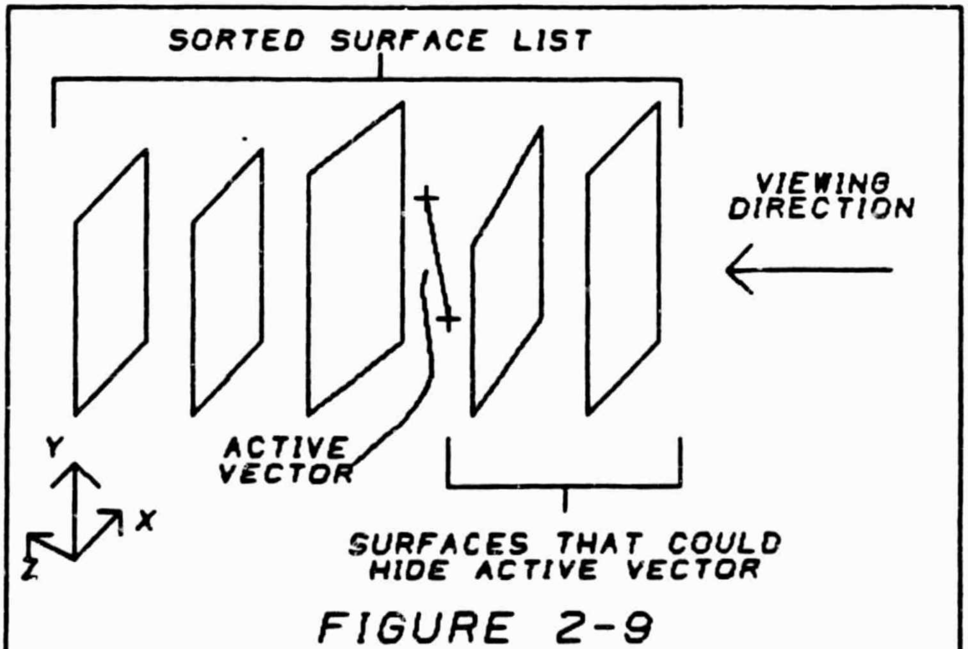
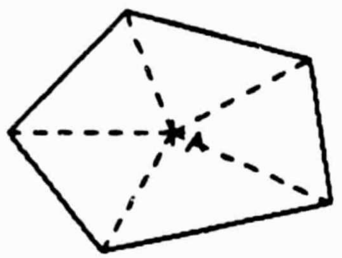
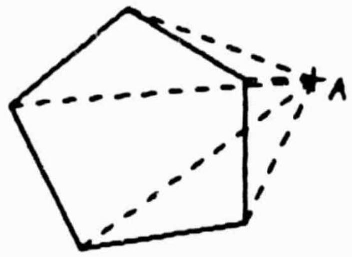


FIGURE 2-9



POINT A CONTAINED BY ELEMENT AREA - AREA OF SUBDIVIDED TRIANGLES = AREA OF WHOLE ELEMENT

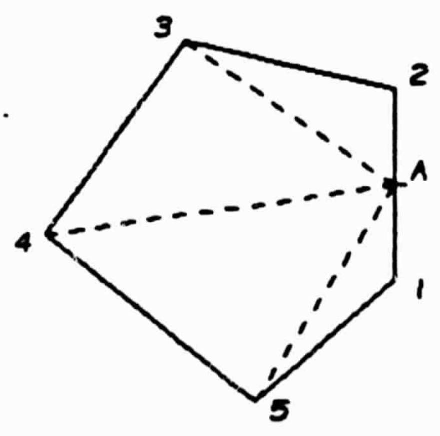
FIGURE 2-10A



POINT A OUTSIDE OF ELEMENT - AREA OF SUBDIVIDED TRIANGLES > AREA OF WHOLE ELEMENT

FIGURE 2-10B

the line segment, as shown in Figure 2-10a. A ratio of this computed area with the area of the element, which was calculated in the surface preparation routine GVPHSP, is then taken. If the point lies within the element this ratio will clearly equal 1.0 (assuming convex elements) as Figure 2-10a shows. If however, the point lies outside the element, the ratio will be greater than 1.0 as is clearly illustrated in Figure 2-10b. Further it will have been determined whether or not this surface hides the line segment being processed. If it does not, the surface will be rejected, and the next potential hiding surface will be considered. If it is determined that the surface does surround the segment midpoint, an initial depth test is performed to see if the minimum Z coordinate of the surface is greater than the Z coordinate of the midpoint. If this is the case, the surface is rejected since the segment is in front of the surface and cannot be hidden by it. If the possibility still exists that the surface is in front of the line segment, the next test performed is one that determines if the segment is actually one of the surface edges. Figure 2-11 illustrates how this is accomplished. If the midpoint is on the surface boundary, then the area of one of the triangles, used in the containment test and as shown in Figure 2-11, will be 0.0; this indicates that the segment is a part of the surface edge, and as such, cannot be hidden by the surface. If the surface makes it through all of the above tests, the plane equation for the surface is determined from the coordinate values of its first three nodes. The X and Y coordinates of the line segment midpoint are then substituted into the plane equation; this yields the Z depth of the plane at the X and Y coordinates of the line segment midpoint. The Z depth of the line segment is



TRIANGLE 1,2,A HAS ZERO AREA THEREFORE "A" IS ON THE SURFACE BOUNDARY

FIGURE 2-11

then compared to the Z depth of the plane, and if the plane is found to be in front of the segment, the segment is flagged as invisible. Once all of the segments which result from a single vector have been visibility tested, the visible ones are drawn, and the endpoints of the vector are flagged for visibility.

The above process is repeated for each vector in the vector list with the end result being an accurate hidden line representation of the model processed if four conditions are maintained. These conditions give rise to a discussion of the limitations of the JonesD hidden line algorithm which will be handled in the next chapter.

Limitations of the JonesD Algorithm

Since Gary Jones wrote his algorithm to process finite element models for the NASTRAN plotting package, there were four considerations in hidden line processing that he did not need to consider. First, there was no need to be concerned with more than four sided elements since the majority of finite element problems deal with three and four sided elements. Second, penetrating polygons were not considered since finite element models require nodes at all intersection points. Third, the capability of handling concave elements was not considered because, again, such elements are not common in finite element models. Fourth and finally, consideration was only given to the handling of planar elements.

CHAPTER 3

MODIFICATION TO THE JONESD ALGORITHM

As briefly mentioned at the conclusion to Chapter 2, there were four limitations to the capabilities of the JonesD algorithm. These included (1) the inability to handle more than four sided elements, (2) the inability to handle penetrating elements, (3) the inability to handle convex elements and (4) the restriction to only planar elements. One of these limitations has been overcome by modifying the JonesD algorithm; suggestions for ways to overcome the remaining three have also been considered. A discussion of both implemented modifications and suggested modifications for the future follows.

N-Sided Element Capability

In its original form, the JonesD algorithm handled four sided elements and line elements. Triangular elements were treated as degenerate four sided elements with the fourth node being set equal to the first node. As a result of this four sided limit, Gary Jones was able to hard code a lot of his program; this means, that in the surface preparation, for example, he could prepare all the surfaces as though they had four sides; he could compute areas, minimum and maximum coordinate values and plane equation information from closed form equations. Figure 3-1 shows an example of the hard coded form of a section of the JonesD algorithm, along with its counterpart n-sided code.

In order to allow n-sided capability, one basic change in the code was required. This change involved keeping track of how many sides there were in each element in the preprocessing section of the code. This knowledge allows the use of "do loops" in area calculation, minimum and maximum coordinate determination and in visibility testing. For example, the area of an element is computed by taking cross products around the element nodes; this is easily

```
C *
C *    SURFACE AREA CALCULATION
       AREA=ABS(SURF(11,NPT)+SURF(12,NPT)+SURF(13,NPT))
      &+ABS(SURF(12,NPT)+XP1*YP4+XP4*SURF(10,NPT)-XP3*YP4)
       IF(AREA.EQ.0.0) THEN
       SURF(22,NPT)=-1.0
       ELSE
       SURF(22,NPT)=1/AREA
       ENDIF
```

JonesD

```
        ARE=0.0
c-----------------------------------------------------------------------
c      Immediately below we are computing the area of this element, by
c      taking cross products around the element vertices. to
c      determine if it can hide anything (an element with 0 area couldn't
c      hide much)   We are also creating the AR array which contains
c      element vertice cross products which will later be used in a
c      containment test
c-----------------------------------------------------------------------
        DO 700 NS=1,NNODES(NPT)
           IF(NS.EQ.NNODES(NPT)) THEN
              AR(NS,NPT)=(XP(NS,NPT)*YP(1,NPT)-XP(1,NPT)*YP(NS,NPT))
           ELSE
              AR(NS,NPT)=(XP(NS,NPT)*YP(NS+1,NPT)-XP(NS+1,NPT)*
     *                      YP(NS,NPT))
           END IF
           ARE=(ARE+AR(NS,NPT))
  700   CONTINUE
        AREA=ABS(ARE)
C
c-----------------------------------------------------------------------
c      Setting SURF(7,NPT) to -1.0 if the area = 0 creates a flag that
c      will later be checked in the determination of line segment
c      visibility
c-----------------------------------------------------------------------
        IF(AREA.EQ.0.0) THEN
           SURF(7,NPT)=-1.0
        ELSE
           SURF(7,NPT)=1/AREA
        END IF
```

Modified JonesD

Figure 3-1

accomplished in a loop that goes from one, to the number of sides in the element. The same type of loops were employed in the determination of element maximum and minimum coordinate values and in line segment visibility testing. This generalization of the code required the creation of three new arrays to handle the X, Y, and Z coordinates of each element; additionally, one new array was required to keep track of the partial areas determined from the nodal cross products for each element. Finally, the connectivity array was expanded to handle n-sided elements as required by the user.

It is the authors opinion that a detailed description of each step taken to make the n-sided enhancement would not provide any new insights to the reader. The fact is, the actual modification is not where the greatest difficulty arose; the greatest difficulty came in the authors efforts to understand how the JonesD algorithm worked. This difficulty came as a result of the sparse comments contained in the original code.

Penetrating Elements

As mentioned above, the JonesD algorithm cannot accurately handle penetrating elements. Figure 3-2 demonstrates the problems that the algorithm has. Notice that at the intersection of the two spheres, the algorithm does not know where to terminate the penetrating lines The reason for this is that the points at which lines intersect surface planes are never

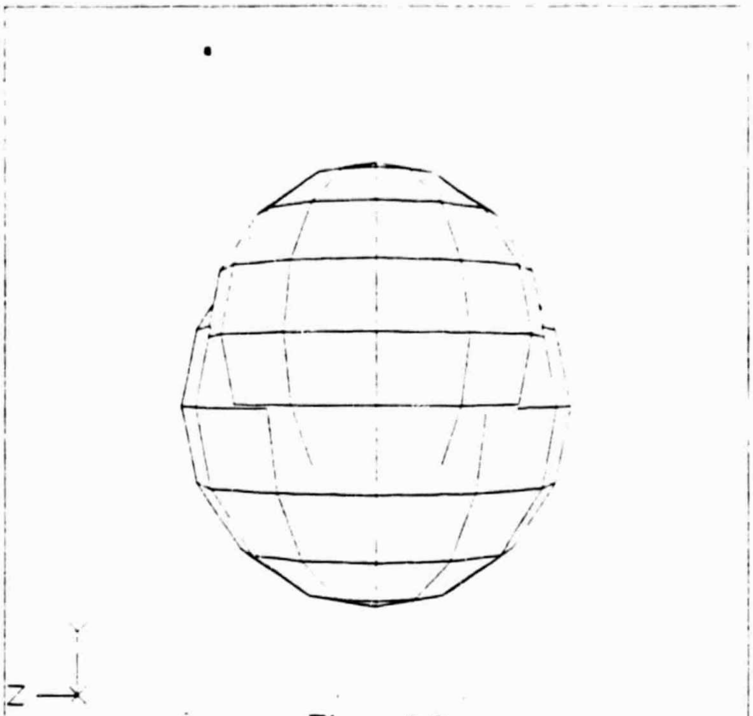


Figure 3-2

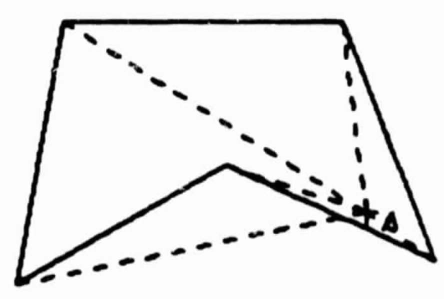
computed in the line intersection portion of the code. That is, intersections are only computed at the locations where lines intersect lines, and not where lines intersect surfaces. Consequently, if the midpoint of a line, that penetrates a surface, is in front of that surface, it will remain completely visible. Conversely, if the midpoint of a line, that penetrates a surface, falls behind the surface, the entire line will be rendered invisible.

The apparent solution to this problem would involve the computation of the points at which penetrating lines intersect planes. This could be accomplished using the line equation of the "penetrating line" and the plane equation of the element it penetrates. This would slow the algorithm down considerably, since a line in a bucket would have to be tested against every surface whose minimum and maximum coordinate values were in common with the minimum and maximum coordinate values of the line. Further, this would only result in the termination of the penetrating lines where they intersected surfaces. The lines of intersection still would not be drawn.

At any rate the implementation of this capability is left to another investigation.

Concave Elements

As Figure 3-3 illustrates, the JonesD algorithm does not handle concave elements with any degree of reliability. This limitation results from the area ratio method, used in determining if a line segment's midpoint is contained

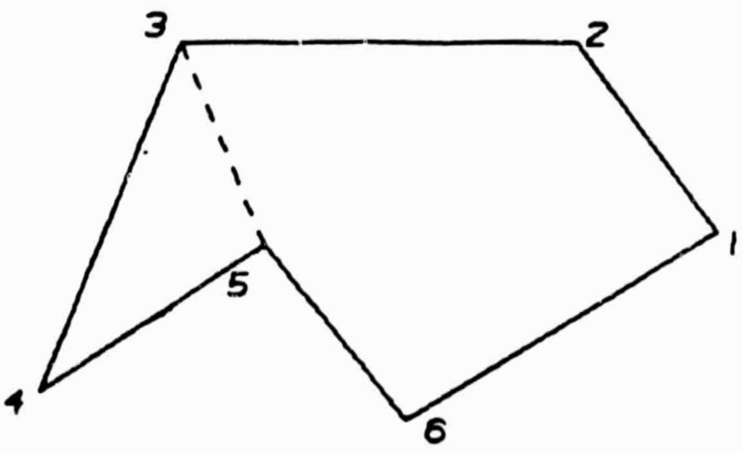


ALTHOUGH POINT "A" LIES WITHIN THE ELEMENT, THE SUM OF THE AREAS OF THE TRIANGLES SUBTENDED FROM IT IS > THE ELEMENT AREA

FIGURE 3-3

within a surface element. In Figure 3-3 the sum of the areas of the triangles, created by drawing lines from point A to each of the element vertices, is clearly greater than the area of the element. Consequently, the containment test would conclude that the point "A" lies outside of the element and, therefore, must be visible. This is of course not the case and results in the hidden line remaining visible.

It is the author's opinion that this problem could be handled with minimal time expense since convex elements are not used very often in modeling. The solution would involve the taking of dot products between element edges which could be accomplished at the time that element areas are



EACH TIME THE DOT PRODUCT CHANGES SIGN, THE ORIGINAL ELEMENT IS DIVIDED BY DRAWING A NEW EDGE FROM THE POINT WHERE THE PRODUCT CHANGED SIGN TO THE NODE 2 PLACES BEFORE IT

FIGURE 3-4
CONCAVE ELEMENT SOLUTION

calculated in subroutine GVPHSP. As an element was traversed, the dot products of the successive edges could then be compared, and if the sign of the product changed from one set of edges to the next, one would know that a concave element had been encountered. With this knowledge, the element could then be divided into two hopefully convex elements as shown in Figure 3-4, and the surface preparation could be done on the two new and now convex elements. If division of the element resulted in a concave and a convex element, then the remaining concave element could be divided in two. This process would continue until only convex elements remained. From this point the hidden line process would proceed as usual and an accurate hidden line view obtained.

Again, as much as the author would like to undertake this modification, time requires that it be left to the future.

Warped Elements

Clearly, the JonesD algorithm would have limited success in processing warped elements since it relies on the plane equation as the ultimate test for line segment visibility. The solution to this problem would be very time expensive since it would require that all elements be tested to find those that are warped. Such testing could be accomplished by comparing the normals around an element. If the normals varied, a warped element would have been found, and the element would have to be divided into triangles. Each triangle could then be accurately processed by the modified JonesD algorithm. It is interesting to note, however, that as long as the elements are not too severely warped, JonesD does a very acceptable job (Figure 3-5).

The implementation of a contour algorithm, as mentioned above, will be the subject of the next chapter.

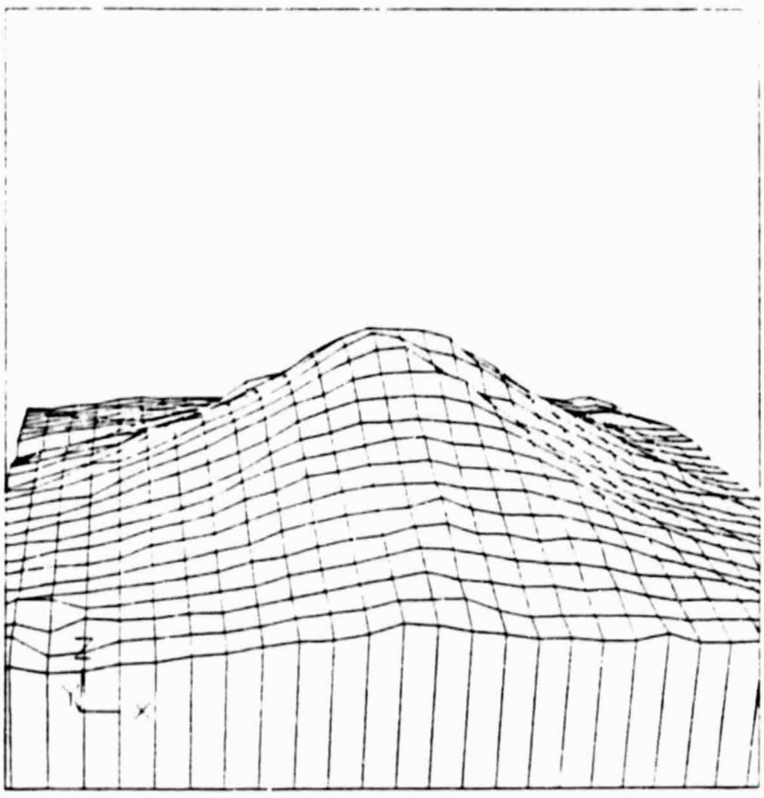


Figure 3-5
Mt. St. Helens

CHAPTER 4

CONTOUR IMPLEMENTATION

The implementation of the contour routine was one of the most interesting and rewarding parts of this work. The algorithm was conceived by an unnamed employee of the John Deer Company with information that was obtained from L. J. Segerlind's book "Applied Finite Element Analysis" [6]. It is very simple, yet produces high quality results. However, to obtain these high quality results, a number of modifications and enhancements had to be made to the basic algorithm. These modifications can be adequately discussed in five divisions which are, (1) the justification for computing the contours prior to entering the hidden routine, (2) the contour problem, (3) the contour generation algorithm, (4) contour sorting and (5) contour hidden processing, output, and labeling.

Justification for Contour Preprocessing

By glancing at Figure 4-1, the reader will quickly be able to see where the contour package fits into the overall hidden line processing system. Now, realizing that many of the contours that are created on a 3-D model are going to be hidden, one might think that generating contour segments prior to

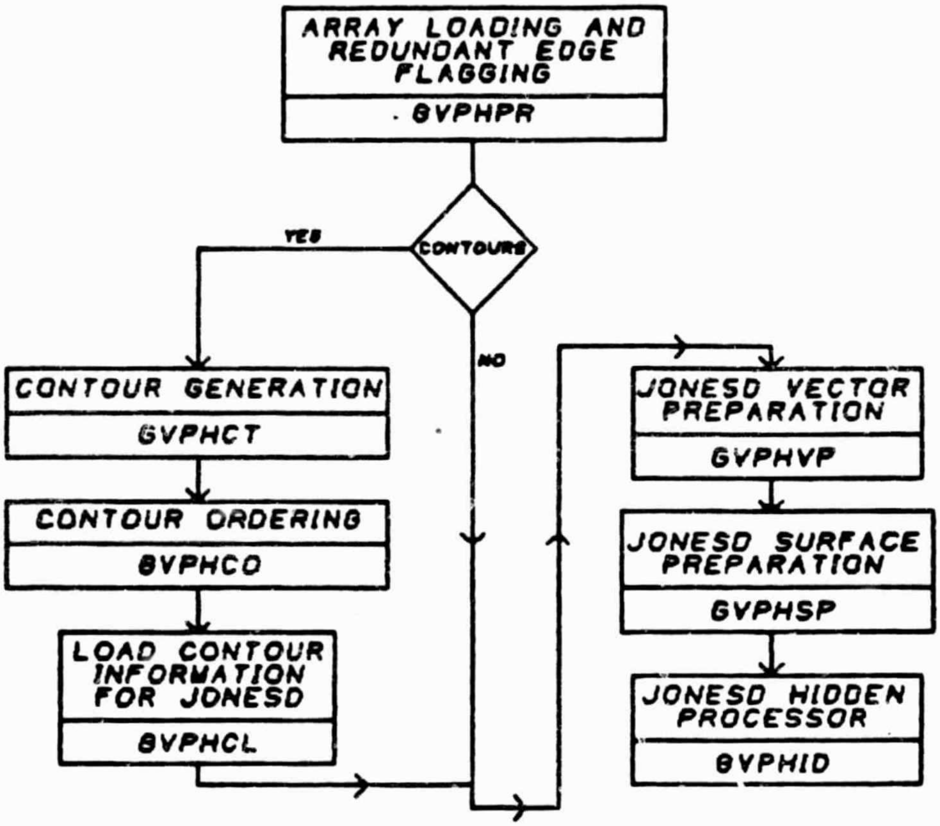
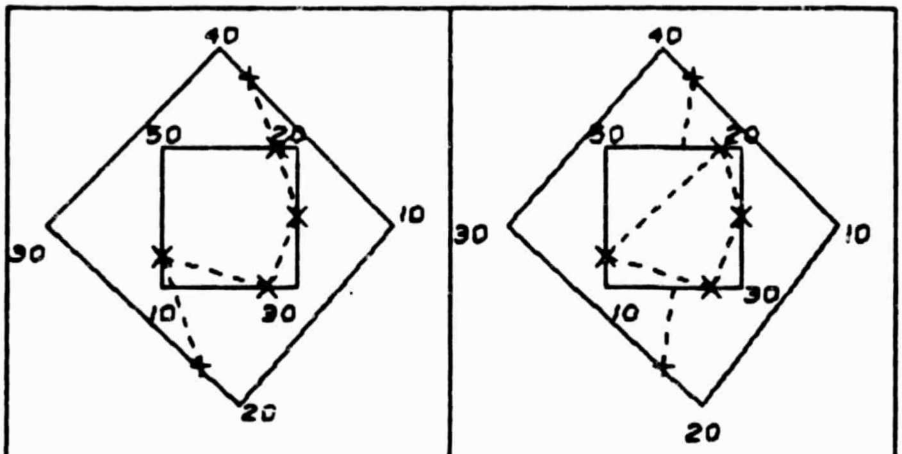


FIGURE 4-1
CONTOURS IN HIDDEN FLOWCHART

hidden line removal is wasteful. However, there are three good reasons for generating contours before hidden processing. First, and most convincing, is that there is no accurate way to create contours given the strictly vector output from the hidden routine. The problem is that once returned from hidden, all there is to work with are coordinates in 2-D space. There is no way to tell, as shown in Figure 4-2, which surface lies in front of which, and therefore, there is no way of knowing how to connect equal function locations



ATTEMPT TO PLOT LEVEL 25 CONTOURS WITH NO DEPTH INFORMATION

FIGURE 4-2A

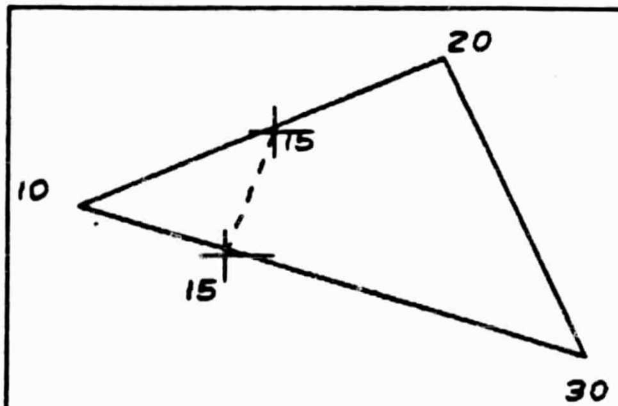
CORRECTLY PLOTTED LEVEL 25 CONTOURS

FIGURE 4-2B

to obtain accurate contour lines. Second, the number of contour segments generated on a typical model is usually quite small relative to the number of lines that combine to make up that model. This normally small number of contour segments is handled quickly by the JonesD hidden algorithm. This is because the algorithm knows that these segments can only be hidden and that they cannot hide anything. Finally, prior to hidden processing, the element connectivity is intact; this makes the accurate generation and proper connection of the contour segments possible.

The Contour Problem

Before going into a detailed discussion of the contour algorithm, a basic description of the contour problem is in order. Contours are created by interpolating between the function values assigned to element coordinates, as diagramed in Figure 4-3. These function values may represent any type of function, and are assigned to the the element nodes according to some type of preprocessing operation. The preprocessor might be a finite element package for example. If one wishes to find the contour line at a function value of 15, in Figure 4-3, interpolation needs to be performed between adjacent element nodes to find all of the points along the element edges that have an interpolated value of 15; these locations are designated with an "X". Once this is accomplished, the two "X" points are connected producing a contour segment.



IF A CONTOUR VALUE EXISTS ON ONE EDGE OF A TRIANGLE, IT MUST EXIST ON ONE, AND ONLY ONE, OTHER EDGE OF THE TRIANGLE

FIGURE 4-3

The Contour Generation Algorithm

The flowchart in Figure 4-4 diagrams the basic processes in the contour generation algorithm as implemented in conjunction with the JonesD hidden line algorithm. The initial portion of the algorithm divides each of the elements into triangles, by creating a center node which is the average of each

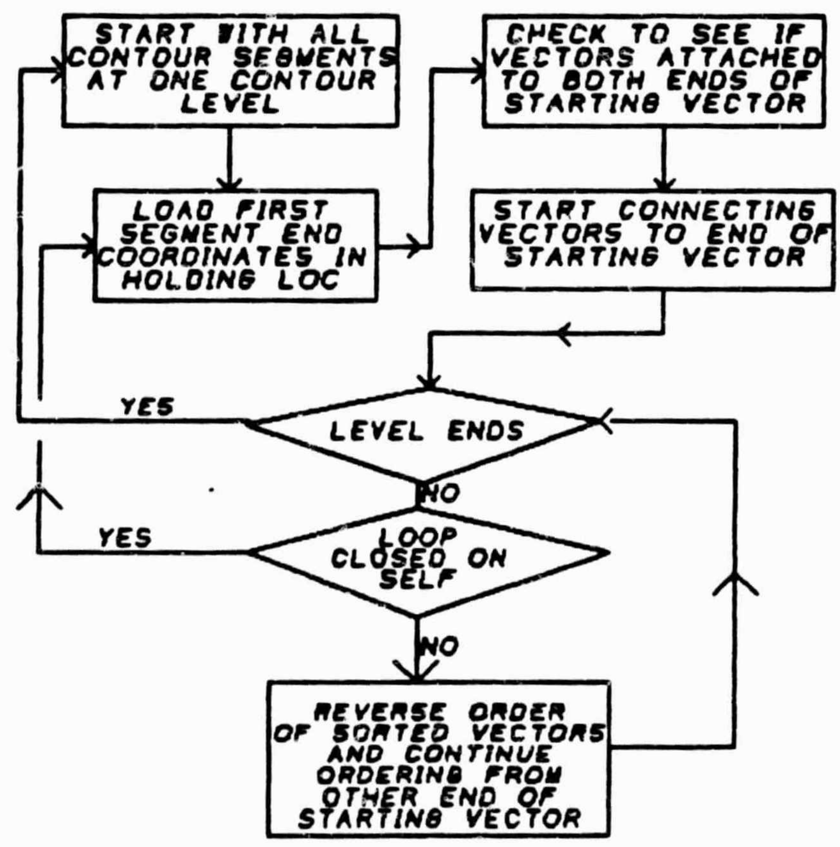


FIGURE 4-4
CONTOUR ORDERING FLOWCHART

of the element nodes. The average function value at the center node is also computed by averaging the function values at each of the element nodes. This process is done to transform the element data into the triangular form required by the contour routine. The main reason for using triangular elements, as a basis for computing contours, is that triangular elements possess a unique interpolation property. This is demonstrated in Figure 4-3 where it is seen that if a contour value is present along one edge of a triangle, it will also be present on one and only one of the other edges of that triangle. Conversely, if a contour value is not present on two sides of a triangle, it cannot exist on the third side. Because of this property one can be assured that contour lines will never cross within triangular elements. Further, one can be assured that contour lines will never cross over a series of elements if each of those elements has been subdivided into a triangles for contour generation purposes. This idea can be visually understood by viewing Figure 4-5.

The contour algorithm next begins to process each of the triangles that make up the first element. One by one each triangle is sent through the interpolation portion of the code. In this portion, contour segments, representing all of the contour levels that have been user requested and that lie within the nodal function values for that triangle, are generated. The process by which this is accomplished is as follows.

The function value at the first contour level is extracted from memory and compared to the function values at each of the nodes that make up the

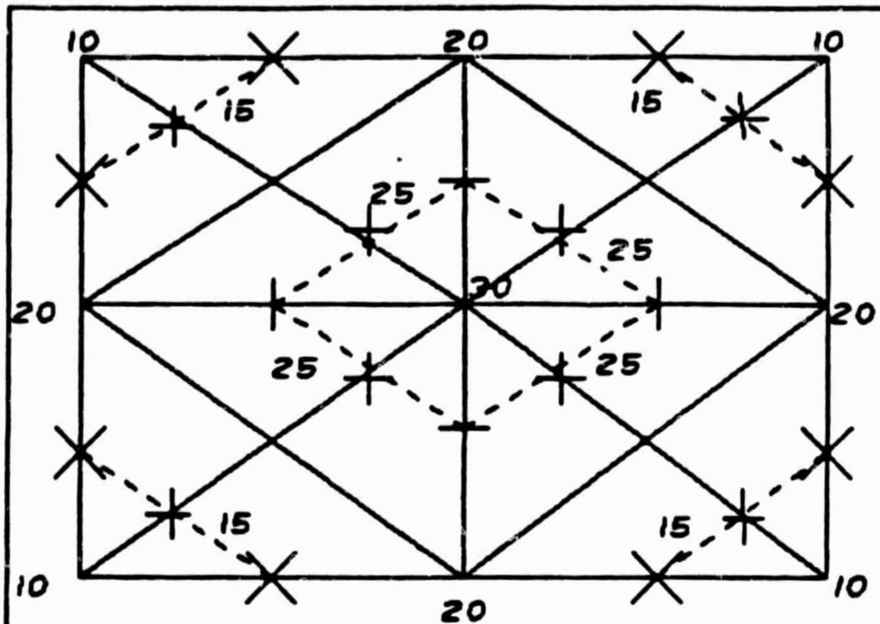


FIGURE 4-5
ELEMENTS DIVIDED INTO TRIANGLES

triangular element being operated on. If the first contour value lies outside the bounds of nodal function values, the element is immediately tested against the next contour value. If, however, it is found that the contour value does lie along the first edge of the triangle, then a contour segment endpoint, on the edge, is interpolated from the edge endpoint function values and X, Y, and Z coordinates. The next edge of the triangle is then tested for contour value intersection, and if it is found, the other endpoint of the contour segment is interpolated along the second edge. The contour segment endpoints are then immediately stored into memory. If it is discovered that the contour value does not lie along the second edge, then a contour segment endpoint is immediately interpolated along the third edge, and the contour segment endpoints are stored into memory. This is done since it is known that such an endpoint must exist because of the interpolating nature of the triangle. This process is repeated for each contour value until all of the contour segments on the triangle have been computed

Succeeding triangles in the element are then processed in a similar manner until all of the triangles making up that element are completed. The same process is then repeated for the next element, and so on until all of the elements have been processed. It should be pointed out, that while the contour segments are being created, the number at each different level, as well as the locations of their endpoints, is tracked. Because of this it is possible to sort the presently random contour segments into nicely ordered contour loops.

Contour Ordering

Contour ordering is required for two reasons - first for accurate labeling and second so that curves can be accurately fit through contour loops if desired. When the contour ordering routine GVPHCO is called, it is passed contour segment coordinate information sorted by contour level; the information is not, however, ordered within those levels. It is also passed the number of contour segments in each level. With this information a systematic process of sorting the contour segments into ordered loops is started. This ordering procedure takes advantage of the fact that adjacent contour segments will have like coordinate endpoints. Consequently, a search starting with the first contour segment at a particular contour level is begun in an attempt to find another one, at the same level, with a like endpoint. This ordering procedure diagramed in Figure 4-4, flows as follows.

(1) The endpoints of the first contour segment, in the segment storage array, are stored in holding locations. The first contour segment will be referred to as the "initial segment" hereafter.

(2) The #1 endpoint (see Figure 4-6) is initially tested against the endpoints of both ends of all the segments in its same contour level. If it is found that one of the other segments, at this level, has an endpoint that matches that of the #1 end of the "initial segment", a logical is set to true. This logical indicates that the "initial segment" has connecting vectors at both of its ends.

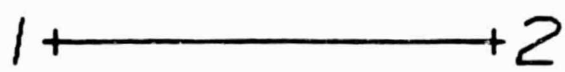


FIGURE 4-6
CONTOUR SEGMENT DESCRIPTION

(3) The #2 endpoint of the "initial segment" is then compared to both ends of the other segments in its contour level. When a vector with a matching endpoint is found, the endpoint storage locations of this vector and those of the second vector in the list are swapped. The sorting then proceeds starting with the #2 end of the vector that newly occupies the number two location in the storage area; the #2 end of this vector is now compared to the remaining vectors in the list at this contour level until a connecting vector is found. Figure 4-7 will clarify this sorting process.

(4) When the end of the contour loop, which progressed from the #2 end of the "initial vector", is encountered, the array order of the contour segments is reversed if the logical indicating that there were contour vectors proceeding from both ends of the "initial vector" is set to true. The process described in part (3) then continues until the last contour

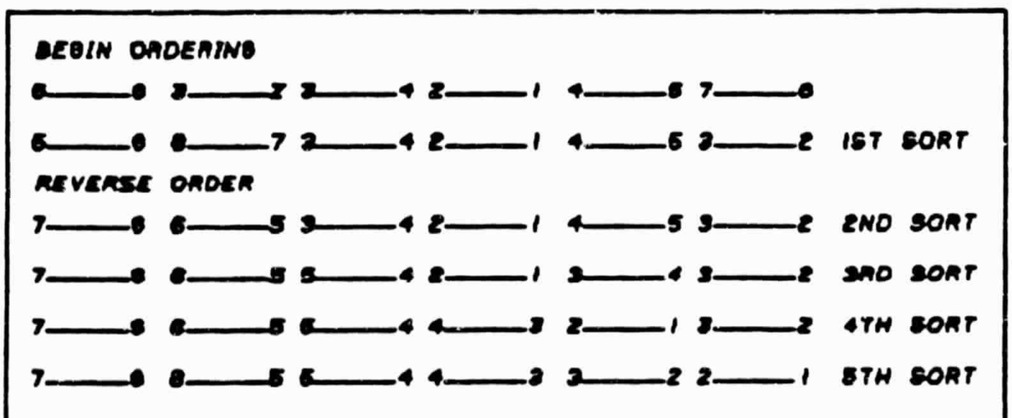


FIGURE 4-7
CONTOUR SORTING

vector at this level has been sorted or until no more connecting vectors can be found. If, however, there is no vector that connects to the #1 end of the "initial vector", and there are still more segments at this level when the end of the loop being processed has been reached, then two possibilities exist. The first one is that the "initial vector" happened to be at one end of the contour loop which does not pose any problem. The second is that the contour loop closed on itself in which case there is no point in reversing the order of the sorted segments; this excess sorting is avoided by storing the #1 end coordinates of the "initial segment", and then checking to see if they are the same as the #2 end coordinates of the last segment found in the loop being processed. If they are the same then order reversal is bypassed.

