INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106
A nonlinear isoparametric contour plotting and filling algorithm

McCleary, Bret Alan, M.S.
Rice University, 1988
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages ✓
2. Colored illustrations, paper or print ✓
3. Photographs with dark background □
4. Illustrations are poor copy □
5. Pages with black marks, not original copy □
6. Print shows through as there is text on both sides of page □
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements □
9. Tightly bound copy with print lost in spine □
10. Computer printout pages with indistinct print □
11. Page(s) □ lacking when material received, and not available from school or author.
12. Page(s) □ seem to be missing in numbering only as text follows.
13. Two pages numbered □. Text follows.
14. Curling and wrinkled pages □
15. Dissertation contains pages with print at a slant, filmed as received □
16. Other □

________________________________________________________
________________________________________________________

UMI
RICE UNIVERSITY

A NONLINEAR ISOPARAMETRIC CONTOUR PLOTTING
AND
FILLING ALGORITHM

by

BRET ALAN McCLEARY

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE

APPROVED, THESIS COMMITTEE:

John E. Akin
Dr. John E. Akin
Professor of Mechanical Engineering,
Chairman

Angelo Miele
Dr. Angelo Miele
Professor of Mechanical Engineering

Frederic A. Wierum
Dr. Frederic A. Wierum
Professor of Mechanical Engineering

Houston, Texas

December, 1987
A Nonlinear Isoparametric Contour Plotting and Filling Algorithm

Bret McCleary

Abstract

An interactive isoparametric contour plotting and filling program, ISOCON, is developed for use with finite element meshes using common two-dimensional triangular and quadrilateral elements. The program uses the local interpolation functions to compute the contour locations. The predictor-corrector algorithm used to compute the contour locations is derived and discussed. The color filling algorithm is also discussed. The program features are given along with a brief discussion of selected algorithms used in the program.

The main feature of ISOCON lies in its portability since most plot drivers are written in FORTRAN 77. Selected contour plots are compared with output from another finite element contour plotting program. The ISOCON-generated plots produce much more realistic looking contours. Partial FORTRAN code along with a flowchart is included.
Acknowledgements

First of all, I would like to thank my advisor, Dr. J. E. Akin, for giving me excellent guidance throughout my studies at Rice University. I would also like to thank my friends and colleagues at Rice University and at McDonnell Douglas Astronautics Company for their many excellent comments and suggestions throughout the development of this thesis. I would especially like to thank my wife, Rebecca, whose understanding, patience and encouragement has helped me pursue and complete an advanced degree.
Table of Contents

Abstract .................................................................................................................. ii
Acknowledgements ................................................................................................. iii
List of Figures .......................................................................................................... v

1. INTRODUCTION ............................................................................................... 1
   1.1 Overview of the Method ............................................................................. 1
   1.2 Other Types of Contour Plotting Algorithms ........................................... 2
   1.3 Other Types of Contour Filling Algorithms .............................................. 3

2. ISOPARAMETRIC ELEMENT THEORY ....................................................... 4
   2.1 Local Coordinate Theory ......................................................................... 4
   2.2 Interpolation Functions ........................................................................... 6
   2.3 Interpolation Function Local Derivatives ................................................. 11
   2.4 Inter-Element Continuity ......................................................................... 11

3. ISOPARAMETRIC CONTOUR THEORY ..................................................... 13
   3.1 Local Coordinate Contour Location ......................................................... 13
   3.2 Local Coordinate Correction .................................................................. 17

4. ISOCON PROGRAM DESCRIPTION ............................................................... 20
   4.1 Fundamental Program Features ............................................................... 20
   4.2 Contour Plotting Algorithm ..................................................................... 21
   4.3 Contour Filling Algorithm ....................................................................... 28
   4.4 Color Selection ......................................................................................... 30
   4.5 Plotting Algorithm Speed ........................................................................ 32
   4.6 Hidden Line Routine ................................................................................ 33
   4.7 Windowing and Clipping Algorithms ...................................................... 34
   4.8 Extension to Three Dimensions ............................................................... 37

5. ISOCON PROGRAM TEST CASES ............................................................... 39
   5.1 ISOCON Interactive Graphics Overview ............................................... 39
   5.2 ISOCON Test Cases ................................................................................ 41

6. SUMMARY ......................................................................................................... 59

References ............................................................................................................. 60
Appendix - Selected ISOCON Subroutines .......................................................... 62

iv
List of Figures

1. Local coordinates for quadrilateral elements ........................................ 5
2. Transformation from local to global coordinates ....................................... 7
3. Geometry of step along contour line ....................................................... 14
4. Contour correction step ........................................................................... 18
5. Pseudo-lines and nodes on quadrilateral elements ..................................... 22
6. Pseudo-lines and nodes on triangular elements ......................................... 23
7. Flowchart of contour line plotting algorithm ............................................ 27
8. Subdivision of quadrilateral and triangular elements in contour filling routine .................................................. 29
9. RGB color model .................................................................................... 31
10. Outcodes used in Cohen-Sutherland clipping algorithm .......................... 36
11. Example of ISOCOM menu display .......................................................... 40
12. Element type number used in MODEL input format ................................. 42
13. Contour plotting example: eight-node quadrilateral element .................... 43
14. Contour plotting example: twelve-node quadrilateral element ................ 45
15. Contour plotting example: six-node triangular element ............................... 46
16. Contour plotting example: nine-node triangular element ........................... 47
17. Comparison of T3 and Q4 elements: potential flow around cylinder example .................................................. 48
18. ISOCON/PIGS comparison: simple stress example ............................................. 49
19. Continuous color plot: simple stress example .................................................. 50
20. ISOCON/PIGS comparison: axisymmetric flywheel example .............................. 51
21. Continuous color plot: axisymmetric flywheel example ..................................... 52
22. ISOCON/PIGS comparison: axisymmetric stress example .................................. 53
23. ISOCON/PIGS comparison: triangular cooling fin example .............................. 55
24. Enlargement showing temperature discontinuity of contour H ............................. 56
25. Continuous color contour plot: triangular cooling fin example .......................... 57
26. Coarse versus fine mesh comparison: simple thermal example .......................... 58
1. INTRODUCTION

Isoparametric elements are among the most common type of element being used to obtain finite element solutions today. Because of the popularity of this type of element, a post-processing finite element graphics program displaying accurate contours over the entire mesh is desired to visually display solutions. Most small contour plotting programs, however, use linear interpolation to compute the contour locations. These programs plot fast but inaccurate and sometimes misleading contours. Thus a contour plotting program has been developed which uses the local interpolation functions to more accurately compute the contour locations. Modern graphics terminal capabilities to plot color-filled contour regions and continuous-color contours using solutions obtained from existing finite element programs are also available in this program. This section of the paper will give an overview of the method, a brief discussion of other types of contour plotting algorithms followed by a brief review of other types of contour filling algorithms.

1.1. Overview of the Method

This paper describes the theory and program structure of the isoparametric contour plotting and filling program, ISOCON, developed for use with existing finite element programs. The program uses the theory developed by Akin and Gray [3], [11] to plot accurate contour lines on two-dimensional isoparametric finite element surfaces. The program also uses the theory developed by Stelzer [17] to fill contour regions with color. The paper will also describe some other features employed in the program such as contour color selection, the hidden line algorithm, and the windowing and clipping algorithms.
1.2. Other Types of Contour Plotting Algorithms

Many other methods of plotting contours over finite element meshes have been developed. Meek and Beer [16] were the first using isoparametric elements to trace a contour in local coordinates before mapping into global or physical coordinates. In this method, the element in local coordinates is divided into sub-domains where the function being contoured is assumed to vary linearly along each sub-domain. The accuracy of this method depends on the number of subdivisions and the element interpolation order. If the element uses quadratic or cubic interpolation, inter-element information is lost using linear interpolation within the element.

Lyness and Asquith [15] developed an algorithm which subdivides elements into triangles and uses a linear interpolation scheme to compute the contour locations. The subdivisions, however, are made in global coordinates with the curved sides of higher order elements approximated with straight sides. Again, piecewise linear contours were found to be inaccurate over quadratic and cubic elements.

Yeo [21] developed a contour plotting algorithm which subdivides each element in local coordinates into three-node triangular elements. Since the function being contoured varies linearly along the three-noded triangle, only the endpoints of the contour need to be identified. The contour is then converted into global coordinates where it is plotted. This results in a fast but inaccurate contour plotting program when quadratic or cubic elements are used.

Steven [20] used minimum surface theory to plot contour lines. The examples in this paper used three-node triangular elements and only plotted contours lying inside the boundaries of the element. The conclusion of the paper noted that the technique broke down at the edges of the element.

Stelzer and Welzel [19] used the relationship between the values to be contoured at
the nodes and the interpolation functions to analytically solve for the local coordinate \( r \) given the value of \( s \). This algorithm has the advantage that the interpolation functions used to transform the finite element mesh from local to global coordinates are also used to determine the locations of the contour lines. A disadvantage of the program is that all values of the \( r \) local coordinate must be used to compute corresponding \( s \) local coordinate values regardless of whether the \( s \) value is inside or outside the element.

1.3. Other Types of Contour Filling Algorithms

The algorithm documented in this paper, ISOCON, fills contour regions by subdividing the elements into triangular or quadrilateral subregions and filling these subregions with a constant color. The resolution of the interface between adjacent contour regions is dependent upon the number of subregions within the element.

Another method of filling contour regions was given by Akin, Gray and Zhang [4]. This paper describes an algorithm which uses a raster scan technique to color an isoparametric element. The program scans along a constant horizontal value in device or screen coordinates. Theoretically, the interpolation functions can be used to compute the value being contoured and the corresponding color at each pixel. In practice, however, values at selected pixels were computed by using the interpolation functions. The values at other pixels were then filled using linear interpolation. A difficulty existed in identifying each pixel of the boundary in screen coordinates since the sides of the elements, in general, are not linear.

Still another approach of coloring contours was given by Christiansen and Stephenson [6]. Color values at the ends of a raster scan line were assigned a value by using linear interpolation between the nodal values on either side of the scan line. The pixels on the interior of the scan line were then determined by linear interpolation using the color values at the ends of the scan line.
2. ISOPARAMETRIC ELEMENT THEORY

This type of element is named isoparametric since the interpolation function used to determine the global coordinates inside the element given the global coordinates at the nodes is also used to interpolate the value to be contoured inside the element given the value at the nodes. Since the contour plotting routine described in this paper is useful on isoparametric elements only, the fundamental theory of this type of element is given below including local coordinate theory, interpolation functions, interpolation function local derivatives and inter-element continuity.

2.1. Local Coordinate Theory

The basis of isoparametric element theory lies in the concept of a transformation from non-dimensional local coordinates to the physical, real-world global coordinates. Local coordinates are non-dimensional, element-based units which are the same from element to element, assuming the same type of element is used throughout the analysis. Use of local coordinates simplifies the algorithm used to determine the location of a contour line.

Two types of local coordinates are used: unit coordinates and natural coordinates. The origin of the unit coordinate system is at the end or a corner of the element and each axis ranges from 0 to +1. On the other hand, the origin of the natural coordinate system for quadrilateral elements is at the midpoint or centroid of the element and each axis ranges from -1 to +1 [2]. Figure 1 shows the unit and natural local coordinates for two-dimensional quadrilateral elements.

Note that the natural coordinates for triangular elements are also named area coordinates. In this system, the element is divided into three regions of area $A_1$, $A_2$ and $A_3$ with the point of interest at one vertex of each of the smaller triangular regions. The ratio of each region to the total element area can be denoted by $L_1$, $L_2$ and $L_3$. Note that
Figure 1 - Local coordinates for quadrilateral elements

Natural Coordinates

Unit Coordinates
these coordinates are related to unit coordinates since \( L_2 \equiv r \) and \( L_3 \equiv s \). Because unit coordinates are used for all triangular elements in ISOCON, area coordinates will not be discussed further.