(5) If it happens that the end of the first contour loop at this level is encountered before all of the vectors have been dealt with, then there is clearly a separate contour loop at this level. This is easily dealt with by making the next vector, in the yet unsorted list, the new "initial vector". When this is done the process continues as described above until all the contour segments at a contour level have been processed. After this, the successive contour levels are processed until all of the contour segments have been ordered.

During the ordering process counters keep track of how many separate contour loops there are at each contour level, and how many contour segments are contained within each loop. This information is used in interfacing the contour sub-system to the hidden line sub-system.

Contour-Hidden Line Interface

There are two parts to the contour-hidden interface; one involves loading the contour information into the hidden algorithm, and the other involves correctly labeling the visible contour segments after hidden line processing. It turns out that once the contour segments have been ordered, the problem of loading them into the hidden line subsystem is quite simple. All that is required is that the contour vectors be added to the model vector array and that their coordinates be added to the model coordinate array. However, rather than load duplicate coordinates from the like ends of connecting vectors, the knowledge of how many segments are in each loop and of how many loops are at each level is used to allow the entering of coordinates only once. This loading operation is performed in subroutine GVPHCL. Now that the contour vector information is in the model vector and coordinate arrays, the hidden algorithm processes them as though they are a part of the model.

The second part of the contour-hidden interface deals with keeping track of where contour loops become invisible due to hidden line processing and where they once again become visible. This operation is done at the time that line visibility is determined in the hidden algorithm. Once again, counters are employed to keep track of how many separate contour loops there are after hidden line processing. The number of segments in each of these loops is also counted. First, this allows curves to be fit through contour loops. Second, it allows for efficient labeling of contours, since each time a contour loop becomes visible, it is labeled. That is, each time a contour loop becomes visible, the screen coordinates of the first segment, along with the contour level number for the segment, are sent to the MOVIE.BYU LABELS subroutine. As a result, every visible loop in every contour level is labeled. Figure 4-8 shows the results of this processes.

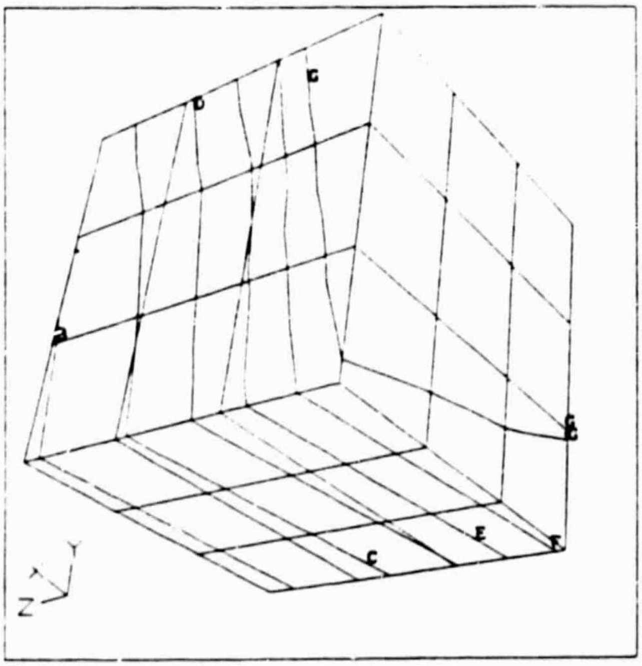


Figure 4-8
Contoured Cube

All that has been discussed from the beginning of this work to this point constitutes the main body of new innovations in this thesis. The remaining two chapters deal with the interfacing of this work to the MOVIE.BYU Display package and with cpu testing of the algorithm to check its actual speed.

CHAPTER 5

INTERFACE TO MOVIE.BYU DISPLAY

The challenge in interfacing the modified JonesD algorithm to the MOVIE.BYU Display package was in pulling out the polygonal data at the correct location and getting it into the correct form. This required the addition of eight new routines that closely parallel eight that are already present in MOVIE.BYU Display. It also required the addition of one completely new routine to order the edges of clipped polygons. An explanation of the interface and the reasons for the methods used to accomplish the task follows.

Data Required by the JonesD Algorithm

The first thing that was considered in creating the interface was the data that the modified JonesD algorithm required. Since the JonesD algorithm's only function is to determine hidden line removal, it was clear that all polygonal data had to be picked up after transformation and clipping operations had been performed. Therefore, the three elements of data required by the JonesD algorithm which are, (1) the element coordinate array, (2) the element connectivity array along with the number of sides in each element and

(3) the function or contour information, were all created or obtained after MOVIE.BYU Display routines had performed all the necessary transformation and clipping operations.

Modifications to MOVIE.BYU Display

The first interface to MOVIE.BYU Display was the 'quick fix' type where one obtains the information without regard for what has taken place in the code prior to the time it is picked up. This type of an interface, although it may work, usually accomplishes three things. First, it makes the existing code less understandable; second, it makes the program run slower since invariably a lot computation, not needed by the new subsystem, has been done; and finally, it removes the desirable feature of modularization. As a result of these considerations, it was determined that a more complete interface would be attempted. This resulted in some close scrutiny of all the routines that preceded the location at which the required data was obtained. Because of this scrutiny, eight new routines were created which represented only small parts of eight MOVIE.BYU routines. These new routines, with their MOVIE.BYU counterparts, are diagramed in Figure 5-1 and are contained in Appendix D. The reduction in routine size came mainly from the deletion of all of the color, shadow, and transparency information contained in the MOVIE.BYU routines, and resulted in a clean, efficient, and modular interface. This interface allows the transformations, clipping, poorman's hidden line

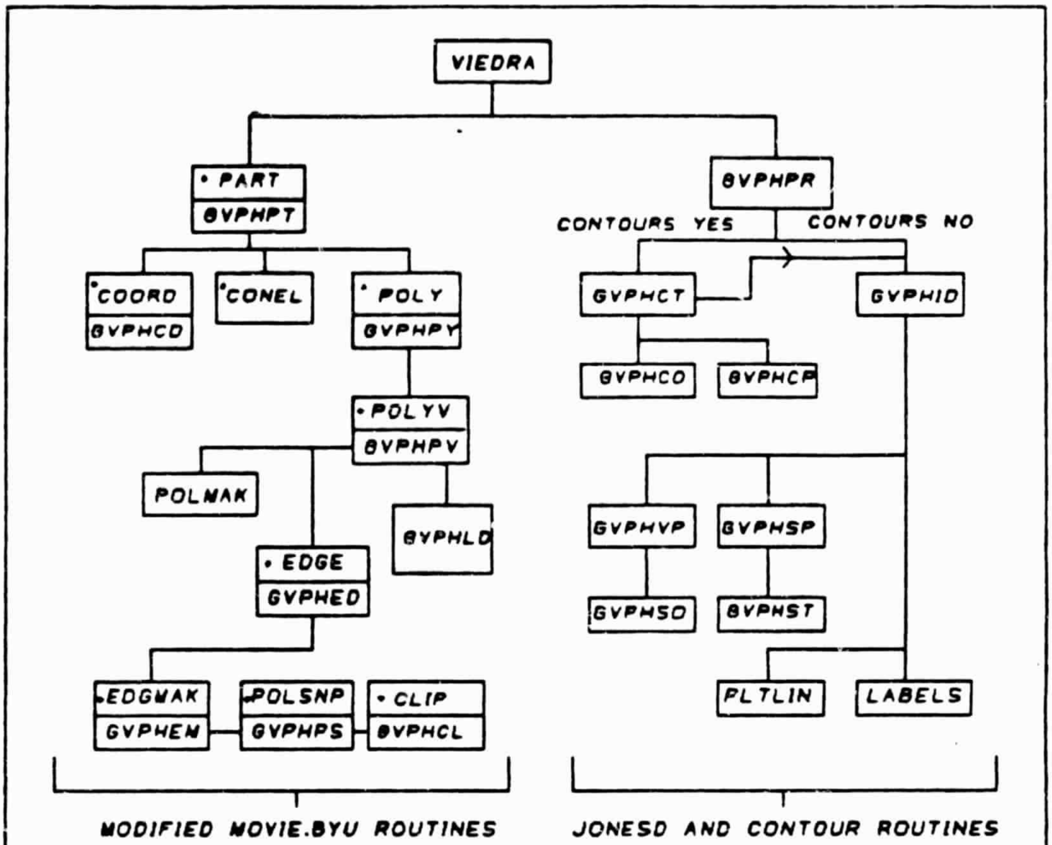


FIGURE 5-1
COMPLETE FLOWCHART

• MOVIE COUNTERPART ROUTINES

removal, shrink option, and contour labeling currently available in the MOVIE.BYU Display package. It does not presently allow node numbering, element numbering, and the dotted line capability, although it could be modified to handle such.

Obtaining the Clipped and Transformed Data

The clipped and transformed data is obtained in the new GVPHPS routine which is a counterpart to the MOVIE.BYU POLSNP routine. This routine calls the GVPHCL routine (counterpart to CLIP) then loads the clipped coordinates, connectivity, and function information into holding arrays. This operation proceeds on an element by element basis with all of the edges of one element being successively clipped. The edge information is then stored before the next element is processed. Because the polygon edge clipper in MOVIE.BYU does not return the polygon edges in consecutive order, they are ordered (GVPHLD) before they are loaded into the holding arrays. Once all of the element edges in the model have been transformed and clipped and the holding arrays have been correspondingly filled, the hashing preprocessor to the modified JonesD algorithm is called and operates as was discussed in Chapter 2. From the preprocessor, the model information goes to the modified JonesD hidden line processor which calls the present MOVIE.BYU PLTLIN routine to output the visible model segments. If contours are enabled, the MOVIE.BYU LABELS routine is called to label the contours.

Now that the modified JonesD algorithm with all of its appendages (Figure 5-1) has been discussed, all that remains is to test the algorithm to see if it is really as fast as its author has claimed it to be.

CHAPTER 6

SPEED COMPARISONS AND CONCLUSIONS

Although the speeds recorded by the JonesD algorithm were not as good as those indicated by its author [4], it did perform very well in a number of cases. It was compared only to the MOVIE.BYU Watkin's algorithm. This comparison included the time necessary to transform and clip all models, and the time to perform the actual hidden computations and output the visible segments to the graphics device. If contours were computed, it also included the time necessary to generate them. All models were tested in exactly the same manner for both the Watkin's and JonesD algorithms; that is, regardless of the algorithm, the same command files were used in generating hidden line pictures. The command file generated pictures are included in Figures 6-2 to 6-11 at the end of this chapter.

The items of interest in making speed comparisons were, (1) how would Watkin's and JonesD compare as the number of model elements was increased, (2) how would the two algorithms compare in the area of contour generation, and (3) how would they compare if a model had high concentrations of elements in localized areas.

Increase Number of Elements

To determine the effect of successively increasing the number of elements on algorithm speed, a prismatic hexahedron was selected. The number of elements per hexahedron face was systematically increased from 4 to 289 (see Figure 6-2 and Figure 6-3). The results of this testing are contained in Figure 6-1 and indicate two things. First, increasing the number of elements between the ranges of 24 and 864 only slightly affects the speed of the JonesD algorithm. The vectors processed per cpu second bares this out. However, when the number of elements exceeded 1000, the speed (see Figure 6-1) of the JonesD algorithm quickly dropped off. On the other hand, the Watkin's algorithm started out very slowly, and then picked up speed as the number of elements was increased. Consequently, the speed advantage of the JonesD algorithm was lost in the 900 to 1000 element range.

Contour Generation

As anticipated, the modified JonesD package is much faster at generating and displaying contours than the Gouraud interpolation method used in the Watkin's algorithm. For small models, the contour generation and hidden representation capability is three to four times faster in the modified JonesD (Figure 6-1). Figures 6-4 and 4-8 are examples of these small models. As models increase in size, the speed of the JonesD package slows down (Figure 6-5). In the area of generating contours on a flat grid as might be the case in

MODEL	FIGURE	NUM. POLYS	NUM. MODEL VECT.	NUM. CONT. VECT.	JONESD		WATKINS		TIMES JONESD FASTER
					CPU SEC	VEC/SEC	CPU SEC	VEC/SEC	
HEX4	6-2	24	96	0	1.6	60.0	7.0	13.7	4.4
HEX16		96	384	0	5.5	69.8	27.2	14.1	5.0
HEX36		216	864	0	13.2	65.4	40.3	21.4	3.1
HEX64		384	1536	0	24.5	62.7	57.1	26.9	2.3
HEX100		600	2400	0	40.4	59.4	75.0	32.0	1.9
HEX144		864	3456	0	60.3	57.3	97.2	35.6	1.6
HEX225	6-3	1350	5400	0	113.1	47.8	136.1	39.7	1.2
HEX289		1734	6936	0	182.3	38.1	174.3	39.8	0.96
2POLY	6-4	2	8	40	2.1	22.9	7.5	6.4	3.6
BIGCUBE	6-5	300	1202	977	55.7	39.1	143	15.2	3.7
MT ST H.	6-6	896	3584	1270	57.3	84.7	264.1	18.4	4.6
VALTEK1	6-7	268	1072	357	21.3	67.1	82.1	17.4	3.9
VALTEK2	6-8	303	1212	577	30	59.6	92.5	19.3	3.1
PLANE	6-10	587	2300	0	77.1	29.8	72.7	31.6	0.94
SPH6	6-11	482	1808	0	49.6	35.5	21.2	85.3	0.25

FIGURE 6-1
JONESD - MOVIE.BYU TIME COMPARISONS

a two dimensional finite element analysis, the JonesD algorithm is four to five times faster. This time savings is demonstrated in the Mount St. Helens model (Figure 6-6) where contours are generated on the flat grid which, when warped, becomes Mount St. Helens (Figure 3-5). Two actual finite element models, Figures 6-7 and 6-8 demonstrate similar time savings. In addition to the time savings, a comparison of Figures 6-8 and 6-9 shows no great difference between the JonesD and MOVIE.BYU contours. In fact, the only difference is that the JonesD contours, because of there vector nature, are rougher than the MOVIE.BYU contours; this problem diminishes as element size decreases.

Element Concentration

The Ame's Plane model (Figure 6-10) was used to test the effect of concentrating large numbers of elements in localized areas of the viewing window. The wheels of the plane are each made of a large number of small elements which satisfy the element concentration requirement. As Figure 6-1 shows, the Watkin's algorithm was slightly faster than the JonesD algorithm, even though there were only 587 elements in the model. This slowing of the JonesD algorithm is a result of a large number of surfaces, and vectors, being mapped into the same X-Y cell for bucket sorting purposes (Figure 2-3). When this happens, the sorting processes used to make the JonesD algorithm faster become inefficient since the sort depth in a bucket is very deep. Figure 6-11 illustrates this same problem.

Conclusions

As a result of this work two conclusions become evident. First, the modified JonesD algorithm provides rapid hidden line processing for relatively small models (ie. 1000 elements or less). This speed is however dependent on the localized density of elements. If elements are spaced evenly throughout the viewing window, the algorithm is fast, and if there are concentrations of elements, the algorithm slows down. Second, the algorithm, in conjunction with the contour generation algorithm, provides very rapid contouring capability. Both of these conclusions present the modified JonesD package as ideal for moderate sized models and in particular, finite element models.

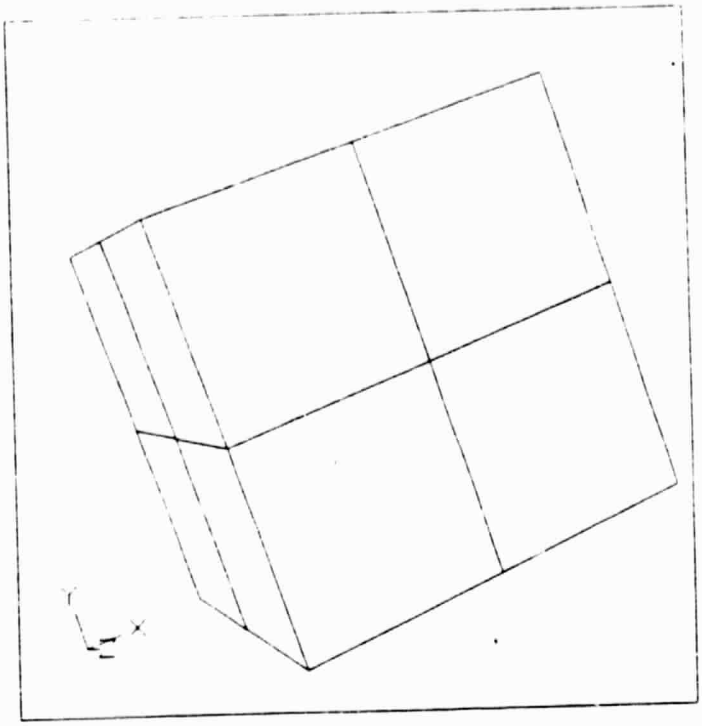


Figure 6-2
Hex4

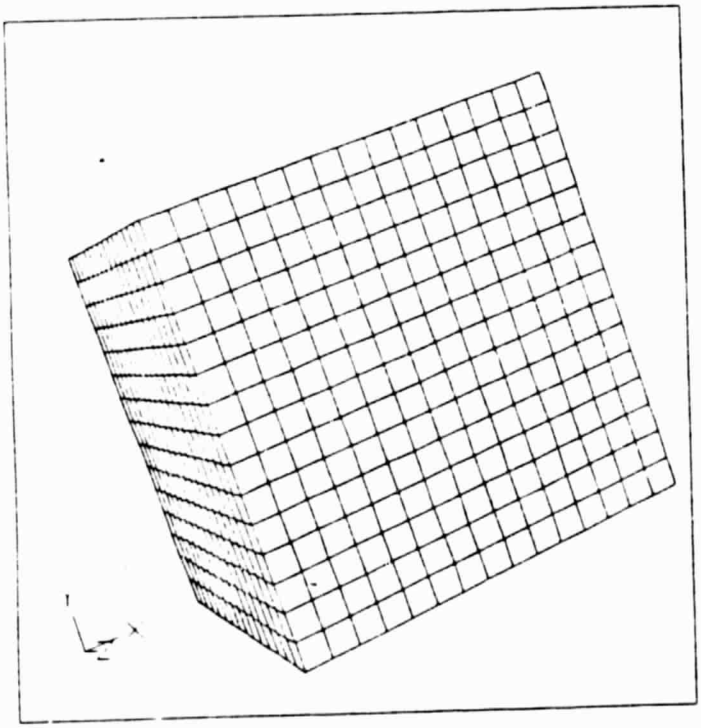


Figure 6-3
Hex225

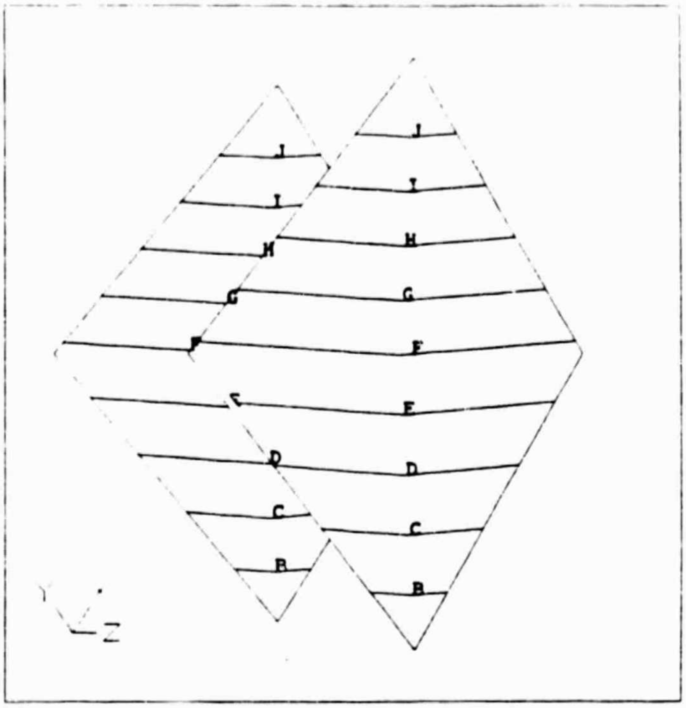


Figure 6-4
2poly

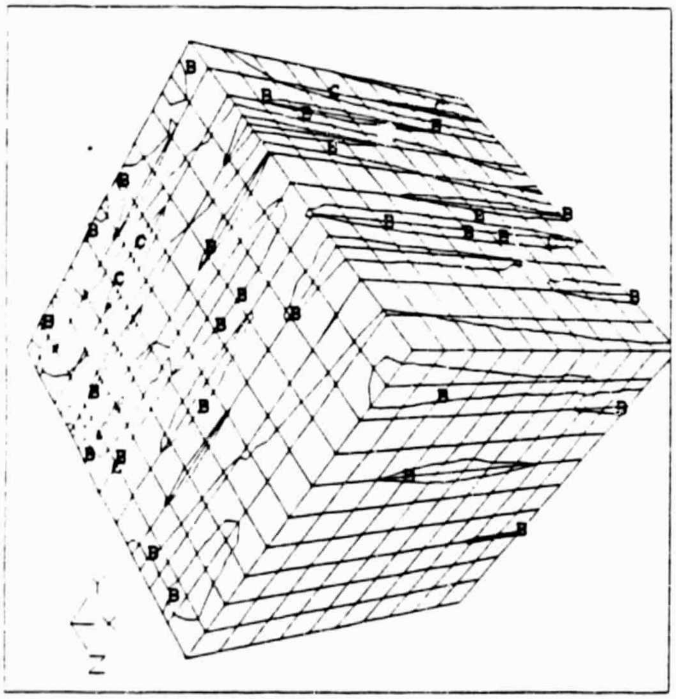


Figure 6-5
Bigcube

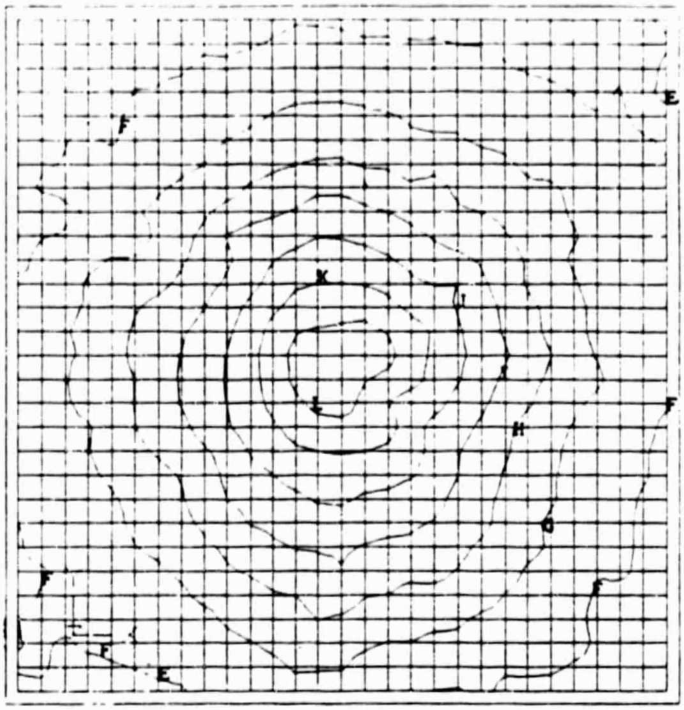


Figure 6-6
Mt. St. Helens

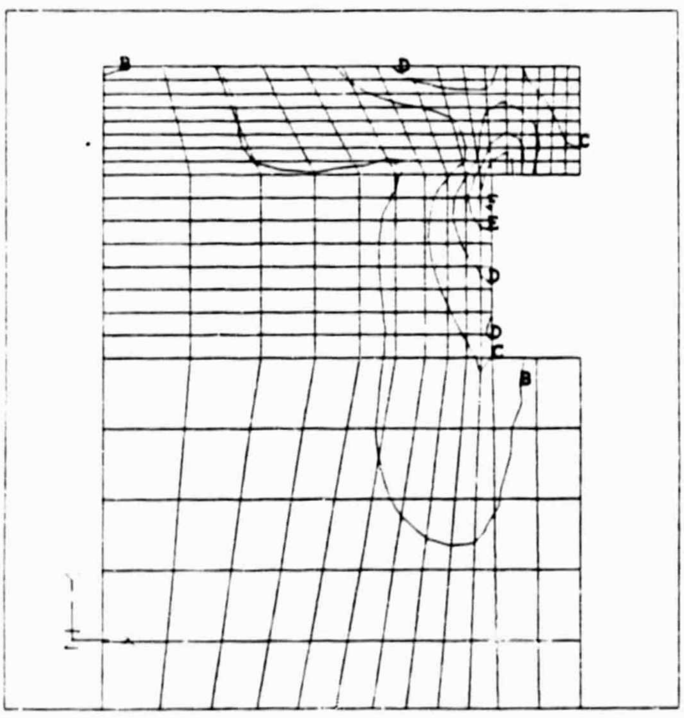


Figure 6-7
Valtek1

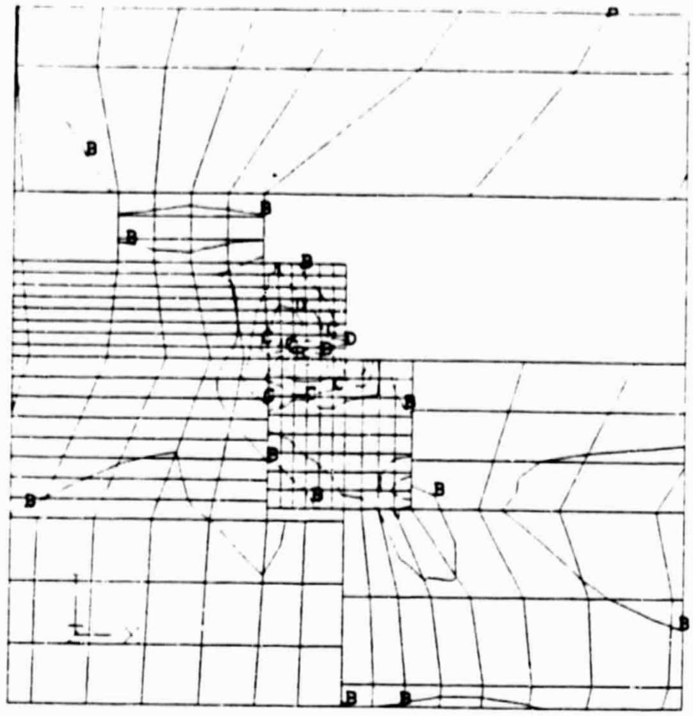


Figure 6-8
Valtek2

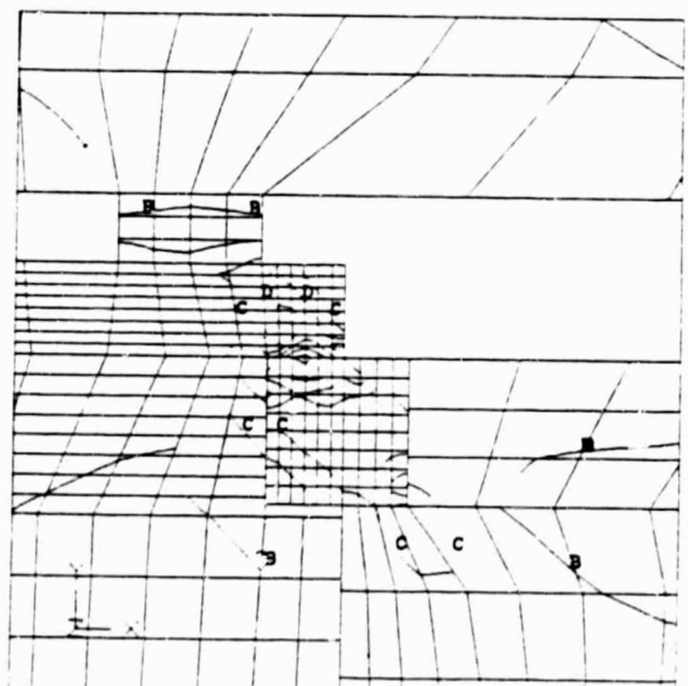


Figure 6-9
Valtek2-MOVIE

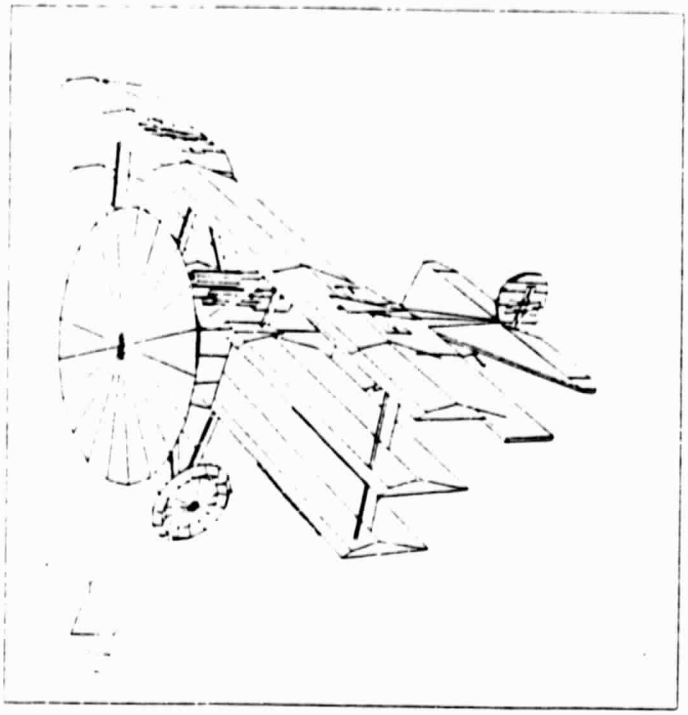


Figure 6-10
Ame's Plane

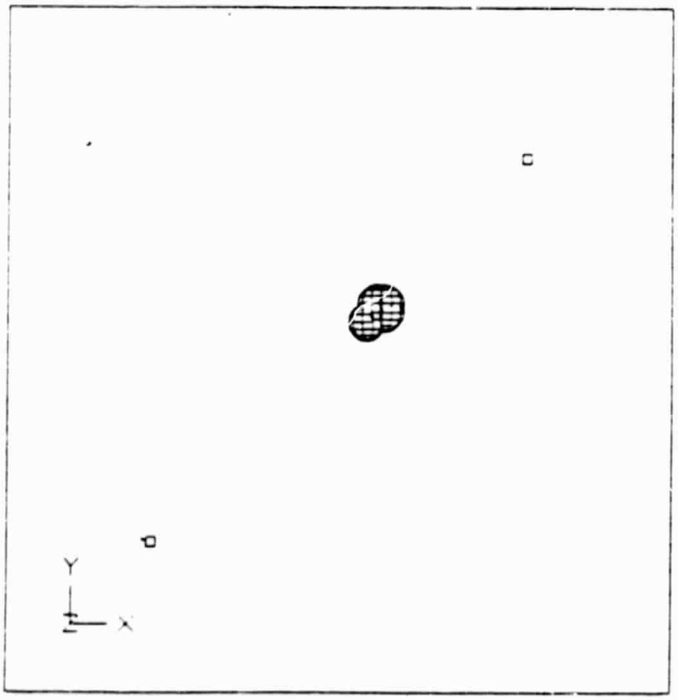


Figure 6-11
Sphere6

BIBLIOGRAPHY

REFERENCES

1. Sutherland, I.E., Sproull, R.F., and Schumacker, R.A. 'A Characterization of 10 Hidden-Surface Algorithms,' *Computing Surveys*, 1-55 March 1974.

2. Newman, W.M. and Sproull, R.F. *Principles of Interactive Computer Graphics*, 367-386 McGraw-Hill Book Company New York, N.Y. 1979

3. Christiansen, H.N. and Stephenson, M.B., *MOVIE.BYU Training Manual*, Brigham Young University Press, Brigham Young University, Provo, UT. 7-2, 1984.

4. Jones, G.K., 'A Fast Hidden Line Algorithm for Plotting Finite Element Models,' *NASA Technical Memorandum TM83981* , NASA/Goddard Space Flight Center, Greenbelt, MD. 1982.

5. Macy, S.C., 'A General Purpose Scatter Storage Subsystem and a Comparison of Hashing Methods,' Unpublished Thesis, Brigham Young University, Provo, UT., December 1984.

6. Segerlind, L.J., *Applied Finite Element Analysis*, 23-33 New York: Wiley 1976

APPENDIX A

Modified JonesD Code

```
      SUBROUTINE GVPHGD (NSVEC,NSUR,XMAX,XMIN,YMAX,YMIN,JBUCKV,JBUCKS,
     $                   XDELTV,XDELTS,YDELTV,YDELTS,NUMV,NUMS)
C
C*********************************************************************
C
C  SUBROUTINE GVPHGD - SETS UP XY BUCKET SORT PARAMETERS FOR SURFACE
C                      ELEMENTS AND SURFACE VECTORS
C
C*********************************************************************
C
C  SUBROUTINE CALLED BY
C     GHIDDN = PERFORMS MODIFIED JONESD HIDDEN LINE PROCESSING
C
C*********************************************************************
C
C  SUBPROGRAMS CALLED
C     NONE
C
C*********************************************************************
C
C  VARIABLES USED
C     JBUCKS = DEFINES THE GRID INTO WHICH THE SCREEN WILL BE DIVIDED
C              FOR THE BUCKET SORT ON THE SURFACES
C     JBUCKV = DEFINES THE GRID INTO WHICH THE SCREEN WILL BE DIVIDED
C              FOR THE BUCKET SORT ON THE VECTORS
C     NSUR   = NUMBER OF SURFACE ELEMENTS IN THE MODEL
C     NSVEC  = NUMBER OF VECTORS BELONGING TO SURFACE ELEMENTS ONLY
C     NUMS   = COUNTS THE NUMBER OF SURFACES IN A SURFACE BUCKET
C     NUMV   = COUNTS THE NUMBER OF SURFACE VECTORS IN A VECTOR BUCKET
C     XDELTS = USED FOR SORTING SURFACES INTO THEIR PROPER BUCKETS
C     XDELTV = USED FOR SORTING VECTORS INTO THEIR PROPER BUCKETS
C     XDIF   = (MAX X MODEL COORDINATE) - (MIN X MODEL COORDINATE)
C     XMAX   = MAXIMUM X MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     XMIN   = MINIMUM X MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     XYSFAC = FACTOR USED TO MAKE POSSIBLE THE DETERMINATION OF WHICH
C              BUCKET OR BUCKETS A SURFACE IS IN
C     XYVFAC = FACTOR USED TO MAKE POSSIBLE THE DETERMINATION OF WHICH
C              BUCKET OR BUCKETS A SURFACE VECTOR IS IN
C     YDELTS = USED FOR SORTING SURFACES INTO THEIR PROPER BUCKETS
C     YDELTV = USED FOR SORTING VECTORS INTO THEIR PROPER BUCKETS
C     YDIF   = (MAX Y MODEL COORDINATE) - (MIN Y MODEL COORDINATE)
C     YMAX   = MAXIMUM Y MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     YMIN   = MINIMUM Y MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C
C*********************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHGD
C     NONE
C
C*********************************************************************
C
      REAL            XMAX,XMIN,YMAX,YMIN,XDELTV,XDELTS,YDELTV,YDELTS
      INTEGER         NSVEC,NSUR,JBUCKV,JBUCKS,
     $                NUMV(18,18),
     $                NUMS(18,18)
C
C                     ************************
C                     ****BEGIN PROCESSING****
C                     ************************
C
C---------------------------------------------------------------------
C
C*********** CALCULATE GRID DENSITIES FOR X-Y BUCKET SORT ***********
C
C---------------------------------------------------------------------
C
      IF(NSVEC.LT.200) THEN
        JBUCKV=8
C---------------------------------------------------------------------
C     For example, if JBUCKV=8 then for the vectors an X-Y grid of 8
C     cells by 8 cells will be set up for bucket sort purposes.
C---------------------------------------------------------------------
      ELSE IF (NSVEC.LT.800) THEN
        JBUCKV=8
      ELSE IF (NSVEC.LT.1800) THEN
        JBUCKV=9
      ELSE IF (NSVEC.LT.2400) THEN
        JBUCKV=10
      ELSE
        JBUCKV=18
      END IF
      XYVFAC=JBUCKV-.01
C---------------------------------------------------------------------
C     XYVFAC = XYVFACtor is set to slightly less than JBUCKV for the
C     purpose of later determining, with certainty, which cells a
C     particular vector is contained within.
C---------------------------------------------------------------------
      IF (NSUR.LT.100) THEN
        JBUCKS=8
      ELSE IF (NSUR.LT.800) THEN
        JBUCKS=9
      ELSE IF (NSUR.LT.1200) THEN
        JBUCKS=10
      ELSE
        JBUCKS=18
      END IF
      XYSFAC=JBUCKS-.01
C---------------------------------------------------------------------
C     XYSFAC = XYSFACtor is set to slightly less than JBUCKS for the
C     purpose of later determining, with certainty, which cells a
C     particular surface is contained within.
C---------------------------------------------------------------------
      XDIF=(XMAX-XMIN)
      IF(XDIF.EQ.0.0) THEN
        XDIF=.01
      END IF
      YDIF=(YMAX-YMIN)
      IF(YDIF.EQ.0.0) THEN
        YDIF=.01
      END IF
      XDELTV=XYVFAC/XDIF
      XDELTS=XYSFAC/XDIF
      YDELTV=XYVFAC/YDIF
      YDELTS=XYSFAC/YDIF
C---------------------------------------------------------------------
C     XDELTV,XDELTS,YDELTV,and YDELTS are representations of the number
```

```
c     X or Y grid cells per unit of model length for vectors or
c     surfaces. These values are used in the determination of which
c     cells any vector or surface lie within - for sorting purposes.
c-----------------------------------------------------------------
c
c-----------------------------------------------------------------
c               ******ZERO THE CELL LENGTH ARRAYS******
c-----------------------------------------------------------------
c
      DO 1000 J=1,JBUCKV
         DO 1000 JJ=1,JBUCKV
            NUMV(JJ,J)=0
 1000    CONTINUE
      DO 2000 J=1,JBUCKS
         DO 2000 JJ=1,JBUCKS
            NUMS(JJ,J)=0
 2000    CONTINUE
c
      RETURN
      END


      SUBROUTINE GVPHID (PGRID,JVECLS,JSURLS,NVEC,NSUR,XMAX,XMIN,
     @                   YMAX,YMIN,NSVEC,JREDUN,JVTYPE,NNODES,LCONT,
     @                   JCSEG,LINES)
c
C*****************************************************************
C
C  SUBROUTINE GVPHID - PERFORMS MODIFIED JONES HIDDEN LINE REMOVAL
C
C*****************************************************************
C
C  SUBROUTINE CALLED BY
C     GHIDDN - PREPARES MODEL GEOMETRY FOR HIDDEN LINE PROCESSING
C
C*****************************************************************
C
C  SUBPROGRAMS CALLED
C     GVPHGD - SETS UP XY BUCKET SORT PARAMETERS FOR SURFACE ELEMENTS
C              AND SURFACE VECTORS
C     GVPHVP - PREPARES MODEL VECTOR INFORMATION FOR HIDDEN LINE
C              PROCESSING
C     GVPHSP - PREPARES MODEL SURFACE INFORMATION FOR HIDDEN LINE
C              PROCESSING
C     PLTLIN - PLOTS A LINE GIVEN THE SCREEN COORDINATES OF ITS
C              END POINTS
C     AFMODE - GRACEFULLY RETURNS THE TERMINAL TO ALPHA MODE
C
C*****************************************************************
C
C  VARIABLES USED
C     A      = CONTAINMENT TEST ARRAY
C     AR     = ARRAY CONTAINING THE CROSS PRODUCT OF ADJACENT NODES IN
C              EACH ELEMENT FOR AREA CALCULATION AND CONTAINMENT TESTING
C     ARATIO = CONTAINMENT TEST VARIABLE
C     ATOT   = CONTAINMENT TEST VARIABLE CONTAINING THE SUM OF THE
C              VALUES IN "A" FOR A SURFACE
C     DENOM  = VECTOR LINE INTERSECTION PARAMETER
C     J3     = SURFACE VECTOR BEING TESTED IN BINARY SEARCH
C     JBUCKS = DEFINES THE GRID INTO WHICH THE SCREEN WILL BE DIVIDED
C              FOR THE BUCKET SORT ON THE SURFACES
C     JBUCKV = DEFINES THE GRID INTO WHICH THE SCREEN WILL BE DIVIDED
C              FOR THE BUCKET SORT ON THE VECTORS
C     JCHK   = VECTOR INTERSECTION COMPUTATION FLAG
C                1 = ABS(DELTA X) > ABS(DELTA Y) FOR VECTOR
C                2 = ABS(DELTA Y) > ABS(DELTA X) FOR VECTOR
C     JCSEG  = CONTAINS THE NUMBER OF DIFFERENT CONTOUR STRINGS IN EACH
C              OF THE CONTOUR LEVELS AND THE NUMBER OF LINE SEGMENTS IN
C              EACH STRING BEFORE HIDDEN LINE REMOVAL
C     JGPA   = ????????????????????
C     JGPB   = ?????????????????????
C     JPC    = THE DISTANCE TO GO INTO THE SORTED LIST OF SURFACES IN A
C              BUCKET (JXYS) TO ACCURATELY DETERMINE THE VISIBILITY
C              OF A LINE SEGMENT IN THAT BUCKET.
C     JREDUN = REDUNDANT EDGE FLAG
C                1 = NOT REDUNDANT
C                2+ = REDUNDANT
C     JSEG   = KEEPS TRACK OF THE NUMBER OF SEPARATE CONTOUR STRINGS IN
C              EACH CONTOUR LEVELAND THE NUMBER OF LINE SEGMENTS IN
C              EACH STRING AFTER HIDDEN LINE REMOVAL
C     JSORTP = Z-DEPTH SHELL SORT PARAMETER
C     JSURLS = ELEMENT CONNECTIVITY ARRAY
C     JVECLS = VECTOR CONNECTIVITY ARRAY (INCLUDES CONTOUR VECTORS IF
C              COMPUTED)
C     JVECVP = VECTOR VISIBILITY FLAG WHICH PREVENTS UNNECESSARY LINE
C              INTERSECTION CALCULATIONS
C     JVISLS = NODE VISIBILITY LIST USED IN LABELING VISIBLE NODES
C     JVP    = ARRAY CONTAINING STARTING AND STOPPING BUCKETS FOR ALL
C              THE VECTORS, SURFACE OR LINE ELEMENT
C     JVTYPE = VECTOR TYPE FLAG
C                0 = LINE ELEMENT TYPE
C                1 = SURFACE VECTOR
C     JX     = INDEX INTO NUMS TO DETERMINE HOW MANY SURFACES LIE IN THE
C              BUCKET THAT THE LINE SEGMENT BEING PROCESSED LIES IN
C     JXYS   = ARRAY INTO WHICH ALL THE SURFACES ARE MAPPED BY BUCKET
C     JXYV   = ARRAY INTO WHICH ALL THE SURFACE VECTORS ARE MAPPED BY
C              BUCKET
C     JY     = INDEX INTO NUMS TO DETERMINE HOW MANY SURFACES LIE IN THE
C              BUCKET THAT THE LINE SEGMENT BEING PROCESSED LIES IN
C     KFLIP  = NODE VISIBILITY FLAG
C     KK     = INDEX INTO JSEG ARRAY (COUNTS THE NUMBER OF CONNECTED
C              CONTOUR STRINGS AT A CONTOUR LEVEL)
C     KKOUNT = COUNTS THE NUMBER OF VECTORS IN A CONNECTED STRING AFTER
C              HIDDEN PROCESSING
C     KOUNT  = COUNTS CONTOUR VECTORS AS PROCESSED IN HIDDEN IN ORDER TO
C              DETERMINE WHEN TO GO TO THE NEXT LEVEL IN JCSEG
C     JCHKS  = ERROR FLAG INDICATING THAT MAX NUMBER OF SURFACE ELEMENTS
C              HAS BEEN EXCEEDED
C     JCHKV  = ERROR FLAG INDICATING THAT MAX NUMBER OF VECTOR ELEMENTS
C              HAS BEEN EXCEEDED
C     LAST   = BINARY SEARCH PARAMETER
C     LCON   = CONTOUR VECTOR CONNECTIVITY LOGICAL
C     LCONT  = LOGICAL INDICATING WHETHER CONTOURS ARE ENABLED
C                .TRUE.   = CONTOURS ENABLED
C                .FALSE.  = CONTOURS DISABLED
C     LIMSUR = MAXIMUM NUMBER OF SURFACES THAT CAN BE IN ANY ONE SURFACE
```

```
C              BUCKET
C     LINVEC = MAXIMUM NUMBER OF SURFACE VECTORS PER BUCKET
C     LINES  = CONTAINS NUMBER OF DIFFERENT CONTOUR LEVELS
C     LOW    = BINARY SEARCH PARAMETER
C     MAXX   = BINARY SEARCH PARAMETER
C     NK     = KEEPS TRACK OF THE CONTOUR LEVEL BEING PROCESSED
C     NN     = INDEX ITO JCSEG ARRAY (INDICATES WHICH CONNECTED STRING
C              OF CONTOURS AT A PARTICULAR CONTOUR LEVEL IS BEING
C              PROCESSED
C     NNODES = ARRAY CONTAINING THE NUMBER OF NODES IN EACH ELEMENT
C     NSEG   = LINE SEGMENT COUNTER (COUNTS THE NUMBER OF LINE SEGMENTS
C              INTO WHICH A VECTOR IS SUBDIVIDED AT INTERSECTION
C              LOCATIONS)
C     NSUR   = NUMBER OF SURFACE ELEMENTS IN THE MODEL
C     NSVEC  = NUMBER OF VECTORS BELONGING TO SURFACE ELEMENTS
C     NUMS   = COUNTS THE NUMBER OF SURFACES IN A SURFACE BUCKET
C     NUMV   = COUNTS THE NUMBER OF VECTORS IN A VECTOR BUCKET
C     NVEC   = NUMBER OF VECTORS IN THE MODEL (INCLUDES CONTOUR VECTORS
C              IF COMPUTED)
C     PGRID  = COORDINATE ARRAY (INCLUDES CONTOUR COORDINATES IF
C              COMPUTED)
C     SURF   = CONTAINS MIN AND MAX X,Y,& Z COORDINATE VALUES SURFACE
C              AREA*2, AND PLANE EQUATION INFORMATION FOR EACH SURFACE
C     VECOUT = BUFFERS VISIBLE LINE SEGMENTS FOR DISPLAY BY STEVE
C              MACEY'S GRAPHICS PACKAGE
C     VECTOR = HOLDS VECTOR INFORMATION SUCH AS COORDINATE ENDPOINTS,
C              MIN AND MAX X,Y,& Z COORDINATE VALUES AND VECTOR
C              INTERSECTION PARAMETERS
C     VXI    = RATIO OF DISTANCE ALONG VECTOR TO POINT OF INTERSECTION
C              AS DETERMINED FROM THE X INTERSECTION LOCATION
C     VYI    = RATIO OF DISTANCE ALONG VECTOR TO POINT OF INTERSECTION
C              AS DETERMINED FROM THE Y INTERSECTION LOCATION
C     XDELTS = USED FOR SORTING SURFACES INTO THEIR PROPER BUCKETS
C     XDELTV = USED FOR SORTING VECTORS INTO THEIR PROPER BUCKETS
C     XINT   = X COORDINATE OF VECTOR INTERSECTION POINT
C     XIT    = XINT - XITF TO HANDLE ROUND OFF
C     XITF   = RATIO OF XINT USED TO HANDLE ROUND OFF PROBLEMS
C     XLINE  = ARRAY CONTAINING LINE SEGMENT ENDPOINT COORDINATES AND
C              THE LINE SEGMENT VISIBILITY FLAG
C     XMAX   = MAXIMUM X MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     XMAXI  = MAXIMUM X VALUE FOR THE VECTOR BEING PROCESSED
C     XMID   = X COORDINATE OF THE MIDPOINT OF THE LINE SEGMENT BEING
C              PROCESSED
C     XMIN   = MINIMUM X MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     XMINI  = MINIMUM X VALUE FOR THE VECTOR BEING PROCESSED
C     XP     = ARRAY CONTAINING THE X COORDINATES OF EACH ELEMENT
C              VERTICE
C     YDELTS = USED FOR SORTING SURFACES INTO THEIR PROPER BUCKETS
C     YDELTV = USED FOR SORTING VECTORS INTO THEIR PROPER BUCKETS
C     YINT   = Y COORDINATE OF VECTOR INTERSECTION POINT
C     YIT    = YINT - YITF TO HANDLE ROUND OFF
C     YITF   = RATIO OF YINT USED TO HANDLE ROUND OFF PROBLEMS
C     YMAX   = MAXIMUM Y MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     YMAXI  = MAXIMUM Y VALUE FOR THE VECTOR BEING PROCESSED
C     YMID   = Y COORDINATE OF THE MIDPOINT OF THE LINE SEGMENT BEING
C              PROCESSED
C     YMIN   = MINIMUM Y MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     YMINI  = MINIMUM Y VALUE FOR THE VECTOR BEING PROCESSED
C     YP     = ARRAY CONTAINING THE Y COORDINATES OF EACH ELEMENT
C              VERTICE
C     ZINT   = Z COORDINATE OF VECTOR INTERSECTION POINT
C     ZMID   = Z COORDINATE OF THE MIDPOINT OF THE LINE SEGMENT BEING
C              PROCESSED
C     ZP     = ARRAY CONTAINING THE Z COORDINATES OF EACH ELEMENT
C              VERTICE
C     ZP1    = VISIBILITY VARIABLE USED IN CALCULATION OF THE PLANE
C              EQUATION FOR A SURFACE
C     ZP2    = VISIBILITY VARIABLE USED IN CALCULATION OF THE PLANE
C              EQUATION FOR A SURFACE
C     ZP3    = VISIBILITY VARIABLE USED IN CALCULATION OF THE PLANE
C              EQUATION FOR A SURFACE
C     ZSUR   = Z-DEPTH OF X-Y POINT ON SURFACE WHICH CORRESPONDS WITH
C              THE X-Y LOCATION OF THE LINE SEGMENT CENTER POINT BEING
C              TESTED FOR VISIBILTY
C
C*********************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE SVPHID
C
      INCLUDE '/g/rt/work/thesis/working/dparam.h'
C
C     MAXNS  = MAXIMUM NUMBER OF NODES PER POLYGON NOT INCLUDING CENTER
C              NODE
C     MAXNJ  = MAXIMUM NUMBER OF COORDINATES (NODES)
C     MAXNPT = MAXIMUM NUMBER OF ELEMENTS
C     MAXVEC = MAXIMUM NUMBER OF LINE SEGMENTS IN MODEL
C     MXCONT = MAXIMUM NUMBER OF CONTOUR LEVELS
C     MXCLEV = MAXIMUM NUMBER OF SEPARATE CONTOUR STRINGS AT THE SAME
C              CONTOUR LEVEL
C     MXHCLV = MAXIMUM NUMBER OF SEPARATE CONTOUR STRINGS AT THE SAME
C              CONTOUR LEVEL AFTER HIDDEN
C     MXBSUR = MAXIMUM NUMBER OF SURFACES PER BUCKET IN JONESD HIDDEN
C     MXBVEC = MAXIMUM NUMBER OF VECTORS PER BUCKET IN JONESD HIDDEN
C
C*********************************************************************
C
      INTEGER        JVECLS(2,MAXVEC),
     $               JSURLS(MAXNS,MAXNPT),
     $               JVISLS(MAXNJ),
     $               JXYV(13,13,MXBVEC),
     $               NUMV(13,13),
     $               JVP(4,MAXVEC),
     $               JXYS(13,13,MXBSUR),
     $               NUMS(13,13),
     $               JSORTP(8),
     $               JVTYPE(MAXVEC),
     $               JREDUN(MAXVEC),
     $               JVECVP(MAXVEC)
      INTEGER        NNODES(MAXNPT),
     $               JCSEG(MXCLEV,MXCONT),
     $               JSEG(MXHCLV,MXCONT)
C
      REAL           PGRID(3,MAXNJ),
     $               VECTOR(15,MAXVEC),
     $               SURF(11,MAXNPT),
     $               XMAX,XMIN,YMAX,YMIN,
```

```
     $                 XLINE(4,MX2VEC),
     $                 XP(MAXNS,MAXNPT),
     $                 YP(MAXNS,MAXNPT),
     $                 AR(MAXNS,MAXNPT),
     $                 A(MAXNS)
C
      LOGICAL          LCONT,LCON
C
      DATA             JSORTP/1,3,7,15,31,63,127,255/
C
      LIMVEC=MXBVEC
      LIMSUR=MXBSUR
      JCHKS=0
      JCHKV=0
      TOLER=.01
C
C                      ************************
C                      ****BEGIN PROCESSING****
C                      ************************
C
C-----------------------------------------------------------------------
C
C***********************PREP WORK FOR HIDDEN LINE***********************
C
C-----------------------------------------------------------------------
C
C-----------------------------------------------------------------------
C        ******CALCULATE GRID DENSITIES FOR X-Y BUCKET SORT******
C-----------------------------------------------------------------------
C
      CALL GVPHGD (NSVEC,NSUR,XMAX,XMIN,YMAX,YMIN,JBUCKV,JBUCKS,
     $               XDELTV,XDELTS,YDELTV,YDELTS,NUMV,NUMS)
C
C-----------------------------------------------------------------------
C                       ******DO VECTOR PREP******
C-----------------------------------------------------------------------
C
      CALL GVPHVP (NVEC,JREDUN,JVECLS,PGRID,X1,Y1,Z1,XDELTV,
     $             YDELTV,XMIN,YMIN,JVTYPE,JSORTP,JBUCKV,LIMVEC,
     $             JVECVP,VECTOR,NUMV,JXYV,JVP,JCHKV)
C
      IF(JCHKV.EQ.1) RETURN
C
C-----------------------------------------------------------------------
C                       ******DO SURFACE PREP******
C-----------------------------------------------------------------------
C
      CALL GVPHSP (NSUR,JSURLS,PGRID,XMIN,YMIN,XDELTS,YDELTS,
     $             JSORTP,JBUCKS,LIMSUR,SURF,NUMS,JXYS,JCHKS,
     $             NNODES,XP,YP,AR)
C
      IF(JCHKS.EQ.1) RETURN
C
C-----------------------------------------------------------------------
C
C*****************************INITIALIZATION****************************
C
C-----------------------------------------------------------------------
C
C-----------------------------------------------------------------------
C     Here we are setting things up for contours if they are requested
C     All of the variables below are used to count contour segments
C     so they can be accurately labeled and curve fit.
C-----------------------------------------------------------------------
      IF(LCONT) THEN
        KK=0
        NK=1
        NN=1
        KOUNT=0
        KKOUNT=0
        LCON=.TRUE.
C
        DO 2700 J=1,LINES
        DO 2700 JJ=1,MXHCLV
          JSEG(JJ,J)=0
 2700   CONTINUE
C
C-----------------------------------------------------------------------
C       If the specified range of contours excludes some of the contour
C       levels, this gets us to the first visible one.  That is NK will
C       be incremented until we find a contour level with some segments
C       in it.
C-----------------------------------------------------------------------
        DO 2702 JX=1,LINES
          IF(JCSEG(1,JX).EQ.0) THEN
            NK=NK+1
          ELSE
            GO TO 2703
          END IF
 2702   CONTINUE
      END IF
 2703 CONTINUE
C
C-----------------------------------------------------------------------
C
C*********************** HIDDEN LINE COMPUTATION ***********************
C
C-----------------------------------------------------------------------
C
      DO 4000 K=1,NVEC
        IF(JREDUN(K).GT.1) GO TO 4000
        KFLIP=0
        NSEG=1
        JGPA=JVECLS(1,K)
        JGPB=JVECLS(2,K)
        JCHK=1
        IF(ABS(VECTOR(7,K)).GT.ABS(VECTOR(8,K))) JCHK=2
        IF(VECTOR(7,K).EQ.0.0.AND.VECTOR(8,K).EQ.0.0) GO TO 4000
C
C     ******THE ARRAY XLINE(1-3,K) HOLDS THE VECTOR END POINTS AND
C           POINTS OF INTERSECTION. XLINE(4,K) INDICATES SEGMENT
C           VISIBILITY: 0 = HIDDEN, 1 = VISIBLE******
C
        XLINE(1,1)=VECTOR(1,K)
        XLINE(2,1)=VECTOR(2,K)
        XLINE(3,1)=VECTOR(3,K)
```

```
      XLINE(4,1)=1.
      XMINI=VECTOR(13,K)
      YMINI=VECTOR(14,K)
      XMAXI=VECTOR(10,K)
      YMAXI=VECTOR(11,K)
      ZMAXI=VECTOR(12,K)
C
C-------------------------------------------------------------------
C
C              ******LINE INTERSECTION COMPUTATION******
C
C-------------------------------------------------------------------
C
      DO 8600 JX=JVP(1,K),JVP(2,K)
      DO 8500 JY=JVP(3,K),JVP(4,K)
C
C-------------------------------------------------------------------
C     ******BINARY SEARCH TO OBTAIN SEARCH DEPTH IN VECTOR MAP******
C-------------------------------------------------------------------
C
       LEN=NUMV(JX,JY)
       IF(LEN.LT.3) GO TO 8425
       MAXX=LEN-1
       LOW=0
       LAST=0
       JPC=LEN/2
8405   IF(LAST.EQ.JPC) GO TO 8415
       LAST=JPC
       J3=JXYV(JX,JY,JPC)
       IF(VECTOR(15,J3).LT.ZMAXI) THEN
         J3=JXYV(JX,JY,JPC+1)
         IF(VECTOR(15,J3).GE.ZMAXI) GO TO 8435
         LOW=JPC
       ELSE
         MAXX=JPC
       END IF
       JPC=(LOW+MAXX+1)/2
       GO TO 8405
8415   CONTINUE
C
C-------------------------------------------------------------------
C              ******DETERMINE LINE INTERSECTIONS******
C-------------------------------------------------------------------
C
       IF(JPC.EQ.1) GO TO 8500
8425   JPC=LEN
8435   DO 8450 JP=1,JPC
         J=JXYV(JX,JY,JP)
C-------------------------------------------------------------------
C     ******JVECVP(J)=0 IF VECTOR J HAS BEEN FOUND TO BE INVISIBLE******
C     This means that this vector was found to be invisible when it came
C     through earlier in the 4000 loop.
C-------------------------------------------------------------------
         IF(JVECVP(J).EQ.0) GO TO 8450
C
C-------------------------------------------------------------------
C     This is a min-max test to see if the active vector and the vector
C     it is being tested against intersect
C-------------------------------------------------------------------
            IF(VECTOR(13,J).LE.XMAXI.AND.VECTOR(10,J).GE.XMINI.AND.
     $      VECTOR(14,J).LE.YMAXI.AND.VECTOR(11,J).GE.YMINI) THEN
C
C-------------------------------------------------------------------
C     This is testing to see if the active vector and the vector it is
C     being tested against are the same.
C-------------------------------------------------------------------
              IF(JVECLS(1,J).EQ.JVECLS(1,K).OR.
     $        JVECLS(1,J).EQ.JVECLS(2,K).OR.
     $        JVECLS(2,J).EQ.JVECLS(2,K).OR.
     $        JVECLS(2,J).EQ.JVECLS(1,K)) GO TO 8450
C
C-------------------------------------------------------------------
C     If the vector wasn't rejected in the first two tests then start
C     intersection calculations.  If the intersection point is outside
C     of the active vector, then go to the next one as in go to 8450
C     below.
C-------------------------------------------------------------------
              DENOM=VECTOR(7,K)*VECTOR(6,J)-VECTOR(7,J)*VECTOR(6,K)
              IF(DENOM.NE.0.0) THEN
C
                IF(JCHK.EQ.1) THEN
                  XINT=(VECTOR(6,K)*VECTOR(9,J)-VECTOR(6,J)
     $            *VECTOR(9,K))/DENOM
                  IF(XINT.GT.XMAXI.OR.XINT.LT.XMINI) GO TO 8450
                  VXI=(VECTOR(1,K)-XINT)/VECTOR(6,K)
                  YINT=VECTOR(2,K)+VXI*VECTOR(7,K)
                  ZINT=VECTOR(3,K)+(VECTOR(6,K)-VECTOR(3,K))*VXI
                ELSE
                  YINT=(VECTOR(9,K)*VECTOR(7,J)-VECTOR(7,K)
     $            *VECTOR(9,J))/DENOM
                  IF(YINT.GT.YMAXI.OR.YINT.LT.YMINI) GO TO 8450
                  VYI=(VECTOR(2,K)-YINT)/VECTOR(7,K)
                  XINT=VECTOR(1,K)+VYI*VECTOR(6,K)
                  ZINT=VECTOR(3,K)+(VECTOR(3,K)-VECTOR(6,K))*VYI
                END IF
C
                IF(ABS(VECTOR(6,J)).GT.ABS(VECTOR(7,J))) THEN
                  IF(XINT.GT.VECTOR(10,J).OR.XINT.LT.VECTOR(13,J)) THEN
                    XITF=.001*(ABS(XINT))
                    XIT=XINT-XITF
                    IF(XIT.GT.VECTOR(10,J)) GO TO 8450
                    XIT=XINT+XITF
                    IF(XIT.LT.VECTOR(13,J)) GO TO 8450
                  END IF
                  ZJNT=VECTOR(3,J)+(VECTOR(3,J)-VECTOR(6,J))*
     $            (XINT-VECTOR(1,J))/VECTOR(6,J)
                ELSE IF (VECTOR(7,J).NE.0.0) THEN
                  IF(YINT.GT.VECTOR(11,J).OR.YINT.LT.VECTOR(14,J)) THEN
                    YITF=.001*(ABS(YINT))
                    YIT=YINT-YITF
                    IF(YIT.GT.VECTOR(11,J)) GO TO 8450
                    YIT=YINT+YITF
                    IF(YIT.LT.VECTOR(14,J)) GO TO 8450
                  END IF
                  ZJNT=VECTOR(3,J)+(VECTOR(3,J)-VECTOR(6,J))*
     $            (YINT-VECTOR(2,J))/(-VECTOR(7,J))
```

```
              ELSE
                GO TO 3450
              END IF
C
C-------------------------------------------------------------------
C             ******DEPTH AT INTERSECTION TEST******
C-------------------------------------------------------------------
C
C-------------------------------------------------------------------
C     If the edge vector is in front of the active vector then divide
C     the active vector into two segments at the calculated intersection
C     point.
C-------------------------------------------------------------------
              IF(ZJNT.LT.ZINT) THEN
                NSEG=NSEG+1
                XLINE(1,NSEG)=XINT
                XLINE(2,NSEG)=YINT
                XLINE(3,NSEG)=ZINT
                XLINE(4,NSEG)=1.
              END IF
C
            END IF
          END IF
 3450     CONTINUE
 3500   CONTINUE
C
C-------------------------------------------------------------------
C         ******INSERT END POINTS AT INTERSECTION LOCATIONS******
C-------------------------------------------------------------------
C
        NSEG=NSEG+1
        XLINE(1,NSEG)=VECTOR(4,K)
        XLINE(2,NSEG)=VECTOR(5,K)
        XLINE(3,NSEG)=VECTOR(6,K)
        XLINE(4,NSEG)=1.
C
C-------------------------------------------------------------------
C           ******SORT INTERSECTION LIST FROM MIN TO MAX******
C-------------------------------------------------------------------
C
C-------------------------------------------------------------------
C     Here we are sorting the segments that have resulted from a single
C     vector - the one presently being processed from JVECLS.
C-------------------------------------------------------------------
        IF(NSEG.GT.3) THEN
          IF(XLINE(JCHK,1).GT.XLINE(JCHK,NSEG)) KFLIP=1
          IF(NSEG.LE.5) THEN
            JCT=1
          ELSE IF (NSEG.LE.13) THEN
            JCT=2
          ELSE IF (NSEG.LE.29) THEN
            JCT=3
          ELSE IF (NSEG.LE.61) THEN
            JCT=4
          ELSE IF (NSEG.LE.125) THEN
            JCT=5
          ELSE
            JCT=6
          END IF
C
          DO 3600 JINDEX=JCT,1,-1
            JUMP=JSORTP(JINDEX)
            JMAX=NSEG-JUMP
 3545       JFLIP=0
            DO 3550 M=1,JMAX
              N=M+JUMP
              IF(XLINE(JCHK,M).GT.XLINE(JCHK,N)) THEN
                DO 3540 JS=1,4
                  TEMP=XLINE(JS,M)
                  XLINE(JS,M)=XLINE(JS,N)
                  XLINE(JS,N)=TEMP
 3540           CONTINUE
                JFLIP=1
              END IF
 3550       CONTINUE
            IF(JFLIP.EQ.1) GO TO 3545
 3600     CONTINUE
        END IF
C
C-------------------------------------------------------------------
C
C
C          ******COMPUTE VISIBILITY OF LINE SEGMENTS******
C
C
C-------------------------------------------------------------------
C
        DO 3700 J=2,NSEG
C-------------------------------------------------------------------
C     We will do all of our visibility testing on the midpoint of the
C     segment being processed since if the midpoint is behind a surface,
C     the whole segment is and vice versa.  Read Ron Thue's thesis
C     to get a better understanding of this.
C-------------------------------------------------------------------
          XMID=(XLINE(1,J-1)+XLINE(1,J))/2
          YMID=(XLINE(2,J-1)+XLINE(2,J))/2
          ZMID=(XLINE(3,J-1)+XLINE(3,J))/2
C-------------------------------------------------------------------
C     Find the bucket that this segment midpoint lies in so we can find
C     all the surfaces to test against it.
C-------------------------------------------------------------------
          JX=1+(XMID-XMIN)*XDELTB
          JY=1+(YMID-YMIN)*YDELTB
C
C-------------------------------------------------------------------
C     ******BINARY SEARCH TO OBTAIN SEARCH DEPTH IN SURFACE MAP******
C-------------------------------------------------------------------
C
C-------------------------------------------------------------------
C     Find the surfaces that lie in front of the segment only
C-------------------------------------------------------------------
          LEN=NUMS(JX,JY)
          IF(LEN.LT.3) GO TO 3825
          MAXX=LEN-1
          LOW=0
          LAST=0
          JPC=LEN/2
 3805     IF(LAST.EQ.JPC) GO TO 3815
```

```
      LAST=JPC
      JS=JXYS(JX,JY,JPC)
      IF(SURF(6,JS).LT.ZMID) THEN
        JS=JXYS(JX,JY,JPC+1)
        IF(SURF(6,JS).GE.ZMID) GO TO 3635
        LOW=JPC
      ELSE
        MAXX=JPC
      END IF
      JPC=(LOW+MAXX+1)/2
      GO TO 3606
3615  CONTINUE
      IF(JPC.EQ.1) GO TO 3700
3625  JPC=LEN
C
C------------------------------------------------------------------------
C          ******LINE SEGMENT MID-POINT VISIBILITY TESTS******
C------------------------------------------------------------------------
3635  DO 3650 JP=1,JPC
C------------------------------------------------------------------------
C     Do a quick minmax test to see if this surface can be eliminated
C     as a possible surface that could hide the segment being processed.
C------------------------------------------------------------------------
        JS=JXYS(JX,JY,JP)
        IF(SURF(1,JS).GE.XMID.AND.SURF(2,JS).LE.XMID.AND.
     *  SURF(3,JS).GE.YMID.AND.SURF(4,JS).LE.YMID) THEN
C
C------------------------------------------------------------------------
C                ******CONTAINMENT TEST******
C------------------------------------------------------------------------
C
C------------------------------------------------------------------------
C     Read Ron Thue's thesis to get a great understanding of this test.
C     Basically it sayes that if a point lies inside of a polygon then
C     the sum of the areas of the triangles, created by drawing lines
C     from the point to the polygon vertices, will equal the area of
C     the element.  If outside the summed area will be larger than the
C     triangle area.
C------------------------------------------------------------------------
          IF(SURF(7,JS).GT.0.0) THEN
            ATOT=0.0
            DO 3641 NS=1,NNODES(JS)
              IF(NS.EQ.NNODES(JS)) THEN
                A(NS)=ABS(XMID*(YP(NS,JS)-YP(1,JS))+YMID*
     *          (XP(1,JS)-XP(NS,JS))+AR(NS,JS))
              ELSE
                A(NS)=ABS(XMID*(YP(NS,JS)-YP(NS+1,JS))+YMID*
     *          (XP(NS+1,JS)-XP(NS,JS))+AR(NS,JS))
              END IF
              ATOT=ATOT+A(NS)
3641        CONTINUE
            ARATIO=ATOT*SURF(7,JS)
            IF(ARATIO.LT.(1.001)) THEN
C
C------------------------------------------------------------------------
C                ******INITIAL DEPTH TEST******
C------------------------------------------------------------------------
C
              IF((ZMID-TOLER).GT.SURF(5,JS)) THEN
                XLINE(4,J)=0.
                GO TO 3700
              END IF
C
C------------------------------------------------------------------------
C       ******IS MID-POINT ON THE SURFACE BOUNDARY?******
C------------------------------------------------------------------------
C
C------------------------------------------------------------------------
C     See Ron Thue's thesis for an explanation of what is happening here.
C------------------------------------------------------------------------
              DO 3642 NS=1,NNODES(JS)
                IF((A(NS)*SURF(7,JS)).LT.(.0005)) GO TO 3650
3642          CONTINUE
              DO 3644 NS=1,NNODES(JS)
                IF(JGPA.EQ.JSURLS(NS,JS)) THEN
                  DO 3643 NSI=1,NNODES(JS)
                    IF(JGPB.EQ.JSURLS(NSI,JS)) GO TO 3650
3643              CONTINUE
                END IF
3644          CONTINUE
C
C------------------------------------------------------------------------
C   ******IF SURFACE DATA NOT ALREADY COMPUTED THEN CALCULATE IT******
C------------------------------------------------------------------------
C
C------------------------------------------------------------------------
C     Set up the plane equation for this element.  With the equation
C     you can plug in the X and Y values for the segment midpoint and
C     obtain the depth of the plane at that point.  Then compare the
C     depth of the plane to the depth of the segment midpoint to
C     determine visibility.
C------------------------------------------------------------------------
             IF(SURF(10,JS).EQ.0.0) THEN
C
               SURF(10,JS)=AR(1,JS)+AR(2,JS)
     *                +(XP(3,JS)*YP(1,JS)-XP(1,JS)*YP(3,JS))
               IF(SURF(10,JS).EQ.0.0) GO TO 3650