2.2. Interpolation Functions

Irrespective of which type of local coordinate scheme is chosen, an interpolation function which maps the element defined in local coordinates into global coordinates must be found. The geometry of a three-node triangular element in both the reference or local configuration (local coordinates) and the actual or global configuration (global coordinates) is shown in Figure 2. The region \( R^l \) in the figure defines the element in local coordinates and the region \( R^g \) defines the element in global coordinates. The transformation, \( \chi^e \), can be expressed in vector notation as [10]:

\[
\chi^e: \mathbf{r} \rightarrow \mathbf{x}^e = \chi^e(r, x_1, x_2, x_3)
\]  

(1)

where \( \mathbf{r} \) represents the vector of local coordinates of the point to be transformed into global coordinates, \( \mathbf{x}^e \) represents the global coordinates of the transformed point and \( X_1, X_2 \) and \( X_3 \) represent the global coordinates at the nodes of the element. The transformation \( \chi^e \) must have the following properties:

1. The transformation from local to global coordinates must be a one-to-one mapping.
2. The order of the nodes on the element must be preserved, in other words, if the element order in local coordinates is 2,3,1, then the same 2,3,1 order must be preserved in the global coordinates.
3. Every point on the boundary between nodes in local coordinates must correspond to the same point on the boundary between the same nodes in global coordinates.

The global coordinates can be computed using the local interpolation function as follows:
where $H^e(r, s)$ is the local interpolation function and $X^e$ and $Y^e$ are the vectors of the $x$ and $y$ nodal global coordinates. Thus, the dimension of the vector $H^e(r, s)$ is $1 \times$ NODEXT and the dimension of vectors $X^e$ and $Y^e$ is NODEXT $\times$ 1 where NODEXT represents the number of nodes in the element.

Most of the common isoparametric interpolation functions can be determined from polynomials. For example, a linear polynomial can be used to derive the interpolation function for the three-node triangular element. A quantity of interest, say $V^e(r, s)$, as a function of local coordinates $r$ and $s$ can be described by

$$V^e(r, s) = P(r, s) C^e$$

where $P(r, s)$ is the linear function $[1 \; r \; s]$ and $C^e$ is a vector of constants. Thus,

$$V_1^e = C_1^e + C_2^e r_1 + C_3^e s_1$$
$$V_2^e = C_1^e + C_2^e r_2 + C_3^e s_2$$
$$V_3^e = C_1^e + C_2^e r_3 + C_3^e s_3$$

or in vector form,

$$V^e = G^e C^e$$

where $G^e$ is a square matrix of dimension NODEXT $\times$ NODEXT. In this example,

$$G^e = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Assuming that $G^e$ is invertible, the vector of constants can be determined by

$$C^e = G^{e^{-1}} V^e.$$  

Thus the quantity of interest as a function of local coordinates $r$ and $s$ can be determined by combining equations (3) and (7) to obtain
\[ V^e(r, s) = P(r, s) G^{e-1} V^e. \] (8)

From equation (2) and the definition of an isoparametric element it can be deduced that the local interpolation function can be computed as

\[ H^e(r, s) = P(r, s) G^{e-1}. \] (9)

In this example,

\[ H^e(r, s) = \begin{bmatrix} 1 & r & s \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}. \] (10)

Therefore,

\[ H^e(r, s) = \begin{bmatrix} 1 - r - s & r & s \end{bmatrix}. \] (11)

Note that it has also been shown [1] that the sum of the isoparametric interpolation functions add to unity.

The systematic procedure demonstrated above can be used to determine most of the interpolation functions used in ISOCON. However, it should be pointed out that the \( G^e \) matrix for some elements defined in natural coordinates may not be unique. Table I shows the polynomials used for each interpolation function. The table also gives the names of the subroutines where the shape functions can be found.

The elements marked incomplete in the table do not use the full quadratic or cubic polynomial in its construction. The complete polynomial in these cases produces interior nodes which are troublesome and are not included in the element libraries of many finite element programs. The interpolation function of the nine-node triangle is computed in a different manner than shown above. This interpolation function is determined by modifying the interpolation function from the complete cubic triangular element (ten nodes). The value at the tenth node, computed using the interpolation function of the complete cubic element, is replaced by a linear combination of the other nine
Table I - Polynomial basis of elements used in ISOCON

<table>
<thead>
<tr>
<th>Geometric Shape</th>
<th>Number of Nodes</th>
<th>Degree of Polynomial</th>
<th>Polynomial Basis ( P(r, s) )</th>
<th>Interpolation Function Subroutines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular</td>
<td>3</td>
<td>1</td>
<td>([1 \ r \ s])</td>
<td>SHP3T</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2</td>
<td>([1 \ r \ s \ r^2 \ rs \ s^2])</td>
<td>SHP6T</td>
</tr>
<tr>
<td></td>
<td>9 (incomplete)</td>
<td>3</td>
<td>([1 \ r \ s \ r^2 \ rs \ s^2 \ r^3 \ r^2s \ rs^2 \ s^3])</td>
<td>SHP9T</td>
</tr>
<tr>
<td>Quadrilateral</td>
<td>4</td>
<td>1</td>
<td>([1 \ r \ s \ rs])</td>
<td>SHP4Q</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2 (incomplete)</td>
<td>([1 \ r \ s \ r^2 \ rs \ s^2 \ r^2s \ rs^2])</td>
<td>SHP8Q</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>1 (incomplete)</td>
<td>([1 \ r \ s \ r^2 \ rs \ s^2 \ r^3 \ r^2s \ rs^2 \ s^3] )</td>
<td>SHP12Q</td>
</tr>
</tbody>
</table>
nodal values in a manner which maintains the polynomial basis of the complete element. The details of this derivation are shown in reference [10].

2.3. Interpolation Function Local Derivatives

In order to compute the location of a contour line, chapter 3 will show that the local derivatives of the interpolation functions are needed. Since the interpolation functions $H(r,s)$ are a function of local coordinates, derivatives can be easily found. For example, the local derivatives of the interpolation function for the three-node triangular element is, from equation (11):

$$\frac{\partial H^e(r,s)}{\partial r} = \begin{bmatrix} -1 & 1 & 0 \end{bmatrix} \quad (12a)$$

$$\frac{\partial H^e(r,s)}{\partial s} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}. \quad (12b)$$

The local derivatives of the shape functions are computed in ISOCON in subroutines DER3T, DER6T and DER9T for the three-, six- and nine-node triangular elements and in subroutines DER4Q, DER8Q and DER12Q for the four-, eight- and twelve-node quadrilateral elements. Since the sum of the interpolation functions is equal to one, the sum of the interpolation function local derivatives must equal zero.

2.4. Inter-Element Continuity

As mentioned previously, an element is isoparametric if the interpolation function used to interpolate a variable (displacement, temperature, etc.) within the element from its nodal values is also used to interpolate global coordinates within the element from its from its nodal values. Nothing has been mentioned about the behavior of a variable value across the element boundaries. An element type is said to possess $C^0$ continuity if the computed value, such as displacement, is continuous across element boundaries. The element type is said to possess $C^1$ continuity if the computed value and its first
derivative are continuous across element boundaries. The element type, in general, is said to possess $C^n$ continuity if the computed value and all derivations up to and including the $n^{th}$ derivative are continuous across element boundaries. All elements used in ISOCON possess $C^0$ continuity, therefore, the contours will be continuous but the slope of the contours may be discontinuous across element boundaries.
3. ISOPARAMETRIC CONTOUR THEORY

To visually show the results of a finite element analysis, an accurate contour plot is usually desired. If higher order quadratic or cubic elements are used in the analysis while contours are computed using linear interpolation, the results are more accurate than the contour plot used to graphically represent these results. Thus, a method using the local interpolation functions to determine the contour locations was developed by Akin and Gray [1], [3] and is the foundation of the ISOCON program. Therefore, the basic theory for the determination of the local coordinate contour location and the local coordinate correction is derived and discussed in this chapter.

3.1. Local Coordinate Contour Location

Since the location of the nodes and boundaries of an element in local coordinates is fixed, it is easier to determine the contour locations in local coordinates and convert these into global coordinates for plotting. Figure 3 shows a sketch of a portion of the contour line and the geometry of this line segment. Since the value along the contour line is constant, the infinitesimal change in the value along the contour, $dV$, is zero. Using the chain rule and assuming $\Delta r$ and $\Delta s$ are small

$$ \frac{\partial V}{\partial r} \Delta r + \frac{\partial V}{\partial s} \Delta s = 0. $$

(13)

The figure also shows the gradient of the value, $\nabla V$. From the definition of a gradient of a function,

$$ \nabla V = \frac{\partial V}{\partial r} \mathbf{i} + \frac{\partial V}{\partial s} \mathbf{j} $$

(14)

where $\mathbf{i}$ and $\mathbf{j}$ denote the unit vectors in the $r$ and $s$ coordinate directions respectively. Thus the magnitude of the gradient is
Figure 3 - Geometry of step along contour line
\[
|\nabla V| = \left[ \left( \frac{\partial V}{\partial r} \right)^2 + \left( \frac{\partial V}{\partial s} \right)^2 \right]^{\frac{1}{2}}.
\] (15)

So, equation (13) is equivalent to

\[
\nabla V \cdot (\Delta r \mathbf{i} + \Delta s \mathbf{j}) = 0
\] (16)

which shows that the gradient is indeed perpendicular to the contour line as shown in the figure.

Assuming that a starting point within the element is found, the new coordinate location can be found from the old coordinate location from

\[
\begin{align*}
    r_{new} &= r_{old} + \Delta r \\
    s_{new} &= s_{old} + \Delta s.
\end{align*}
\] (17a) (17b)

If the step \( \Delta l \) is small enough to be assumed linear, the small change in the local coordinates \( \Delta r \) and \( \Delta s \) can be computed as

\[
\begin{align*}
    \Delta r &= \Delta l \cos \theta \\
    \Delta s &= \Delta l \sin \theta.
\end{align*}
\] (18a) (18b)

Using trigonometric identities and the sketch in the figure, the values of \( \cos \theta \) and \( \sin \theta \) can be determined by

\[
\cos \theta = \frac{\frac{\partial V}{\partial s}}{|\nabla V|}
\] (19a)

\[
\sin \theta = -\frac{\frac{\partial V}{\partial r}}{|\nabla V|}.
\] (19b)

Therefore, the changes in local coordinates \( \Delta r \) and \( \Delta s \) are

\[
\Delta r = \frac{\Delta l \left( \frac{\partial V}{\partial s} \right)}{|\nabla V|}
\] (20a)
\[\Delta s = -\frac{\Delta l \left[ \frac{\partial V}{\partial r} \right]}{|\nabla V|}.\]  

(20b)

Since the element is isoparametric, the value of any variable to be contoured, \(V\), as a function of local coordinates \(r\) and \(s\) is

\[V(r, s) = H^e(r, s) \mathbf{V}^e\]  

(21)

where \(H^e(r, s)\) is the local interpolation function and \(\mathbf{V}^e\) is the vector of nodal values to be contoured. Note that the dimension of the vector \(\mathbf{V}^e\) is NODEXT x 1 and the dimension of \(H^e(r, s)\) is 1 x NODEXT where NODEXT is the number of nodes on the element as mentioned previously. The values \(\frac{\partial V}{\partial r}\) and \(\frac{\partial V}{\partial s}\) can be computed from (21), thus

\[\frac{\partial V}{\partial r} = \frac{\partial H^e}{\partial r} \mathbf{V}^e\]  

(22a)

\[\frac{\partial V}{\partial s} = \frac{\partial H^e}{\partial s} \mathbf{V}^e.\]  

(22b)

So the new local coordinates can be computed from the old local coordinates using equations (17), (20) and (22) obtaining

\[r_{new} = r_{old} + \frac{\Delta l \left[ \frac{\partial H^e}{\partial s} \right] \mathbf{V}^e}{|\nabla V|}\]  

(23a)

\[s_{new} = s_{old} - \frac{\Delta l \left[ \frac{\partial H^e}{\partial r} \right] \mathbf{V}^e}{|\nabla V|}\]  

(23b)

where \(|\nabla V|\) is given by equations (15) and (22) obtaining

\[|\nabla V| = \left\{ \left[ \left( \frac{\partial H^e}{\partial r} \right) \mathbf{V}^e \right]^2 + \left[ \left( \frac{\partial H^e}{\partial s} \right) \mathbf{V}^e \right]^2 \right\}^{\frac{1}{2}}.\]  

(24)
3.2. Local Coordinate Correction

If the step size were infinitesimal, no further computation would be required since equation (23) was derived on the premise that the step size $\Delta l$ was infinitesimal. Since $\Delta l$ is finite, albeit small, and the contour is not linear in general, a small error will be produced at each step and is cumulative. This cumulative error is noticeable on closed contours and will lead to a spiraling contour which will not close upon itself. Thus a correction step is needed to bring the computed contour back to the correct contour value.