               ZP1=PGRID(3,JSURLS(1,JS))
               ZP2=PGRID(3,JSURLS(2,JS))
               ZP3=PGRID(3,JSURLS(3,JS))
               SURF(8,JS)=ZP1*(YP(2,JS)-YP(3,JS))
     *                +ZP2*(YP(3,JS)-YP(1,JS))
     *                +ZP3*(YP(1,JS)-YP(2,JS))
               SURF(9,JS)=ZP1*(XP(3,JS)-XP(2,JS))
     *                +ZP2*(XP(1,JS)-XP(3,JS))
     *                +ZP3*(XP(2,JS)-XP(1,JS))
               SURF(11,JS)=ZP1*AR(2,JS)+ZP2
     *                *(XP(3,JS)*YP(1,JS)-XP(1,JS)*YP(3,JS))
     *                +ZP3*AR(1,JS)
C
             END IF
C
C------------------------------------------------------------------------
C     ******DEPTH TEST: IF MID-POINT IS BEHIND SURFACE, THEN VECTOR
```

```
C            SEGMENT IS NOT VISIBLE******
C--------------------------------------------------------------------
C
             ZSUR=(SURF(11,JS)+XMID*SURF(8,JS)+
                  YMID*SURF(9,JS))/SURF(10,JS)
             IF((ZMID-TOLER).GT.ZSUR) THEN
               XLINE(4,J)=0.
               GO TO 8700
C
             END IF
            END IF
           END IF
          END IF
 8650     CONTINUE
 8700   CONTINUE
C
C--------------------------------------------------------------------
C
C********************** DRAW VISIBLE LINE SEGMENTS *********************
C
C--------------------------------------------------------------------
C
        JVECVP(K)=0
C
C--------------------------------------------------------------------
C      ******DETERMINE VISIBILITY OF CONTOUR STRINGS IF CONTOUR OPTION
C                                                  ENABLED******
C
C--------------------------------------------------------------------
C
        IF(LCONT) THEN
C--------------------------------------------------------------------
C     We only want to send contour vectours through the labeling and
C     counting code, not the model vectors.
C--------------------------------------------------------------------
          IF(K.GT.NVVEC) THEN
            TOLER=.06
            KOUNT=KOUNT+1
            DO 8800 J=2,NSEG
              IF(XLINE(4,J).NE.0.) THEN
C
C--------------------------------------------------------------------
C        ******OUTPUT VISIBLE SEGMENTS TO GRAPHICS DEVICE******
C                     ***CONTOURS ENABLED***
C
C--------------------------------------------------------------------
C
                KKOUNT=KKOUNT+1
C--------------------------------------------------------------------
C     If LCON is true then we know that a loop of vectors just became
C     visible and we need to label it.
C--------------------------------------------------------------------
                IF(LCON) THEN
                  KK=KK+1
                  LCON=.FALSE.
                  CALL LABELS (XLINE(1,J),XLINE(2,J),64+NK,1)
                END IF
                CALL PLTLIN(XLINE(1,J-1),XLINE(2,J-1),
                            XLINE(1,J),XLINE(2,J))
                JVECVP(K)=1
C--------------------------------------------------------------------
C     The JSEG array is counting up how many loops are at each level
C     after hidden processing.  The idea here is to enable the fitting
C     of splines to the contour points
C--------------------------------------------------------------------
                JSEG(KK,NK)=KKOUNT
              ELSE
                LCON=.TRUE.
                KKOUNT=0
              END IF
 8800       CONTINUE
C--------------------------------------------------------------------
C     Find out if we are at the end of a loop at this level and if so
C     go on to the next one.
C--------------------------------------------------------------------
            IF(JCSEG(NN,NK).EQ.KOUNT) THEN
              LCON=.TRUE.
              NN=NN+1
              KOUNT=0
              KKOUNT=0
              IF(JCSEG(NN,NK).EQ.0) THEN
                DO 8900 JX=NK+1,LINES
                  IF(JCSEG(1,JX).NE.0) THEN
                    NK=NK+1
                    KK=0
                    NN=1
                    GO TO 8815
                  ELSE
                    NK=NK+1
                  END IF
 8900           CONTINUE
              END IF
            END IF
          ELSE
C--------------------------------------------------------------------
C     This is the normal segment drawing code which the model vectors
C     go through when contours are enabled.
C--------------------------------------------------------------------
            DO 8810 J=2,NSEG
              IF(XLINE(4,J).NE.0.) THEN
                CALL PLTLIN(XLINE(1,J-1),XLINE(2,J-1),
                            XLINE(1,J),XLINE(2,J))
                JVECVP(K)=1
              END IF
 8810       CONTINUE
 8815     END IF
        ELSE
C
C--------------------------------------------------------------------
C        ******OUTPUT VISIBLE SEGMENTS TO GRAPHICS DEVICE******
C               ******CONTOURS DISABLED******
C--------------------------------------------------------------------
C
          DO 8820 J=2,NSEG
            IF(XLINE(4,J).NE.0.) THEN
              CALL PLTLIN(XLINE(1,J-1),XLINE(2,J-1),
                          XLINE(1,J),XLINE(2,J))
              JVECVP(K)=1
```

```
            END IF
 3920      CONTINUE
C
C-----------------------------------------------------------------------
C        ******CHECK AND RECORD VISIBILITY OF END POINTS******
C-----------------------------------------------------------------------
C
           IF(NSEG.LT 3.OR.KFLIP.EQ.0) THEN
             IF(XLINE(4,2).EQ.1.) THEN
               JVISLS(JVECLS(1,K))=2
             END IF
             IF(XLINE(4,NSEG).EQ.1.) THEN
               JVISLS(JVECLS(2,K))=2
             END IF
           ELSE
             IF(XLINE(4,2).EQ.1.) THEN
               JVISLS(JVECLS(2,K))=2
             END IF
             IF(XLINE(4,NSEG).EQ.1.) THEN
               JVISLS(JVECLS(1,K))=2
             END IF
           END IF
         END IF
C
 4000 CONTINUE
      CALL ANNODE
C
      RETURN
      END


      SUBROUTINE GVPRSP (NSUR,JSURLS,PGRID,XMIN,YMIN,XDELTS,YDELTS,
     $                   JSORTP,JBUCKS,LINSUR,SURF,NUMS,JXYS,JCHKS,
     $                   NNODES,XP,YP,AR)
C
C***********************************************************************
C
C  SUBROUTINE GVPRSP - PREPARES MODEL SURFACE INFORMATION FOR MODIFIED
C                      JONESD HIDDEN LINE COMPUTATION
C
C***********************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHID - PERFORMS MODIFIED JONESD HIDDEN LINE PROCESSING
C
C***********************************************************************
C
C  SUBPROGRAMS CALLED
C     GVPHST - COMPUTES SHELL SORT PARAMETER JCT
C
C***********************************************************************
C
C  VARIABLES USED
C     AR      = ARRAY CONTAINING THE CROSS PRODUCT OF ADJACENT NODES IN
C               EACH ELEMENT FOR AREA CALCULATION AND CONTAINMENT TESTING
C     ARE     = USED TO OBTAIN THE AREA OF AN ELEMENT
C     AREA    = CONTAINS 2 TIMES THE AREA OF THE ELEMENT AS DETERMINED
C               BY CROSS PRODUCT
C     HXPOLD  = HOLDING VARIABLE USED IN DETERMINING MAX X ELEMENT
C               COORDINATE VALUES
C     HYPOLD  = HOLDING VARIABLE USED IN DETERMINING MAX Y ELEMENT
C               COORDINATE VALUES
C     HZPOLD  = HOLDING VARIABLE USED IN DETERMINING MAX Z ELEMENT
C               COORDINATE VALUES
C     J3      = Z-DEPTH SHELL SORT HOLDING LOCATION
C     J4      = Z-DEPTH SHELL SORT HOLDING LOCATION
C     JBUCKS  = DEFINES THE GRID INTO WHICH THE SCREEN WILL BE DIVIDED
C               FOR THE BUCKET SORT ON THE SURFACES
C     JCT     = NUMBER OF TIMES TO GO THROUGH THE OUTER LOOP OF THE SHELL
C               SORT TO GET ALL THE VECTORS IN A BUCKET SORTED IN Z DEPTH
C     JFLIP   = Z-DEPTH SHELL SORT FLAG
C     JLEN    = NUMBER OF ITEMS IN A BUCKET TO BE SORTED IN Z DEPTH
C     JMAX    = MAXIMUM DEPTH TO WHICH SORT IS PERFORMED IN JXYV OR JXYS
C     JOUT    = OUTPUT UNIT NUMBER
C     JSORTP  = Z-DEPTH SHELL SORT PARAMETER
C     JSPX    = LOCATION OF BUCKET IN WHICH SURFACE STOPS IN X
C     JSPY    = LOCATION OF BUCKET IN WHICH SURFACE STOPS IN Y
C     JSTX    = LOCATION OF BUCKET IN WHICH SURFACE STARTS IN X
C     JSTY    = LOCATION OF BUCKET IN WHICH SURFACE STARTS IN Y
C     JSURLS  = ELEMENT CONNECTIVITY ARRAY
C     JUMP    = Z-DEPTH SHELL SORT PARAMETER
C     JXYS    = ARRAY INTO WHICH ALL THE SURFACES ARE MAPPED BY BUCKET
C     JCHKS   = ERROR FLAG INDICATING THAT MAX NUMBER OF SURFACE ELEMENTS
C               PER BUCKET HAS BEEN EXCEEDED
C     LEN     = NUMBER OF ITEMS IN THE BUCKET BEING PROCESSED
C     LINSUR  = MAXIMUM NUMBER OF SURFACES THAT CAN BE IN ANY ONE SURFACE
C               BUCKET
C     N       = Z-DEPTH SHELL SORT INDEX
C     M       = Z-DEPTH SHELL SORT INDEX
C     NNODES  = ARRAY CONTAINING THE NUMBER OF NODES IN EACH ELEMENT
C     NSUR    = NUMBER OF SURFACE ELEMENTS IN THE MODEL
C     NUMS    = COUNTS THE NUMBER OF SURFACES IN A SURFACE BUCKET
C     PGRID   = COORDINATE ARRAY (INCLUDES CONTOUR COORDINATES IF
C               COMPUTED)
C     SURF    = CONTAINS MIN AND MAX X,Y,& Z COORDINATE VALUES SURFACE
C               AREA*2, AND PLANE EQUATION INFORMATION FOR EACH SURFACE
C     SXPOLD  = HOLDING VARIABLE USED IN DETERMINING MIN X ELEMENT
C               COORDINATE VALUES
C     SYPOLD  = HOLDING VARIABLE USED IN DETERMINING MIN Y ELEMENT
C               COORDINATE VALUES
C     SZPOLD  = HOLDING VARIABLE USED IN DETERMINING MIN Z ELEMENT
C               COORDINATE VALUES
C     XDELTS  = USED FOR SORTING SURFACES INTO THEIR PROPER BUCKETS
C               PROCESSED
C     XMIN    = MINIMUM X MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     XP      = ARRAY CONTAINING THE X COORDINATES OF EACH ELEMENT
C               VERTICE
C     YDELTS  = USED FOR SORTING SURFACES INTO THEIR PROPER BUCKETS
C     YMIN    = MINIMUM Y MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     YP      = ARRAY CONTAINING THE Y COORDINATES OF EACH ELEMENT
C               VERTICE
C     ZP      = ARRAY CONTAINING THE Z COORDINATES OF EACH ELEMENT
C               VERTICE
C
C***********************************************************************
C
```

```
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHSP
C
      include '/g/rt/work/thesis/working/dparam.h'
C
C     MAXNS  = MAXIMUM NUMBER OF NODES PER POLYGON NOT INCLUDING CENTER
C              NODE
C     MAXNJ  = MAXIMUM NUMBER OF COORDINATES (NODES)
C     MAXNPT = MAXIMUM NUMBER OF ELEMENTS
C     MXBSUR = MAXIMUM NUMBER OF SURFACES PER BUCKET IN JONESD HIDDEN
C
C**********************************************************************
      REAL             PGRID(3,MAXNJ),
     &                 SURF(11,MAXNPT),
     &                 XDELTS,YDELTS,
     &                 XMIN,YMIN,
     &                 XP(MAXNS,MAXNPT),
     &                 YP(MAXNS,MAXNPT),
     &                 ZP(MAXNS,MAXNPT),
     &                 AR(MAXNS,MAXNPT)
C
      INTEGER          NSUR,
     &                 JSURLS(MAXNS,MAXNPT),
     &                 NUMS(13,13),
     &                 JXYS(13,13,MXBSUR),
     &                 JBUCKS,LIMSUR,JCHKS,
     &                 JSORTP(8),
     &                 NNODES(MAXNPT)
C
      JOUT = 6
C
C                      ************************
C                      ****BEGIN PROCESSING****
C                      ************************
C
C----------------------------------------------------------------------
C
C************************** BEGIN SURFACE PREP ************************
C
C----------------------------------------------------------------------
C
      DO 1000 NPT=1,NSUR
C
C----------------------------------------------------------------------
C         ******CREATE THE ARRAYS XP, YP, & ZP WHICH CONTAIN THE
C               X, Y, & Z COORDINATES OF EACH OF THE VERTICIES
C               OF EACH ELEMENT******
C----------------------------------------------------------------------
C
        XP(1,NPT)=PGRID(1,JSURLS(1,NPT))
        YP(1,NPT)=PGRID(2,JSURLS(1,NPT))
        ZP(1,NPT)=PGRID(3,JSURLS(1,NPT))
c----------------------------------------------------------------------
c     I put the element X,Y, & Z coordinate values into XP,YP, & ZP
c     to facilitate the hradling of "n" sided elements. These arrays
c     will be used in area calculations, minmax tests, hidden line
c     determination, etc.  Just below we are initializing hold variables
c     to enable the determination of the max and min values in X,Y, & Z
c     for each element.
c----------------------------------------------------------------------
          NXPOLD=XP(1,NPT)
          SXPOLD=XP(1,NPT)
          NYPOLD=YP(1,NPT)
          SYPOLD=YP(1,NPT)
          NZPOLD=ZP(1,NPT)
          SZPOLD=ZP(1,NPT)
C
          DO 500 NS=2,NNODES(NPT)
C
            XP(NS,NPT)=PGRID(1,JSURLS(NS,NPT))
            YP(NS,NPT)=PGRID(2,JSURLS(NS,NPT))
            ZP(NS,NPT)=PGRID(3,JSURLS(NS,NPT))
C
C----------------------------------------------------------------------
C     ******FIND MAX AND MIN X, Y, & Z FOR ELEMENT NUMBER NPT******
C----------------------------------------------------------------------
C
            IF(XP(NS,NPT).GE.NXPOLD) NXPOLD=XP(NS,NPT)
            IF(XP(NS,NPT).LE.SXPOLD) SXPOLD=XP(NS,NPT)
C
            IF(YP(NS,NPT).GE.NYPOLD) NYPOLD=YP(NS,NPT)
            IF(YP(NS,NPT).LE.SYPOLD) SYPOLD=YP(NS,NPT)
C
            IF(ZP(NS,NPT).GE.NZPOLD) NZPOLD=ZP(NS,NPT)
            IF(ZP(NS,NPT).LE.SZPOLD) SZPOLD=ZP(NS,NPT)
C
  500   CONTINUE
C
c----------------------------------------------------------------------
c     Now we will load SURF 1-6 with the min and max X,Y, & Zcoordinate
c     values for this element.
c----------------------------------------------------------------------
          SURF(1,NPT)=NXPOLD
          SURF(2,NPT)=SXPOLD
          SURF(3,NPT)=NYPOLD
          SURF(4,NPT)=SYPOLD
          SURF(5,NPT)=NZPOLD
          SURF(6,NPT)=SZPOLD
C
C----------------------------------------------------------------------
C    ***BUCKET SORT (X,Y) ON SURFACES TO PRODUCE SURFACE MAP (JXYS)***
C----------------------------------------------------------------------
C
c----------------------------------------------------------------------
c     Now we will determine which bucket this surface starts in and
c     which one it stops in.  This will tell us which buckets it could
c     possibly be in and when we know this we will map this surface
c     into JXYS to speed hidden computation down the line; this mapping
c     is done in the 800 loop.
c----------------------------------------------------------------------
          JSTX=1+(SURF(2,NPT)-XMIN)*XDELTS
          JSPX=1+(SURF(1,NPT)-XMIN)*XDELTS
          JSTY=1+(SURF(4,NPT)-YMIN)*YDELTS
          JSPY=1+(SURF(3,NPT)-YMIN)*YDELTS
          DO 800 JJ=JSTX,JSPX
          DO 800 J=JSTY,JSPY
```

```
           NUMS(JJ,J)=NUMS(JJ,J)+1
           JXYS(JJ,J,NUMS(JJ,J))=NPT
  800   CONTINUE
C
C----------------------------------------------------------------------
C              ******SURFACE AREA CALCULATION******
C----------------------------------------------------------------------
C
        ARE=0.0
C----------------------------------------------------------------------
C     Immediately below we are computing the area of this element, by
C     taking cross products around the element vertices, to
C     determine if it can hide anything (an element with 0 area couldn't
C     hide much).  We are also creating the AR array which contains
C     element vertice cross products which will later be used in a
C     containment test.
C----------------------------------------------------------------------
        DO 700 NS=1,NNODES(NPT)
          IF(NS.EQ.NNODES(NPT)) THEN
            AR(NS,NPT)=(XP(NS,NPT)*YP(1,NPT)-XP(1,NPT)*YP(NS,NPT))
          ELSE
            AR(NS,NPT)=(XP(NS,NPT)*YP(NS+1,NPT)-XP(NS+1,NPT)*
     $                 YP(NS,NPT))
          END IF
          ARE=(ARE+AR(NS,NPT))
  700   CONTINUE
        AREA=ABS(ARE)
C
C----------------------------------------------------------------------
C     Setting SURF(7,NPT) to -1.0 if the area = 0 creates a flag that
C     will later be checked in the determination of line segment
C     visibility.
C----------------------------------------------------------------------
        IF(AREA.EQ.0.0) THEN
          SURF(7,NPT)=-1.0
        ELSE
          SURF(7,NPT)=1/AREA
        END IF
C----------------------------------------------------------------------
C     Setting SURF(10,NPT)=0.0 tells us that down the line when we have
C     to compute the plane equation for this element, that there is
C     some information that must be computed.  However it is only
C     necessary to compute this information once which is what happens
C     since a value of greater than 0 in SURF(10,NPT) says that the
C     information has already been computed.
C----------------------------------------------------------------------
        SURF(10,NPT)=0.0
C
 1000 CONTINUE
C
C----------------------------------------------------------------------
C       ******SHELL SORT BY DEPTH (Z) OF SURFACE MAP (JXYS)******
C----------------------------------------------------------------------
C
C----------------------------------------------------------------------
C     Below we are sorting all of the surface vectors in each bucket in
C     increasing Z-depth.  This is a sorting method developed by a
C     fellow named Shell and is therefore called a shell sort.
C----------------------------------------------------------------------
      kkkk=0
      DO 2000 JX=1,JBUCKS
      DO 2000 JY=1,JBUCKS
        JLEN=NUMS(JX,JY)
        IF(JLEN.GT.1) THEN
          CALL GVPRST(JLEN,JCT)
        END IF
        DO 1500 JINDEX=JCT,1,-1
          JUMP=JSORTP(JINDEX)
          JMAX=JLEN-JUMP
 1445     JFLIP=0
          DO 1450 M=1,JMAX
            kkkk=kkkk+1
            N=M+JUMP
            J8=JXYS(JX,JY,M)
            J4=JXYS(JX,JY,N)
            IF(SURF(8,J8).GT.SURF(8,J4)) THEN
              JXYS(JX,JY,M)=J4
              JXYS(JX,JY,N)=J8
              JFLIP=1
            END IF
 1450     CONTINUE
          IF(JFLIP.EQ.1) GO TO 1445
 1500   CONTINUE
 2000 CONTINUE
c      print*,'kkkk=',kkkk
C
C----------------------------------------------------------------------
C         ******CHECK FOR OVERFLOW IN SURFACE MAP, JXYS******
C----------------------------------------------------------------------
C
C----------------------------------------------------------------------
C     We have a limit on the number of surface elements that can be
C     in any one bucket which is currently 600.
C----------------------------------------------------------------------
      DO 3000 J=1,JBUCKS
      DO 3000 JJ=1,JBUCKS
        IF(NUMS(JJ,J).GT.LIMSUR) THEN
          WRITE(JOUT,6000) LIMSUR
          JCHKS = 1
          RETURN
        END IF
 3000 CONTINUE
C
 6000 FORMAT(/,1X,'OVERFLOW IN SURFACE MAP JXYS',8X,'LIMIT =',I4)
C
      RETURN
      END


      SUBROUTINE GVPRST(JLEN,JCT)
C
C**********************************************************************
C
C  SUBROUTINE GVPRST - COMPUTES SHELL SORT PARAMETER JCT FOR SHELL SORT
C
C**********************************************************************
```

```
C
C  SUBROUTINE CALLED BY
C     GVPHVP = PERFORMS VECTOR PREPARATION FOR HIDDEN LINE COMPUTATION
C
C*******************************************************************
C
C  SUBPROGRAMS CALLED
C     NONE
C
C*******************************************************************
C
C  VARIABLES USED
C     JCT     = NUMBER OF TIMES TO GO THROUGH THE OUTER LOOP OF THE SHELL
C               SORT TO GET ALL THE VECTORS IN A BUCKET SORTED IN Z DEPTH
C     JLEN    = NUMBER OF ITEMS IN A BUCKET TO BE SORTED IN Z DEPTH
C
C*******************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHST
C     NONE
C
C*******************************************************************
C
      INTEGER JLEN,JCT
C
C                        ************************
C                        ****BEGIN PROCESSING****
C                        ************************
C
      IF(JLEN.LE.6) THEN
        JCT=1
      ELSE IF(JLEN.LE.13) THEN
        JCT=2
      ELSE IF(JLEN.LE.29) THEN
        JCT=3
      ELSE IF(JLEN.LE.61) THEN
        JCT=4
      ELSE IF(JLEN.LE.125) THEN
        JCT=5
      ELSE IF(JLEN.LE.253) THEN
        JCT=6
      ELSE IF(JLEN.LE.510) THEN
        JCT=7
      ELSE
        JCT=8
      END IF
C
      RETURN
      END


      SUBROUTINE GVPHVP (NVEC,JREDUN,JVECLS,PGRID,JX1,JY1,JZ1,XDELTV,
     $                   YDELTV,XMIN,YMIN,JVTYPE,JSORTP,JBUCKV,LINVEC,
     $                   JVECVP,VECTOR,NUMV,JXYV,JVP,LCHKV)
C
C*******************************************************************
C
C  SUBROUTINE GVPHVP - PREPARES MODEL VECTOR INFORMATION FOR HIDDEN
C                         LINE COMPUTATION
C
C*******************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHID - PERFORMS MODIFIED JONES HIDDEN LINE PROCESSING
C
C*******************************************************************
C
C  SUBPROGRAMS CALLED
C     GVPHST - CALCULATES SHELL SORT PARAMETERS FOR Z-DEPTH SORTING
C
C*******************************************************************
C
C  VARIABLES USED
C     J3     = Z-DEPTH SHELL SORT HOLDING LOCATION
C     J4     = Z-DEPTH SHELL SORT HOLDING LOCATION
C     JBUCKV = DEFINES THE GRID INTO WHICH THE SCREEN WILL BE DIVIDED
C              FOR THE BUCKET SORT ON THE VECTORS
C     JCT    = NUMBER OF TIMES TO GO THROUGH THE OUTER LOOP OF THE SHELL
C              SORT TO GET ALL THE VECTORS IN A BUCKET SORTED IN Z DEPTH
C     JFLIP  = Z-DEPTH SHELL SORT FLAG
C     JINDEX = SHELL SORT PARAMETER
C     JLEN   = NUMBER OF ITEMS IN A BUCKET TO BE SORTED IN Z DEPTH
C     JMAX   = MAXIMUM DEPTH TO WHICH SORT IS PERFORMED IN JXYV OR JXYS
C     JOUT   = OUTPUT UNIT NUMBER
C     JREDUN = REDUNDANT EDGE FLAG
C                1 = NOT REDUNDANT
C                2+ = REDUNDANT
C     JSORTP = Z-DEPTH SHELL SORT PARAMETER
C     JSPX   = LOCATION OF BUCKET IN WHICH VECTOR STOPS IN X
C     JSPY   = LOCATION OF BUCKET IN WHICH VECTOR STOPS IN Y
C     JSTX   = LOCATION OF BUCKET IN WHICH VECTOR STARTS IN X
C     JSTY   = LOCATION OF BUCKET IN WHICH VECTOR STARTS IN Y
C     JUMP   = Z-DEPTH SHELL SORT PARAMETER
C     JVECLS = VECTOR CONNECTIVITY ARRAY (INCLUDES CONTOUR VECTORS IF
C              COMPUTED)
C     JVECVP = VECTOR VISIBILITY FLAG WHICH PREVENTS UNNECESSARY LINE
C              INTERSECTION CALCULATIONS
C     JVP    = ARRAY CONTAINING STARTING AND STOPPING BUCKETS FOR ALL
C              THE VECTORS, SURFACE OR LINE ELEMENT
C     JVTYPE = VECTOR TYPE FLAG
C                0 = LINE ELEMENT TYPE
C                1 = SURFACE VECTOR
C     JXYV   = ARRAY INTO WHICH ALL THE SURFACE VECTORS ARE MAPPED BY
C              BUCKET
C              BUCKET
C     LINVEC = MAXIMUM NUMBER OF SURFACE VECTORS PER BUCKET
C     M      = Z-DEPTH SHELL SORT INDEX
C     N      = Z-DEPTH SHELL SORT INDEX
C     N1     = NODE NUMBER OF ONE END OF VECTOR BEING PROCESSED
C     N2     = NODE NUMBER OF OTHER END OF VECTOR BEING PROCESSED
C     NUMV   = COUNTS THE NUMBER OF VECTORS IN A VECTOR BUCKET
C     NVEC   = NUMBER OF VECTORS IN THE MODEL (INCLUDES CONTOUR VECTORS
C              IF COMPUTED)
C     PGRID  = COORDINATE ARRAY (INCLUDES CONTOUR COORDINATES IF
C              COMPUTED)
C     VECTOR = HOLDS VECTOR INFORMATION SUCH AS COORDINATE ENDPOINTS,
```

```
c              MIN AND MAX X,Y,& Z COORDINATE VALUES AND VECTOR
c              INTERSECTION PARAMETERS
c     XDELTV = USED FOR SORTING VECTORS INTO THEIR PROPER BUCKETS
c     XMIN   = MINIMUM X MODEL COORDINATE FOR EFFECTIVE BUCKET SORT
c     YDELTV = USED FOR SORTING VECTORS INTO THEIR PROPER BUCKETS
c     YMIN   = MINIMUM Y MODEL COORDINATE FOR EFFECTIVE BUCKET SORT
c
C**********************************************************************
c
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHVP
c
      include '/g/rt/work/thesis/working/dparam.h'
c
c     MAXNJ  = MAXIMUM NUMBER OF COORDINATES (NODES)
c     MAXVEC = MAXIMUM NUMBER OF LINE SEGMENTS IN MODEL(INCLUDING
c              CONTOURS)
c     MXBVEC = MAXIMUM NUMBER OF VECTORS PER BUCKET IN JONESD HIDDEN
c
C**********************************************************************
c
c
c
      REAL              PGRID(3,MAXNJ),
     $                  XDELTV,YDELTV,
     $                  VECTOR(15,MAXVEC)
c
      INTEGER           NVEC,
     $                  JREDUN(MAXVEC),
     $                  JVECLS(2,MAXVEC),
     $                  JX1,JY1,JZ1,
     $                  JVTYPE(MAXVEC),
     $                  JBUCKV,LIMVEC,
     $                  NUMV(13,13),
     $                  JXYV(13,13,MXBVEC),
     $                  JVP(4,MAXVEC),
     $                  JVECVP(MAXVEC),
     $                  LCHKV,
     $                  JSORTP(8)
c
      JOUT=6
c
c                        ************************
c                        ****BEGIN PROCESSING****
c                        ************************
c
C----------------------------------------------------------------------
C
C*********************** BEGIN VECTOR (LINE) PREP *********************
C
C----------------------------------------------------------------------
C----------------------------------------------------------------------
c     To prepare vectors means to perform the calculations necessary to
c     determine which X-Y cells the vectors lie within so they can be
c     sorted by cell areas in X and Y.
c----------------------------------------------------------------------
      kount=0
      DO 1000 J=1,NVEC
         IF(JREDUN(J).GT.1) then
            GO TO 1000
         end if
         kount=kount+1
c----------------------------------------------------------------------
c     If JREDUN is greater than 1 the vector is a redundant edge and
c     therefore has already been processed or will be processed later.
c----------------------------------------------------------------------
         JVECVP(J)=1
         N1=JVECLS(1,J)
         N2=JVECLS(2,J)
c----------------------------------------------------------------------
c     N1 and N2 represent the node numbers at the ends of the vector
c     being operated on.  JVECLS then contains the vector conectivity.
c----------------------------------------------------------------------
         VECTOR(1,J)=PGRID(1,N1)
         VECTOR(2,J)=PGRID(2,N1)
         VECTOR(3,J)=PGRID(3,N1)
         VECTOR(4,J)=PGRID(1,N2)
         VECTOR(5,J)=PGRID(2,N2)
         VECTOR(6,J)=PGRID(3,N2)
c----------------------------------------------------------------------
c     VECTOR 1-6 contain the coordinate endpoints(XYZ) of the vector
c     being operated on.
c----------------------------------------------------------------------
         VECTOR(7,J)=VECTOR(5,J)-VECTOR(2,J)
         VECTOR(8,J)=VECTOR(1,J)-VECTOR(4,J)
         VECTOR(9,J)=-VECTOR(7,J)*VECTOR(4,J)-VECTOR(8,J)*
     9              VECTOR(5,J)
c----------------------------------------------------------------------
c     VECTOR 7-9 contain information used in the computation of the line
c     equation for each vector which will be used in the computation of
c     vector intersection points.
c----------------------------------------------------------------------
         VECTOR(10,J)=MAX(VECTOR(1,J),VECTOR(4,J))
         VECTOR(11,J)=MAX(VECTOR(2,J),VECTOR(5,J))
         VECTOR(12,J)=MAX(VECTOR(3,J),VECTOR(6,J))
         VECTOR(13,J)=MIN(VECTOR(1,J),VECTOR(4,J))
         VECTOR(14,J)=MIN(VECTOR(2,J),VECTOR(5,J))
         VECTOR(15,J)=MIN(VECTOR(3,J),VECTOR(6,J))
c----------------------------------------------------------------------
c     VECTOR 10-15 contain the maximum and minimum X,Y, and Z values for
c     the vector being processed.  These will be used to locate the
c     starting and ending points for each vector processed (for bucket
c     sort) and to do Z depth comparisons for computing intersections in
c     GHIDDN.
c----------------------------------------------------------------------
C
C----------------------------------------------------------------------
C    ***BUCKET SORT (X,Y) ON VECTORS TO PRODUCE VECTOR MAP (JXYV)***
C----------------------------------------------------------------------
C
         JSTX=1+(VECTOR(13,J)-XMIN)*XDELTV
         JSPX=1+(VECTOR(10,J)-XMIN)*XDELTV
         JSTY=1+(VECTOR(14,J)-YMIN)*YDELTV
         JSPY=1+(VECTOR(11,J)-YMIN)*YDELTV
c----------------------------------------------------------------------
c     JSTX stands for "I START IN X" and JSPX stands for "I STOP IN X".
c     These values have reference to which cell a vector starts or stops
```

```
c     in and are therefore used to map the starting and ending points of
c     "surface" vectors onto an XY grid; this grid information is
c     contained in JXYV. These grid areas are the buckets into which the
c     vectors are sorted.
c-----------------------------------------------------------------------
          IF(JVTYPE(J).GE.1) THEN
            DO 500 KK=JSTX,JSPX
            DO 500 K=JSTY,JSPY
              NUMV(KK,K)=NUMV(KK,K)+1
              JXYV(KK,K,NUMV(KK,K))=J
 500        CONTINUE
          END IF
c-----------------------------------------------------------------------
c     In the above code all vectors except those that are soley line
c     element types are mapped into an XY vector map according to there
c     cell starting and stoping points as determined in JSTX, JSPX,
c     JSTY, and JSPY.  We know the vector belongs to a surface if JVTYPE
c     is greater than or equal to 1.
c-----------------------------------------------------------------------
          JVP(1,J)=JSTX
          JVP(2,J)=JSPX
          JVP(3,J)=JSTY
          JVP(4,J)=JSPY
c-----------------------------------------------------------------------
c     The JVP array contains the starting and stopping locations of all
c     vectors processed regardless of type.  This array is used to
c     compare all of the vectors to the surface vectors (bucket by
c     bucket as required) so that intersections can be computed in
c     HIDDEN.  There is no need to compare line element types to line
c     element types since they cannot hide each other.  This is why only
c     the surface vectors were mapped into the JXYV mapping array above.
c-----------------------------------------------------------------------
 1000 CONTINUE
      write(6,*) 'THE NUMBER OF VECTORS PROCESSED = ',kount
C
C-----------------------------------------------------------------------
C       ******SHELL SHORT BY DEPTH (Z) OF VECTOR MAP (JXYV)******
C-----------------------------------------------------------------------
C
      DO 2000 JX=1,JBUCKV
      DO 2000 JY=1,JBUCKV
        JLEN=NUMV(JX,JY)
        IF(JLEN.GT.1) THEN
          CALL GVPRST(JLEN,JCT)
        END IF
        DO 1500 JINDEX=JCT,1,-1
          JUMP=JSORTP(JINDEX)
          JMAX=JLEN-JUMP
 1445     JFLIP=0
          DO 1450 M=1,JMAX
            N=M+JUMP
            J3=JXYV(JX,JY,M)
            J4=JXYV(JX,JY,N)
            IF(VECTOR(15,J3).GT.VECTOR(15,J4)) THEN
              JXYV(JX,JY,M)=J4
              JXYV(JX,JY,N)=J3
              JFLIP=1
            END IF
 1450     CONTINUE
          IF(JFLIP.EQ.1) GO TO 1445
 1500   CONTINUE
 2000 CONTINUE
c-----------------------------------------------------------------------
c     In the above, a cell by cell sort in the Z direction is
c     accomplished. Given the number of vectors that could lie inside a
c     cell (JLEN), the GHSORT routine is called and according to JLEN
c     returns a value for JCT; JCT is the number of times we must go
c     through the 1500 loop to get all the vectors in a cell sorted in
c     depth Z.  The minimum Z values between two vectors are compared
c     and if necessary the vectors are flipped in order to get the
c     vectors in increasing Z depth order; if there were three vectors
c     in cell (1,1), then the closest vector in Z would be flagged as
c     JXYV(1,1,1), the middle one as JXYV(1,1,2) and the far one as
c     JXYV(1,1,3).  The M=1,JMAX loop is passed through as many times as
c     vectors are flipped in order to get the correct Z depth sort.  The
c     JSORTP values used in defining JUMP are defined in a data
c     statement.  For more information on this sorting procedure read
c     about the sorting method developed by Shell.
c-----------------------------------------------------------------------
C
C-----------------------------------------------------------------------
C          ******CHECK FOR OVERFLOW IN VECTOR MAP, JXYV******
C-----------------------------------------------------------------------
C
      DO 3000 J=1,JBUCKV
      DO 3000 JJ=1,JBUCKV
        IF(NUMV(JJ,J).GT.LINVEC) THEN
          WRITE(JOUT,5000) LINVEC
          LCHKV = 1
          RETURN
        END IF
c-----------------------------------------------------------------------
c     In the above a check is made to insure that the maximum allowed
c     number of vectors per cell is not exceeded;  the maximum allowable
c     is defined in a parameter statement as MXNVEC and in this version
c     equals 600.
c-----------------------------------------------------------------------
 3000 CONTINUE
C
 5000 FORMAT(/,1X,'OVERFLOW IN VECTOR MAP JXYV',8X,'LIMIT =',I4)
C
      RETURN
      END
```