Figure 4 shows the position of the new and corrected values relative to the contour. The difference between the correct contour value, $V_{cor}$, and the new value at the local coordinates obtained from equation (23), $V_{new}$, can be approximated by taking the first term in the Taylor series expansion,

$$\Delta V = \frac{\partial V}{\partial r} \Delta r + \frac{\partial V}{\partial s} \Delta s.$$  \hspace{1cm} (25)

Realizing that $\Delta V = V_{cor} - V_{new}$ and ignoring the effects of higher order terms of the Taylor series expansion,

$$\frac{\partial V}{\partial r} \Delta r + \frac{\partial V}{\partial s} \Delta s + V_{new} - V_{cor} = 0.$$  \hspace{1cm} (26)

The above equation has an infinite number of solutions. The desired solution reaches the contour line in the minimum distance. The square of the distance to the contour line can be represented by

$$G = (\Delta r)^2 + (\Delta s)^2 + \lambda \left[ \frac{\partial V}{\partial r} \Delta r + \frac{\partial V}{\partial s} \Delta s + V_{new} - V_{cor} \right].$$  \hspace{1cm} (27)

where $\lambda$ is a constant. The minimum distance is found by taking the derivative of (27) with respect to $\Delta r$ and $\Delta s$ and setting it equal to zero, thus
Figure 4 - Contour correction step
\[
\frac{\partial G}{\partial (\Delta r)} = 2 \Delta r + \lambda \frac{\partial V}{\partial r} = 0 \quad (28a)
\]

\[
\frac{\partial G}{\partial (\Delta s)} = 2 \Delta s + \lambda \frac{\partial V}{\partial s} = 0. \quad (28b)
\]

Solving for \(\Delta r\) and \(\Delta s\) yields

\[
\Delta r = -\frac{\lambda}{2} \frac{\partial V}{\partial r} \quad (29a)
\]

\[
\Delta s = -\frac{\lambda}{2} \frac{\partial V}{\partial s}. \quad (29b)
\]

The constant \(\lambda\) must satisfy equation (26), so

\[
\lambda = \frac{2 (V_{\text{new}} - V_{\text{cor}})}{\left( \frac{\partial V}{\partial r} \right)^2 + \left( \frac{\partial V}{\partial s} \right)^2}. \quad (30)
\]

Substituting equation (15) into (30) and then into (29) gives

\[
\Delta r = -\left( \frac{1}{|\nabla V|} \right)^2 \frac{\partial V}{\partial r} [V_{\text{new}} - V_{\text{cor}}] \quad (31a)
\]

\[
\Delta s = -\left( \frac{1}{|\nabla V|} \right)^2 \frac{\partial V}{\partial s} [V_{\text{new}} - V_{\text{cor}}]. \quad (31b)
\]

Therefore, the coordinates of the corrected position are

\[
r_{\text{cor}} = r_{\text{new}} - \left( \frac{1}{|\nabla V|} \right)^2 \frac{\partial V}{\partial r} [V_{\text{new}} - V_{\text{cor}}] \quad (32a)
\]

\[
s_{\text{cor}} = s_{\text{new}} - \left( \frac{1}{|\nabla V|} \right)^2 \frac{\partial V}{\partial s} [V_{\text{new}} - V_{\text{cor}}]. \quad (32b)
\]
4. ISOCON PROGRAM DESCRIPTION

To implement the isoparametric contour theory shown in the previous chapter, an interactive plotting program, ISOCON, has been written. This section describes the fundamental program structure and features, the contour plotting algorithm, the contour filling algorithm, color selection, plotting algorithm speed, the hidden line routine and the windowing and clipping algorithms used in the program.

4.1. Fundamental Program Features

The program ISOCON is written in FORTTRAN 77 and contains over 6900 lines of code and calls 79 subroutines. Therefore, because of space limitations the complete code will not be included in this paper. Only the main program along with first level subroutines are included in the Appendix. The program is fully interactive allowing the user to change responses after viewing the first plot of contour lines or regions. The program reads nodal and connectivity data from a file which can be output by the finite element code MODEL [1]. ISOCON also has the capability to read the binary backing store file (.bs file) output by the Program for Automatic Finite Element Calculations (PAFEC) finite element program. The program can plot contour lines, color contour regions or continuous color contours of any value that can be represented at each node. This includes, among other quantities, temperatures, averaged stresses and streamlines. The mesh can be eliminated from the plot to see the contours more clearly by using a hidden line algorithm. A closeup view of a portion of the plot can be made by using windowing and clipping routines. Some other minor features include a title, sub-title, axes, axes labels and legend. The plots showing contour lines can be sent to the laser printer to generate a document quality hardcopy.

The main feature of this program lies in its portability. Most plot driver subroutines were written using FORTTRAN 77. Thus the only subroutines that need to be
modified when adding a new plotting terminal or when changing graphics packages are the color values subroutine (PLTCOL), the absolute draw subroutine (PLTDRW), the plot closing subroutine (PLTFIN), the plot initialization subroutine (PLTIPD), the absolute draw subroutine (PLTMOV), the polygon draw subroutine (PLTPLY) and the screen parameter subroutine (PLTSRT).

4.2. Contour Plotting Algorithm

The contour plotting algorithm derived in the previous chapter is relatively easy to implement if a contour line simply passes once through an element. The starting point along an edge can be estimated using linear interpolation from the nodes and then using Newton's method or a bisection algorithm to accurately find the intersection of the contour line with the element interface. The contour is then drawn across the element in increments of length $\Delta l$ (in local coordinates) until the element boundary is reached. This procedure has a drawback, namely, no closed contours will be found.

Another procedure that has been suggested by Gray and Akin [11] is to check the value to be contoured on vertical and horizontal lines passing through the centroid of the element. The starting point for internal contours can be obtained through linear interpolation and refined by using Newton's method or bisection as before between values computed at specified points along the lines. This is the basic method employed in ISOCON, however, more lines are used within each element than indicated above.

The line segments mentioned above can be called pseudo-lines and the points can be called pseudo-nodes. Figure 5 shows the pseudo-lines and nodes for the quadrilateral isoparametric elements and Figure 6 shows the pseudo-lines and nodes for triangular isoparametric elements. Note that the pseudo-lines are the short line segments between the pseudo-nodes. In ISOCON, the pseudo-lines and nodes are numbered and the coordinates of the pseudo-nodes are placed in common variables CLR and CLS, while the
Figure 5 - Pseudo-lines and nodes on quadrilateral elements

- Linear Quadrilateral
- Quadratic Quadrilateral
- Cubic Quadrilateral

- Actual and pseudo-nodes
- Pseudo-nodes only
Figure 6 - Pseudo-lines and nodes on triangular elements
beginning and ending nodes of each pseudo-line are placed in common variables LBNOD and LENOD.

Now that the coordinates of each pseudo-node are known, the value of whatever is being contoured can be evaluated at each pseudo-node by

$$V_{pn} = H^e(r_{pn}, s_{pn}) V^e$$

(33)

where $V_{pn}$ is the value to be contoured at the pseudo-node. These pseudo-node values are compared with the value of the present contour line to determine if the contour line intersects one of these pseudo-nodes. The value of the present contour line is also compared with the pseudo-node values at the end of each pseudo-line to determine whether the contour passes through the pseudo-line. If either of the above conditions do not occur, the routine goes to the next element since the contour line does not pass through this element.

The numbers of the pseudo-nodes which are intersected are placed in an array (INTNOD) while the numbers of all pseudo-lines intersected are placed in another array (INTLIN). If the contour line intersects a pseudo-node, the starting point of the contour is the lowest numbered pseudo-node intersected. Otherwise, the contour line starts from the intersection of the contour and the lowest numbered pseudo-line. Thus, the contour does not necessarily start at an element boundary.

The computation of the intersection between the contour and a pseudo-line is determined by either the Newton's method or the bisection method. The initial local coordinates are obtained by a linear interpolation from the pseudo-line end points. Since the function is rather expensive to evaluate, a Newton's method is tried before the bisection method is employed because Newton's method usually requires fewer iterations for convergence. The classical form of Newton's method for a function of one variable is [8]
\[ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \]  

(34)

The value of the contour line, however, is a function of two variables. In most cases, the pseudo-line is either horizontal or vertical making \( s \) or \( r \) constant respectively. Thus, equation (34) can be used to find the non-constant variable with the derivative of the contour value at \((r_{pn}, s_{pn})\) being computed by taking the derivative of (33) obtaining

\[ \frac{\partial V}{\partial r}(r_{pn}, s_{pn}) = \frac{\partial H^e}{\partial r}(r_{pn}, s_{pn}) V^e \]  

(35a)

\[ \frac{\partial V}{\partial s}(r_{pn}, s_{pn}) = \frac{\partial H^e}{\partial s}(r_{pn}, s_{pn}) V^e \]  

(35b)

where \( \frac{\partial H^e}{\partial r} \) and \( \frac{\partial H^e}{\partial s} \) are the local coordinate derivatives of the interpolation function having combined dimensions of NSPACE x NODEXT. Note that NSPACE is the dimension of the element. If the pseudo-line is diagonal, one coordinate is chosen to be computed by (35a) or (35b). The other coordinate is computed from the equation of the diagonal pseudo-line. If the derivative of the contour value with respect to a local coordinate at this point is zero or the computed coordinates are outside the element, the Newton’s method is assumed to diverge and a bisection method is used to compute the initial contour location.

After the starting point of the contour in local coordinates is computed, it is converted into global coordinates and stored in a plot buffer array. The INTLIN or INTNOD array element signifying this line or node is set to its negative value to mark the starting point. The remaining coordinates of the contour line are computed by using equations (23) and (32), then converted into global coordinates and finally stored in a plot buffer array. After each point of the contour line is computed, the coordinates of this point are compared to the previous point’s coordinates to determine if a pseudo-line or pseudo-node has been crossed. If so, the INTLIN or INTNOD array element for that
particular line or node is set equal to zero. If the crossed pseudo-line or pseudo-node is located on the exterior of the element, the INTNOD and INTLIN arrays are searched for more non-negative and non-zero values. If no positive values exist, the final point is found by linear interpolation. The local coordinates are converted to global coordinates and stored in the plot buffer array. If positive values exist, this means the contour line passes through the element again. The contour line is resumed at the lowest numbered pseudo-node or at the intersection of the contour and the lowest numbered pseudo-line.

If the crossed pseudo-line or node is in the interior of the element, the value of the INTLIN or INTNOD array element is checked to see if it is negative. A negative value indicates the contour line has closed on itself. The final point is found, as above, by linear interpolation. The local coordinates are again converted to global coordinates and stored in the plot buffer array. If the array element is positive, it is set equal to zero and the contour march is continued.

The above procedure differs from that in references [1], [3] and [11]. The procedure in the references simply computes a maximum number of steps along a contour line based on the perimeter of the element. This leads to contours that can plot over themselves many times before the maximum number of steps is reached. The above procedure allows contour plotting to stop when the contour is closed.

Figure 7 shows a top level flowchart of the contour line plotting algorithm. Note that in the actual program the contour line is not plotted after each new position is computed. The global coordinates of the computed contour are first stored in a buffer array. Once the buffer is full, a straight line between each point in the buffer is plotted. The buffer is then reset in preparation to store more points. This procedure allows faster contour plotting.
Figure 7 - Flowchart of contour line plotting algorithm
4.3. Contour Filling Algorithm

Color-filled contour regions can be produced by the ISOCON program. The major advantage of color-filled regions is that continuous color plots of values such as stress or temperature can be shown. These plots give quick and detailed views similar to those produced by photoelasticity or infrared photography.

One way of developing a continuous color plot algorithm was mentioned briefly before and is described in reference [4]. This method starts with a horizontal line in projected graphics display or screen coordinates called a scan line. Thus, if the horizontal screen coordinate is denoted by I and the vertical screen coordinate is denoted by J, this scan line would have a constant value of J. Using the theory provided in chapter three, equations similar to (23) and (32) can be developed assuming constant J (scan line). The local coordinates obtained can be converted into global coordinates and then into screen coordinates to compute the values of the other screen coordinate, I, and the corresponding value, V, along the scan line. These values are assigned colors and plotted. The gaps between the computed pixels are then approximated using linear interpolation. A major disadvantage of this method is that the boundary of each element must be stored to obtain the starting and stopping I coordinates along the scan line. Also, the bottom or top edge of the element may be convex requiring rather complex starting and stopping logic.