APPENDIX B

Contour Subsystem Code

```
      SUBROUTINE GVPHCO (CONT,LINES,JMM,JCSEG)
C
C**********************************************************************
C
C  SUBORUTINE GVPHCO - ORDERS CONTOUR VECTORS INTO CONTINUOUS STRINGS
C
C**********************************************************************
C
C  SUBROUTINE CALLED BY
C      GVPHCT - GENERATES CONTOUR VECTORS
C
C**********************************************************************
C
C  SUBPROGRAMS CALLED
C      NONE
C
C**********************************************************************
C
C  VARIABLES USED
C      CONT   = CONTAINS VECTOR ENDPOINT COORDINATES BY CONTOUR LEVEL
C      CONX1  = HOLDS X COORDINATE OF THE #1 END OF THE VECTOR FOR WHICH
C               A CONNECTING VECTOR IS BEING SOUGHT
C      CONX2  = HOLDS X COORDINATE OF THE #2 END OF THE VECTOR FOR WHICH
C               A CONNECTING VECTOR IS BEING SOUGHT
C      CONY1  = HOLDS Y COORDINATE OF THE #1 END OF THE VECTOR FOR WHICH
C               A CONNECTING VECTOR IS BEING SOUGHT
C      CONY2  = HOLDS Y COORDINATE OF THE #2 END OF THE VECTOR FOR WHICH
C               A CONNECTING VECTOR IS BEING SOUGHT
C      CONZ1  = HOLDS Z COORDINATE OF THE #1 END OF THE VECTOR FOR WHICH
C               A CONNECTING VECTOR IS BEING SOUGHT
C      CONZ2  = HOLDS Z COORDINATE OF THE #2 END OF THE VECTOR FOR WHICH
C               A CONNECTING VECTOR IS BEING SOUGHT
C      HOLDX1 = HOLDS X COORDINATE OF #1 END OF CONTOUR VECTOR FOR
C               PURPOSES OF REVERSING THE ORDER OF A STRING OF
C               CONTOUR VECTORS
C      HOLDX2 = HOLDS X COORDINATE OF #2 END OF CONTOUR VECTOR FOR
C               PURPOSES OF REVERSING THE ORDER OF A STRING OF
C               CONTOUR VECTORS
C      HOLDY1 = HOLDS Y COORDINATE OF #1 END OF CONTOUR VECTOR FOR
C               PURPOSES OF REVERSING THE ORDER OF A STRING OF
C               CONTOUR VECTORS
C      HOLDY2 = HOLDS Y COORDINATE OF #2 END OF CONTOUR VECTOR FOR
C               PURPOSES OF REVERSING THE ORDER OF A STRING OF
C               CONTOUR VECTORS
C      HOLDZ1 = HOLDS Z COORDINATE OF #1 END OF CONTOUR VECTOR FOR
C               PURPOSES OF REVERSING THE ORDER OF A STRING OF
C               CONTOUR VECTORS
C      HOLDZ2 = HOLDS Z COORDINATE OF #2 END OF CONTOUR VECTOR FOR
C               PURPOSES OF REVERSING THE ORDER OF A STRING OF
C               CONTOUR VECTORS
C      JCSEG  = CONTAINS THE NUMBER OF DIFFERENT CONTOUR STRINGS IN EACH
C               OF THE CONTOUR LEVELS AND THE NUMBER OF LINE SEGMENTS IN
C               EACH STRING BEFORE HIDDEN LINE REMOVAL
C      JMM    = CONTAINS THE NUMBER OF CONTOUR VECTORS GENERATED AT EACH
C               CONTOUR LEVEL
C      KKOUNT = CONTOUR ORDERING COUNTER THAT KEEPS TRACK OF THE VALUE OF
C               KOUNT AT THE TIME THE LAST STRING IN A PARTICULAR CONTOUR
C               LEVEL WAS COMPLETED
C      KOUNT  = CONTOUR VECTOR ORDERING COUNTER THAT COUNTS THE NUMBER OF
C               VECTORS ORDERED AT A PARTICULAR CONTOUR LEVEL
C      L2TIME = CONTOUR VECTOR ORDERING LOGICAL
C               .TRUE.  = CONTOUR STRING HAS BEEN REVERSED IN ORDER
C               .FALSE. = CONTOUR STRING NOT REVERSED IN ORDER
C      LCON   = CONTOUR VECTOR ORDERING LOGICAL
C               .TRUE.  = CONTOUR STRING RUNS BOTH WAYS FROM
C                         'STARTING VECTOR'
C               .FALSE. = CONTOUR STRING RUNS ONE WAY FROM
C                         'STARTING VECTOR'
C      LINES  = CONTAINS NUMBER OF DIFFERENT CONTOUR LEVELS
C      LPCHK  = CONTOUR VECTOR ORDERING LOGICAL
C               .TRUE.  = CONTOUR STRING CLOSED OF ITSELF
C               .FALSE. = CONTOUR STRING DOESN'T CLOSE
C      NUMCON = CONTOUR ORDERING COUNTER THAT KEEPS TRACK OF THE NUMBER
C               OF SEPARATE CONTOUR STRINGS FOR A PARTICULAR CONTOUR
C               LEVEL.
C      XLOOP  = HOLDS X COORDINATE OF #1 END OF 'STARTING VECTOR' TO
C               CHECK IF A CONTOUR STRING CLOSED ON ITSELF
C      YLOOP  = HOLDS Y COORDINATE OF #1 END OF 'STARTING VECTOR' TO
C               CHECK IF A CONTOUR STRING CLOSED ON ITSELF
C      ZLOOP  = HOLDS Z COORDINATE OF #1 END OF 'STARTING VECTOR' TO
C               CHECK IF A CONTOUR STRING CLOSED ON ITSELF
C
C**********************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHCO
C
      include '/g/rt/work/thesis/working/dparam.h'
C
C      MXCONT = MAXIMUM NUMBER OF CONTOUR LEVELS
C      MXCLEV = MAXIMUM NUMBER OF SEPARATE CONTOUR STRINGS AT THE SAME
C               CONTOUR LEVEL
C      MXCSEG = MAXIMUM NUMBER OF CONTOUR VECTORS IN A CONTOUR LEVEL
C
C**********************************************************************
C
      INTEGER          LINES,
     $                 JMM(MXCONT),
     $                 JCSEG(MXCLEV,MXCONT)
      REAL             CONT(6,MXCONT,MXCSEG)
C
      LOGICAL          LCON,L2TIME,LPCHK
C
C----------------------------------------------------------------------
C                    ******INITIALIZATION******
C----------------------------------------------------------------------
C
      DO 1000 J=1,MXCONT
      DO 1000 JJ=1,MXCLEV
        JCSEG(JJ,J)=0
 1000 CONTINUE
C
C                    ***********************
C                    ****BEGIN PROCESSING****
C                    ***********************
C
C----------------------------------------------------------------------
```

```
c     This program creates an ordered list of all contours by
c     connectivity and by contour value.  This will allow the removal of
c     duplicate nodes created in the contour algorithm and will also
c     allow the flagging of connecting contours for the purpose of later
c     fitting spline curves through contour points. It will also allow
c     the accurate labeling of contours.
c-----------------------------------------------------------------------
      DO 3000 J=1,LINES
c-----------------------------------------------------------------------
c     Set counters and logicals to keep track of how many contour
c     vectors of a certain value are connected together.  Also set the
c     holding variables used to check for contour strings that loop back
c     on themselves.
c-----------------------------------------------------------------------
        KOUNT = 1
        KKOUNT = 0
        NUMCON = 0
        LCON = .FALSE.
        L2TIME = .FALSE.
        LPCHK = .FALSE.
        XLOOP = CONT(1,J,1)
        YLOOP = CONT(2,J,1)
        ZLOOP = CONT(3,J,1)
C
        DO 2500 JJ=1,JMM(J)
c-----------------------------------------------------------------------
c     Start looping through the vector coordinate endpoint values to
c     decide how contour vectors go together.
c-----------------------------------------------------------------------
          CONX1=CONT(1,J,JJ)
          CONY1=CONT(2,J,JJ)
          CONZ1=CONT(3,J,JJ)
          CONX2=CONT(4,J,JJ)
          CONY2=CONT(5,J,JJ)
          CONZ2=CONT(6,J,JJ)
c-----------------------------------------------------------------------
c     The following IF checks to see if a string of contours at a
c     contour level has looped back on itself.  If this is the case
c     and there are still vectors to be processed at this contour
c     level, then it is important that we set up a way to check if
c     another string at this same level loops back on itself. This
c     check is accomplished by setting LPCHK to true if it is
c     discovered that a contour does loop back on itself; the
c     setting of LPCHK  is done later in this code at the point
c     where it is possible to make the check.
c-----------------------------------------------------------------------
          IF(LPCHK) THEN
            XLOOP=CONX1
            YLOOP=CONY1
            ZLOOP=CONZ1
            LPCHK=.FALSE.
          END IF
C
          IF(.NOT.LCON) THEN
            DO 2450 JJJ=JJ+1,JMM(J)
              IF((ABS(CONX1-CONT(1,J,JJJ)).LE.(.001)
     $          .AND.ABS(CONY1-CONT(2,J,JJJ)).LE.(.001)
     $          .AND.ABS(CONZ1-CONT(3,J,JJJ)).LE.(.001))
     $          .OR.(ABS(CONX1-CONT(4,J,JJJ)).LE.(.001)
     $          .AND.ABS(CONY1-CONT(5,J,JJJ)).LE.(.001)
     $          .AND.ABS(CONZ1-CONT(6,J,JJJ)).LE.(.001)))
     $          THEN
C-----------------------------------------------------------------------
C     We am setting the logical LCON to true if we find that there
C     are contour vectors that connect to both ends of the starting
C     or initial vector.  If vectors connect to both ends of the initial
C     contour vector then we will process the connectivity starting at
C     the #2 end of the vector first  (#1 ------------ #2).  When we
C     come to the end of possible vectors connecting to the #2 end,
C     we will then start at the #1 end of the initial contour
C     vector and find the vectors that connect to it.
C-----------------------------------------------------------------------
                LCON=.TRUE.
                GO TO 2455
              END IF
 2450       CONTINUE
          END IF
 2455     CONTINUE
c-----------------------------------------------------------------------
c     Now we am starting to find, and connect in order, all of the
c     contour vectors that propogate from the #2 end of the initial
c     vector.
c-----------------------------------------------------------------------
          DO 2460 JJJ=JJ+1,JMM(J)
            IF((ABS(CONX2-CONT(1,J,JJJ)).LE.(.001)
     $        .AND.ABS(CONY2-CONT(2,J,JJJ)).LE.(.001)
     $        .AND.ABS(CONZ2-CONT(3,J,JJJ)).LE.(.001))) THEN
c-----------------------------------------------------------------------
c     If the preceding "IF" is satisfied then we have found a vector
c     that connects to the initial vector. If this is the first time
c     through, or we have found a vector that connects to the "n"th
c     vector down the line from the initial
c     vector if this is the "n"th time through the loop.  Now,
c     having found this vector, we will swap its endpoint locations
c     with the vector presently in the "n+1" vector location. That is,
c     #1 ----- #2, #3 ----- #4, #5 ----- #4, #2 ----- #3 will become
c     #1 ----- #2, #2 ----- #3, #5 ----- #4, #3 ----- #4 the first
c     time through. The second time through it will become
c     #1 ----- #2, #2 ----- #3, #3 ----- #4, #5 ----- #4 and finally
c     the last time through we will jump to the "ELSE IF" below
c     which will switch the end point locations of #5 ----- #4 to
c     #4 ----- #5 and at the same time will produce the following
c     #1 ----- #2, #2 ----- #3, #3 ----- #4, #4 ----- #5.
c-----------------------------------------------------------------------
              HOLDX1=CONT(1,J,(JJ+1))
              HOLDY1=CONT(2,J,(JJ+1))
              HOLDZ1=CONT(3,J,(JJ+1))
              HOLDX2=CONT(4,J,(JJ+1))
              HOLDY2=CONT(5,J,(JJ+1))
              HOLDZ2=CONT(6,J,(JJ+1))
              CONT(1,J,(JJ+1))=CONT(1,J,JJJ)
              CONT(2,J,(JJ+1))=CONT(2,J,JJJ)
              CONT(3,J,(JJ+1))=CONT(3,J,JJJ)
              CONT(4,J,(JJ+1))=CONT(4,J,JJJ)
              CONT(5,J,(JJ+1))=CONT(5,J,JJJ)
              CONT(6,J,(JJ+1))=CONT(6,J,JJJ)
```

```
          CONT(1,J,JJJ)=HOLDX1
          CONT(2,J,JJJ)=HOLDY1
          CONT(3,J,JJJ)=HOLDZ1
          CONT(4,J,JJJ)=HOLDX2
          CONT(5,J,JJJ)=HOLDY2
          CONT(3,J,JJJ)=HOLDZ2
          KOUNT = KOUNT +1
          GO TO 2500
c-----------------------------------------------------------------
c     As long as we continue to find connecting vectors we will jump
c     to 2500 because we do not want to continue past this point until
c     we come to the end of a line of contours.  The reasoning for this
c     will be discussed below.
c-----------------------------------------------------------------
         ELSE IF((ABS(CONX2-CONT(4,J,JJJ)).LE.(.001)
     $    .AND.ABS(CONT2-CONT(5,J,JJJ)).LE.(.001)
     $    .AND.ABS(CONZ2-CONT(6,J,JJJ)).LE.(.001))) THEN
          HOLDX1=CONT(1,J,(JJ+1))
          HOLDY1=CONT(2,J,(JJ+1))
          HOLDZ1=CONT(3,J,(JJ+1))
          HOLDX2=CONT(4,J,(JJ+1))
          HOLDY2=CONT(5,J,(JJ+1))
          HOLDZ2=CONT(6,J,(JJ+1))
          CONT(1,J,(JJ+1))=CONT(4,J,JJJ)
          CONT(2,J,(JJ+1))=CONT(5,J,JJJ)
          CONT(3,J,(JJ+1))=CONT(6,J,JJJ)
          CONT(4,J,(JJ+1))=CONT(1,J,JJJ)
          CONT(5,J,(JJ+1))=CONT(2,J,JJJ)
          CONT(6,J,(JJ+1))=CONT(3,J,JJJ)
          CONT(1,J,JJJ)=HOLDX2
          CONT(2,J,JJJ)=HOLDY2
          CONT(3,J,JJJ)=HOLDZ2
          CONT(4,J,JJJ)=HOLDX1
          CONT(5,J,JJJ)=HOLDY1
          CONT(6,J,JJJ)=HOLDZ1
          KOUNT = KOUNT +1
          GO TO 2500
         END IF
 2460   CONTINUE
c-----------------------------------------------------------------
c     If there are contour vectors that connect to the #1 (#1 ----- #2)
c     end of the initaial vector picked, then we take the string of
c     vectors that we have got so far and switch them end for end so
c     that all the vectors of this contour, in this string, can be in
c     order starting at one end and going to the other.  That is,
c     #1 ----- #2 ----- #3 ----- #4 ----- #5 ----- #6 becomes
c     #6 ----- #5 ----- #4 ----- #3 ----- #2 ----- #1 so that when we
c     add the 7th vector to the string it will attach to #6 instead of
c     #1.  If LCON is true below, then the initial contour vector does
c     have a vector connected to its #1 end.  If L2TIME is false then
c     we know that this is the first time we have come to this location
c     for this particular string of vectors.  Consequently if LCON is
c     true and L2TIME is false we have to reorder the vectors processed
c     thus far ,end for end, in preparation for adding on the vectors
c     which propogate from the #1 end of the initial vector.  However,
c     if L2TIME is true, then we know that we have already back sorted
c     the vectors propogating from the #2 end of the initial vector and
c     don't need to hack sort again; this is because there can, at most,
c     be two end points for a contour string.
c-----------------------------------------------------------------
            IF(LCON.AND.(.NOT.L2TIME)) THEN
             IF(JJ.EQ.JMM(J)) THEN
c-----------------------------------------------------------------
c     We have finished with this contour level so lets go to the next
c     one after we record how many vectors were in the last loop at this
c     level.
c-----------------------------------------------------------------
               NUMCON=NUMCON+1
               JCSEG(NUMCON,J)=(KOUNT-KKOUNT)
               GO TO 8000
             END IF
C
             IF(ABS(XLOOP-CONT(4,J,KOUNT)).LE.(.001).AND.
     $        ABS(YLOOP-CONT(5,J,KOUNT)).LE.(.001).AND.
     $        ABS(ZLOOP-CONT(6,J,KOUNT)).LE.(.001)) THEN
c-----------------------------------------------------------------
c     If we find that this loop of contour vectors closed on itself
c     then we will set LPCHK to true since the next time through
c     we want to save the coordinates of the #1 end of the next
c     contour vector (XLOOP, YLOOP, XLOOP) just in case the next
c     loop also closes on itself.  LCON is set to false since we
c     know that the next vector processed does not connect to the
c     last one processed.  NUMCON (counts the number of loops in a
c     particular contour level) is incremented by 1 since we want to
c     know how many vectors are in the loop that just ended which
c     number is stored in JCSEG.  KKOUNT is set equal to KOUNT
c     for possible sorting requirements later.  KOUNT is then
c     incremented by 1 because the next vector to be processed belongs
c     to a new loop.
c-----------------------------------------------------------------
               LPCHK=.TRUE.
               LCON=.FALSE.
               NUMCON=NUMCON+1
               JCSEG(NUMCON,J)=(KOUNT-KKOUNT)
               KKOUNT=KOUNT
               KOUNT=KOUNT+1
               GO TO 2500
             END IF
c-----------------------------------------------------------------
c     If we get to this point then we know that we must reverse
c     the segments already processed in this loop since there are
c     more vectors that connect to this loop going the other way
c     from the starting vector.  The magic in the determination of
c     KK and KKK below is required due to the possibility that more
c     than one loop of contours may exist at any one contour level.
c     Therefore, if there has already been a loop processed at this
c     level, we want to leave it alone in its array location (that
c     is we don't want to start reversing what is already nicely
c     sorted.  So we start our reversing of coordinate locations
c     from the beginning of the loop presently being processed which
c     happens to be KKOUNT+1.  We go to a point half way through the
c     list of contours we have already sorted in this loop(XMM), and by
c     doing so we flip end for end the connectivity locations of the
c     vectors processed in this loop.  If you don't believe me, test
c     this out for yourself and you will see that it always works.
c-----------------------------------------------------------------
```

```
      KK=KKOUNT+1
      KKK=KKOUNT+(((KOUNT-KKOUNT)+1)/2)
      LL=0
      DO 2470 JJJ=KK,KKK
        HOLDX1=CONT(1,J,JJJ)
        HOLDY1=CONT(2,J,JJJ)
        HOLDZ1=CONT(3,J,JJJ)
        HOLDX2=CONT(4,J,JJJ)
        HOLDY2=CONT(5,J,JJJ)
        HOLDZ2=CONT(6,J,JJJ)
        CONT(1,J,JJJ)=CONT(4,J,(KOUNT-LL))
        CONT(2,J,JJJ)=CONT(5,J,(KOUNT-LL))
        CONT(3,J,JJJ)=CONT(6,J,(KOUNT-LL))
        CONT(4,J,JJJ)=CONT(1,J,(KOUNT-LL))
        CONT(5,J,JJJ)=CONT(2,J,(KOUNT-LL))
        CONT(6,J,JJJ)=CONT(3,J,(KOUNT-LL))
        CONT(1,J,(KOUNT-LL))=HOLDX2
        CONT(2,J,(KOUNT-LL))=HOLDY2
        CONT(3,J,(KOUNT-LL))=HOLDZ2
        CONT(4,J,(KOUNT-LL))=HOLDX1
        CONT(5,J,(KOUNT-LL))=HOLDY1
        CONT(6,J,(KOUNT-LL))=HOLDZ1
        LL=LL+1
 2470 CONTINUE
      L2TIME=.TRUE.
c-----------------------------------------------------------------
c     We set the counter(JJ) for the 2500 loop back one because we want
c     to repeat the 2500 loop for the vector in the present JJ vector
c     location; this is because the vector in the present JJ location
c     is now (after the order reversal above) the initial vector with
c     its old #1 end now being its new #2 end.  we know that there is
c     a vector that connects to the new #2 end of the JJ (initial)
c     vector from the fact that we ever made to the place where the
c     vector locations were reversed.
c-----------------------------------------------------------------
          JJ=JJ-1
        ELSE
c-----------------------------------------------------------------
c     Now, if we get to this "ELSE", we know that we have got a nicely
c     ordered set of contour vectors that start at one end and go to the
c     other.  We still however may have one or more other separate
c     strings of contours at this function value, and by setting both
c     LCON and L2TIME to false we can go through the process just
c     discussed and obtain any number of separate contour strings at
c     this value.  This is because setting LCON to false will result in
c     a new initial vector being selected just below the DO 2500
c     statement, and this will be one of the vectors remaining in the
c     list of possible vectors at this contour value.  We should point
c     out that every time a connectivity is discovered it is tossed out
c     of the list of possible vectors to sort through.
c-----------------------------------------------------------------
          LCON=.FALSE.
          L2TIME=.FALSE.
c-----------------------------------------------------------------
c     NUMCON counts the number of vectors in a string.
c     JCSEG contains the number of vectors in each separate string
c     at each of the different contour values.
c     KOUNT contains the number of vectors ordered to this point in
c     time for a particular contour value.
c     KKOUNT contains the number of vectors that had been processed
c     for a particular contour value the last time we passed this way.
c-----------------------------------------------------------------
            NUMCON=NUMCON+1
            JCSEG(NUMCON,J)=(KOUNT-KKOUNT)
            KKOUNT=KOUNT
            KOUNT=KOUNT+1
            LPCHK=.TRUE.
          END IF
C
C
 2500 CONTINUE
c-----------------------------------------------------------------
c     Now we have ordered all the vectors in the above contour level and
c     will go on to the next one.
c-----------------------------------------------------------------
 3000 CONTINUE
C
      RETURN
      END


      SUBROUTINE GVPHCP (LINES,NUMCON,JMM,CUMMOD,CONT,PGRID,NVEC,
     &                   JCSEG,JVECLS)
C
C*****************************************************************
C
C  SUBORUTINE GVPHCP - PACKS CONTOUR COORDINATES AND CONNECTIVITY INTO
C                      PGRID AND JVECLS
C
C*****************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHCT - GENERATES CONTOUR VECTORS
C
C*****************************************************************
C
C  SUBPROGRAMS CALLED
C     NONE
C
C*****************************************************************
C
C  VARIABLES USED
C     CONT   = CONTAINS VECTOR ENDPOINT COORDINATES BY CONTOUR LEVEL
C     JCSEG  = CONTAINS THE NUMBER OF DIFFERENT CONTOUR STRINGS IN EACH
C              OF THE CONTOUR LEVELS AND THE NUMBER OF LINE SEGMENTS IN
C              EACH STRING BEFORE HIDDEN LINE REMOVAL
C     JMM    = CONTAINS THE NUMBER OF CONTOUR VECTORS GENERATED AT EACH
C              CONTOUR LEVEL
C     JVECLS = VECTOR CONNECTIVITY ARRAY (INCLUDES CONTOUR VECTORS IF
C              COMPUTED)
C     KCON   = COUNTER USED IN LOADING CONTOUR VALUES INTO PGRID AND
C              JVECLS
C     LCONEC = LOGICAL USED IN THE LOADING OF JVECLS AND PGRID WITH
C              CONTOUR VALUES
C     LINES  = CONTAINS NUMBER OF DIFFERENT CONTOUR LEVELS
C     NUMCON = CONTOUR ORDERING COUNTER THAT KEEPS TRACK OF THE NUMBER
```

```
c              OF SEPARATE CONTOUR STRINGS FOR A PARTICULAR CONTOUR
c              LEVEL.
c     NUMNOD = NUMBER OF NODES IN PGRID
c     NVEC   = NUMBER OF VECTORS IN THE MODEL (INCLUDES CONTOUR VECTORS
c              IF COMPUTED)
c     PGRID  = COORDINATE ARRAY (INCLUDES CONTOUR COORDINATES IF
c              COMPUTED)
c
C**********************************************************************
c
c  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHCP
c
      include '/g/rt/work/thesis/working/dparam.h'
c
c     MAXNJ  = MAXIMUM NUMBER OF COORDINATES (NODES)
c     MAXVEC = MAXIMUM NUMBER OF LINE SEGMENTS IN MODEL
c     MXCONT = MAXIMUM NUMBER OF CONTOUR LEVELS
c     MXCLEV = MAXIMUM NUMBER OF SEPARATE CONTOUR STRINGS AT THE SAME
c              CONTOUR LEVEL
c     MXCSEG = MAXIMUM NUMBER OF CONTOUR VECTORS IN A CONTOUR LEVEL
c
C**********************************************************************
c
      INTEGER LINES,NUMCON,NUMNOD,NVEC,
     $        JMM(MXCONT),
     $        JVECLS(2,MAXVEC),
     $        JCSEG(MXCLEV,MXCONT),
     $        KCON
c
      REAL    CONT(6,MXCONT,MXCSEG),
     $        PGRID(3,MAXNJ)
C
      LOGICAL LCONEC
c
c                         ************************
c                         ****BEGIN PROCESSING****
c                         ************************
c
      DO 2000 J=1,LINES
c---------------------------------------------------------------------
c     NUMCON is an index into the JCSEG array and its function is to
c     separate distinct contour loops which share the same contour
c     value.  KCON is a counter which keeps track of how many vectors
c     we have processed at a prticular contour level.  LCONEC is a
c     logical which allows the separation of distinct contour loops
c     at the same level (ie. LCONEC insures that the end of the last
c     vector in loop 1 at level 1 will not connect to the first vector
c     in loop 2 at level 1.
c---------------------------------------------------------------------
        NUMCON=1
        KCON=0
        LCONEC=.FALSE.
        DO 1500 JJ=1,JMM(J)
          KCON=KCON+1
          IF(.NOT.LCONEC) THEN
c---------------------------------------------------------------------
c     If the last vector processed should not connect to the next one
c     being processed then LCONEC will be .false.  LCONEC is also
c     false when this loop is first entered since we must start with
c     the #1 end of the first vector in the contour level being
c     processed.
c---------------------------------------------------------------------
            NUMNOD = NUMNOD+1
            PGRID(1,NUMNOD)=CONT(1,J,JJ)
            PGRID(2,NUMNOD)=CONT(2,J,JJ)
            PGRID(3,NUMNOD)=CONT(3,J,JJ)
            NVEC = NVEC+1
            JVECLS(1,NVEC)=NUMNOD
          END IF
C
          NUMNOD = NUMNOD+1
c---------------------------------------------------------------------
c     If it turns out that my vectors are connecting to each other
c     then I am deleting the common nodes between vectors.  This is
c     evident in the fact that I skip the 1, 2, & 3 locations in the
c     CONT array.  I can do this because of the way I have ordered the
c     contour vectors in the GHCORD routine.
c---------------------------------------------------------------------
          PGRID(1,NUMNOD)=CONT(4,J,JJ)
          PGRID(2,NUMNOD)=CONT(5,J,JJ)
          PGRID(3,NUMNOD)=CONT(6,J,JJ)
          JVECLS(2,NVEC)=NUMNOD
C
          IF(JJ.NE.JMM(J)) THEN
c---------------------------------------------------------------------
c     In the preceding IF I am saying that if JJ is not equal the
c     max number of contours at this level (JMM) then do the
c     following.  If JJ is equal JMM then I am finished with
c     this contour level and I am ready to go to the next one.
c---------------------------------------------------------------------
            IF(KCON.EQ.JCSEG(NUMCON,J)) THEN
c---------------------------------------------------------------------
c     First if the number of vectors counted to this point (KCON)
c     equals the the total number of contour vectors in this
c     distinct loop of contours (JCSEG(NUMCON,J)), then I know that
c     there are more contours at this level, but that they are in one
c     or more different loops; therefore I (one) set the segment counter
c     back to zero, (two) I increment the the loop counter (NUMCON)
c     by 1 so I know how many segments are in the next loop at this
c     level, and (three) I set LCONEC to false because I know that
c     the vector just processed, and the one to be processed next,
c     do not connect since they belong to different loops.
c---------------------------------------------------------------------
              KCON=0
              NUMCON=NUMCON+1
              LCONEC=.FALSE.
            ELSE
c---------------------------------------------------------------------
c     Otherwise knowing that the vector just processed connects to the
c     one to be processed next I increment the number of vectors by
c     1 and enter the current nod number (NUMNOD) into the vector
c     connectivity array (JVECLS) as the #1 end of a new vector.  I
c     then set LCONEC to true since I know I want the next vector to
c     connect to this one or rather that the next nod in PGRID should
c     be the #2 end of this vector.
c---------------------------------------------------------------------
```

```
            NVEC=NVEC+1
            JVECLS(1,NVEC)=NUMNOD
            LCONEC=.TRUE.
          END IF
        END IF
 1800   CONTINUE
 2000 CONTINUE
C
      RETURN
      END


      SUBROUTINE GVPHCT (JVECLS,PGRID,FUNC,LINES,XLINE,NNODES,JSURLS,
     $                   NSUR,NVEC,NUMNOD,JCSEG)
C
C*************************************************************************
C
C  SUBORUTINE GVPHCT - GENERATES CONTOUR VECTORS
C
C*************************************************************************
C
C  SUBROUTINE CALLED BY
C     GHIDDN - PREPARES MODEL GEOMETRY FOR HIDDEN LINE PROCESSING
C
C*************************************************************************
C
C  SUBPROGRAMS CALLED
C     GVPHCO - ORDERS CONTOUR VECTORS INTO CONTINOUS STRINGS
C     GVPHCP - LOADS ORDERED CONTOUR COORDINATE AND CONNECTIVITY
C              INFORMATION INTO PGRID AND JVECLS
C
C*************************************************************************
C
C  VARIABLES USED
C     CFUN   = FUNCTION VALUE AT THE CENTER NODE
C     CONT   = CONTAINS VECTOR ENDPOINT COORDINATES BY CONTOUR LEVEL
C     CX     = X COORDINATE OF THE CENTER NODE
C     CY     = Y COORDINATE OF THE CENTER NODE
C     CZ     = Z COORDINATE OF THE CENTER NODE
C     FUN    = FUNCTION VALUES AT ELEMENT VERTICES
C     FUNC   = NODAL FUNCTION VALUES
C     JCSEG  = CONTAINS THE NUMBER OF DIFFERENT CONTOUR STRINGS IN EACH
C              OF THE CONTOUR LEVELS AND THE NUMBER OF LINE SEGMENTS IN
C              EACH STRING BEFORE HIDDEN LINE REMOVAL
C     JMM    = CONTAINS THE NUMBER OF CONTOUR VECTORS GENERATED AT EACH
C              CONTOUR LEVEL
C     JOUT   = OUTPUT UNIT NUMBER
C     JSURLS = ELEMENT CONNECTIVITY ARRAY
C     JVECLS = VECTOR CONNECTIVITY ARRAY (INCLUDES CONTOUR VECTORS IF
C              COMPUTED)
C     L      = INDEX TO CONTOUR VECTOR ENDPOINTS (1 OR 2)
C     LINES  = CONTAINS NUMBER OF DIFFERENT CONTOUR LEVELS
C     MM     = COUNTER USED IN DETERMINING THE NUMBER OF CONTOUR VECTORS
C              IN A CONTOUR LEVEL
C     NSEGMT = MAXIMUM NUMBER OF CONTOUR VECTORS AT ANY ONE CONTOUR
C              LEVEL
C     NNOD   = NUMBER OF NODES IN AN ELEMENT
C     NNODES = ARRAY CONTAINING THE NUMBER OF NODES IN EACH ELEMENT
C     NUMCON = CONTOUR ORDERING COUNTER THAT KEEPS TRACK OF THE NUMBER
C              OF SEPARATE CONTOUR STRINGS FOR A PARTICULAR CONTOUR
C              LEVEL.
C     NUMNOD = NUMBER OF NODES IN PGRID
C     NVEC   = NUMBER OF VECTORS IN THE MODEL (INCLUDES CONTOUR VECTORS
C              IF COMPUTED)
C     PGRID  = COORDINATE ARRAY (INCLUDES CONTOUR COORDINATES IF
C              COMPUTED)
C     XLINE  = ARRAY CONTAINING THE FUNCTION VALUES AT WHICH EACH
C              CONTOUR LINE IS REQUIRED
C     XT     = CONTOUR VALUES
C     XX     = X COORDINATE OF CONTOUR ENDPOINT
C              VERTICE
C     YY     = Y COORDINATE OF CONTOUR ENDPOINT
C     ZZ     = Z COORDINATE OF CONTOUR ENDPOINT
C
C*************************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHCT
C
      include '/g/rt/work/thesis/working/dparam.h'
C
C     MAXNS  = MAXIMUM NUMBER OF NODES PER POLYGON NOT INCLUDING CENTER
C              NODE
C     MAXNJ  = MAXIMUM NUMBER OF COORDINATES (NODES)
C     MAXNPT = MAXIMUM NUMBER OF ELEMENTS
C     MAXVEC = MAXIMUM NUMBER OF LINE SEGMENTS IN MODEL
C     MXCONT = MAXIMUM NUMBER OF CONTOUR LEVELS
C     MXCLEV = MAXIMUM NUMBER OF SEPARATE CONTOUR STRINGS AT THE SAME
C              CONTOUR LEVEL
C     MXCSEG = MAXIMUM NUMBER OF CONTOUR VECTORS IN A CONTOUR LEVEL
C
C*************************************************************************
C
      INTEGER             JVECLS(2,MAXVEC),
     $                    LINES,
     $                    JMM(MXCONT),
     $                    NNODES(MAXNPT),
     $                    JSURLS(MAXNS,MAXNPT),
     $                    NSUR,NUMNOD,NVEC,
     $                    JCSEG(MXCLEV,MXCONT)
      REAL                PGRID(3,MAXNJ),
     $                    FUNC(MAXNJ),
     $                    XLINE(MXCONT),
     $                    X(MAXNS),
     $                    Y(MAXNS),
     $                    Z(MAXNS),
     $                    FUN(MAXNS),
     $                    XX(2),
     $                    YY(2),
     $                    ZZ(2),
     $                    CONT(6,MXCONT,MXCSEG)
C
      JOUT=6
C
C                         ************************
C                         ****BEGIN PROCESSING****
C                         ************************
```

```
C
C-----------------------------------------------------------------------
C                       ******INITIALIZE******
C-----------------------------------------------------------------------
C
C-----------------------------------------------------------------------
c     Set the maximum number of contour vectors at any one contour level
c     to MXCSEG and initialize JMM which will contain the number of
c     contour vectors created at each of the requested contour levels.
C-----------------------------------------------------------------------
      MSEGMT = MXCSEG
      DO 1000 JI=1,MXCONT
        JMM(JI)=0
 1000 CONTINUE
C
      DO 8000 J=1,NSUR
        NNOD = NNODES(J)
        IF(NNOD.GT.3) THEN
C
C-----------------------------------------------------------------------
C ******CREATE CENTER NODE FOR ELEMENT AND GIVE IT AVG. FUN VALUE******
C ******ALSO STORE THE X, Y, & Z COORDINATES WITH THEIR FUN VALUES******
C-----------------------------------------------------------------------
C
c-----------------------------------------------------------------------
c     Now for each element I will create a center node by averaging the
c     nodal coordinates (X,Y, & Z) at each of the element veticies. I
c     will give this center node a function value equal to the average
c     of the function values at each of the element verticies.  Now I
c     can transform an element into a series of triangles upon which
c     this contour algorithm can operate.  The number of triangles will
c     be equal to the number of element verticies.  The algorithm
c     operates on triangles because of the special quality of the
c     triangle in  computing contours; mainly there are two qualities,
c     one being that if a contour comes in on one edge of a triangle it
c     must go out on one and only one other edge of the triangle(this
c     makes the correct conection of equal function value nodes a simple
c     matter.  The other quality is that a contour value can cross any
c     one edge of a triangle only once.
c     We will now initialize the center node values for this element.
c-----------------------------------------------------------------------
            CX=0.
            CY=0.
            CZ=0.
            CFUN=0.
c-----------------------------------------------------------------------
c     Sum up the coordinate and function values at each node of this
c     element.
c-----------------------------------------------------------------------
            DO 20 JJ=1,NNOD
              X(JJ) = PGRID(1,JSURLS(JJ,J))
              CX = CX + X(JJ)
              Y(JJ) = PGRID(2,JSURLS(JJ,J))
              CY = CY + Y(JJ)
              Z(JJ) = PGRID(3,JSURLS(JJ,J))
              CZ = CZ + Z(JJ)
              FUN(JJ) = FUNC(JSURLS(JJ,J))
              CFUN =CFUN + FUN(JJ)
   20       CONTINUE
c-----------------------------------------------------------------------
c       Create the center node by averaging the coordinate values and
c       give this node a function value by averaging the vertice
c       function values.
c-----------------------------------------------------------------------
            CX = CX/NNOD
            CY = CY/NNOD
            CZ = CZ/NNOD
            CFUN = CFUN/NNOD
c-----------------------------------------------------------------------
c     Now I have all the coordinates of each node on this element
c     stored in the NOD* arrays and I have all the nodal function values
c     stored in the FUN array; the center node values are in CX, CY, CZ,
c     and CFUN.  Now I can treat this element as though it is a system
c     of triangles (each having the center node as a common vertice
c     location) for purposes of contour computation.
c-----------------------------------------------------------------------
C
        ELSE
c-----------------------------------------------------------------------
c     Process triangular elements as they are. (don't divide them up)
c-----------------------------------------------------------------------
            NNOD=1
            X(1)=PGRID(1,JSURLS(1,J))
            Y(1)=PGRID(2,JSURLS(1,J))
            Z(1)=PGRID(3,JSURLS(1,J))
            FUN(1)=FUNC(JSURLS(1,J))
            CX=PGRID(1,JSURLS(2,J))
            CY=PGRID(2,JSURLS(2,J))
            CZ=PGRID(3,JSURLS(2,J))
            CFUN=FUNC(JSURLS(2,J))
            X(2)=PGRID(1,JSURLS(3,J))
            Y(2)=PGRID(2,JSURLS(3,J))
            Z(2)=PGRID(3,JSURLS(3,J))
            FUN(2)=FUNC(JSURLS(3,J))
        END IF
C
C-----------------------------------------------------------------------
C          ******COMPUTE THE CONTOUR VECTORS BY ELEMENT******
C-----------------------------------------------------------------------
C
        DO 2500 JJ=1,NNOD
c-----------------------------------------------------------------------
c      Now I am looping through the triangular connectivity I created
c      above.  When I get to the last triangle in this element I need
c      to have it close back on the first triangle in the element;
c      that is the first and last triangles share a common edge. This
c      closure is handled in the IF statement below.
c-----------------------------------------------------------------------
            IF(JJ.EQ.NNOD.AND.NNOD.GT.1) THEN
              FUN(JJ+1) = FUN(1)
              X(JJ+1) = X(1)
              Y(JJ+1) = Y(1)
              Z(JJ+1) = Z(1)
            END IF
c-----------------------------------------------------------------------
c     Now I will take a triangle in the element being processed and
```

```
c     locate all the wanted contour locations that may be on it and
c     I will create the corresponding contour vectors.  When I finish
c     one triangle I will do the same with the next one.
c----------------------------------------------------------------------
         DO 2450 JJJ=1,LINES
c----------------------------------------------------------------------
c     LINES contains the number of contours wanted. XT contains
c     the actual value of a contour interval wanted. JMM in
c     combination with MM keeps track of the number of contour
c     vectors created at each interval.
c----------------------------------------------------------------------
          XT = XLINE(JJJ)
          MM = JMM(JJJ)
c----------------------------------------------------------------------
c     From this point to statement 85 below I am checking to see
c     if the contour interval in question crosses any of the edges
c     of the triangle being processed.  If the difference between
c     the contour value being sougat and the the function value
c     at a node is positive at one node and negative at another
c     node, then I can say that the contour value wanted crosses
c     the edge between the two nodes in question. If the difference
c     does not change signs between nodes I can say that the
c     contour wanted does not exist on the edge betweer the two
c     nodes in question.  At this point I check the third node on
c     the triangle and if the difference at this node and either of
c     the other two does not change sign, I know that the contour
c     wanted does not fall anywhere in this triangle.
c----------------------------------------------------------------------
          IF(XT-FUN(JJ)) 2404,2401,2401
 2401     CONTINUE
          IF(XT-CFUN) 2406,2402,2402
 2402     CONTINUE
          IF(XT-FUN(JJ+1)) 2406,2403,2403
 2403     GO TO 2450
C
 2404     CONTINUE
          IF(XT-CFUN) 2405,2406,2406
 2405     CONTINUE
          IF(XT-FUN(JJ+1)) 2403,2406,2406
 2406     CONTINUE
c----------------------------------------------------------------------
c     I increment my contour vector counter since I now know a vector
c     is going to be created.
c----------------------------------------------------------------------
          MM = MM+1
c----------------------------------------------------------------------
c     L specifies which end of the vector we are creating.
c----------------------------------------------------------------------
          L=0
          IF(XT-FUN(JJ)) 2407,2407,2408
 2407     CONTINUE
          IF(XT-CFUN) 2410,2409,2409
 2408     CONTINUE
          IF(XT-CFUN) 2409,2409,2410
 2409     L=L+1
c----------------------------------------------------------------------
c     Compute the locations of the X,Y, & Z coordinate endpoints of one
c     end of a contour vector.
c----------------------------------------------------------------------
          XX(L) = (CX-X(JJ))*(XT-FUN(JJ))/(CFUN-FUN(JJ))+X(JJ)
          YY(L) = (CY-Y(JJ))*(XT-FUN(JJ))/(CFUN-FUN(JJ))+Y(JJ)
          ZZ(L) = (CZ-Z(JJ))*(XT-FUN(JJ))/(CFUN-FUN(JJ))+Z(JJ)
C
 2410     CONTINUE
C
          IF(XT-CFUN) 2411,2411,2412
 2411     CONTINUE
          IF(XT-FUN(JJ+1)) 2414,2413,2413
 2412     CONTINUE
          IF(XT-FUN(JJ+1)) 2413,2413,2414
 2413     L=L+1
c----------------------------------------------------------------------
c     Compute the locations of the X,Y, & Z coordinate endpoints of one
c     end of a contour vector.
c----------------------------------------------------------------------
          XX(L) = (X(JJ+1)-CX)*(XT-CFUN)/(FUN(JJ+1)-CFUN)+CX
          YY(L) = (Y(JJ+1)-CY)*(XT-CFUN)/(FUN(JJ+1)-CFUN)+CY
          ZZ(L) = (Z(JJ+1)-CZ)*(XT-CFUN)/(FUN(JJ+1)-CFUN)+CZ
C
 2414     CONTINUE
C
          IF(XT-FUN(JJ+1)) 2415,2415,2416
 2415     CONTINUE
          IF(XT-FUN(JJ)) 2418,2417,2417
 2416     CONTINUE
          IF(XT-FUN(JJ)) 2417,2417,2418
 2417     L=L+1
c----------------------------------------------------------------------
c     Compute the locations of the X,Y, & Z coordinate endpoints of one
c     end of a contour vector.
c----------------------------------------------------------------------
          XX(L) = (X(JJ)-X(JJ+1))*(XT-FUN(JJ+1))/
     *              (FUN(JJ)-FUN(JJ+1))+X(JJ+1)
          YY(L) = (Y(JJ)-Y(JJ+1))*(XT-FUN(JJ+1))/
     *              (FUN(JJ)-FUN(JJ+1))+Y(JJ+1)
          ZZ(L) = (Z(JJ)-Z(JJ+1))*(XT-FUN(JJ+1))/
     *              (FUN(JJ)-FUN(JJ+1))+Z(JJ+1)
 2418     CONTINUE
c----------------------------------------------------------------------
c     The coordinates for this contour segment are as follows in CONT.
c----------------------------------------------------------------------
          CONT(1,JJJ,MM)=XX(1)
          CONT(2,JJJ,MM)=YY(1)
          CONT(3,JJJ,MM)=ZZ(1)
          CONT(4,JJJ,MM)=XX(2)
          CONT(5,JJJ,MM)=YY(2)
          CONT(6,JJJ,MM)=ZZ(2)
c----------------------------------------------------------------------
c     JMM contains the total number of contour vectors generated at
c     level JJJ
c----------------------------------------------------------------------
          JMM(JJJ)=MM
C
C----------------------------------------------------------------------
C           ******CHECK ON OVERFLOW OF ARRAY BOUNDS******
C----------------------------------------------------------------------
```

```
C
            IF(MM.GT.MSEGMT) THEN
              WRITE(JOUT,5000) MSEGMT,JJJ
            END IF
C
 2450      CONTINUE
 2500    CONTINUE
 3000 CONTINUE
C
C-----------------------------------------------------------------------
C         ******ORDER THE CONTOURS INTO CONTINUOUS STRINGS******
C-----------------------------------------------------------------------
C
      CALL GVPHCO(CONT,LINES,JMM,JCSEG)
C-----------------------------------------------------------------------
C     Now I need to tac all of the coordinates for the contour segments
C     on to the end of PGRID and I need to add the contour segment
C     conectivity on to the end of JVECLS.  When this is done I will
C     be already to go to GHIDDN.  I will handle this problem one
C     contour level at a time in the GVPHCP routine.
C-----------------------------------------------------------------------
C
C-----------------------------------------------------------------------
C       ******PACK CONTOUR INFORMATION INTO PGRID AND JVECLS******
C-----------------------------------------------------------------------
C
      CALL GVPHCP(LINES,NUMCON,JMM,NUMNOD,CONT,PGRID,NVEC,JCSEG,JVECLS)
C
 5000 FORMAT('EXCEEDED MAX NUMBER OF CONTOUR LINE SEGMENTS = ',I4,/
     $        'CONTOUR LEVEL = ',I2)
C
      RETURN
      END
```

APPENDIX C

Hashing Subsystem Code

C-2

```
      SUBROUTINE GVPHPR
C
C*****************************************************************************
C
C  SUBROUTINE GVPHPR - THIS ONE TAKES THE HOLDING ARRAYS GENERATED IN
C                      MOVIE AND THEN HASHES REDUNDANT NODES AND
C                      HASHES IN ORDER TO FLAG REDUNDANT EDGES   IT
C                      CALLS THE CONTOUR ROUTINE IF CONTOURS ENABLED
C                      AND THEN CALLS THE JONESD PROCESSOR.
C
C*****************************************************************************
C
C  SUBROUTINE CALLED BY
C     VIEDRA
C
C*****************************************************************************
C
C  SUBPROGRAMS CALLED
C     GVPHCT - GENERATES CONTOUR SEGMENTS PRIOR TO JONESD HIDDEN
C              PROCESSING
C     GVPRID - JONESD HIDDEN LINE PROCESSOR
C     THIS ROUTINE ALSO CALLS HASHING ROUTINES THAT STEVE MACY WROTE.
C     IF YOU WANT TO KNOW MORE ABOUT THEM LOOK IN HIS THESIS OR IN
C     APPENDIX C OF RON THUE'S THESIS.  ALSO PLEASE NOTE THAT ALL
C     OF THE X AND Y COORDS BROUGHT INTO THIS ROUTINE FROM MOVIE ARE
C     IN A 1024 X 1024 SCREEN COORDINATE SYSTEM.  THE Z COORDINATES
C     RANGE FROM 0 TO 1023.
C
C*****************************************************************************
C
C  VARIABLES USED
C     CCONT  = HOLDING LOC FOR CONTOUR VALUES FOR JONESD HIDDEN
C     HOLDX  = HOLDING LOC FOR X EDGE COORDS FOR JONESD HIDDEN
C     HOLDY  = HOLDING LOC FOR Y EDGE COORDS FOR JONESD HIDDEN
C     HOLDZ  = HOLDING LOC FOR Z EDGE COORDS FOR JONESD HIDDEN
C     NNODES = NUMBER OF NODES IN A POLYGON
C     NPOLYS = COUNTS NUMBER OF POLYGONS
C     FUNC   = NODAL FUNCTION VALUES
C     JREDUN = REDUNDANT EDGE FLAG
C                1 = NOT REDUNDANT
C                2+ = REDUNDANT
C     JSURLS = ELEMENT CONNECTIVITY ARRAY
C     JVECLS = VECTOR CONNECTIVITY ARRAY (INCLUDES CONTOUR VECTORS IF
C              COMPUTED)
C     JVTYPE = VECTOR TYPE FLAG
C                0 = LINE ELEMENT TYPE
C                1 = SURFACE VECTOR
C     LCONT  = LOGICAL INDICATING WHETHER CONTOURS ARE ENABLED
C              .TRUE.   = CONTOURS ENABLED
C              .FALSE.  = CONTOURS DISABLED
C     NINPGR = NUMBER OF NODES IN THE MODEL
C     NNODES = ARRAY CONTAINING THE NUMBER OF NODES IN EACH ELEMENT
C     NPOLYS = NUMBER OF POLYGONS IN MODEL
C     NSUR   = NUMBER OF SURFACE ELEMENTS IN THE MODEL
C     NSVEC  = NUMBER OF VECTORS BELONGING TO SURFACE ELEMENTS
C     NUMNOD = NUMBER OF NODES IN PGRID
C     NVEC   = NUMBER OF VECTORS IN THE MODEL (INCLUDES CONTOUR VECTORS
C              IF COMPUTED)
C     PGRID  = COORDINATE ARRAY (INCLUDES CONTOUR COORDINATES IF
C              COMPUTED)
C     XHOLD  = X COORDINATES OF MODEL NODES FROM WHICH PGRID IS BUILT
C     XLINE  = ARRAY CONTAINING THE FUNCTION VALUES AT WHICH EACH
C              CONTOUR LINE IS REQUIRED
C     XMX    = MAXIMUM X MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     XMN    = MINIMUM X MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     YHOLD  = Y COORDINATES OF MODEL NODES FROM WHICH PGRID IS BUILT
C     YMX    = MAXIMUM Y MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     YMN    = MINIMUM Y MODEL COORDINATE FOR EFFICIENT BUCKET SORT
C     ZHOLD  = Z COORDINATES OF MODEL NODES FROM WHICH PGRID IS BUILT
C     FOR EXPLANATION OF HASHING VARIABLES SEE CODE IN RON THUE'S
C     THESIS.
C
C*****************************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHPR
C
C     MAXNS  = MAXIMUM NUMBER OF NODES PER POLYGON NOT INCLUDING CENTER
C              NODE
C     MAXNJ  = MAXIMUM NUMBER OF COORDINATES (NODES)
C     MAXNPT = MAXIMUM NUMBER OF ELEMENTS
C     MAXVEC = MAXIMUM NUMBER OF LINE SEGMENTS IN MODEL
C     MXCONT = MAXIMUM NUMBER OF CONTOUR LEVELS
C     MXCLEV = MAXIMUM NUMBER OF SEPARATE CONTOUR STRINGS AT THE SAME
C              CONTOUR LEVEL
C
      INCLUDE '/g/rt/work/thesis/working/dparam.h'
C
C*****************************************************************************
C
      COMMON/CONLEV/   CHI,CLO,NCONLV,CLEVEL(MXCONT)
      COMMON/GRID/     XHOLD(MAXNS,MAXNPT),
     $                 YHOLD(MAXNS,MAXNPT),
     $                 ZHOLD(MAXNS,MAXNPT),
     $                 CCONT(MAXNS,MAXNPT),
     $                 NNODES(MAXNPT)
      COMMON/QFORIO/   CONTRS,IDIVICE,IBAD,SHOSHR,LBLSPC
      COMMON/SPARE/    NNNOD,NPOLYS
      EQUIVALENCE      (XLINE,CLEVEL)
      DIMENSION        JSURLS(MAXNS,MAXNPT),
     +                 JVECLS(2,MAXVEC),
     +                 PGRID(3,MAXNJ),
     +                 JREDUN(MAXVEC),
     +                 JVTYPE(MAXVEC),
     +                 XLINE(MXCONT),
     +                 FUNC(MAXNJ),
     +                 JCSEG(MXCLEV,MXCONT)
      LOGICAL          LCONT,CONTRS
C
C     THIS IS HASH STUFF
C
      PARAMETER        (LMEMRY=MAXNJ)
      DIMENSION        MEMORY(LMEMRY),
     +                 JPGINB(LMEMRY)
      PARAMETER        (TOLER=.001)
      PARAMETER        (MAXINT=2147483647)
C
```

```
C                    ****************************
C                    *****-BEGIN PROCESSING******
C                    ****************************
C
C-----------------------------------------------------------------------
C         INITIALIZE REDUNDANT EDGE FLAG AND VECTOR TYPE FLAG
C-----------------------------------------------------------------------
C
      DO 100 J=1,MAXVEC
        JREDUN(J)=0
        JVTYPE(J)=0
  100 CONTINUE
      NSUR = NPOLYS
      NINPGR = 0
C-----------------------------------------------------------------------
C         INIT HASH SUBSYSTEM
C-----------------------------------------------------------------------
      CALL UHINIT(MAXINT,MAXNJ,LMEMRY,28,MEMORY,LMEMRY,1,LREQD,JERR)
      IF (JERR.NE.0) CALL UABORT
     +    ('*A*ERROR FROM HASH SUBSYSTEM *GHIDDN*',37)
C-----------------------------------------------------------------------
C     BEGIN PROCESS NECESSARY TO STORE THE MODEL MIN AND MAX COORDINATES
C     TO BE USED IN BUCKET SORT SET UP IN GVPHGD
C-----------------------------------------------------------------------
      XMX = XHOLD(1,1)
      XMN = XHOLD(1,1)
      YMX = YHOLD(1,1)
      YMN = YHOLD(1,1)
C-----------------------------------------------------------------------
C         BUILD *JSURLS* AND *PGRID*
C
C         LOOP FOR EACH POLYGON IN THE LIST
C-----------------------------------------------------------------------
C
C     I DO THIS BECAUSE I BEGAN IN MOVIE WITH NPOLYS =1 SO EVEN IF
C     THERE ARE NO POLYGONS NPOLYS WILL SAY THERE IS 1.
C
      IF(NPOLYS.LE.1) RETURN
C-----------------------------------------------------------------------
      DO 2500 J=1,NPOLYS
C-----------------------------------------------------------------------
C        LOOP FOR EACH NODE IN THIS POLYGON
C-----------------------------------------------------------------------
      DO 2500 JJ=1,NNODES(J)
C
C         LOOP FOR EACH NODE IN PGRID TO DETERMINE IF THIS NODE
C           IS ALREADY IN THE LIST.
C           - CREATE A KEY FROM THE NORMALIZED X,Y,Z COORDINATE VALUES.
C              MAP THIS KEY INTO AN 8-BIT SPACE SO IT CAN BE UNIQUELY
C              DEFINED BY A ONE WORD KEY.
C           - PUT KEY IN TABLE. IF PUT FAILS, COORDINATE IS ALREADY
C              IN THE TABLE.
C
      NX = XHOLD(JJ,J)
      NY = YHOLD(JJ,J)
      NZ = ZHOLD(JJ,J)
C
      JKEY = NX
      JKEY = JKEY * (2**10)
      JKEY = JKEY + NY
      JKEY = JKEY * (2**10)
      JKEY = JKEY + NZ
C
      CALL UHPUT(JKEY,NBINT,MEMORY,JERR)
C-----------------------------------------------------------------------
C         NODE DOESNT EXIST IN *PGRID* - PUT THE NODE INTO *PGRID*
C-----------------------------------------------------------------------
      IF (JERR.EQ.0) THEN
         NINPGR = NINPGR + 1
         PGRID(1,NINPGR) = XHOLD(JJ,J)
         PGRID(2,NINPGR) = YHOLD(JJ,J)
         PGRID(3,NINPGR) = ZHOLD(JJ,J)
         FUNC(NINPGR) = CCONT(JJ,J)
C-----------------------------------------------------------------------
C         FIND THE MIN AND MAX MODEL COORDS FOR BUCKET SORT LATER
C-----------------------------------------------------------------------
         IF(PGRID(1,NINPGR).GE.XMX) XMX=PGRID(1,NINPGR)
         IF(PGRID(1,NINPGR).LE.XMN) XMN=PGRID(1,NINPGR)
         IF(PGRID(2,NINPGR).GE.YMX) YMX=PGRID(2,NINPGR)
         IF(PGRID(2,NINPGR).LE.YMN) YMN=PGRID(2,NINPGR)
C
         JSURLS(JJ,J) = NINPGR
C-----------------------------------------------------------------------
C         STORE THE PGRID INDEX AT THE NODES INTERNAL NUMBER LOCATION
C           IN THE JPGINB ARRAY. THE INDEX INTO THE JPGINB IS THE KEYS
C           ACTUAL INTERNAL NUMBER. THE VALUE STORED IS THE PGRID INDEX.
C-----------------------------------------------------------------------
         JPGINB(NBINT) = NINPGR
C-----------------------------------------------------------------------
C         NODE KEY ALREADY EXISTS. ASSUME THAT THE NODE IS IN THE LIST -
C          STORE THE PGRID NODE NUMBER INTO THE CONNECTIVITY ARRAY
C-----------------------------------------------------------------------
      ELSE
         CALL UHETOI(JKEY,NBINT,MEMORY)
         JSURLS(JJ,J) = JPGINB(NBINT)
      ENDIF
C
 2500 CONTINUE
C
      IF(NINPGR.EQ.0) RETURN
C
C-----------------------------------------------------------------------
C         BUILD *JVECLS* - ASSUMES N-SIDED POLYGONS
C         BUILD *JREDUN* - FLAGS  REDUNDANT EDGES
C         BUILD *JVTYPE* - FLAGS VECTORS AS BEING SURFACE OR
C                            SOLID ELEMENT EDGES
C-----------------------------------------------------------------------
C
      CALL UHINIT(2147483647,MAXNJ,LMEMRY,28,MEMORY,LMEMRY,1,LREQD,JERR)
C
      NVEC = 0
      DO 2900 J=1,NPOLYS
        DO 2800 JJ=1,NNODES(J)-1
          NVEC = NVEC + 1
          JVECLS(1,NVEC) = JSURLS(JJ,J)
          JVECLS(2,NVEC) = JSURLS(JJ+1,J)
```

```
      IF(JVECLS(1,NVEC).GT.JVECLS(2,NVEC)) THEN
        JTEMP = JVECLS(1,NVEC)
        JVECLS(1,NVEC) = JVECLS(2,NVEC)
        JVECLS(2,NVEC) = JTEMP
      END IF
C-----------------------------------------------------------------------
C         BUILD REDUNDANT EDGE AND VECTOR TYPE FLAGS
C-----------------------------------------------------------------------
      JKEY = JVECLS(1,NVEC)
      JKEY = JKEY * (2**15)
      JKEY = JKEY + JVECLS(2,NVEC)
C
      CALL URPUT(JKEY,NBINT,MEMORY,JERR)
C-----------------------------------------------------------------------
C       FIRST TIME EDGE ENCOUNTERED. THE NUMBER OF ENCOUNTERS FOR
C        EDGE IS STORED IN *JPGINB* AT THE KEYS INTERNAL NUMBER.
C-----------------------------------------------------------------------
      IF (JERR.EQ.0) THEN
C       PRINT*,'NVEC=',NVEC,'JKEY=',JKEY
        JPGINB(NBINT) = 1
        JREDUN(NVEC) = 1
        JVTYPE(NVEC) = 1
C-----------------------------------------------------------------------
C       EDGE ALREADY IN THE LIS
C-----------------------------------------------------------------------
      ELSE
        CALL URETOI(JKEY,NBINT,MEMORY)
        JPGINB(NBINT) = JPGINB(NBINT) + 1
        JREDUN(NVEC) = JPGINB(NBINT)
        JVTYPE(NVEC) = 1
      ENDIF
 2800 CONTINUE
      NVEC = NVEC + 1
      JVECLS(1,NVEC) = JSURLS(NNODES(J),J)
      JVECLS(2,NVEC) = JSURLS(1,J)
      IF(JVECLS(1,NVEC).GT.JVECLS(2,NVEC)) THEN
        JTEMP = JVECLS(1,NVEC)
        JVECLS(1,NVEC) = JVECLS(2,NVEC)
        JVECLS(2,NVEC) = JTEMP
      END IF
C-----------------------------------------------------------------------
C         BUILD REDUNDANT EDGE AND VECTOR TYPE FLAGS
C-----------------------------------------------------------------------
      JKEY = JVECLS(1,NVEC)
      JKEY = JKEY * (2**15)
      JKEY = JKEY + JVECLS(2,NVEC)
C
      CALL URPUT(JKEY,NBINT,MEMORY,JERR)
C-----------------------------------------------------------------------
C       FIRST TIME EDGE ENCOUNTERED. THE NUMBER OF ENCOUNTERS FOR
C        EDGE IS STORED IN *JPGINB* AT THE KEYS INTERNAL NUMBER.
C-----------------------------------------------------------------------
      IF (JERR.EQ.0) THEN
        JREDUN(NVEC) = 1
        JVTYPE(NVEC) = 1
        JPGINB(NBINT) = 1
C-----------------------------------------------------------------------
C       EDGE ALREADY IN THE LIST.
C-----------------------------------------------------------------------
      ELSE
        CALL URETOI(JKEY,NBINT,MEMORY)
        JPGINB(NBINT) = JPGINB(NBINT) + 1
        JREDUN(NVEC) = JPGINB(NBINT)
        JVTYPE(NVEC) = 1
      ENDIF
 2900 CONTINUE
      LCONT=.FALSE.
C-----------------------------------------------------------------------
C         CALL THE CONTOUR ROUTINE IF CONTOURS ENABLED
C-----------------------------------------------------------------------
      IF(CONTRS) THEN
        LCONT=.TRUE.
        LINES=NCONLV
C-----------------------------------------------------------------------
C         SAVE THE NUMBER OF VECTORS THAT BELONG TO SURFACES
C-----------------------------------------------------------------------
        NSVEC=NVEC
C
        CALL GVPHCT(JVECLS,PGRID,FUNC,LINES,XLINE,NNODES,JSURLS,NSUR,
     $             NVEC,NINPGR,JCSEG)
C
      END IF
C-----------------------------------------------------------------------
C         CALL TO PROCESS HIDDEN LINES
C-----------------------------------------------------------------------
        CALL GVPHID(PGRID,JVECLS,JSURLS,NVEC,NSUR,
     +              XMX,XMN,YMX,YMN,NSVEC,JREDUN,JVTYPE,
     +              NNODES,LCONT,JCSEG,LINES)
C
99999 RETURN
      END


      SUBROUTINE URDEL(JKEY,JINTNB,MEMORY,JERR)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:   HASHING SUBSYTEM
CXX
CXX   PURPOSE:
CXX      TO DELETE AN EXTERNAL KEY FROM THE HASH TABLE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX      ARGUMENT  TYPE  I/O  DESCRIPTION
CXX       JKEY       I    I   EXTERNAL KEY TO DELETE.
CXX       JINTNB     I    O   INTERNAL NUMBER ASSIGNED TO THE KEY.
CXX      MEMORY()    I   I/O  MEMRORY ARRAY CONTAINING THE
CXX                            HASHING DATA BASE
CXX       JERR       I    O   ERROR FLAG:
CXX                            0=NO ERRORS
CXX                            1=TABLE EMPTY
CXX                            2=*JKEY* NOT IN THE TABLE.
CXX
```

```
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX       /CRASH/   CONTAINS THE HASH TABLE - SEE *URINIT*
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX       THIS MODULE CALLS TO GET THE LOOK-UP TABLE INDEX FOR THIS
CXX        EXTERNAL NUMBER. AN ERROR OCCURS IF THE NUMBER (*JKEY*)
CXX        IS NOT IN THE LOOK-UP TABLE. ONCE THE HASH
CXX        TABLE INDEX IS OBTAINED, THE MODULE CALLS TO HAVE
CXX        THE KEY DELETED.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CRASH/  IMAPHE,LENTBL,NINTBL,MXKEYS,MAXKEY,JOFFST,JDIVSR
C
      DIMENSION MEMORY(*)
C
C  * * * * * * * * * * * * * * * * * * * *
C   P R O C E E D U R E   S E C T I O N
C  * * * * * * * * * * * * * * * * * * * *
C
C
C..      INITIALIZE
C
      JERR = 0
C
C..      ERROR IF TABLE IS EMPTY
C
      IF (NINTBL.EQ.0) THEN
         JERR = 1
         GO TO 99990
      ENDIF
C
C..      CALL TO GET THE LOOK-UP TABLE INDEX
C
      CALL URLOOK(JKEY,JCODE,JWHAT,MEMORY)
      IF (JWHAT.NE.1) GO TO 90000
C
C..      UPDATE THE HASH TABLES
C
C.       DELETE THE EXTERNAL KEY FROM THE TABLE (*IMAPHE*)
C
      CALL UHUDEL(JCODE,JINTNB,MEMORY)
C
C..      DONE - NO ERRORS
C
      GO TO 99990
C
C..      PROCESS ERRORS
C
90000 CONTINUE
      IF (JWHAT.EQ.2) THEN
         JERR = 2
         GO TO 99990
      ELSEIF (JWHAT.NE.5) THEN
         CALL UABORT('*A*UNKNOWN ERROR FLAG FROM *URLOOK*',35)
      ENDIF
C
C  * * * * * * *
C   R E T U R N
C  * * * * * * *
C
99990 RETURN
      END
      SUBROUTINE UHDIR(MEMORY)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:   HASHING SUBSYSTEM
CXX
CXX   PURPOSE:
CXX       TO OUTPUT THE HASH TABLES TO THE SCREEN (DEBUG)
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX      ARGUMENT   TYPE  I/O  DESCRIPTION
CXX       MEMORY()   I    I/O  MEMORY ARRAY CONTAINING THE
CXX                              HASHING DATA BASE
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX       /CRASH/    CONTAINS THE HASH TABLE - SEE *URINIT*
CXX       IMAPHE     LIST FVA OFFSET FOR THE HASH TABLE.
CXX                   THIS TABLE IS USED TO STORE KEYS AT
CXX                   THE CALCULATED HASH ADDRESS. THIS PROVIDES A
CXX                   MAPPING OF HASH TO EXTERNAL KEYS.
CXX                   THE TABLE IS INITIALIZED TO ZERO.
CXX                   WHEN KEYS ARE DELETED FROM THE TABLE, A NEGATIVE
CXX                   VALUE MUST BE PLACED IN THE DELETED LOCATION.
CXX                   THOUGH A DELETED LOCATION IS AVAILABLE FOR USE,
CXX                   IT MUST NOT TERMINATE THE SEARCHS FOR KEYS.
CXX       LENTBL     DIMENSION LENGTH OF THE TABLE. IF *JOFFST* IS
CXX                   GREATER THAN 1, THEN THIS VALUE NEEDS TO BE
CXX                   PRIME. IF IT IS NOT PRIME, THEN THE NEXT
CXX                   LARGEST PRIME INTEGER WILL BE AUTOMATICALLY
CXX                   ASSIGNED TO IT.
```

```
CXX       NINTBL    INDICATES THE NUMBER OF ACTIVE KEYS IN THE TABLE.
CXX       MXKEYS    INDICATES THE MAXIMUM NUMBER OF KEYS THAT MAY BE
CXX                  IN THE TABLE AT ANY ONE TIME. THE TABLE IS
CXX                  DECLARED FULL WHEN THIS NUMBER OF ETRIES HAVE
CXX                  BEEN ENTERED.
CXX       MAXKEY    LARGEST ALLOWABLE VALUE OF AN EXTERNAL KEY.
CXX       JOFFST    THIS IS THE AMOUNT TO OFFSET FROM THE LAST
CXX                  POSITION WHEN CHECKING A NEW POSITION IN
CXX                  THE TABLE. IF THIS VALUE IS GREATER THAN 1 THEN
CXX                  IT MUST BE A PRIME NUMBER AND THE TABLE LENGTH
CXX                  MUST ALSO BE PRIME. IF IT IS NOT PRIME, THEN THE
CXX                  NEXT LARGEST PRIME INTEGER WILL BE AUTOMATICALLY
CXX                  ASSIGNED TO IT.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CRASH/  IMAPHE,LENTBL,NINTBL,MXKEYS,MAXKEY,JOFFST,JDIVSR
C
      DIMENSION MEMORY(*)
C
      CHARACTER MSG*70
C
      CHARACTER KINPUT*8
C
C  * * * * * * * * * * * * * * * * * * * *
C   P R O C E E D U R E   S E C T I O N
C  * * * * * * * * * * * * * * * * * * * *
C
C
C..        PERFORM DIALOG TO ALLOW USER TO ABORT COMMAND
C
      CALL UDOMSG('*** HASH UTILITY TABLES ***',28)
      MSG = ' THE OUTPUT WILL BE OVER       LINES LONG.'
      WRITE(MSG(26:29),'(I4)') LENTBL
      CALL UDOMSG(MSG,41)
      CALL UDOMSG(' DO YOU WISH TO CONTINUE?',25)
      CALL UDGETC(8,KINPUT,LINPUT)
      IF (KINPUT(1:1).NE.'Y'.AND.KINPUT(1:1).NE.'y') GO TO 99990
C
C..        OUTPUT HEADER
C
      CALL UDOMSG(' ',1 )
      CALL UDOMSG('*** HASH TABLE DIRECTORY ***',28)
C
C..        OUTPUT THE HASH TABLE
C
      MSG = 'NUMBER OF KEYS IN HASH TABLE = '
      WRITE(MSG(32:35),'(I4)') NINTBL
      CALL UDOMSG(MSG,35)
C
      CALL UDOMSG(' ',1 )
      MSG = 'INDEX        MAPHE'
      CALL UDOMSG(MSG,16)
C
      DO 100 J=1,LENTBL
      MSG = ' '
      WRITE(MSG(1:5),'(I5)') J
      WRITE(MSG(8:16),'(I9)') MEMORY(IMAPHE+J)
C
      CALL UDOMSG(MSG,16)
  100 CONTINUE
C
C..        OUTPUT THE MEMORY ARRAY
C
      CALL UDOMSG('*** HASH UTILITY MEMORY DUMP ***',32)
      MSG = ' THE OUTPUT WILL BE OVER       LINES LONG.'
      WRITE(MSG(26:30),'(I5)') LENTBL+50
      CALL UDOMSG(MSG,41)
      CALL UDOMSG(' DO YOU WISH TO CONTINUE?',25)
      CALL UDGETC(8,KINPUT,LINPUT)
      IF (KINPUT(1:1).NE.'Y'.AND.KINPUT(1:1).NE.'y') GO TO 99990
C
      CALL UDOMSG(' ',1 )
      MSG = 'INDEX        VALUE'
      CALL UDOMSG(MSG,16)
C
      DO 200 J=1,LENTBL+50
      MSG = ' '
      WRITE(MSG(1:5),'(I5)') J
      WRITE(MSG(8:16),'(I9)') MEMORY(J)
      CALL UDOMSG(MSG,16)
  200 CONTINUE
C
C  * * * * * * *
C   R E T U R N
C  * * * * * * *
C
99990 RETURN
      END
      SUBROUTINE URETOI(JKEY,JINTNB,MEMORY)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:     HASHING SUBSYTEM
CXX
CXX   PURPOSE:
CXX        TO RETURN THE INTERNAL NUMBER OF A GIVEN EXTERNAL KEY.
CXX
```

```
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX      ARGUMENT  TYPE  I/O  DESCRIPTION
CXX       JKEY      I     I   EXTERNAL KEY.
CXX       JINTNB    I     O   INTERNAL NUMBER ASSOCIATED WITH *JKEY*.
CXX                            O=*JKEY* DOES NOT EXIST.
CXX       MEMORY()  I    I/O  MEMRORY ARRAY CONTAINING THE
CXX                            HASHING DATA BASE
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX       /CHASH/   CONTAINS THE HASH TABLE - SEE *UHINIT*
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX       THIS MODULE CALLS TO GET THE HASH CODE FOR THE GIVEN KEY.
CXX        USING THE HASH CODE, THE INTERNAL NUMBER IS LOOKED UP
CXX        AND RETURNED.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CHASH/  IMAPHE,LEXTBL,NINTBL,MXKEYS,MAXKEY,JOFFST,JDIVSR
C
      DIMENSION MEMORY(*)
C
C  * * * * * * * * * * * * * * * * * * * *
C   P R O C E E D U R E   S E C T I O N
C  * * * * * * * * * * * * * * * * * * * *
C
C
C..       VERIFY REQUEST
C
      IF (JKEY.LT.1.OR.JKEY.GT.MAXKEY) CALL UABORT
     +    ('*A*INVALID KEY REQUEST IN *UHETOI*',84)
C
C..       INITIALIZE
C
      JINTNB = 0
C
C..       DONE IF TABLE IS EMPTY
C
      IF (NINTBL.EQ.0) GO TO 99990
C
C..       CALL TO GET THE LOOK-UP TABLE INDEX
C
      CALL UHLOOK(JKEY,JCODE,JWHAT,MEMORY)
      IF (JWHAT.NE.1) GO TO 99990
C
C..       GET THE INTERNAL NUMBER
C
      JINTNB = JCODE
C
C  * * * * * * *
C   R E T U R N
C  * * * * * * *
C
99990 RETURN
      END
      SUBROUTINE UHINBS(JIN,JBUF,LBUF,JOUT,NINBUF,MEMORY)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:   HASHING SUBSYTEM
CXX
CXX   PURPOSE:
CXX       TO BUFFER OUT THE CURRENT LIST OF INTERNAL NUMBERS.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX      ARGUMENT  TYPE  I/O  DESCRIPTION
CXX       JIN       I     I   INPUT FLAG:
CXX                            O=INITIALIZE THE ROUTINE. THIS ROUTINE
CXX                               MUST BE REINITIALIZED EACH TIME
CXX                               THE LIST IS TO BE PROCESSED.
CXX                            1=FILL THE BUFFER WITH VALUES.
CXX                               THE BUFFER WILL BE FILLED
CXX                               WITH EACH SUCCESSIVE CALL TO
CXX                               UNTIL ALL VALUES ARE EXHAUSTED.
CXX       JBUF()    I     I   BUFFER SUPPLIED BY CALLING ROUTINE.
CXX       LBUF      I     I   DIMENSIONED LENGTH OF JBUF.
CXX       JOUT      I     O   OUTPUT FLAG:
CXX                            O=LIST EXHAUSTED BY THIS CALL.
CXX                            1=MORE VALUES IN LIST. THESE WILL
CXX                              BE RETURNED BY SUBSEQUENT CALLS.
CXX       NINBUF    I     O   NUMBER OF VALUES RETURNED IN THE
CXX                            BUFFER *JBUF*.
CXX       MEMORY()  I    I/O  MEMORY ARRAY CONTAINING DATA BASE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX       /CHASH/   CONTAINS THE HASH TABLE - SEE *UHINIT*
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX       .NONE.
```

```
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX       THIS MODULE IS USED TO RETURN TO THE CALLER THE LIST
CXX        OF ACTIVE INTERNAL NUMBERS. THESE INTERNAL NUMBERS ARE
CXX        THE NUMBERS ASSIGNED TO KEYS WHEN THEY WERE ENTERED INTO
CXX        THE HASH TABLE. SINCE THE CALLER MAY NOT BE ABLE TO