Another approach, which is used by ISOCON, was pointed out by Stelzer [17]. This procedure uses the capability of some graphics packages to fill polygonal regions with color. The only data needed by the graphics package is the location of the vertices in projected or screen coordinates and the value of the color to use when filling the polygon.

Figure 8 shows both a quadrilateral and triangular element in local coordinates
Figure 8 - Subdivision of quadrilateral and triangular elements in contour filling routine
divided into smaller quadrilaterals and triangles. In this figure, shown as an example, each element has three subdivisions. Thus, for the quadrilateral element the dimension \( \text{SUBDIV} \) is 0.667 and for the triangular element the dimension \( \text{SUBDIV} \) is 0.333. The location marked by the 'X' denotes the centroid of each subregion. In the quadrilateral case these centroids are \( \text{SUBDIV} \) apart and in the triangular case the centroids are one-half \( \text{SUBDIV} \) apart.

The algorithm starts by looping over all the elements. The number of subdivisions is computed along with the dimension \( \text{SUBDIV} \). The value at each centroid of the subregions is computed from equation (21). The local coordinates of the vertices of the polygonal subregions are then converted to global coordinates using equation (2). All these values are stored in a buffer array. When the buffer array is full, the coordinates are plotted. The algorithm then closes the loop over the elements. The algorithm ends by plotting the colors along with the values associated with these colors in the legend.

4.4. Color Selection

The color scheme used in ISOCON is the red, green and blue (RGB) color model [7]. This primary color model is used because most raster display's color schemes are based on this model. These pigments are used as the basis for this color model since the human eye has peak sensitivity at about 630nm (red), 530nm (green) and 450nm (blue) [12]. The RGB model can be represented by a unit cube with primary axes of red, green and blue as shown in Figure 9. Note that the origin is black and thus the intensity level of all three primary colors is zero (0,0,0). The white vertex is denoted by an intensity level of all three primary colors of one (1,1,1).

In ISOCON, the color scale ranges from blue to red using a single scale numbered from 1 (blue) to 100 (red). The scale is divided into three parts of 33 units each. The lines with arrows in Figure 9 show the color paths. The first 33 units, for example, start
Figure 9 - RGB color model
with one unit of blue and zero units of both green and red. The green intensity is linearly ramped until it reaches one unit at the value of 33 on the single scale. The blue intensity is still unity and the red intensity is still zero. Thus, 33 on the single scale gives a cyan color.

4.5. Plotting Algorithm Speed

The plotting speed of the contour line algorithm is proportional to the step size $\Delta l$. The step size necessary for smooth accurate contour lines is mainly a function of the element size and radius of curvature of the contour. For example, an element which displaces a small percentage of the total screen area would require a larger local coordinate value of $\Delta l$ than an element that takes up the entire screen area. In ISOCON, the element area in projected screen coordinates is approximated by using the polygon area equation from analytic geometry. This area is converted to a circle of equivalent area where the diameter in inches is computed. Three ranges of step sizes are used based on the diameter of the circle. A diameter of less than one inch uses a $\Delta l$ of 0.1, between one inch and three inches a $\Delta l$ of 0.05 is used and a diameter greater than three inches uses a $\Delta l$ of 0.02.

The plotting speed of the continuous color algorithm is, of course, proportional to the number of element subdivisions. Since the radius of curvature of the contour cannot quickly be computed, the number of subdivisions is a function of the element area only. The program ISOCON uses 50 subdivisions for all elements that occupy more than eight percent of the total screen area. The number of subdivisions in smaller elements approaches 10 when the element area approaches zero.

The contour filling algorithm presently computes the interpolation function at each vertex of each subregion of each element. The plotting speed could be improved by first computing the values of the interpolation functions at all vertices within the ele-
ment and storing these in an array. These values can then simply be recalled instead of recomputed for use in converting local to global coordinates or in computing the values at the centroid of each subregion.

4.6. Hidden Line Routine

Plots of two- and three-dimensional meshes showing all lines make objects difficult to visualize and the position of the contour line hard to determine from the plot. The desired plot would have all interior lines removed on two-dimensional surfaces and hidden surfaces not shown on three-dimensional objects. Many algorithms have been written which remove hidden lines from three-dimensional objects. These will not be discussed in this paper, however, a few are listed in the references: [9], [13], [14].

The algorithm used in ISOCON can remove both the interior lines in a two-dimensional mesh and the hidden surfaces in a three-dimensional mesh. The subroutine was written from a paper which described the algorithm [5]. The main feature of the algorithm is that it uses the finite element nodal assembly procedure to assemble normal vectors on element edges. Normals are computed for each element edge and located at each node for two-dimensional meshes. Three-dimensional meshes use normals to each element surface instead of each edge. When these normals are assembled, the normal vectors at the interior nodes cancel out. This section describes the procedure used to eliminate interior lines from two-dimensional elements in detail, then gives a brief description of the changes necessary for three-dimensional elements.

The first objective of this algorithm is to compute the normal vectors of each element edge and assign these vectors to each corner node of the element. Thus, the algorithm starts by beginning a loop over the elements. The corner node numbers are first identified using local node numbering. These local node numbers are then converted to global node numbers using the connectivity matrix. The normal vector at each corner
node is computed by using the normal vector to the element edge bounded by points with global coordinates \((x_1,y_1)\) and \((x_2,y_2)\). Thus,

\[
v^e = \begin{bmatrix} y_2 - y_1 \\ x_2 - x_1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \quad x_2 \neq x_1
\]

\[
n^e = \frac{v^e}{|v^e|}.
\]

Once the normal vector at each boundary node is computed, these vectors are assembled by adding the components of the normal vectors at each node. When the components of the normal vectors at each node are assembled, the vectors are investigated. The nodes that have a zero normal vector are interior nodes. An element edge that is bounded by one or more interior nodes is an interior edge and is not plotted if the mesh is not desired to be plotted. In practice, because of roundoff error, a tolerance \((10^{-6})\) is used to determine if a node is interior.

The same basic procedure can be followed for three-dimensional elements. Instead of computing normal vectors of element edges, normal vectors of each element surface are determined. The vectors are assembled by adding the normal vector components at each node of each surface. Nodes at the back of the mesh are identified and eliminated by examining the equation

\[
v \cdot n < 0
\]

where \(v\) is a unit vector from the node to the viewer and \(n\) is the assembled or global normal vector of the node.

4.7. Windowing and Clipping Algorithms

An essential part of any interactive contour plotting program is the ability to zoom in on a particular region of the plot to view it more closely. This procedure is called windowing or window-to-viewpoint mapping. Once the windowing occurs on a portion
of the screen, all lines outside the window must be clipped. In this section, the methods used for windowing and clipping will be described.

A window in ISOCON is defined as a rectangular area in world coordinates [12]. In ISOCON, world coordinates are also called screen coordinates since they usually represent the resolution of the terminal screen. The window is mapped onto a viewport which is a rectangular area defined by device coordinates.

Since ISOCON can be used with different types of terminals, an internal viewport and windowing scheme has been developed. The device coordinates are defined to be the same as the screen (world) coordinates. The plot area is defined in screen coordinates using variables PLTMIN and PLTMAX to specify the corners of the plottable screen area. A window can be defined within the plot area by using a mouse or other locator device to define the window corners. These corner coordinates are used to compute a different scale factor for the conversion from plot to screen coordinates.

Before the plot can be redrawn to view a particular region of the original plot, every line outside of the plot area must be eliminated. The procedure which eliminates these lines is called line clipping. The routine selected for use in ISOCON is the Cohen-Sutherland clipping algorithm [7]. This algorithm uses a four-bit outcode to determine if the endpoints of each line segment are inside or outside the window and the location of the endpoints if outside the window. Figure 10 shows the nine regions where the outcodes are specified. The outcodes actually consist of four logical variables (bits) set to either FALSE (0) or TRUE (1). If both endpoints are above, below, to the right or to the left of the window, the points can be immediately erased from the plot buffer. Thus, when bits describing the location of a line's endpoints are equal, the line is not plotted. The entire line is plotted if both points are inside the window. A line that has endpoints in different regions may or may not intersect the window. The part of the line inside the window is found by systematically discarding line segments above,
Figure 10 - Outcodes used in Cohen-Sutherland clipping algorithm
below, to the right or to the left of the window.

The contour filling portion of ISOCON includes an area clipping routine which clips the filled triangles or quadrilaterals when windowing is used. The algorithm uses a technique similar to that used in the line clipping algorithm for trivially rejecting polygons outside the window. Polygons which contain some vertices inside the window and some outside are clipped using the following approach. An n-sided polygon is broken up into n lines representing the edges of the polygon. Each line is terminated by two points. The first point on the line can be denoted by S and the other point by P. The table below gives action taken depending on whether point S or point P is inside or outside the window.

<table>
<thead>
<tr>
<th>Point S</th>
<th>Point P</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inside</td>
<td>Inside</td>
<td>Point P is saved</td>
</tr>
<tr>
<td>Inside</td>
<td>Outside</td>
<td>Find intersection with window and save that point</td>
</tr>
<tr>
<td>Outside</td>
<td>Inside</td>
<td>Find intersection with window and save that point and point P</td>
</tr>
<tr>
<td>Outside</td>
<td>Outside</td>
<td>Neither point is saved</td>
</tr>
</tbody>
</table>

The area clipping routine is repeated for each polygon in each element. Because of the large number of polygons plotted, the algorithm contains code to quickly determine if the entire polygon is inside or outside the window.

4.8. Extension to Three Dimensions

The contour plotting program ISOCON presently contains no three-dimensional contour plotting capability. However, the basic method to plot three-dimensional contours is given in this section. The contours of three-dimensional isoparametric elements can be drawn by using the hidden line algorithm to identify the visible faces, if any, of
each element. The procedure of stepping through an element in local coordinates is then similar to the two-dimensional case. The exception is that the three-dimensional interpolation function, $H(r,s,t)$, is evaluated using the constant value of $r$, $s$ or $t$ on this element surface (+1 or -1).

Of course, the transformation from global coordinates to screen coordinates has an additional step. This step involves a projection of the physical object onto the screen. One method has been suggested by Stelzer [18] which uses a parallel transformation. However, since most high level graphics languages have this transformation built in, the details are not given in this paper.
5. ISOCON PROGRAM TEST CASES

A number of test cases have been run using ISOCON to show the contour plots generated by the program. A few examples are compared to the output from the PAFEC Interactive Graphics Suite (PIGS) program using the same input data. Before these examples are shown, however, an overview of the ISOCON interactive graphics capabilities is given.

5.1. ISOCON Interactive Graphics Overview

ISOCON is designed to be an interactive graphics program allowing the user to change items like the input file, number of contours plotted, contour values plotted, titles and axes labels during execution of the program. Most modern graphics packages use a menu with choices represented by either character strings or graphical icons on one side or at the bottom of the screen. The menu items can then be simply selected by a mouse or other input device. The main advantage of this method is that menu choices can be made while viewing the plot. The main disadvantage of this method is that the menu takes up screen area which cannot be used for the plot. Since only one item in ISOCON’s menu is changed while viewing the plot (windowing), all other changes are made through a menu with a simple question and answer format. Windowing is selected simply by pressing the right mouse button on SUN terminals or w followed by a carriage return on Selanar graphics terminals.

All other menu options available in ISOCON require either simple yes or no answers or short responses such as the title of the plot or the number of contours to be plotted. All responses are checked for validity before changes to the plot are made. Invalid responses are identified and the user is prompted to try again. An example of the menu is shown in Figure 11. This menu corresponds to the one printed after Figure 13 is displayed.
1. Plot type ............................................................................................................. contour lines

2. Data file name ..................................................................................................... test1.10

3. Main title of the plot
   Example of Contours on Eight-Node Quadrilateral

4. Sub-title of the plot
   Values at Each Node: 1=2.0, 2=1.0, 3=2.0, 4=1.0, 5=2.0, 6=1.0, 7=2.0, 8=1.0

5. Mesh plotted flag (True/False) .............................................................................. f

6. Node numbers only flag (True/False). .................................................................. t

7. Axis plotted flag (True/False). .............................................................................. t

8. X-axis label
   X Coordinate

9. Y-axis label
   Y Coordinate

10. Legend label
    Contour Values

Do you wish to change something on the plot (y/n)?