CXX        DIMENSION AN ARAY LARGE ENOUGH TO HANDLE THE COMPLETE
CXX        LIST OF NUMBERS, THE LIST IS BUFFERED TO THE USER
CXX        BY SUCCESSIVE CALLS TO THIS ROUTINE. THE CALLER
CXX        DECIDES THE LENGTH OF THE BUFFERING ARRAY.
CXX
CXX       IT IS IMPORTANT THAT THE CALLER INITIALIZE THIS ROUTINE
CXX        EACH TIME BEFORE REQUESTING A COMPLETE LIST OF INTERNAL
CXX        NUMBERS.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX        .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CHASH/  IMAPHE,LENTBL,NINTBL,MXKEYS,MAXKEY,JOFFST,JDIVSR
C
      DIMENSION JBUF(LBUF),
     +          MEMORY(*)
C
      SAVE NLEFT,JLAST
C
C  * * * * * * * * * * * * * * * * * * * *
C   P R O C E E D U R E   S E C T I O N
C  * * * * * * * * * * * * * * * * * * * *
C
C
C..       INITIALIZE
C
      IF (JIN.NE.0) GO TO 2000
C
      NLEFT = NINTBL
      JLAST = 0
      GO TO 99990
C
C..       PUT VALUES INTO THE BUFFER - INCREMENT FROM THE
C..        LAST POSITION. THE INTERNAL NUMBER IS THE INDEX (*JAT*)
C..        INTO THE TABLE *IMAPIH*.
C
 2000 CONTINUE
      IF (JIN.NE.1) CALL UABORT
     +   ('*A*INVALID INPUT FLAG (*UHINBS*)',82)
C
      IF (NLEFT.EQ.0) THEN
         NINBUF = 0
         JOUT = 0
         GO TO 99990
      ENDIF
C
C.        LOOP TO GET EACH INTERNAL NUMBER TO PUT INTO THE BUFFER.
C
      NINBUF = NLEFT
      IF (NLEFT.GT.LBUF) NINBUF = LBUF
      JAT =  LAST
      DO 2500 J=1,NINBUF
C
C         INCREMENT THE INTERNAL NUMBER POSITON.
C          IF INTERNAL NUMBER IS DELETED OR NOT IN USE, INCREMENT
C          THE INTERNAL NUMBER AND CHECK THE NEW ONE.
C
 2100 CONTINUE
      JAT = JAT + 1
      IF (MEMORY(IMAPHE+JAT).LE.0) THEN
         IF (JAT.GT.LENTBL) CALL UABORT
     +    ('*A*FEWER VALUES IN TABLE THAN ANTICIPATED IN *UHINBS*',83)
         GO TO 2100
      ENDIF
C
C         PUT THE INTERNAL NUMBER IN THE BUFFER
C
      JBUF(J) = JAT
 2500 CONTINUE
      NLEFT = NLEFT - NINBUF
      JLAST = JAT
C
C.        SET THE OUT FLAG
C
      IF (NLEFT.EQ.0) THEN
         JOUT = 0
      ELSE
         JOUT = 1
      ENDIF
C
C  * * * * * * *
C   R E T U R N
C  * * * * * * *
C
99990 RETURN
      END
      SUBROUTINE UHINIT(MAXITM,MXITMS,LTABLE,JOFSET,MEMORY,LMEMRY,
     +                  JFWA,LREQD,JERR)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:   HASHING SUBSYTEM
CXX
CXX   PURPOSE:
CXX       TO INITITIALIZE THE HASH TABLES
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX      ARGUMENT  TYPE  I/O  DESCRIPTION
CXX       MAXITM    I     I    LARGEST ALLOWABLE ITEM (KEY). IT IS
```

```
CXX                              ASSUMED THAT THE MINIMUM ALLOWABLE
CXX                              ITEM (KEY) IS 1.
CXX      MXITMS    I    O       MAXIMUM NUMBER OF ITEMS (KEYS) THAT MAY
CXX                              BE INSERTED INTO THE TABLE. THIS VALUE
CXX                              DETERMINES THE PERCENTAGE AT WHICH THE
CXX                              HASH TABLE WILL BE DECLARED FULL.
CXX      LTABLE    I    I       LENGTH TO DIMENSION THE HASH TABLE.
CXX      JOFSET    I    I       THIS IS THE AMOUNT TO OFFSET FROM THE
CXX                              LAST POSITION WHEN CHECKING A NEW
CXX                              POSITION IN THE TABLE. IF THIS VALUE
CXX                              IS NOT EQUAL TO 1, THEN IT MUST BE
CXX                              CO-PRIME WITH *LTABLE*. IF IT IS
CXX                              NOT CO-PRIME, THEN THE NEXT LARGEST
CXX                              CO-PRIME INTEGER WILL BE AUTOMATICALLY
CXX                              ASSIGNED TO IT.
CXX      MEMORY    I    I/O     ARRAY AVAILABLE FOR HASH INFO STORAGE.
CXX      LMEMRY    I    I       ABSOLUTE LENGTH OF THE *MEMORY* ARRAY.
CXX      JFWA      I    I       FWA AT WHICH TO BEGIN THE HASH TABLES
CXX                              IN THE MEMORY ARRAY.
CXX      LREQD     I    O       AMOUNT OF THE MEMORY ARRAY REQUIRED FOR
CXX                              HASH TABLE STORAGE.
CXX      JERR      I    O       ERROR FLAG:
CXX                              0=NO ERRORS
CXX                              1=MEMORY REQUIREMENT EXCEEDED.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX      /CHASH/      CONTAINS THE HASH TABLE - SEE *UHINIT*
CXX      IMAPHE       LIST FWA OFFSET FOR THE HASH TABLE.
CXX                    THIS TABLE IS USED TO STORE KEYS AT
CXX                    THE CALCULATED HASH ADDRESS. THIS PROVIDES A
CXX                    MAPPING OF HASH TO EXTERNAL KEYS.
CXX                    THE TABLE IS INITIALIZED TO ZERO.
CXX                    WHEN KEYS ARE DELETED FROM THE TABLE, A NEGATIVE
CXX                    VALUE MUST BE PLACED IN THE DELETED LOCATION.
CXX                    THOUGH A DELETED LOCATION IS AVAILABLE FOR USE,
CXX                    IT MUST NOT TERMINATE THE SEARCHS FOR KEYS.
CXX      LENTBL       DIMENSION LENGTH OF THE TABLE. IF *JOFFST* IS
CXX                    GREATER THAN 1, THEN THIS VALUE NEEDS TO BE
CXX                    PRIME. IF IT IS NOT PRIME, THEN THE NEXT
CXX                    LARGEST PRIME INTEGER WILL BE AUTOMATICALLY
CXX                    ASSIGNED TO IT.
CXX      NINTBL       INDICATES THE NUMBER OF ACTIVE KEYS IN THE TABLE.
CXX      MXKEYS       INDICATES THE MAXIMUM NUMBER OF KEYS THAT MAY BE
CXX                    IN THE TABLE AT ANY ONE TIME. THE TABLE IS
CXX                    DECLARED FULL WHEN THIS NUMBER OF ETRIES HAVE
CXX                    BEEN ENTERED.
CXX      MAXKEY       LARGEST ALLOWABLE VALUE OF AN EXTERNAL KEY.
CXX      JOFFST       THIS IS THE AMOUNT TO OFFSET FROM THE LAST
CXX                    POSITION WHEN CHECKING A NEW POSITION IN
CXX                    THE TABLE. IF THIS VALUE IS GREATER THAN 1 THEN
CXX                    IT MUST BE A PRIME NUMBER AND THE TABLE LENGTH
CXX                    MUST ALSO BE PRIME. IF IT IS NOT PRIME, THEN THE
CXX                    NEXT LARGEST PRIME INTEGER WILL BE AUTOMATICALLY
CXX                    ASSIGNED TO IT.
CXX
CXX   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX      IMMEDIATLY AFTER CALLING THIS ROUTINE, THE CALLER MUST
CXX        VERIFY THAT THE AMOUNT OF MEMORY REQUIRED BY THE HASH
CXX        SUBSYSTEM  (*LREQD*) DOES NOT EXCEED THE AVAILABLE
CXX        SPACE IN THE MEMORY ARRAY PROVIDED (*MEMORY*). THIS IS
CXX        ACCOMPLISHED BY CHECKING THE ERROR FLAG OR THE *LREQD*
CXX        PARAMETER.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX      ALL TABLES ARE INITIALIZED IN THIS ROUTINE. A DESCRIPTION
CXX        OF EACH TABLE IS GIVEN IN THE COMMON BLOCK DESCRIPTION.
CXX        THE HASH TABLES RESIDE IN THE *MEMORY* ARRAY PASSED BY
CXX        THE CALLING ROUTINES. THE APPLICATION MULT FIRST CALL
CXX        THIS ROUTINE WITH THE *MEMORY* ARRAY AND WITH THE
CXX        STARTING FWA (FIRST WORD ADDRESS) TO BEGIN THE TABLES.
CXX        THIS ROUTINE WILL IN TURN ESTABLISH FWA'S FOR EACH
CXX        REQUIRED TABLE AND RETURN TO THE CALLING APPLICATION
CXX        THE AMOUNT OF MEMORY REQUIRED (IN WORDS).
CXX
CXX      THERE IS AN ENTERY POINT IN THIS ROUTINE BY THE NAME
CXX        OF *UHITBL* (INIT TABLE). THIS ALLOWS THE REINITIALIZING
CXX        OF THE HASH TABLES WHENEVER THE TABLE IS DETERMINED TO BE
CXX        EMPTY OR COMPLETELY DELETED.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX      .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CHASH/  IMAPHE,LENTBL,NINTBL,MXKEYS,MAXKEY,JOFFST,JDIVSR
C
C******************
C******************
C
      DIMENSION MEMORY(*)
C
C   * * * * * * * * * * * * * * * * * * * *
C    P R O C E E D U R E   S E C T I O N
C   * * * * * * * * * * * * * * * * * * * *
C
      JERR = 0
C
C.        INIT COMMON BLOCK *CHASH* VARIABLES
C
      IF (MAXITM.LT.1) CALL UABORT
     +   ('*A*MAX KEY TOO SMALL',20)
      MAXKEY = MAXITM
C
C.        CREATE A HASH DIVISOR THAT IS THE LARGEST PRIME NUMBER
C.         THAT IS LESS THAN OR EQUAL TO THE TABLE SIZE.
```

```
C
      JDIVSR = LTABLE
  100 CONTINUE
      DO 200 J=2,(JDIVSR/2)+1
      JTEST = MOD(JDIVSR,J)
      IF (JTEST.EQ.0) THEN
         JDIVSR = JDIVSR - 1
         GO TO 100
      ENDIF
  200 CONTINUE
C
C..       VERIFY THAT THE TABLE LENGTH AND THE OFFSET VALUE
C..        ARE CO-PRIME. (NOT REQUIRED IF OFFSET = 1). ADJUST
C..        OFFSET IF NECESSARY.
C
      LENTBL = LTABLE
      JOFFST = JOFSET
      IF (JOFFST.EQ.1.OR.JOFFST.EQ.-1) GO TO 800
  300 CONTINUE
      DO 400 J=2,JOFFST
      JTEST1 = MOD(JOFFST,J)
      JTEST2 = MOD(LENTBL,J)
      IF (JTEST1.EQ.0.AND.JTEST2.EQ.0) THEN
         JOFFST = JOFFST + 1
         GO TO 300
      ENDIF
  400 CONTINUE
C
   IF(NPOLYS.LE.1) RETURN
C-----------------------------------------------------------------
      DO 2500 J=1,NPOLYS
C-----------------------------------------------------------------
C         LOOP FOR EACH NODE IN THIS POLYGON
C-----------------------------------------------------------------
      DO 2500 JJ=1,NNODES(J)
C
C          LOOP FOR EACH NODE IN PGRID TO DETERMINE IF THIS NODE
C           IS ALREADY IN THE LIST.
C           - CREATE A KEY FROM THE NORMALIZED X,Y,Z COORDINATE VALUES.
C              MAP THIS KEY INTO AN 8-BIT SPACE SO IT CAN BE UNIQUELY
C              DEFINED BY A ONE WORD KEY.
C           - PUT KEY IN TABLE. IF PUT FAILS, COORDINATE IS (LREQD.GT.MAVAIL) THEN
          JERR = 1
          GO TC 99990
      ENDIF
C
C..       VERIFY THAT THE MAXIMUM NUMBER OF KEYS ALLOWED DOES
C..        NOT EXCEED THE TABLE LENGTH
C
      IF (MXITMS.LT.1.OR.MXITMS.GT.LENTBL) CALL UABORT
     +   ('*A*MAX ITEMS EXCEEDS TABLE LENGTH',33)
      MXKEYS = MXITMS
C
C..       *** ENTRY POINT *UNITBL* ***
C
      ENTRY UNITBL(MEMORY)
C
C.        INIT THE NUMBER OF KEYS IN THE TABLE TO ZERO
C
      NINTBL = 0
C
C..       INITIALIZE HASH TABLES
C
      DO 900 J=1,LENTBL
      MEMORY(IMAPHE+J) = 0
  900 CONTINUE
C
C  * * * * * * *
C   R E T U R N
C  * * * * * * *
C
99990 RETURN
      END
      SUBROUTINE UNITOE(JINTNB,JKEY,MEMORY)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:   HASHING SUBSYTEM
CXX
CXX   PURPOSE:
CXX       TO RETURN THE EXTERANL KEY FOR A GIVEN INTERNAL NUMBER.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX       ARGUMENT  TYPE  I/O  DESCRIPTION
CXX        JINTNB    I     U   INTERNAL NUMBER ASSOCIATED WITH *JKEY*.
CXX        JKEY      I     I   EXTERNAL KEY.
CXX                             0=JINTNB NOT IN USE.
CXX        MEMORY()  ?    I/O  MEMRORY ARRAY CONTAINING THE
CXX                             HASHING DATA BASE
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX       /CHASH/   CONTAINS THE HASH TABLE - SEE *UNINIT*
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX       THIS MODULE VERIFIES THAT THE INTERNAL NUMBER IS IN USE,
CXX        THEN LOOKS UP THE EXTERNAL KEY AND RETURNS IT TO THE
CXX        CALLER.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX       *A*INVALID INTERNAL NUMBER IN *UNITOE*
CXX
```

```
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CHASH/  IMAPHE,LENTBL,NINTBL,MXKEYS,MAXKEY,JOFFST,JDIVSR
C
      DIMENSION MEMORY(*)
C
C  * * * * * * * * * * * * * * * * * * *
C   P R O C E E D U R E   S E C T I O N
C  * * * * * * * * * * * * * * * * * * *
C
C
C..         VERIFY REQUEST
C
      IF (JINTNB.LT.1.OR.JINTNB.GT.LENTBL)  CALL UABORT
     +   ('*A*INVALID INTERNAL NUMBER IN *UHITOE*',88)
C
C..         INITIALIZE
C
      JKEY = 0
C
C..         DONE IF INTERNAL NUMBER IS NOT IN USE
C
      IF (MEMORY(IMAPHE+JINTNB).LT.1) GO TO 99990
C
C..         GET THE EXTERNAL KEY
C
      JKEY = MEMORY(IMAPHE+JINTNB)
C
C  * * * * * * *
C   R E T U R N
C  * * * * * * *
C
99990 RETURN
      END
      SUBROUTINE UHLOOK(JKEY,JINDEX,JWHAT,MEMORY)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:   HASHING SUBSYTEM
CXX
CXX   PURPOSE:
CXX        TO RETURN THE HASH TABLE INDEX FOR AN EXTERNAL KEY.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX      ARGUMENT  TYPE  I/O  DESCRIPTION
CXX        JKEY      I     I   EXTERNAL KEY TO PROCESS
CXX        JINDEX    I     O   TABLE INDEX FOR THE EXTERNAL KEY *JKEY*.
CXX        JWHAT     I     O   WHAT FLAG:
CXX                             1=*JKEY* IS IN THE TABLE.
CXX                               *JINDEX* = THE HASH TABLE INDEX
CXX                               WHERE *JKEY* RESIDES.
CXX                             2=*JKEY* IS NOT IN THE TABLE.
CXX                               *JINDEX* = THE AVAILABLE TABLE INDEX
CXX                               IF *JKEY* IS TO BE ENTERED.
CXX                             3=*JKEY* NOT IN TABLE AND TABLE IS FULL.
CXX        MEMORY()  I    I/O  MEMRORY ARRAY CONTAINING THE
CXX                               HASHING DATA BASE
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX        /CHASH/   CONTAINS THE HASH TABLE - SEE *UHINIT*
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX        NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX        THIS MODULE USES AN OPEN HASHING TECHNIQUE.
CXX         ALL FIELDS IN THE HASH TABLE ARE INITIALIZED TO ZERO.
CXX         DELETED ENTRIES ARE MARKED WITH A -1.
CXX         THE HASH CODE IS THE REMAINDER AFTER DIVIDING THE
CXX         INTEGER KEY BY THE LENGTH OF THE TABLE. COLLISIONS
CXX         ARE HANDLED BY SEARCHING THE LIST IN A SEQUENTIAL
CXX         BUT OFFSET FASHION UNTIL THE KEY IS FOUND OR UNTIL
CXX         A ZERO FIELD IS FOUND. IF THE KEY IS NOT FOUND, THE
CXX         MODULE DETERMINES THE INDEX OF THE FIRST AVAILABLE
CXX         SPOT (I.E. DELETED FIELD OR ZERO FIELD) ENCOUNTERED
CXX         WHILE SEARCHING FOR THE KEY AND RETURNS THIS VALUE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX        *A*KEY OUT OF ALLOWABLE RANGE
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CHASH/  IMAPHE,LENTBL,NINTBL,MXKEYS,MAXKEY,JOFFST,JDIVSR
C
C*************************
C*************************
C
      DIMENSION MEMORY(*)
C
C  * * * * * * * * * * * * * * * * * * *
C   P R O C E E D U R E   S E C T I O N
C  * * * * * * * * * * * * * * * * * * *
C
C
C..         INITIALIZE
C
      JINDEX = 0
      JFIRST = 0
      NCHEKS = 0
C
```

```
C..        HASH TO OBTAIN THE TABLE INDEX AT WHICH TO START LOOKING
C..         FOR THE EXTERNAL NUMBER. (THIS HASH CODE IS TO DIVIDE BY
C..         A PRIME DIVISOR JUST SMALLER THAN THE LENGTH OF THE TABLE
C..         AND USE THE REMAINDER AS THE CODE)
C
      JINDEX = MOD(JKEY-1,JDIVSR) + 1
C
C..        INCREMENT THE NUMBER OF CHECKS FOR THIS EXTERNAL NUMBER
C..         AND CHECK THE TABLE AT THIS INDEX.
C
 1000 CONTINUE
      NCHEKS = NCHEKS + 1
C
C.         IF EXTERNAL NUMBER AT THIS INDEX, SET WHAT FLAG AND RETURN.
C
      IF (MEMORY(IMAPHE+JINDEX).EQ.JKEY) THEN
         JWHAT = 1
         GO TO 99990
      ENDIF
C
C.         IF THIS IS A ZERO FIELD, INDICATE THAT THE KEY WAS NOT FOUND.
C.          RETURN THE FIRST AVAILABLE FIELD FOUND WHILE SEARCHING AS
C.          THE INDEX AND RETURN.
C.
C.         IF THIS IS NOT A ZERO FIELD (AND IS NOT THE KEY), THEN
C.          IF IT IS THE FIRST DELETED FIELD ENCOUNTERED IN THE
C.          SEARCH, STORE THE INDEX IN •JFIRST• AND CONTINUE SEARCHING.
C
      IF (MEMORY(IMAPHE+JINDEX).EQ.0) THEN
         JWHAT = 2
         IF (JFIRST.NE.0) JINDEX = JFIRST
         GO TO 99990
      ELSEIF (MEMORY(IMAPHE+JINDEX).LT.0) THEN
         IF (JFIRST.EQ.0) JFIRST = JINDEX
      ENDIF
C
C.         THE TABLE HAS BEEN COMPLETELY SEARCHED AND THE KEY
C.          HAS NOT BEEN FOUND.
C.           - IF NO AVAILABLE SPOTS HAVE BEEN FOUND DURING THE
C.             SEARCHING PROCESS, THEN THE TABLE IS FULL. SET
C.             FLAG AND RETURN.
C..          - IF AN AVAILABLE SPOT WAS FOUND DURING SEARCHING, SET
C..            THE INDEX TO THIS VALUE AND RETURN.
C
      IF (NCHEKS.LT.LENTBL) GO TO 9000
      IF (JFIRST.EQ.0) THEN
         JWHAT = 3
      ELSE
         JWHAT = 2
         JINDEX = JFIRST
      ENDIF
      GO TO 99990
C
C..        COLLISION OCCURED - CALCULATE A NEW INDEX TO
C..         LOOK AT AND JUMP TO PROCESS.
C
 9000 CONTINUE
      JINDEX = MOD(JINDEX+JOFFST-1,LENTBL) + 1
```

```
      GO TO 1000
C
C  * * * * * * *
C   R E T U R N
C  * * * * * * *
C
99990 RETURN
      END
      SUBROUTINE UNPUT(JKEY,JINTNB,MEMORY,JERR)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:   HASHING SUBSYSTEM
CXX
CXX   PURPOSE:
CXX       TO PUT A KEY INTO THE HASH TABLE AND RETURN THE ASSIGNED
CXX        INTERNAL NUMBER.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX      ARGUMENT  TYPE  I/O  DESCRIPTION
CXX       JKEY      I     I   EXTERNAL INTEGER KEY.
CXX       JINTNB    I     O   INTERNAL NUMBER ASSIGNED TO THE KEY.
CXX       MEMORY()  I    I/O  MEMRORY ARRAY CONTAINING THE
CXX                            HASHING DATA BASE.
CXX       JERR      I     O   ERROR FLAG:
CXX                            0=NO ERRORS
CXX                            1=TABLE ALREADY FULL
CXX                            2=•JKEY• ALREADY IN THE TABLE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX       /CHASH/   CONTAINS THE HASH TABLE - SEE •UHINIT•
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX        NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX       THIS MODULE PUTS A NEW KEY INTO THE HASH TABLE AND THEN
CXX        ASSIGNS THE NEXT AVAILABLE EXTERNAL NUMBER TO THE KEY
CXX        THE INTERNAL NUMBER IS THE INDEX USED TO ACCESS THE KEYS
CXX        RELATED INFORMATION IN A SEPARATE TABLE.
CXX
CXX       THE KEYS IN THE HASH TABLE ARE MAPPED TO THE INTERNAL NUMBER
CXX        BY MEANS OF A SEPARATE TABLE (•IMAPHI•). THE HASH INDEX FOR
CXX        THE KEY IS THE LOCATION OF THE INTERNAL NUMBER IN THE TABLE.
CXX
CXX       THE INTERNAL NUMBERS ARE MAPPED TO THE HASH TABLE KEYS BY
CXX        MEANS OF ANOTHER TABLE (•IMAPIH•). THE INTERNAL NUMBER IS
CXX        THE LOCATION IN THE HASH TABLE OF THE EXTERNAL KEY.
```

```
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX       *A*KEY INVALID
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CHASH/  IMAPHE,LENTBL,NINTBL,MXKEYS,MAXKEY,JOFFST,JDIVSR
C
      DIMENSION MEMORY(*)
C
C  * * * * * * * * * * * * * * * * * * *
C   P R O C E E D U R E   S E C T I O N
C  * * * * * * * * * * * * * * * * * * *
C
C
C..       INITIALIZE
C
      JERR = 0
C
C..       ABORT IF KEY IS OUT OF ALLOWABLE RANGE
C
      IF (JKEY.LT.1.OR.JKEY.GT.MAXKEY) CALL UABORT
     +   ('*A*BAD KEY PASSED TO *UHPUT*',28)
C
C..       ERROR IF TABLE ALREADY FULL
C
      IF (NINTBL.GE.MXKEYS) THEN
         JERR = 1
         GO TO 99990
      ENDIF
C
C..       INITIALIZE THE SUBSYSTEM IF THE TABLE IS EMPTY
C
      IF (NINTBL.EQ.0) CALL UHITBL(MEMORY)
C
C..       CALL TO GET THE LOOK-UP TABLE INDEX
C
      CALL UHLOOK(JKEY,JCODE,JWHAT,MEMORY)
      IF (JWHAT.NE.2) GO TO 90000
C
C..       UPDATE THE HASH TABLES
C
C.        PUT THE EXTERNAL NUMBER INTO THE TABLE (*IMAPHE*)
C
      CALL UHUPUT(JKEY,JCODE,JINTNB,MEMORY)
C
C..       DONE - NO ERRORS
C
      GO TO 99990
C
C..       PROCESS ERRORS
C
90000 CONTINUE
      IF (JWHAT.EQ.1) THEN
         JERR = 2
         GO TO 99990
      ELSEIF (JWHAT.EQ.3) THEN
         CALL UABORT('*A*FULL TABLE NOT PREVIOUSLY DETECTED',37)
      ELSE
         CALL UABORT('*A*UNKNOWN ERROR FLAG FROM *UHLOOK*',35)
      ENDIF
C
C  * * * * * * *
C   R E T U R N
C  * * * * * * *
C
99990 RETURN
      END
      SUBROUTINE UHUDEL(JINDEX,JINTNB,MEMORY)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:   HASHING SUBSYTEM
CXX
CXX   PURPOSE:
CXX       TO DELETE A KEY FROM THE HASH TABLE AT A GIVEN ADDRESS.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX      ARGUMENT  TYPE  I/O  DESCRIPTION
CXX       JINDEX    I     I    HASH TABLE ADDRESS TO DELETE *JKEY* FROM.
CXX       JINTNB    I     O    INTERNAL NUMBER ASSOCIATED WITH THE KEY.
CXX       MEMORY()  I    I/O   MEMRORY ARRAY CONTAINING THE
CXX                             HASHING DATA BASE
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX       /CHASH/   CONTAINS THE HASH TABLE - SEE *UHINIT*
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX       THIS MODULE DELETES THE KEY RESIDING IN THE HASH TABLE
CXX        AT THE GIVEN ADDRESS.
CXX
CXX       DELETED TABLE ENTRIES ARE ALWAYS REPLACED BY A -1.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX       *A*INVALID HASH TABLE ADDRESS
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CHASH/  IMAPHE,LENTBL,NINTBL,MXKEYS,MAXKEY,JOFFST,JDIVSR
C
      DIMENSION MEMORY(*)
C
C  * * * * * * * * * * * * * * * * * * *
C   P R O C E D U R E   S E C T I O N
C  * * * * * * * * * * * * * * * * * * *
C
C
C..        VERIFY THE REQUEST
C
      IF (JINDEX.LT.1.OR.JINDEX.GT.LENTBL) CALL UABORT
     *   ('*A*INVALID HASH TABLE ADDRESS',20)
C
      JKEY = MEMORY(IMAPHE+JINDEX)
      IF (JKEY.LT.1.OR.JKEY.GT.MAXKEY) CALL UABORT
     *   ('*A*ATTEMPTED TO DELETE INVALID KEY FROM HASH TABLE',50)
C
C..        DELETE THE KEY FROM THE TABLE
C
      MEMORY(IMAPHE+JINDEX) = -1
C
C.         PLACE THE ASSOCIATED INTERNAL NUMBER IN THE GARBAGE STACK
C.          AND ZERO OUT THE ASSOCIATED FIELDS IN THE INTERNAL TO
C.          HASH CODE MAPPING TABLES.
C
      JINTNB = JINDEX
C
C..         - DECREMENT THE NUMBER OF ITEMS IN THE TABLE
C
      NINTBL = NINTBL - 1
C
C  * * * * * * *
C   R E T U R N
C  * * * * * * *
C
99999 RETURN
      END
      SUBROUTINE UHPUT(JKEY,JINDEX,JINTNB,MEMORY)
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXX BEGIN PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   SYSTEM:    HASHING SUBSYTEM
CXX
CXX   PURPOSE:
CXX       TO PUT A KEY INTO THE HASH TABLE AT A GIVEN ADDRESS.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   CALL PARAMETERS:
CXX      ARGUMENT  TYPE  I/O  DESCRIPTION
CXX        JKEY      I    I   INTEGER KEY TO ENTER INTO THE HASH TABLE.
CXX        JINDEX    I    I   HASH TABLE ADDRESS TO PUT *JKEY* INTO.
CXX        JINTNB    I    O   INTERNAL NUMBER ASSIGNED TO THE KEY.
CXX        MEMORY()  I   I/O  MEMORY ARRAY CONTAINING THE
CXX                            HASHING DATA BASE
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   COMMON USAGE:
CXX
CXX       /CHASH/   CONTAINS THE HASH TABLE - SEE *UHINIT*
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   INTERFACE REQUIREMENTS:
CXX       .NONE.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   MODULE DESCRIPTION:
CXX       THIS MODULE PUTS A GIVEN INTEGER KEY INTO THE HASH TABLE
CXX       AT THE GIVEN ADDRESS.
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
CXX   ERROR MESSAGES:
CXX       *A*INVALID HASH TABLE ADDRESS
CXX       *A*INVALID HASH KEY
CXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXX END PROLOGUE XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CXX
      COMMON /CHASH/  IMAPHE,LENTBL,NINTBL,MXKEYS,MAXKEY,JOFFST
C
      DIMENSION MEMORY(*)
C
C  * * * * * * * * * * * * * * * * * * *
C   P R O C E D U R E   S E C T I O N
C  * * * * * * * * * * * * * * * * * * *
C
C
C..        VERIFY THE REQUEST
C
      IF (JKEY.LT.1.OR.JKEY.GT.MAXKEY) CALL UABORT
     *   ('*A*INVALID HASH KEY',19)
C
      IF (JINDEX.LT.1.OR.JINDEX.GT.LENTBL) CALL UABORT
     *   ('*A*INVALID HASH TABLE ADDRESS',29)
C
C..        PUT THE KEY INTO THE TABLE
C
      MEMORY(IMAPHE+JINDEX) = JKEY
C
      JINTNB = JINDEX
C
C..         - INCREMENT THE NUMBER OF ITEMS IN THE TABLE
C
      NINTBL = NINTBL + 1
C
C  * * * * * * *
```

```
C   R E T U R N
C   . . . . . . .
C
99990 RETURN
      END
```