Figure 11 - Example of ISOCOM menu display
As mentioned previously, two types of input files are allowed, the PAFEC backing store (.bs) file or a file containing data in the MODEL program output form [1]. Only the first four letters of the PAFEC backing store file are needed to uniquely identify the file. Because only four letters are read into ISOCON, you must be in the directory containing the backing store file before running the program. Note that the program only reads in the results from the first PAFEC load case. Multiple load cases should be handled as separate PAFEC jobs.

The file containing data in MODEL program output form is read in free format. The first line contains the following: (1) number of nodes in the entire mesh (NODES), (2) number of different element types or groups in the mesh (NGROUP), (3) number of contours to be plotted (NCONTR) and (4) a flag (.FALSE. or .TRUE.) indicating whether the contour values are to be automatically selected by ISOCON. If the contour values are not automatically selected, the next NCONTR lines specify the contour values. The next group of lines (NODES long) specify the global node number, the global x and y coordinates of the node and the value to be contoured at that node for each node in the mesh. The following line contains the number of elements in the first group (NELTS) and the element type number of that group. Figure 12 shows the element type numbers for each acceptable element in ISOCON. The next NELTS lines give the global node number followed by element connectivity data. This element data is repeated until all elements in all groups are specified.

5.2. ISOCON Test Cases

Test cases were run to verify proper ISOCON program operation and to compare the contour plots with those generated by the PIGS program. Figure 13 shows an example of a contour plot over a single eight-node quadrilateral element. Other single elements were plotted to verify that the proper interpolation functions were input correctly
Figure 12 - Element type number used in MODEL input format
Example of Contours on Eight-Node Quadrilateral

Values at Each Node: 1=2.0, 2=1.0, 3=2.0, 4=1.0, 5=2.0, 6=1.0, 7=2.0, 8=1.0

Contour Values:
A = 5.000 x 10^-3
B = 1.000 x 10^-1
C = 3.000 x 10^-1
D = 7.000 x 10^-1
E = 1.100 x 10^0
F = 1.500 x 10^0

Figure 13 - Contour plotting example: eight-node quadrilateral element
into ISOCON. Figures 14, 15 and 16 show single twelve-node quadrilateral, six-node triangular and nine-node triangular elements respectively.

Figure 17 shows two plots of a finite element model of a potential flow around a cylinder bounded by parallel walls. The top plot uses 120 linear triangular elements. Note that this type of element generates linear contours within each element. These elements give a poor contour representation of the lower streamline values near the cylinder since the radius of curvature of the contours in this region is small. The middle plot shows the same flow region analyzed using 60 bi-linear quadrilateral elements. Note that even the lowest value streamline (0.05) compares very well to the plot of the same streamline computed analytically.

The next example uses the finite element method to analyze stresses in a flat plate with a shoulder under an axially applied load. The mesh in this example is very coarse containing only three eight-node quadrilateral elements. The model includes only one half of the plate because of symmetry about the x axis. Figure 18 shows a plot of stress contours generated by ISOCON and stress contours generated by PIGS for this case. Since the PIGS program uses a form of linear interpolation to compute contour locations, many of the contours over these quadratic elements are poorly defined. In fact, the contour line labeled C has a rather large discrepancy between the two plots. More insight into the behavior of the stress contours is gained by viewing the continuous color contour plot shown in Figure 19. Note the area where contour line C is located in the previous figure. This can be seen from the figure to be a saddle point.

Two examples of axisymmetric stress contours are shown in Figures 20 through 22. Figures 20 and 21 show a line contour plot and a continuous color contour plot of an axisymmetric flywheel segment. The stresses are caused from a rotation rate of 3000 rpm. The flywheel was modeled with 14 eight-node quadrilateral elements. Figure 22 shows the loads and boundary constraints along with a comparison of the Von Mises
Example of Contours on Six-Node Triangle

Values at Each Node:
1 = 2.0, 2 = 2.0, 3 = 2.0, 4 = 1.5, 5 = 1.0, 6 = 1.0

Figure 15 - Contour plotting example: six-node triangular element
Example of Contours on Nine-Node Triangle

Values at Each Node 1=0.0, 2=1.0, 3=1.5, 4-7=2.5, 8=2.0, 9=1.0

Contour Values
A = 4.167 x 10^{-1}
B = 6.250 x 10^{-1}
C = 8.333 x 10^{-1}
D = 1.042 x 10^0
E = 1.250 x 10^0
F = 1.450 x 10^0
G = 1.667 x 10^0
H = 1.875 x 10^0
I = 2.083 x 10^0
J = 2.292 x 10^0

Figure 16 - Contour plotting example: nine-node triangular element
Figure 17 - Comparison of T3 and Q4 elements: potential flow around cylinder example
Figure 18 - ISOCON/PIGS comparison: simple stress example
Figure 19 - Continuous color plot: simple stress example
Figure 20 - ISOCON/PIGS comparison: axisymmetric flywheel example
Figure 21 - Continuous color plot: axisymmetric flywheel example
Axisymmetric Stress Example
Using Eight-Node Quadrilateral Elements

ISOCON contour plot

PIGS contour plot

VON MISES
A = 6.000 \times 10^4
B = 1.100 \times 10^5
C = 1.600 \times 10^5
D = 2.000 \times 10^5
E = 2.500 \times 10^5
F = 3.000 \times 10^5
G = 3.500 \times 10^5
H = 4.000 \times 10^5
I = 4.500 \times 10^5
J = 5.000 \times 10^5

VON MISES / 10
A = 0.06
B = 0.11
C = 0.16
D = 0.20
E = 0.25
F = 0.30
G = 0.35
H = 0.40
I = 0.45
J = 0.50

Figure 22 - ISOCON/PIGS comparison: axisymmetric stress example
stresses plotted by both ISOCON and PIGS. In both examples, ISOCON produces smoother and more realistic contours than those produced by PIGS.

Figure 23 shows thermal contour plots generated from a finite element analysis of a triangular cooling fin. This example uses both quadratic triangular and quadrilateral elements in the analysis. Figure 24 shows an enlargement of the contour labeled H. Note this contour has a large discontinuity in slope between the elements. The main reason for this discontinuity is the very coarse mesh used in the analysis. Figure 25 also shows this discontinuity in the continuous color contour plot. Thus the ISOCON program can be used to aid in the development of an adequate mesh.

Figure 26 shows an example with a fixed temperature constraint of 0 °C on the left side’s bottom, center and top nodes. Also, a fixed temperature constraint of 100 °C is applied to the right side’s bottom, center and top nodes. Note the warmer (left side) or cooler (right side) spots between the nodes on the fine mesh contour plot. The coarse mesh shows no temperature variations between the nodes. Therefore, since these temperature variations are expected, the fine mesh gives more realistic contours. Thus, the mesh resolution is one factor to consider when viewing ISOCON generated contour plots.
Thermal Analysis of Triangular Cooling Fin

ISOCON contour plot

TEMPERATURES
A = 5.000 x 10^6
B = 1.500 x 10^1
C = 2.500 x 10^1
D = 3.500 x 10^1
E = 4.500 x 10^1
F = 5.500 x 10^1
G = 6.500 x 10^1
H = 7.550 x 10^1
I = 8.550 x 10^1
J = 9.550 x 10^1

PIGS contour plot

TEMP FIELD / 10^6
A = 0.05
B = 0.15
C = 0.25
D = 0.35
E = 0.45
F = 0.55
G = 0.65
H = 0.75
I = 0.85
J = 0.95

Figure 23 - ISOCON/PIGS comparison: triangular cooling fin example
Figure 24 - Enlargement showing temperature discontinuity of contour H
Figure 2.5 - Continuous color contour plot: triangular cooling fin example
Figure 26 - Coarse versus fine mesh comparison: simple thermal example
6. SUMMARY

The ability to plot contours over isoparametric finite elements is a useful addition to any finite element post-processing library. Many contour plotting programs use linear interpolation to approximate the contour locations. The ISOCON program, however, uses the element interpolation functions to plot smoother, more realistic contours. The program also has the capability to fill elements with color to represent various contour levels. This allows the user to view a nearly continuous range of contours limited only by the number of colors available on the terminal.

Comparisons between plots from ISOCON and from the PAFEC interactive graphics suite, PIGS, show differences in the contour locations. The PIGS-generated contours sometimes have large discontinuities in slope within elements. ISOCON, however, shows no contours with discontinuous slopes inside element boundaries. One cause of large changes in the slope of contours between elements is a coarse mesh. Of course, the slope of contours between elements is not guaranteed to be continuous since the elements used in ISOCON all possess $C^0$ continuity.

Another advantage of ISOCON over some other contour plotting programs is that most of the plot drivers are written in FORTRAN 77 and included within the program. Thus, only basic plot commands need to be added to ISOCON before other graphics devices are used.
References


Appendix

Selected ISOCON Subroutines
PROGRAM ISOCON

PURPOSE: This main program uses the inter-element information
(i.e. the shape functions and their derivatives) to compute
and plot contour lines for quadrilaterals and triangular
isoparametric elements.

PARAMETER (MAXAXS = 76, MAXBUF = 500, MAXCON = 26, MAXELT = 500,
1 MAXLEG = 14, MAXLND = 12, MAXNOD = 2000,
2 MAXPNT = 4000, MAXRNG = 15, MAXTIT = 51, MAXVAL = 1000,
3 NSPACE = 2)

Common IOUNIT defines the units numbers of all external files
used in ISOCON.

COMMON /IOUNIT/ LASSCR, LASDMP, IDBDMP

CHARACTER*16 FILE
CHARACTER*80 AXIS, IDFORM, LEGEND, STITLE, TITLE
LOGICAL CALCCV, DAXIS, DEBUG, INDFLG, NODFLG, PLTMSH, WFLAG

Dimension the internal variables

DIMENSION AFIRST(NSPACE), AXIS(NSPACE), BUFLIN(MAXBUF,NSPACE),
1 CONBUF(MAXBUF,NSPACE), COORMN(NSPACE), COORMX(NSPACE),
2 CSIZE(3), DER(NSPACE), DH(NSPACE,MAXLND), DLUSE(MAXELT),
3 GVNORM(MAXNOD,NSPACE), H(MAXLND), INDFLG(MAXNOD),
4 LEMCON(MAXLND,MAXELT), LEMTPE(MAXELT), NDIV(MAXELT),
5 PLTMAX(NSPACE), PLTMIN(NSPACE), PSIZE(NSPACE),
6 SCRMX(NSPACE), SCRMN(NSPACE), UNPIN(NSPACE),
7 VALUES(MAXNOD), VALBUF(MAXVAL), VERBUF(MAXPNT,NSPACE),
8 VLEVEL(MAXCON), VMAX(MAXELT), VMID(MAXRNG),
9 VMIN(MAXELT), VNODE(MAXLND), XYZ(MAXNOD,NSPACE),
. XZELT(NSPACE,MAXLND)

Indicate status of the debug printout flag and initialize
the window flag

DATA DEBUG, WFLAG / .FALSE., .FALSE. /

Open the laser scratch file and debug file, if used

OPEN (UNIT = LASSCR, STATUS = 'SCRATCH', FORM = 'UNFORMATTED')
IF (DEBUG) OPEN (UNIT = IDBDMP, FILE = 'datout')

Input the data

CALL GETDAT (MAXAXS, MAXCON, MAXELT, MAXLEG, MAXLND, MAXNOD,
1 MAXRNG, MAXTIT, NSPACE, AXIS, CALCCV, DAXIS,
2 DFILE, ICONTR, IDFORM, ITERM, LEGEND, LEMCON,
3 LEMTPE, NCONTR, NEILTS, NGROUP, NODES, NODFLG,
4 PLTMSH, STITLE, TITLE, VALUES, VLEVEL, XYZ)

Initialize the plot device

CALL PLTIN (AXIS, COORMN, COORMX, DAXIS, ICONTR, ITERM, LEGEND,
 MAXNOD, NODES, NODFLG, NSPACE, PLTMSH, STITLE, TITLE,
 WFLAG, XYZ, AFIRST, CSIZE, ITERM, PLTMAX, PLTMIN,
 PSSCALE, SCRMAX, SCRMIN, UNPIN)

C Draw the global outlines
CALL DRWGLB (AFIRST, BUFLIN, COORMN, COORMX, CSIZE, GVNORM,
  H, INDFLG, ITERM, LEMCON, LEMTPE, MAXBUF,
  MAXELT, MAXLND, MAXNOD, NELTS, NODES, NODFLG,
  NSPACE, PLTMAX, PLTMIN, PLTMSH, PSSCALE, XYZ,
  XZELT)

C Determine the element maxima and minima
IF (ICONTR .EQ. 1 .OR. ICONTR .EQ. 2) THEN
  CALL ELMXXN (CALCVC, H, LEMCON, LEMTPE, MAXCON, MAXELT, MAXLND,
   MAXNOD, NCONTR, NELTS, NSPACE, PLTMAX, PLTMIN,
   PSSCALE, UNPIN, VALUES, VNODE, XYZ, XZELT, DLUSE,
   NDIV, VLEVEL, VMAX, VMIN, VOMAX, VOMIN)
ENDIF

C Draw the element contours
IF (ICONTR .EQ. 1) THEN
  CALL DCONTR (AFIRST, CONBUF, COORMN, COORMX, CSIZE, DEBUG,
    DER, DH, DLUSE, H, ITERM, LEMCON, LEMTPE,
    MAXBUF, MAXCON, MAXELT, MAXLND, MAXNOD, NCONTR,
    NELTS, NSPACE, PLTMAX, PLTMIN, PSSCALE, UNPIN,
    VALUES, VLEVEL, VMAX, VMIN, VNODE, XYZ, XZELT)
ELSE
  CALL PCONTR (AFIRST, COORMN, COORMX, CSIZE, DEBUG, H,
    ITERM, LEMCON, LEMTPE, MAXBUF, MAXCON,
    MAXELT, MAXLND, MAXNOD, MAXPNT, MAXRNG, MAXVAL,
    NCONTR, NDIV, NELTS, NSPACE, PLTMAX, PLTMIN,
    PSSCALE, VALBUF, VALUES, VERBUF, VLEVEL, VMID,
    VNODE, VOMAX, VOMIN, XYZ, XZELT)
ENDIF

C Determine if window option is to be used
CALL WINDOW (AFIRST, CSIZE, ITERM, NSPACE, PLTMAX, PLTMIN,
  PSSCALE, COORMN, COORMX, WFLAG)

C Finalize the plotter
CALL PLTFIN (ITERM)

C If windowing, reset plot file and start again
IF (WFLAG) THEN
  CLOSE (LASSCR)
  OPEN (UNIT = LASSCR, STATUS = 'SCRATCH', FORM = 'UNFORMATTED')
  GOTO 10
ENDIF
Determine if anything on the plot needs to be changed

CALL INTDAT (ITERM, MAXAXS, MAXCON, MAXELT, MAXLEG, MAXIND,
1 MAXNOD, MAXRNG, MAXTIT, NSPACE, AXIS, CALCCV,
2 DAXIS, DFILE, ICONTR, IDFORM, INTFLG, LEGEND,
3 LECMD, LEMTPE, NCONTR, MELTS, NGROUP, NODES,
4 NODFLG, PLMTSH, STITE, TITLE, VALUES, VLEVEL,
5 XYZ)

If something on plot is to be changed, start again

IF (INTFLG .GT. 0) GOTO 10

Open the laser dump file

OPEN (UNIT = LASDMP, FILE = 'lasout')

Write out plot data to be used by the laser printer

CALL LASWRT (SCRMAX, SCRMIN)

Close the output files

CLOSE (LASSCR)
CLOSE (LASDMP)
IF (DEBUG) CLOSE(IDBDMP)

Indicate a normal ending to ISOCON

STOP 'Normal ending of ISOCON'
END
SUBROUTINE DCONTR (AFIRST, CONBUF, COORMN, COORMX, CSIZE, DEBUG,
  DER, DH, DLUSE, H, ITERM, LEMCON, LEMTPE,
  MAXBUF, MAXCON, MAXELT, MAXLND, MAXNOD, NCONTR,
  NELTS, NSPACE, PLTMAX, PLTMIN, PSCALE, UNPIN,
  VALUES, VLEVEL, VMAX, VMIN, VNOD, XYZ, XYZELT)

SUBROUTINE DCONTR

PURPOSE:  This subroutine uses subroutines CONTRC, FNDLRS, GUELRS,
          PRECON, and SETSEC to compute and draw the element
          contours.

Common IOUNIT defines the units numbers of all external files
used in ISOCON

 COMMON /IOUNIT/ LASSCR, LASDMP, IDBDMP

 COMMON /LOCNOD/ - Local coordinate node storage

 COMMON /LOCNOD/ LENOD(84,6), LENV(84,6), CLR(49,6), CLS(49,6)

 LOGICAL DEBUG, ELNFLG, LPFLAG

 Dimension the external variables

 DIMENSION AFIRST(NSPACE), CONBUF(MAXBUF,NSPACE), COORMN(NSPACE),
  COORMX(NSPACE), CSIZE(3), DER(NSPACE),
  DH(NSPACE,MALND), DLUSE(MALTE), H(MALND),
  LEMCON(MALND,MALTE), LEMTPE(MALTE), PLTMAX(2),
  PLTMIN(2), PSCALE(NSPACE), UNPIN(NSPACE),
  VALUES(MALND), VLEVEL(MAXCON), VMAX(MALTE),
  VMIN(MALTE), VNOD(MALND), XYZ(MALND,NSPACE),
  XYZELT(NSPACE,MALND)

 Dimension the internal variables

 DIMENSION INTLIN(84), INTNOD(49), VNLOC(49)

 ** Begin the loop on the contour levels **

 DO 80 NCONCT = 1,NCONTR
  ITOTAL = 0
  LPFLAG = .FALSE.
  VCONTR = VLEVEL(NCONCT)

 Determine the color of this contour line

 CALL PLTCOL (ITERM, VLEVEL(NCONTR), VLEVEL(1), VCONTR)

 ** Begin the loop over the elements **

 DO 70 IELT = 1,NELTS
  ELNFLG = .TRUE.

 Print out the contour level and the minimum and maximum levels

   IF (DEBUG) WRITE (IDBDMP, 1000) NCONCT, VCONTR, VMIN(IELT),
                            VMAX(IELT)
1000      FORMAT('//' , ' * DCONTR ','*,5X,'CONTOUR=','*,I3, ' VCONTR=','*,  
           E10.3, ' VMIN=','*,E10.3, ' VMAX=','*,E10.3)
1
C Does the element contain a contour line?
C        IF (VCONTR .GT. VMIN(IELT) .AND.
C                  VCONTR .LT. VMAX(IELT)) THEN
C Determine number of nodes (NODEXT), number of pseudo-nodes (NLLNOD),
C and the number of pseudo-lines (NLLIN) for this element
C
   LEMTYP = LEMTYP(IELT)
   IF (LEMTPY .LE. 3) THEN
      NODEXT = LEMTYP*4
      NLLNOD = 1.5*LEMTPY**2 + 16.5*LEMTPY - 14
      NLLIN = (LEMTPY**2 + 6*LEMTPY - 6) * 4
   ELSE IF (LEMTPY .LE. 6) THEN
      NODEXT = (LEMTPY - 3)*3
      NLLNOD = (LEMTPY**2 + 15*LEMTPY - 70) / 2
      NLLIN = 1.5*(LEMTPY**2 + 5*LEMTPY - 34)
   ELSE
      STOP '?DCONTR - ICORRECT ELEMENT TYPE'
   END IF

C Print out the element number and the number of external nodes, number
C of pseudo-nodes, and number of pseudo-lines
C
1010      FORMAT('//' , ' * DCONTR ','*,5X,'I3, ' NODEXT=','*,I2,  
           ' NLLNOD=','*,I2, ' NLLIN=','*,I2)

C Fill up the element matrices
C
   CALL PRECON (H, LEMCON, LEMTYP, MAXELT, MAXNOD, MAXNOD,  
                NLLNOD, NODEXT, IELT, NSFACE, VALUES, XYZ,  
                VNLOC, VNOD, XYZELT)

C Decide if this element is inside the window
C
   NODCNT = 0
   DO 10 I = 1,NODEXT
      IF (XYZELT(2,I) .LT. COORMN(2)) NODCNT = NODCNT + 1
         CONTINUE
      IF (NODCNT .EQ. NODEXT) ELNFLG = .FALSE.
         NODCNT = 0
   DO 20 I = 1,NODEXT
      IF (XYZELT(2,I) .GT. COORMX(2)) NODCNT = NODCNT + 1
         CONTINUE
      IF (NODCNT .EQ. NODEXT) ELNFLG = .FALSE.
         NODCNT = 0
   DO 30 I = 1,NODEXT
      IF (XYZELT(1,I) .LT. COORMN(1)) NODCNT = NODCNT + 1
         CONTINUE
      IF (NODCNT .EQ. NODEXT) ELNFLG = .FALSE.
         NODCNT = 0
   DO 40 I = 1,NODEXT
      IF (XYZELT(1,I) .GT. COORMX(1)) NODCNT = NODCNT + 1
         CONTINUE

10  GO TO 10
20  GO TO 20
30  GO TO 30
40  GO TO 40
IF (NODCNT .EQ. NODEXT) ELNFLG = .FALSE.
IF (ELNFLG) THEN

Define a step increment (DL) for this element
DL = DLUSE(IELT)
IF (LEMTYP .GE. 4) DL = DL*0.5

Set the contour intersection vectors
CALL SETSEC (DER, DH, INTLIN, INTNOD, LEMTYP, LINHIT, 
MAXLND, NLLIN, NLNOD, NODEXT, NODHIT, 
NSPACE, VCONTR, VNLOC, VNOD)
ITOTAL = ITOTAL + NODHIT + LINHIT

Check for the special case of a contour intersecting an element at 
one point
IF (NODHIT .GT. 1 .OR. LINHIT .GT. 0) THEN

Reset the plot counter for T3 elements
KOUNT = 0

** Begin loop to draw contours which have nodal intersections **
IF (NODHIT .GT. 0) THEN
DO 50 I = 1,NODHIT

Print out the contour number and the node hit number
IF (DEBUG) WRITE (IDBDMP, 1020) NCONCT, I
1020
FORMAT(' * DCONTR *',5X, 'CONTOUR= ',12, 
' * NODHIT= ',I2)

Has the pseudo-node already been hit by the contour?
NODE = INTNOD(I)
IF (NODE .GT. 0) THEN

Negate the intersection pointer to indicate a beginning
INTNOD(I) = -NODE

Get the starting coordinates
RLOCAL = CLR(NODE, LEMTYP)
SLOCAL = CLS(NODE, LEMTYP)

If the element is a three node triangle, simply use linear 
interpolation and draw a straight line through the element
IF (LEMTYP .EQ. 4) THEN
CALL CONT3T (AFIRST, CONBUF, CSIZE, H, ITERM, 
KOUNT, LEMTYP, LFLAG, MAXBUF, 
MAXLND, NCONCT, NODEXT, NSPACE, 
PLTMAX, PLTMIN, PSCALE, RLOCAL, 
SLOCAL, UNPIN, XYZELT)
ELSE

C March along the contour
C
    CALL CONTRC (AFIRST, CONBUF, CSIZE, DEBUG,
1      DER, DH, DL, H, INTLIN, INTNOD,
2      ITERM, LEMTYP, LINE, LINHIT,
3      LFLAG, MAXBUF, MAXLND, NCONCT,
4      NODE, NODEXT, NODEHIT, NSPACE,
5      PLTMAX, PLTMIN, PSCALE, RLOCAL,
6      RNEW, SLOCAL, SNEW, UNPIN,
7      VCONTR, VNOID, XYZELT)

END IF
END IF
CONTINUE

50

C ** End of loop on the pseudo-node intersections **
C
END IF

C ** Begin loop to draw contours which have line intersections **
C
IF (LINHIT .GT. 0) THEN
DO 60 I = 1, LINHIT

C Print out the contour number and the line hit number
C
    IF (DEBUG) WRITE (IDBDMP, 1030) NCONCT, I
1030        FORMAT(5X, 'CONTR ', 'CONTOUR= ', 'LINEHIT= ', I2)

C Has the pseudo-line already been intersected by the contour line?
C
    LINE = INTLIN(I)
    IF (LINE .GT. 0) THEN

C No, so begin contour by negating the intersection pointer
C
        INTLIN(I) = -LINE