APPENDIX D

MOVIE.BYU Interface Code

```
      SUBROUTINE GVPHCD(ICEN)
C
C**********************************************************************
C
C  SUBROUTINE GVPHCD - GETS COORDINATES AND NORMALS FOR POLYGONS
C     INCLUDING SPECIFIED OR AVERAGE CENTER POINT IF NEEDED
C
C**********************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHPT    = PROCESSES POLYGONS BY PART
C
C**********************************************************************
C
C  SUBPROGRAMS CALLED
C     NORMAL = CALCULATES NORMALS AT NODES
C     POINTS = SETS UP POLYGON COORDINATE ARRAY
C     SHRINK = MOVES NODES OF POLYGONS TOWARD CENTER OF POLYGON
C
C**********************************************************************
C
C  VARIABLES USED
C     ICEN   = ELEMENT(POLYGON) NUMBER
C     IPOOR  = IF .TRUE. INVOKES POOR MANS HIDDEN SURFACE REMOVAL
C     LINELN = 1  FOR LINE ELEMENTS (IGNORE EXCEPT ON A DRAW WITH
C              NO POOR MANS)
C            = 0  FOR ALL OTHER ELEMENTS
C     MEDGE  = NUMBER OF SIDES OF POLYGON + 1
C     NEDGE  = NUMBER OF SIDES OF POLYGON
C     SHRK   = SHRINK FACTOR
C     XP     = COORDINATE ARRAY FOR A POLYGON
C
C**********************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHCD
C
C     MAXNS1 = MAX NUMBER OF POLYGON NODES INCLUDING THE CENTER NODE
C
      INCLUDE '../dparam.h'
C
C**********************************************************************
C
      COMMON/COOR/     XP(3,MAXNS1)
      COMMON/DELTH/    DELTA,SHRK
      COMMON/LIN/      LINELM
      COMMON/LOGI/     ISMOTH,IPOOR,IMIX,DIRC,ISPEC,ISMA,LINEAR,
     $                 IHLR,IFRING
      COMMON/NEME/     NEDGE,MEDGE,IPART
      LOGICAL          IPOOR
C
C                          ***************************
C                          ******BEGIN PROCESSING*****
C                          ***************************
C
      CALL POINTS(ICEN)
      IF (.NOT.IPOOR.AND.SHRK.EQ.0.0) RETURN
      XEDGE=NEDGE
C----------------------------------------------------------------------
C     CENTER POINT COORDINATE CALCULATED = AVERAGE OF
C     OTHER NODES (UNLESS SPECIFIED IN DATA). NEED THIS NODE TO HAVE
C     SOMETHING TO SHRINK ELEMENT AROUND.
C----------------------------------------------------------------------
      DO 10 I=1,3
   10 XP(I,MEDGE)=0.0
      DO 20 I=1,NEDGE
        DO 20 J=1,3
   20 XP(J,MEDGE)=XP(J,MEDGE)+XP(J,I)
      DO 30 I=1,3
   30 XP(I,MEDGE)=XP(I,MEDGE)/XEDGE
C----------------------------------------------------------------------
C     CALL SHRINK IF SPECIFIED
C----------------------------------------------------------------------
      IF(SHRK.EQ.0.0) GO TO 50
      CALL SHRINK
   50 IF(LINELM.EQ.1) RETURN
C----------------------------------------------------------------------
C     GET NORMALS FOR LATER POORMANS IF REQUIRED
C----------------------------------------------------------------------
      CALL NORMAL
C
      RETURN
      END


      SUBROUTINE GVPHCL
C
C**********************************************************************
C
C  SUBROUTINE GVPHCL - CLIPS EDGES TO ALL CLIPPING BOUNDARYS
C
C**********************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHPS - LOADS CLIPPED POLYGON INFORMATION INTO JONESD HOLDING
C              ARRAYS
C
C**********************************************************************
C
C  SUBPROGRAMS CALLED
C     NONE
C
C**********************************************************************
C
C  VARIABLES USED
C     IC          = EDGE COLOR VALUES OR GLOBAL NODE NUMBERS
C     ICNT        = COUNT OF(3*NUMBER OF EDGES IN A POLYGON) + ADDED
C                   EDGES DUE TO CLIPPING
C     T1, T2      = Z-CLIP PARAMETERS
C     VC          = FUNCTION VALUES AT CLIPPED ENDS
C     VX,VY,VZ    = EDGE STACK COORDINATES
C
C**********************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHCL
C
      INCLUDE '../dparam.h'
```

```
C
C     MAXCLP = MAX NUMBER OF CLIPPED EDGES IN A POLYGON
C
C***********************************************************************
C
      COMMON/COMMIO/ ICNT,NT,VX(MAXCLP),VY(MAXCLP),VZ(MAXCLP),VN(MAXCLP)
     *               ,IC(MAXCLP),VC(MAXCLP),VTX(MAXCLP),VTY(MAXCLP),
     *               VTZ(MAXCLP),VTN(MAXCLP),VTC(MAXCLP),ITC(MAXCLP),
     *               JC(MAXCLP)
C
      COMMON/SNPDAT/T1,T2,I
C
C                    ****************************
C                    ******BEGIN PROCESSING******
C                    ****************************
C
C-----------------------------------------------------------------------
C           RETURN IF LINE IS OUTSIDE FRUSTUM OF VISION
C-----------------------------------------------------------------------
C
      IF(IC(I).LT.33554431) RETURN
C
C-----------------------------------------------------------------------
C        RETURN IF LINE DOES NOT INTERSECT PLANE, OTHERWISE CLIP
C-----------------------------------------------------------------------
C
      IF(T1.LT.0.) THEN
        IF(T2.LT.0.) THEN
          IC(I)=0
          IC(I+1)=0
          JC(I)=0
          JC(I+1)=0
          RETURN
        END IF
      ELSE
        IF(T2.GE.0.) RETURN
      END IF
C
C-----------------------------------------------------------------------
C                    THE LINE IS TO BE CLIPPED
C-----------------------------------------------------------------------
C
      ALPHA=T1/(T1-T2)
C
C-----------------------------------------------------------------------
C        DETERMINE WHICH INDEX WILL RECEIVE THE CLIPPED POINT
C-----------------------------------------------------------------------
C
      I1=I+1
      IF(T1.LT.0.0) I1=I
C
C-----------------------------------------------------------------------
C                             CLIP
C-----------------------------------------------------------------------
C
      VX(I1)=ALPHA*(VX(I+1)-VX(I))+VX(I)
      VY(I1)=ALPHA*(VY(I+1)-VY(I))+VY(I)
      VZ(I1)=ALPHA*(VZ(I+1)-VZ(I))+VZ(I)
      IF(IC(I).EQ.IC(I+1)) THEN
        IC(I1)=IC(I)
      END IF
      VC(I1)=ALPHA*(VC(I+1)-VC(I))+VC(I)
C
C-----------------------------------------------------------------------
C                   JUMP IF EDGE STACK IS FULL
C-----------------------------------------------------------------------
C
      IF(ICNT.LT.32) THEN
        ICNT=ICNT+1
        VX(ICNT)=VX(I1)
        VY(ICNT)=VY(I1)
        VZ(ICNT)=VZ(I1)
        VC(ICNT)=VC(I1)
        IC(ICNT)=MOD(IC(I1),16777216)+33554432
        RETURN
      END IF
      PRINT*,'EDGE STACK FULL'
      RETURN
      END


      SUBROUTINE GVPHED(I1,I2)
C
C***********************************************************************
C
C  SUBROUTINE GVPHED - COLLECTS COORDINATE AND FUNCTION INFORMATION
C                      FOR AN EDGE AND SHIPS IT ON.  ALSO DOES PART
C                      OF THE PERSPECTIVE TRANSFORMATION OF THE Z
C                      COORDINATES OF EDGE ENDPOINTS.
C
C***********************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHPV  = SENDS POLYGON EDGES ON FOR HIDDEN LINE PROCESSING
C
C***********************************************************************
C
C  SUBPROGRAMS CALLED
C     GVPHEM = STORES POLYGON EDGES FOR LATER PROCESSING
C
C***********************************************************************
C
C  VARIABLES USED
C     CB,CE   = BEGINNING AND ENDING CONTOUR VALUES
C     CONT    = CONTOUR ARRAY
C     DOZ     = DISTANCE TO ORIGIN
C     TANAL   = TANGENT OF PERSPECTIVE HALF ANGLE
C     XB,XE   = BEGINNING AND ENDING (X) EDGE COORDINATES
C     YB,YE   = BEGINNING AND ENDING (Y) EDGE COORDINATE
C     ZB,ZE   = BEGINNING AND ENDING (Z) EDGE COORDINATE
C     XP      = COORDINATE ARRAY FOR POLYGON
C
C***********************************************************************
C
C   VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHED
C
```

```
      INCLUDE '../dparam.h'
C
C     MAXNS1 = MAXIMUM NUMBER OF EDGES IN A POLYGON INCLUDING THE CENTER
C              NODE
C
C**********************************************************************
C
      COMMON/CLIPS/  XB,YB,ZB,KB,CB,XE,YE,ZE,KE,CE,LAS,ISHARE,NTR,
     $               LD,LE
      COMMON/CONTO/  CONT(MAXNS1)
      COMMON/COOR/   XP(3,MAXNS1)
      COMMON/BCMR/   DOZ,FIOLD,FIELD,TANAL,RES
C
C                    ****************************
C                    ******BEGIN PROCESSING******
C                    ****************************
C
C---------------------------------------------------------------------
C     DO A LITTLE PERSPECTIVE WORK ON THE Z COORDS OF THIS EDGE.  PICK
C     UP THE EDGE ENDPOINT COORDINATES AND FUNCTION VALUES
C---------------------------------------------------------------------
C
      XB=XP(1,I1)
      YB=XP(2,I1)
      ZB=(DOZ-XP(3,I1))*TANAL
      XE=XP(1,I2)
      YE=XP(2,I2)
      ZE=(DOZ-XP(3,I2))*TANAL
      CB=(XB-XE)+(YB-YE)+(ZB-ZE)
      IF(CB.EQ.0.0) RETURN
      CB=CONT(I1)
      CE=CONT(I2)
C
C---------------------------------------------------------------------
C       CALL THE EDGMAK ROUTINE WHICH COULD PROBABLY BE SKIPPED
C---------------------------------------------------------------------
C
      CALL GVPHEM
C
      RETURN
      END


      SUBROUTINE GVPHEM
C
C**********************************************************************
C
C  SUBROUTINE CVPHEM - LOAD EDGE STACK AND SET UP VISIBILITY FLAG
C
C**********************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHED - COLLECTS ENDPOINT COORDINATE DATA AND FUNCTION DATA FOR
C              AN EDGE AND SHIPS IT ON
C
C**********************************************************************
C
C  SUBPROGRAMS CALLED
```

```
C     GVPHPS - SETS UP PARAMATERS FOR CALL TO CLIP AND LOADS JONESD
C              HOLDING ARRAYS
C
C**********************************************************************
C
C  VARIABLES USED
C     C1,C2     = BEGINNING AND ENDING CONTOUR VALUE
C     IC        = VISIBILITY FLAG
C     ICNT      = COUNT OF(2*NUMBER OF EDGES IN A POLYGON) + ADDED
C                 EDGES DUE TO CLIPPING
C     IPOLY     = COUNT OF POLYGONS
C     LASEDG    = IF .TRUE., LAST EDGE
C     VC        = FUNCTION VALUE
C     VX,VY,VZ  = EDGE STACK COORDINATES
C     X1,X2     = BEGINNING AND ENDING (X) EDGE COORDINATE
C     Y1,Y2     = BEGINNING AND ENDING (Y) EDGE COORDINATE
C     Z1,Z2     = BEGINNING AND ENDING (Z) EDGE COORDINATE
C
C**********************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHEM
C
      INCLUDE '../dparam.h'
C
C     MAXCLP = MAXIMUM NUMBER OF CLIPPED LINE SEGMENTS
C
C**********************************************************************
C
      LOGICAL        LASEDG,ISHARE
      COMMON/CLIPS/  X1,Y1,Z1,K1,C1,X2,Y2,Z2,K2,C2,LASEDG,ISHARE,NTR,
     $               ITR1,ITR2
C
      COMMON/COMMIO/ ICNT,NT,VX(MAXCLP),VY(MAXCLP),VZ(MAXCLP),
     $               VN(MAXCLP),IC(MAXCLP),VC(MAXCLP),VTX(MAXCLP),
     $               VTY(MAXCLP),VTZ(MAXCLP),VTN(MAXCLP),VTC(MAXCLP),
     $               ITC(MAXCLP),JC(MAXCLP)
C
C                     ****************************
C                     ******BEGIN PROCESSING******
C                     ****************************
C
C---------------------------------------------------------------------
C                 JUMP IF EDGE STACK WILL OVERFLOW
C---------------------------------------------------------------------
C
        IF(ICNT.LT. (MAXCLP-1)) THEN
C
C---------------------------------------------------------------------
C     SET 10TH BIT IF EDGE IS SHARED AND SET 10TH BIT FOR EDGE IS
C     VISIBLE FLAG.  I REALLY DON'T KNOW WHAT IS GOING ON HERE OTHER
C     THAN IT SEEMS TO WORK.  THIS HAS TO DO WITH PACKING INFORMATION
C     INTO AS FEW WORDS AS POSSIBLE.
C---------------------------------------------------------------------
C
         I=33554432
         IF(ISHARE) I=50331648
C
C---------------------------------------------------------------------
```

```
C                    PUT BEGIN POINT INTO EDGE STACK
C------------------------------------------------------------------------
C
          ICNT=ICNT+1
          VX(ICNT)=X1
          VY(ICNT)=Y1
          VZ(ICNT)=Z1
          VC(ICNT)=C1
          IC(ICNT)=I
C
C------------------------------------------------------------------------
C                      PUT END POINT INTO EDGE STACK
C------------------------------------------------------------------------
C
          ICNT=ICNT+1
          VX(ICNT)=X2
          VY(ICNT)=Y2
          VZ(ICNT)=Z2
          VC(ICNT)=C2
          IC(ICNT)=I
C
C------------------------------------------------------------------------
C        CALL GVPHPS IF THIS IS THE LAST EDGE OF THIS POLYGON.
C        WE WILL CLIP THE WHOLE POLYGON AND LOAD IT INTO THE
C        JONESD HOLDING ARRAYS IN GVPHPS
C------------------------------------------------------------------------
C
          IF(LASEDG) CALL GVPHPS
        ELSE
          PRINT*, 'MAXCLP = ',MAXCLP,'EXCEEDED'
        END IF
        RETURN
      END


      SUBROUTINE GVPHLD
C
C************************************************************************
C
C  SUBROUTINE GVPHLD - ORDERS THE EDGES OF ANY CLIPPED POLYGONS AND
C                      LOADS THE APPROPRIATE ARRAYS FOR THE JONESD
C                      HIDDEN ROUTINE
C
C************************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHPV - SENDS POLYGON EDGES ON FOR HIDDEN LINE PROCESSING
C
C************************************************************************
C
C  SUBPROGRAMS CALLED
C     NONE
C
C************************************************************************
C
C  VARIABLES USED
C     CCONT  = HOLDING LOC FOR CONTOUR VALUES FOR JONESD HIDDEN
C     HOLDX  = HOLDING LOC FOR X EDGE COORDS FOR JONESD HIDDEN
C     HOLDY  = HOLDING LOC FOR Y EDGE COORDS FOR JONESD HIDDEN
C     HOLDZ  = HOLDING LOC FOR Z EDGE COORDS FOR JONESD HIDDEN
C     NMNOD  = NUMBER OF NODES IN THE POLYGON BEING PROCESSED
C     NNODES = NUMBER OF NODES IN A POLYGON
C     NPOLY  = COUNTS NUMBER OF POLYGONS
C     TEMP*  = TEMPORARY ARRAYS USED TO STORE CLIPPED EDGE
C              INFORMATION SO THAT IT CAN BE ORDERED IN THIS
C              ROUTINE AND LOADED INTO THE JONESD HOLDING ARRAYS.
C              IT COMES FROM GVPHPS.
C     *KEEP  = HOLDING VARIABLES USED IN ORDERING POLYGON EDGES
C
C************************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR THE SUBROUTINE GVPHLD
C
      INCLUDE '/g/rt/work/thesis/working/dparam.h'
C
C     MAXNPT = MAX NUMBER OF ELEMENTS ALLOWED
C     MAXNS  = MAX NUMBER OF NODES ALLOWED IN A POLYGON
C
C************************************************************************
C
      COMMON/GHID/   HOLDX(MAXNS,MAXNPT),
     $               HOLDY(MAXNS,MAXNPT),
     $               HOLDZ(MAXNS,MAXNPT),
     $               CCONT(MAXNS,MAXNPT),
     $               NNODES(MAXNPT)
      COMMON/SPARE/  NMNOD,NPOLY
      COMMON/SORT/   TEMPX1(MAXNS),
     $               TEMPY1(MAXNS),
     $               TEMPZ1(MAXNS),
     $               TCONT1(MAXNS),
     $               TEMPX2(MAXNS),
     $               TEMPY2(MAXNS),
     $               TEMPZ2(MAXNS),
     $               TCONT2(MAXNS)
C
C                   ************************
C                   ****BEGIN PROCESSING****
C                   ************************
C
C------------------------------------------------------------------------
C     LOAD IN THE COORDS OF ONE END OF AN EDGE IN THIS POLYGON AND
C     START TO ATTACH THE OTHER EDGES OF THE POLYGON TO IT IN ORDER.
C     WE'RE JOINING EDGES WITH THE SAME COORDINATE ENDPOINTS.
C------------------------------------------------------------------------
C
      DO 8000 I=1,NMNOD-1
        HOLDX2=TEMPX2(I)
        HOLDY2=TEMPY2(I)
        HOLDZ2=TEMPZ2(I)
        DO 2500 J=I+1,NMNOD
          IF(ABS(HOLDX2-TEMPX1(J)).LT.(.1)
     $      .AND.ABS(HOLDY2-TEMPY1(J)).LT.(.1)
     $      .AND.ABS(HOLDZ2-TEMPZ1(J)).LT.(.1)) THEN
C             PRINT*,'ORDERING'
            IF(J.EQ.(I+1)) GO TO 8000
            X1KEEP=TEMPX1(I+1)
```

```
              Y1KEEP=TEMPY1(I+1)
              Z1KEEP=TEMPZ1(I+1)
              C1KEEP=TCONT1(I+1)
              X2KEEP=TEMPX2(I+1)
              Y2KEEP=TEMPY2(I+1)
              Z2KEEP=TEMPZ2(I+1)
              C2KEEP=TCONT2(I+1)
              TEMPX1(I+1)=TEMPX1(J)
              TEMPY1(I+1)=TEMPY1(J)
              TEMPZ1(I+1)=TEMPZ1(J)
              TCONT1(I+1)=TCONT1(J)
              TEMPX2(I+1)=TEMPX2(J)
              TEMPY2(I+1)=TEMPY2(J)
              TEMPZ2(I+1)=TEMPZ2(J)
              TCONT2(I+1)=TCONT2(J)
              TEMPX1(J)=X1KEEP
              TEMPY1(J)=Y1KEEP
              TEMPZ1(J)=Z1KEEP
              TCONT1(J)=C1KEEP
              TEMPX2(J)=X2KEEP
              TEMPY2(J)=Y2KEEP
              TEMPZ2(J)=Z2KEEP
              TCONT2(J)=C2KEEP
              GO TO 3000
            ELSE IF(ABS(HOLDX2-TEMPX2(J)).LT.(.1)
     $       .AND.ABS(HOLDY2-TEMPY2(J)).LT.(.1)
     $       .AND.ABS(HOLDZ2-TEMPZ2(J)).LT.(.1)) THEN
C               PRINT*,'SWITCHED ENDS'
              X1KEEP=TEMPX1(I+1)
              Y1KEEP=TEMPY1(I+1)
              Z1KEEP=TEMPZ1(I+1)
              C1KEEP=TCONT1(I+1)
              X2KEEP=TEMPX2(I+1)
              Y2KEEP=TEMPY2(I+1)
              Z2KEEP=TEMPZ2(I+1)
              C2KEEP=TCONT2(I+1)
              TEMPX1(I+1)=TEMPX2(J)
              TEMPY1(I+1)=TEMPY2(J)
              TEMPZ1(I+1)=TEMPZ2(J)
              TCONT1(I+1)=TCONT2(J)
              TEMPX2(I+1)=TEMPX1(J)
              TEMPY2(I+1)=TEMPY1(J)
              TEMPZ2(I+1)=TEMPZ1(J)
              TCONT2(I+1)=TCONT1(J)
              TEMPX1(J)=X2KEEP
              TEMPY1(J)=Y2KEEP
              TEMPZ1(J)=Z2KEEP
              TCONT1(J)=C2KEEP
              TEMPX2(J)=X1KEEP
              TEMPY2(J)=Y1KEEP
              TEMPZ2(J)=Z1KEEP
              TCONT2(J)=C1KEEP
              GO TO 3000
            END IF
 2500   CONTINUE
 3000 CONTINUE
C
C----------------------------------------------------------------------
C     LOAD THE JONESD HOLDING ARRAYS WITH A NICE ORDERED POLYGON
C----------------------------------------------------------------------
C
      DO 4000 J=1,NNNOD
        HOLDX(J,NPOLY)=TEMPX1(J)
        HOLDY(J,NPOLY)=TEMPY1(J)
        HOLDZ(J,NPOLY)=TEMPZ1(J)
        CCONT(J,NPOLY)=TCONT1(J)
C         PRINT*,'NODE',J
C         PRINT*,'X=',TEMPX1(J)
C         PRINT*,'Y=',TEMPY1(J)
C         PRINT*,'Z=',TEMPZ1(J)
 4000 CONTINUE
      RETURN
C
      END


      SUBROUTINE GVPHPS
C
C**********************************************************************
C
C  SUBROUTINE GVPHPS - SETS UP CLIPPING PARAMETERS AND LOADS THE JONESD
C                      HOLDING ARRAYS
C
C**********************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHEM - LOADS UP EDGE DATA AND INITIALIZES VISIBILITY FLAG
C
C**********************************************************************
C
C  SUBPROGRAMS CALLED
C     GVPHCL - CLIPS EDGES TO ALL DEFINED CLIPPING PLANES
C
C**********************************************************************
C
C  VARIABLES USED
C     CURZEE   = Z-CLIP LOGICAL PARAMETER
C     IC       = EDGE COLOR VALUES
C     ICNT     = COUNT OF(2*NUMBER OF EDGES IN A POLYGON) + ADDED
C                EDGES DUE TO CLIPPING
C     IPOLY    = COUNT OF POLYGONS
C     NT       = IF .TRUE., INTERIOR POLYGON
C     VC       = FUNCTION VALUE
C     VX,VY,VZ = EDGE STACK COORDINATES
C     ZMAX     = Z MAXIMUM CLIPPING PLANE LOCATION
C     ZMIN     = Z MINIMUM CLIPPING PLANE LOCATION
C     CCONT  = HOLDING LOC FOR CONTOUR VALUES FOR JONESD HIDDEN
C     HOLDX  = HOLDING LOC FOR X EDGE COORDS FOR JONESD HIDDEN
C     HOLDY  = HOLDING LOC FOR Y EDGE COORDS FOR JONESD HIDDEN
C     HOLDZ  = HOLDING LOC FOR Z EDGE COORDS FOR JONESD HIDDEN
C     NNODES = NUMBER OF NODES IN A POLYGON
C     NNNOD  = COUNTS NUMBER OF EDGES IN THIS POLYGON
C     NPOLY  = COUNTS NUMBER OF POLYGONS
C     TEMP*  = TEMPORARY ARRAYS USED TO STORE CLIPPED EDGE
C              INFORMATION SO THAT IT CAN BE ORDERED IN THIS
C              ROUTINE AND LOADED INTO THE JONESD HOLDING ARRAYS.
```

```
C              IT COMES FROM GVPHPS.
C
C***********************************************************************
C
C   VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHPS
C
      INCLUDE '../dparam.h'
C
C     MAXCLP = MAX NUMBER OF CLIPPED EDGES IN A POLYGON
C     MAXNS  = MAX NUMBER OF NODES IN A POLYGON
C     MAXNPT = MAX NUMBER OF ELEMENTS ALLOWED
C
C***********************************************************************
C
      LOGICAL         NT,VN
      COMMON/COMMIO/  ICNT,NT,VX(MAXCLP),VY(MAXCLP),VZ(MAXCLP),
     $                VN(MAXCLP),IC(MAXCLP),VC(MAXCLP),VTX(MAXCLP),
     $                VTY(MAXCLP),VTZ(MAXCLP),VTN(MAXCLP),
     $                VTC(MAXCLP),ITC(MAXCLP),JC(MAXCLP,
C
      COMMON/CONLEV/  CHI,CLO,NCONLV,CLEVEL(26)
C
      COMMON/GRID/    HOLDX(MAXNS,MAXNPT),
     $                HOLDY(MAXNS,MAXNPT),
     $                HOLDZ(MAXNS,MAXNPT),
     $                CCONT(MAXNS,MAXNPT),
     $                NNODES(MAXNPT)
      COMMON/SPARE/   NNNOD,NPOLY
      COMMON/SORT/    TEMPX1(MAXNS),
     $                TEMPY1(MAXNS),
     $                TEMPZ1(MAXNS),
     $                TCONT1(MAXNS),
     $                TEMPX2(MAXNS),
     $                TEMPY2(MAXNS),
     $                TEMPZ2(MAXNS),
     $                TCONT2(MAXNS)
C
      COMMON/EYES/    XR,YR,IXRES,IYRES,DELINT,DELCON,IFLRCO,ICLGCO
C
      LOGICAL         CURSHD,CURFAC
      COMMON/SHDFG/   CURSHD,CURFAC
C
      COMMON/SHPDAT/  T1,T2,IDS
      COMMON/XNO/     SCOORD(3,MAXNJ)
C
      LOGICAL         CURZEE
      COMMON/ZFIXER/  ZMIN,ZMAX,ZSPRED,CURZEE
C
C                     ****************************
C                     ******BEGIN PROCESSING******
C                     ****************************
C
C----------------------------------------------------------------------
C     JUMP IF CURRENT POLYGON IS THE ZMIN CLIPPED
C----------------------------------------------------------------------
        IF(.NOT.CURZEE) THEN
C----------------------------------------------------------------------
C     CLIP TO THE PLANE Z=ZMIN
C----------------------------------------------------------------------
        IF(.NOT.CURFAC) THEN
        J=ICNT
          DO 10 I=1,J,2
            IDS=I
            T1=VZ(I)-ZMIN
            T2=VZ(I+1)-ZMIN
   10     CALL GVPHCL
        END IF
C----------------------------------------------------------------------
C     CLIP TO THE PLANE Z=ZMAX
C----------------------------------------------------------------------
        J=ICNT
        DO 20 I=1,J,2
          IDS=I
          T1=ZMAX-VZ(I)
          T2=ZMAX-VZ(I+1)
   20   CALL GVPHCL
C----------------------------------------------------------------------
C     CLIP TO THE PLANE Y=Z
C----------------------------------------------------------------------
      END IF
      J=ICNT
      DO 30 I=1,J,2
        IDS=I
        T1=VZ(I,-VY(I)
        T2=VZ(I+1)-VY(I+1)
   30 CALL GVPHCL
C----------------------------------------------------------------------
C     CLIP TO THE PLANE Y=-Z
C----------------------------------------------------------------------
      J=ICNT
      DO 40 I=1,J,2
        IDS=I
        T1=VZ(I)+VY(I)
        T2=VZ(I+1)+VY(I+1)
   40 CALL GVPHCL
C----------------------------------------------------------------------
C     CLIP TO THE PLANE X=Z
C----------------------------------------------------------------------
      J=ICNT
      DO 50 I=1,J,2
        IDS=I
        T1=VZ(I)-VX(I)
        T2=VZ(I+1)-VX(I+1)
   50 CALL GVPHCL
C----------------------------------------------------------------------
C     CLIP TO THE PLANE X=-Z
C----------------------------------------------------------------------
      J=ICNT
      DO 60 I=1,J,2
        IDS=I
        T1=VZ(I)+VX(I)
        T2=VZ(I+1)+VX(I+1)
   60 CALL GVPHCL
C----------------------------------------------------------------------
C     THE CLIPPING IS NOW COMPLETE. GO THROUGH THE LIST
C     OF EDGES AND SEE WHICH ARE OUTSIDE THE FRUSTUM OF VISION.
```

```
C     LOAD THE VISIBLE EDGES INTO THE JONESD HOLDING ARRAYS
C-----------------------------------------------------------------------
        IF(NT) RETURN
          DO 140 I=1,ICNT,2
            IF(IC(I).GE.83554432) THEN
              K=I+1
              L=I
              NMNOD=NMNOD+1
              TEMPX1(NMNOD)=VX(L)*XR/VZ(L)+XR+.1
              TEMPY1(NMNOD)=VY(L)*YR/VZ(L)+YR+.1
              TEMPZ1(NMNOD)=((VZ(L)-ZMIN)/VZ(L))*1022+.1
              TCONT1(NMNOD)=VC(L)
              TEMPX2(NMNOD)=VX(K)*XR/VZ(K)+XR+.1
              TEMPY2(NMNOD)=VY(K)*YR/VZ(K)+YR+.1
              TEMPZ2(NMNOD)=((VZ(K)-ZMIN)/VZ(K))*1022+.1
              TCONT2(NMNOD)=VC(K)
              HOLDX(NMNOD,NPOLY) = TEMPX1(NMNOD)
              HOLDY(NMNOD,NPOLY) = TEMPY1(NMNOD)
              HOLDZ(NMNOD,NPOLY) = TEMPZ1(NMNOD)
              CCONT(NMNOD,NPOLY) = TCONT1(NMNOD)
            END IF
            NNODES(NPOLY)=NMNOD
 140      CONTINUE
          RETURN
      END


      SUBROUTINE GVPHPT
C
C***********************************************************************
C
C  SUBROUTINE GVPHPT - PROCESSES POLYGONS BY PART
C
C***********************************************************************
C
C  SUBROUTINE CALLED BY
C     VIEDRA = CALLS FOR NORMALS,LIGHT INTENSITY,ETC. NEEDED TO
C              DISPLAY SCENE (EITHER VIEW OR DRAW)
C
C***********************************************************************
C
C  SUBPROGRAMS CALLED
C     CONEL   = SETS UP LOCAL CONNECTIVITY FOR POLYGON
C     GVPHCD  = GETS COORDINATES AND NORMALS FOR POLYGONS,INCLUDING
C               SPECIFIED OR AVERAGE CENTER POINT IF NEEDED
C     GVPHPT  = SENDS POLYGON EDGE TO CORRECT SUBROUTINE - DRAW OR
C               VIEW
C     VISIP   = TESTS FOR VISIBILITY OF POLYGONS
C
C***********************************************************************
C
C  VARIABLES USED
C     DELTA   = LOCAL MOTION SCALE FACTOR
C     ICEN    = ELEMENT (POLYGON) NUMBER
C     IHLR    = HIDDEN LINE REMOVAL(LOGICAL VARIABLE)
C     IPART   = PART NUMBER
C     IPOOR   = IF .TRUE. INVOKES POOR MANS HIDDEN SURFACE REMOVAL
C     IVIS    = -1  DO NOT DISPLAY
C             =  0  ON EDGE
C             =  1  DISPLAY
C     JLAST   = LOCATION OF LAST NODE OF PREVIOUS ELEMENT IN TOTAL
C               CONNECTIVITY ARRAY
C     LASP    = CONNECTIVITY ARRAY LOCATION OF LAST NODE OF A PART
C     LINELM  = 1  FOR LINE ELEMENT(IGNORE EXCEPT ON DRAW WITH NO
C                  POOR MANS)
C             = 0  FOR ALL OTHER ELEMENTS
C     MEDGE   = NUMBER OF SIDES OF POLYGON + 1
C     NEDGE   = NUMBER OF SIDES OF POLYGON
C     NNN     = NODE ARRAY FOR POLYGON THAT IS SENT ON FOR DISPLAY
C     NPL     = PARTS ARRAY
C     XD      = TRANSLATION ARRAY
C     XX      = LOCAL MOTION TRANSLATION ARRAY
C     YY      = INCREMENTED TRANSLATION ARRAY
C
C***********************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHPT
      INCLUDE '../dparam.h'
C
C     MAXNP   = MAX NUMBER OF PARTS
C     MAXNP1  = MAX NUMBER OF PARTS + 1 (COORD TRIAD)
C
C***********************************************************************
C
      COMMON/DELSH/  DELTA,SHRK
      COMMON/LAIF/   JLAST,IFLAG
      COMMON/LASPO/  LASP(MAXNP1)
      COMMON/LINES/  LINELM
      COMMON/LOGI/   ISMOTH,IPOOR,INIX,DIRC,ISPEC,ISMA,LINEAR,
     $               IHLR,IFRING
      COMMON/NEME/   NEDGE,MEDGE,IPART
      COMMON/PAAR/   NPL(2,MAXNP),NPLS
      COMMON/VISI/   NNN(MAXNS),IVIS
      DIMENSION      XX(3,MAXNP)
      DIMENSION      XD(3),YY(3)
      LOGICAL        IHLR,IPOOR
C
C                    ****************************
C                    ******BEGIN PROCESSING******
C                    ****************************
C
      I1=NPL(1,IPART)
      I2=NPL(2,IPART)
      YY(1)=XD(1)-DELTA*XX(1,IPART)
      YY(2)=XD(2)-DELTA*XX(2,IPART)
      YY(3)=XD(3)-DELTA*XX(3,IPART)
C
      JLAST=LASP(IPART)
      DO 100 J=I1,I2
C-----------------------------------------------------------------------
C     GET POLYGON CONNECTIVITY
C-----------------------------------------------------------------------
          IVIS=1
          CALL CONEL
          ICEN=J
C-----------------------------------------------------------------------
```

```
C     GET COORDINATES AND NORMALS FOR POLYGON
C-----------------------------------------------------------------------
         CALL GVPHCD(ICEN)
C-----------------------------------------------------------------------
C     IF LINE ELEMENT, IGNORE EXCEPT ON A DRAW
C-----------------------------------------------------------------------
         IF(IHLR.AND.LINELM.EQ.1) GO TO 180
         IF(IPOOR.AND.LINELM.EQ.1) GO TO 180
C
         IF(IHLR.OR.IPOOR) CALL VISIP
         IF(IVIS.LT.0) GO TO 180
         IF(IPOOR) IHID=0
         DO 120 I=1,NEDGE
  120    NNN(I)=I
         CALL GVPHPV
         GO TO 180
  180 CONTINUE
      RETURN
      END


      SUBROUTINE GVPHPV
C
C***********************************************************************
C
C  SUBROUTINE GVPHPV - SENDS POLYGON EDGES ON FOR JONESD HIDDEN LINE
C                      PROCESSING
C
C***********************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHPT    = SENDS POLYGON EDGE TO GET PROCESSED IN PREPARATION
C                 FOR JONESD HIDDEN LINE REMOVAL
C
C***********************************************************************
C
C  SUBPROGRAMS CALLED
C     GVPHED  = COLLECTS DATA FOR AN EDGE AND SHIPS IT ON
C     POLMAK = BEGINS NEW POLYGON IN PICTURE
C     GVPHSO = SORTS THE EDGES OF A CLIPPED POLYGON IN PREPARATION
C              FOR HIDDEN LINE PROCESSING BY THE JONESD ALGORITHM
C
C***********************************************************************
C
C  VARIABLES USED
C     IPOLY  = COUNT OF POLYGONS
C     LAS    = LAST EDGE IN POLYGON
C     MEDGE  = NUMBER OF SIDES OF POLYGON + 1
C     NEDGE  = NUMBER OF SIDES OF POLYGON
C     CCONT  = HOLDING LOC FOR CONTOUR VALUES FOR JONESD HIDDEN
C     HOLDX  = HOLDING LOC FOR X EDGE COORDS FOR JONESD HIDDEN
C     HOLDY  = HOLDING LOC FOR Y EDGE COORDS FOR JONESD HIDDEN
C     HOLDZ  = HOLDING LOC FOR Z EDGE COORDS FOR JONESD HIDDEN
C     NMNOD  = NUMBER OF NODES IN THE POLYGON BEING PROCESSED
C     NNODES = NUMBER OF NODES IN A POLYGON
C     NPOLY  = COUNTS NUMBER OF POLYGONS
C
C***********************************************************************
C
C  VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHPV
      INCLUDE '../dparam.h'
C
C     MAXNS  = MAX NUMBER OF NODES IN AN ELEMENT
C     MAXNPT = MAX NUMBER OF ELEMENTS IN MODEL
C
C***********************************************************************
C
      COMMON/CLIPS/  XB,YB,ZB,KB,CB,XE,YE,ZE,KE,CE,LAS,ISHARE,NTR,
     $               ITR1,ITR2
      COMMON/NEME/   NEDGE,MEDGE,IPART
      COMMON/GHID/   HOLDX(MAXNS,MAXNPT),
     $               HOLDY(MAXNS,MAXNPT),
     $               HOLDZ(MAXNS,MAXNPT),
     $               CCONT(MAXNS,MAXNPT),
     $               NNODES(MAXNPT)
      COMMON/VISI/   NNN(MAXNS),IVIS
      COMMON/SPARE/  NMNOD,NPOLY
      LOGICAL        LAS
C
C
C                    **************************
C                    ******BEGIN PROCESSING******
C                    **************************
C
      CALL POLMAK
      LAS=.FALSE.
C
C-----------------------------------------------------------------------
C     IF THERE ARE NO NODES IN THIS POLYGON (IE. IT HAS BEEN CLIPPED
C     AWAY) THEN WE DON'T WANT TO INCREASE OUR POLYGON COUNTER FOR
C     JONESD
C-----------------------------------------------------------------------
C
         IF(NMNOD.EQ.0) GO TO 61
         NMNOD = 0
         NPOLY = NPOLY+1
   61    CONTINUE
C-----------------------------------------------------------------------
C     PROCESS THIS POLYGON
C-----------------------------------------------------------------------
         DO 64 I=1,NEDGE
           I1 = NNN(I)
           I2 = NNN(I+1)
           IF(I.NE.NEDGE) GO TO 64
           I2 = NNN(1)
           LAS = .TRUE.
   64    CALL GVPHED(I1,I2)
         IF(NMNOD.EQ.0) RETURN
C
C-----------------------------------------------------------------------
C     IF ICNT IS .GT. NEDGE*2 THEN WE KNOW THAT THIS POLYGON WAS
C     CLIPPED AND WE MUST ORDER ITS EDGES.
C-----------------------------------------------------------------------
C
         IF(ICNT.GT.NEDGE*2) THEN
           CALL GVPHLD
```

```
      END IF
C
C-----------------------------------------------------------------------
C     LOAD THE NUMBER OF NODES IN THIS POLYGON INTO THE NNODES ARRAY
C     FOR JONESD
C-----------------------------------------------------------------------
C
      NNODES(NPOLY)=NNNOD
      RETURN
      END


      SUBROUTINE GVPHPT
C
C**********************************************************************
C
C  SUBROUTINE GVPHPT - SENDS POLYGON EDGE TO CORRECT SUBROUTINE FOR
C                      EITHER DRAW OR VIEW
C
C**********************************************************************
C
C  SUBROUTINE CALLED BY
C     GVPHPT   = PROCESSES POLYGONS BY PART
C
C**********************************************************************
C
C  SUBPROGRAMS CALLED
C     GVPHPV  = SENDS POLYGON EDGES ON FOR JONES HIDDEN LINE
C               PROCESSING
C
C**********************************************************************
C
C  VARIABLES USED
C     NONE
C
C**********************************************************************
C
C   VARIABLE DIMENSION INFORMATION FOR SUBROUTINE GVPHPT
C      NONE
C
C**********************************************************************

C                        ****************************
C                        ******BEGIN PROCESSING******
C                        ****************************
C
       CALL GVPHPV
C
      RETURN
      END
```

A FAST HIDDEN LINE ALGORITHM

WITH CONTOUR OPTION

Ronald B. Thue

Department of Civil Engineering

M.S. Degree, December 1984

ABSTRACT

There is an on going desire in the computer graphics community to develop faster hidden line algorithms that can give accurate hidden line representations of geometric models. This thesis presents the JonesD algorithm, modified to allow the processing of N-sided elements and implemented in conjunction with a 3-D contour generation algorithm. The total hidden line and contour subsystem is implemented in the MOVIE.BYU Display package, and is compared to the subsystems already existing in the MOVIE.BYU package. The comparison reveals that the modified JonesD hidden line and contour subsystem yields substantial processing time savings, when processing moderate sized models comprised of 1000 elements or less. There are, however, some limitations to the modified JonesD subsystem.

COMMITTEE APPROVAL:

Michael B. Stephenson
Committee Chairman

Steven E. Benzley
Committee Member

Henry N. Christiansen
Department Chairman