C Guess the intersected local coordinates of the point of intersection
C
    CALL GUELRS (LEMTYP, LINE, LSEGID, RB, RE,
1      RNEW, SB, SE, SLOPE, SNEW,
2      VCONTR, VNLOC, YINTER)

C If the element is a three node triangle, simply use linear
C interpolation and draw a straight line through the element
C
    IF (LEMTYP .EQ. 4) THEN
        CALL CONT3T (AFIRST, CONBUF, CSIZE, H, ITERM,
1          KOUNT, LEMTYP, LFLAG, MAXBUF,
2          MAXLND, NCONCT, NODEXT, NSPACE,
3          PLTMAX, PLTMIN, PSCALE, RNEW,
4          SNEW, UNPIN, XYZELT)

    ELSE

C Iterate to approximate the correct initial values

      END IF
50

C END IF
CALL FNDLRS (DER, DH, H, LEMTYP, LSEGID,
   MAXLND, NODEXT, NSPACE, RB,
   RE, RNEW, SB, SE, SLOPE, SNEW,
   VCONTR, VNOD, YINTER)

C March along the contour
C
RLOCAL = RNEW
SLOCAL = SNEW
CALL CONTRC (AFIRST, CONBUF, CSIZE, DEBUG,
   DER, DH, DL, H, INTLIN, INTNOD,
   ITERM, LEMTYP, LINE, LINHIT,
   LPFLAG, MAXBUF, MAXLND, NCONCT,
   NODE, NODEXT, NODENT, NSPACE,
   PLTMAX, PLTMIN, PSCALE, RLOCAL,
   RNEW, SLOCAL, SNEW, UNPIN,
   VCONTR, VNOD, XYZELT)

END IF
END IF
CONTINUE

C ** End of loop on line intersections **
C

END IF

C Plot the remaining points in the buffer for T3 elements
C
IF (LEMTYP .EQ. 4) CALL PLTBUF (AFIRST, CONBUF, ITERM,
   KOUNT, LPFLAG, MAXBUF,
   NSPACE, PLTMAX,
   PLTMIN, PSSCALE)

END IF
END IF
CONTINUE

C ** End of loop over the elements **
C
Print out the contour level in the legend
C
IF (ITOTAL .GT. 1 .AND. LPFLAG) THEN
   CALL PLTLEG (CSIZE, ITERM, NCONCT, PLTMAX, VCONTR)
END IF
CONTINUE

C ** End of loop on the contour levels **
C
RETURN
END
SUBROUTINE DRWGLB (AFIRST, BUFLIN, COORMN, COORMX, CSIZE, GVNORM, 
1 H, INDFLG, ITERM, LEMCON, LEMTPE, MAXBUF, 
2 MAXELT, MAXLND, MAXNOD, NELTS, NODES, NODFLG, 
3 NSPACE, PLTMAX, PLTMIN, PLTMSH, PSSCALE, XYZ, 
4 XYZELT)

SUBROUTINE DRWGLB

PURPOSE: This subroutine uses subroutines BEGLIN, DRWLIN, and ENDLIN 
1 to draw the global boundaries.

COMMON LCOORD - Local boundary coordinates

COMMON /LCOORD/ CORLOC(8,3,3)

LOGICAL ELMFLG, IELMFLG, INDFLG, NODFLG, PLTMSH

Dimension the external variables

DIMENSION AFIRST(NSPACE), BUFLIN(MAXBUF,NSPACE), COORMN(NSPACE), 
1 COORMX(NSPACE), CSIZE(3), GVNORM(MAXNOD,NSPACE), 
2 H(MAXNOD), INDFLG(MAXNOD), LEMCON(MAXLND,MAXELT), 
3 LEMTPE(MAXELT), PLTMAX(2), PLTMIN(2), PSSCALE(NSPACE), 
4 XYZ(MAXNOD,NSPACE), XXYZELT(NSPACE,MAXLND)

Dimension the internal variable

DIMENSION AVGNRM(3)

DATA NLINE, TOL /40, 0.5/

Plot the node numbers if desired

IF (NODFLG) THEN
1 CALL PLTND (AFIRST, CSIZE(1), ITERM, INDFLG, LEMCON, LEMTPE, 
1 MAXELT, MAXLND, MAXNOD, NELTS, NSPACE, PLTMAX, 
2 PLTMIN, PSSCALE, XYZ)
END IF

Identify the interior nodes using the Assembled Surface Normal 
Algorithm

IF (.NOT. PLTMSH) THEN
1 CALL ASNALG (H, LEMCON, LEMTPE, MAXELT, MAXLND, MAXNOD, NELTS, 
1 NODES, NSPACE, XYZ, XXYZELT, GVNORM, INDFLG)
END IF

** Begin loop over all elements **

DO 130 IELT = 1,NELTS
1 ELMFLG = .TRUE.

Determine element number, element type, number of nodes per element, 
and whether the element is interior

LEMTYP = LEMTPE(IELT)
IF (LEMTYP .LE. 3) THEN
1 NSTYPE = 1
NODBDY = 4
NODEXT = 4*LEMTYP
ELSE IF (LEMTYP .LE. 6) THEN
  NTYPE = 2
  NODBDY = 3
  NODEXT = (LEMTYP - 3)*3
ELSE
  STOP 'DRWGLB = INCORRECT ELEMENT TYPE'
END IF

C Determine if the element is interior if the mesh is not plotted
C
IF (.NOT.(PLTMSH)) THEN
  IELFLG = .TRUE.
  DO 50 I = 1, NODBDY
      NOD = LEMCON(I,IELT)
      IF (.NOT.INDFLG(NOD)) IELFLG = .FALSE.
  CONTINUE
  IF (IELFLG) GOTO 130
END IF

C Fill the XYZELT array with values
C
DO 70 I = 1, NODEXT
  NOD = LEMCON(I,IELT)
  DO 60 J = 1, NSFACE
      XYZELT(J,I) = XYZ(NOD,J)
  60 CONTINUE
  70 CONTINUE

C Decide if this element is inside the window
C
NODCNT = 0
DO 10 I = 1, NODEXT
    IF (XYZELT(2,I) .LT. COORMN(2)) NODCNT = NODCNT + 1
    10 CONTINUE
    IF (NODCNT .EQ. NODEXT) ELNFLG = .FALSE.
    NODCNT = 0
    DO 20 I = 1, NODEXT
        IF (XYZELT(2,I) .GT. COORMX(2)) NODCNT = NODCNT + 1
        20 CONTINUE
        IF (NODCNT .EQ. NODEXT) ELNFLG = .FALSE.
        NODCNT = 0
        DO 30 I = 1, NODEXT
            IF (XYZELT(1,I) .LT. COORMN(1)) NODCNT = NODCNT + 1
            30 CONTINUE
            IF (NODCNT .EQ. NODEXT) ELNFLG = .FALSE.
            NODCNT = 0
            DO 40 I = 1, NODEXT
                IF (XYZELT(1,I) .GT. COORMX(1)) NODCNT = NODCNT + 1
                40 CONTINUE
                IF (NODCNT .EQ. NODEXT) ELNFLG = .FALSE.
                IF (.NOT. ELNFLG) GOTO 130

C Draw the two-dimensional linear elements (q4 and t3)
C
IF (LEMTYP .EQ. 1 .OR. LEMTYQ .EQ. 4) THEN
  DO 90 I = 1, NODBDY
  90 CONTINUE

\begin{verbatim}
J = I + 1
IF (I .EQ. NODBDY) J = 1
NOD1 = LEMCON(I,IELT)
NOD2 = LEMCON(J,IELT)
IF (NOT.(INDFLG(NOD1)) .AND. NOT.(INDFLG(NOD2)))
1 OR. PLTMSH) THEN
C This prevents plotting interior lines at corners
C
IF (LEMTYP .EQ. 4 .AND. NOT.PLTMH) THEN
   K = J + 1
IF (J .EQ. NODBDY) K = J + 1
NOD3 = LEMCON(K,IELT)
IF (NOT.(INDFLG(NOD3))) THEN
   TEST = 0.0
   DO 80 L = 1,NSPACE
      AVGNRM(L) = (Gvnrm(NOD1,L) + Gvnrm(NOD2,L))/2.0
      TEST = TEST + AVGNRM(L) - Gvnrm(NOD3,L)
   80 CONTINUE
   IF (ABS(TEST) .LT. TOL) GOTO 90
END IF
END IF
C Plot a line between node 1 and node 2
C
RLOCAL = CORLOC(I,1,NSTYPE)
SLOCAL = CORLOC(I,2,NSTYPE)
CALL BEGLIN (AFIRST, BUFLIN, H, ITERM, LEMTYL,
   1 LINLST, MAXBUF, MAXIND, NODEXT, NSPACE,
   2 PLTMAX, PLTMIN, FSCALE, RLOCAL, SLOCAL,
   3 XZELT)
RLOCAL = CORLOC(J,1,NSTYPE)
SLOCAL = CORLOC(J,2,NSTYPE)
CALL DRWLIN (AFIRST, BUFLIN, H, ITERM, LEMTYL,
   1 LINLST, MAXBUF, MAXIND, NODEXT, NSPACE,
   2 PLTMAX, PLTMIN, FSCALE, RLOCAL, SLOCAL,
   3 XZELT)
CALL ENDLIN (AFIRST, BUFLIN, ITERM, LINLST, MAXBUF,
   1 NSPACE, PLTMAX, PLTMIN, FSCALE)
END IF
CONTINUE
C Draw the quadratic and cubic elements by plotting a straight line
between points defined by increment DLINN. Thus DLINN is the
local coordinate increment size and NLINE is the number of points
per side on a quadrilateral element.
C
ELSE IF (LEMTYP .EQ. 2 .OR. LEMTYL .EQ. 3 .OR.
1 LEMTYL .EQ. 5 .OR. LEMTYL .EQ. 6) THEN
   DO 120 I = 1,NODBDY
      J = I + 1
      IF (I .EQ. NODBDY) J = 1
      NOD1 = LEMCON(I,IELT)
      NOD2 = LEMCON(J,IELT)
      IF (NOT.(INDFLG(NOD1)) .AND. NOT.(INDFLG(NOD2)))
1 OR. PLTMH) THEN
C This prevents plotting interior lines at corners
C
\end{verbatim}
C
IF (LEMTYP .GE. 5 .AND. .NOT.PLTMSH) THEN
   K = J + 1
IF (J .EQ. NOEDDY) K = J + 1
NOD3 = LEMCON(K, IELT)
   IF (.NOT.(INDFLG(NOD3))) THEN
      TEST = 0.0
      DO 100 L = 1, NSPACE
         AVGNRM(L) = (GVNORM(NOD1,L) + GVNORM(NOD2,L))/2.0
      TEST = TEST + AVGNRM(L) - GVNORM(NOD3,L)
      CONTINUE
100 IF (ABS(TEST) .LT. TOL) GOTO 120
   END IF
END IF
C
C  Plot a line between node 1 and node 2
C
   RLOCAL = CORLOC(I,1,NSTYPE)
   SLOCAL = CORLOC(I,2,NSTYPE)
   CALL BEGLIN (AFIRST, BUFLIN, H, ITERM, LEMTPY,
1 LINLST, MAXBUF, MAXLND, NODEXT, NSPACE,
2 PLTMAX, PLTMIN, PSCALE, RLOCAL, SLOCAL,
3 XYZELT)
   DO 110 K = 1, NLINE
      RLOCAL = RLOCAL + (CORLOC(J,1,NSTYPE) -
1 CORLOC(I,1,NSTYPE)) / NLINE
      SLOCAL = SLOCAL + (CORLOC(J,2,NSTYPE) -
1 CORLOC(I,2,NSTYPE)) / NLINE
      CALL DRWLIN (AFIRST, BUFLIN, H, ITERM, LEMTPY,
1 LINLST, MAXBUF, MAXLND, NODEXT, NSPACE,
2 PLTMAX, PLTMIN, PSSCALE, RLOCAL, SLOCAL,
3 XYZELT)
110 CONTINUE
   CALL ENDLIN (AFIRST, BUFLIN, ITERM, LINLST, MAXBUF,
1 NSPACE, PLTMAX, PLTMIN, PSSCALE)
   END IF
END IF
C
C  Else draw the three-dimensional element boundaries
C
   ELSE
      STOP 'DRWGLB - ERROR, NO 3-D CAPABILITIES'
   END IF
C
C  Plot the lines in the buffer, even if the buffer is not full
C
C
130 CONTINUE
C
C ** End the loop over the elements **
C
   RETURN
END
SUBROUTINE ELMAXMN (CALCCV, H, LEMCON, LEMTPE, MAXCON, MAXELT,
1     MAXLND, MAXNOD, NCONTR, NELTS, NSPACE, PLTMAX,
2     PLTMIN, PSSCALE, UNFIN, VALUES, VNODE, XYZ,
3     XYZELT, DLUSE, NDIV, VLEVEL, VMAX, VMIN,
4     VOMAX, VOMIN)

SUBROUTINE ELMAXMN

PURPOSE: This subroutine uses subroutines DOT, ESHAPE, and PRECON to determine the maximum and minimum nodal values of the variable (VALUE) to be contoured in a specific element. This subroutine also determines the step size dl.

PARAMETER (PI = 3.141592653589793)

COMMON /LOCNOD/- Local coordinate node storage

COMMON /LOCNOD/ LBNOD(84,6), LENOD(84,6), CLR(49,6), CLS(49,6)

LOGICAL CALCCV

Dimension the external variables

DIMENSION DLUSE(MAXELT), H(MAXLND), LEMCON(MAXLND,MAXELT),
1     LEMTPE(MAXELT), NDIV(MAXELT), PLTMAX(2), PLTMIN(2),
2     PSSCALE(NSPACE), UNFIN(NSPACE), VALUES(MAXNOD),
3     VLEVEL(MAXCON), VMAX(MAXELT), VMIN(MAXELT),
4     VNODE(MAXLND), XYZ(MAXLND,NSPACE),
5     XYZELT(NSPACE,MAXLND)

Dimension the internal variables

DIMENSION DLBASE(3), NDIVBS(2)

DATA DLBASE /0.1, 0.05, 0.02/
DATA NDIVBS /10, 50/

VOMAX = VALUES(1)
VOMIN = VOMAX

Compute the plottable screen area in plot units

AREASC = ((PLTMAX(1) - PLTMIN(1))/PSSCALE(1))*
1    ((PLTMAX(2) - PLTMIN(2))/PSSCALE(2))

** Begin loop over the elements **

DO 50 IELT = 1, NELTS

Get element type and calculate the number of exterior nodes

LEMTYP = LEMTPE(IELT)
IF (LEMTYP .LE. 3) THEN
    NODEXT = LEMTYP*4
    NLNOD = 1.5*LEMTYP**2 + 16.5*LEMTYP - 14
ELSE IF (LEMTYP .LE. 6) THEN
    NODEXT = (LEMTYP - 3)*3
    NLNOD = 0.5*LEMTYP**2 + 7.5*LEMTYP - 35

50 CONTINUE
ELSE
    STOP 'ELMXML - INCORRECT ELEMENT TYPE'
END IF

C C Fill up the VNOD and XYZELT arrays
C
DO 10 I = 1,NODEXT
    NOD = LEMCON(I,IELT)
    VNOD(I) = VALUES(NOD)
    DO 20 J = 1,NSPACE
        XYZELT(J,I) = XYZ(NOD,J)
    20   CONTINUE
10   CONTINUE
C C Fill up VMAX and VMIN
C
    VVTMP = VNOD(1)
    VVMAX = VVTMP
    VVMIN = VVTMP
C
C C Loop over the pseudo-nodes and determine the minimum and maximum
C values within the element
C
DO 30 I = 1,NLNOD
    RLOCAL = CLR(I,LEMTYP)
    SLOCAL = CLS(I,LEMTYP)
    CALL ESHAPE (LEMTYP, NODEXT, RLOCAL, SLOCAL, H)
    VVTMP = DOT(H,VNOD,NODEXT)
    IF (VVTMP .GT. VVMAX) VVMAX = VVTMP
    IF (VVTMP .LT. VVMIN) VVMIN = VVTMP
30   CONTINUE
C C Store VVMAX and VVMIN
C
    VVMAX(IELT) = VVMAX
    VVMIN(IELT) = VVMIN
C C Fill up the DLUSE vector
C
    AREAEL = 0.0
    DO 40 I = 1,NODEXT
        J = I + 1
        IF (I .EQ. NODEXT ) J = 1

    40   CONTINUE

C C Approximate the element area using the polygonal area formula
C
    AREAEL = AREAEL + 0.5* (XYZELT(1,I)*XYZELT(2,J) -
                             XYZELT(1,J)*XYZELT(2,I))
C
C C Determine the percentage of screen area the element takes up
C
    APCENT = AREAEL / AREASC
C C
C C Compute the diameter of a circle with the same area as the element
C
    DIAMTR = SQRT(AREAEL*4.0/PI)
Convert this diameter from plotting units to inches

\[ \text{DIAMIN} = \text{DIAMTR}/\text{UNPIN}(1) \]

Based upon the guessed diameter of the polygon and the plotting scale factor choose a DL.

\[ \text{LB} = 1 \]
\[ \text{IF} (\text{DIAMIN} \ .GT. 1.0) \text{ LB} = 2 \]
\[ \text{IF} (\text{DIAMIN} \ .GT. 3.0) \text{ LB} = 3 \]

Store the increment DL in the DLUSE array

\[ \text{DLUSE}(\text{IELT}) = \text{DLBASE}(\text{LB}) \]

Determine the increment DL in the DLUSE array and the number of polygon divisions (NDIVSN) in the NDIV array

\[ \text{IF} (\text{APCENT} \ .LT. 0.08) \text{ THEN} \]
\[ \text{NDIV}(\text{IELT}) = -\text{APCENT}^2 \times 8617.1 + \text{APCENT} \times 1191.3 + \text{NDIVBS}(1) \]
\[ \text{ELSE} \]
\[ \text{NDIV}(\text{IELT}) = \text{NDIVBS}(2) \]
\[ \text{END IF} \]

Now keep track of the overall maxima and minima

\[ \text{VOMAX} = \text{AMAX1}(\text{VOMAX}, \text{VVMAX}) \]
\[ \text{VOMIN} = \text{AMIN1}(\text{VOMIN}, \text{VVMIN}) \]

** End of loop on elements **

Calculate the contour levels if necessary

\[ \text{IF} (\text{CALCCV}) \text{ THEN} \]
\[ \text{DC} = (\text{VOMAX} - \text{VOMIN})/\text{FLOAT}(\text{NCONTR} + 2) \]
\[ \text{DO 60 I} = 1, \text{NCONTR} \]
\[ \text{VLEVEL}(I) = \text{VOMIN} + \text{DC} \times \text{FLOAT}(I + 1) \]
\[ \text{60 CONTINUE} \]
\[ \text{END IF} \]

Sort the contour values

\[ \text{CALL BSORT (VLEVEL, NCONTR)} \]
\[ \text{RETURN} \]
\[ \text{END} \]
SUBROUTINE GETDAT (MAXAXS, MAXCON, MAXELT, MAXLEG, MAXLND, MAXNOD, 
MAXRNG, MAXTIT, NSPACE, AXIS, CALCCV, DAXIS, 
DFILE, ICONTR, IDFORM, ITERM, LEGEND, LEMCON, 
LEMTPE, NCONTR, NELTS, NGROUP, NODES, NODFIG, 
PLTMSH, STITLE, TITLE, VALUES, VLEVEL, XYZ)

SUBROUTINE GETDAT

PURPOSE : This subroutine is the main driver for data input

CHARACTER*(* ) AXIS, DFILE, IDFORM, LEGEND, STITLE, TITLE
CHARACTER*80 TERM
LOGICAL CALCCV, DAXIS, NODFLG, PLTMSH

Dimension the external variables

DIMENSION AXIS(NSPACE), LEMCON(MAXLND,MAXELT), LEMTPE(MAXELT), 
VALUES(MAXNOD), VLEVEL(MAXCON), XYZ(MAXNOD,NSPACE)

INTFLG = 0
CALL SYSTEM ('clear')

Determine the plot device type

CALL GETENV ('TERM',TERM)
IF (TERM(1:5) .EQ. 'vt100') THEN
  ITERM = 1
ELSE IF (TERM(1:7) .EQ. 'cel7180') THEN
  ITERM = 2
ELSE IF (TERM(1:3) .EQ. 'sun') THEN
  ITERM = 3
ELSE
  STOP 'GETDAT - Improper terminal type'
END IF
WRITE (*.1000)
1000 FORMAT ('')

Determine whether the data is in MODEL format or is to be read from
a PAFEC backing store file

IF (ITEM .EQ. 3) THEN
  WRITE (*.1010)
1010 FORMAT('Is the data in (M)ODEL format or (P)AFEC format?')
READ (*,'(A)') IDFORM
IF (.NOT. (IDFORM(1:1) .EQ. 'm' .OR. IDFORM(1:1) .EQ. 'M' 
1 .OR. IDFORM(1:1) .EQ. 'p' .OR. IDFORM(1:1) .EQ. 'P'))
  THEN
    WRITE (*.1020)
1020 FORMAT('Improper response - try again')
GOTO 10
END IF
ELSE
  IDFORM = 'm'
END IF

Get the interactive data

CALL GINDAT (MAXAXS, MAXCON, MAXELT, MAXLEG, MAXLND, MAXNOD, 
MAXRNG, MAXTIT, NSPACE, AXIS, CALCCV, DAXIS, 
DFILE, ICONTR, IDFORM, ITERM, LEGEND, LEMCON, 
LEMTPE, NCONTR, NELTS, NGROUP, NODES, NODFIG, 
PLTMSH, STITLE, TITLE, VALUES, VLEVEL, XYZ)
DFILE, ICONTR, IDFORM, INTFLG, ITERM, LEGEND,
LEMCON, LEMTPE, NCONTR, NELTS, NGROUP, NODES,
NODFLG, PLTMSH, STITLE, TITLE, VALUES, VLEVEL,
RETURN
END
SUBROUTINE INTDAT (ITERM, MAXAXS, MAXCON, MAXELT, MAXLEG, MAXLND,
1 MAXNOD, MAXNRG, MAXIT, NSPACE, AXIS, CALCCV,
2 DAXIS, DFILE, ICONDRA, IDFORM, INTFLG, LEGEND,
3 LEMCON, LEMTPE, NCONDRA, NELTS, NGROUP, NODES,
4 NOGFILG, PLTMSH, STITLE, TITLE, VALUES, VLEVEL,
5 XYZ)
C
SUBROUTINE INTDAT
C
PURPOSE: This subroutine, INTEractive DATa (INTDAT), lets the
user change items such as title, axis labels, whether
node numbers are to be plotted, etc. without
re-running the program.
C
Common IOUNIT defines the units numbers of all external files
used in ISOCON
C
COMMON /IOUNIT/ LASSCR, LASDM, IDBDMP
C
CHARACTER*80 AXIS, DFILE, IDFORM, LEGEND, STITLE, TITLE
C
CHARACTER*80 ANS

LOGICAL CALCCV, DAXIS, NODFLG, PLTMSH
C
Dimension the external variables
C
DIMENSION AXIS(NSPACE), LEMCON(MAXLND,MAXELT), LEMTPE(MAXELT),
1 VALUES(MAXNOD), VLEVEL(MAXCON), XYZ(MAXNOD,NSPACE)
C
Determine if plot needs changing
C
INTFLG = 0
10 CALL SYSTEM ('clear')
IF (ICONDRA.EQ. 1) THEN
WRITE (*,1000)
1000 FORMAT(/'1. Plot type',54('.'),'contour lines')
ELSE IF (ICONDRA.EQ. 2) THEN
WRITE (*,1010)
1010 FORMAT(/'1. Plot type',52('.'),'contour regions')
ELSE
WRITE (*,1020)
1020 FORMAT(/'1. Plot type',56('.'),'no contour')
END IF
WRITE (*,1030) DFILE, TITLE, STITLE, PLTMSH, NODFLG, DAXIS
1030 FORMAT(/'2. Data file name',46('.'),A16,'/3. Main ',
2 'title of the plot',/4X,A51,'/4. Sub-title of ',
3 'the plot',/4X,A76,'/5. Mesh plotted flag ',
4 '(True/False)',/45('.'),/6. Node numbers only ',
5 'flag (True/False)',/40('.'),/7. Axis plotted ',
6 'flag (True/False)',/45('.'),/I1/
IF (DAXIS) THEN
WRITE (*,1040) AXIS(1), AXIS(2), LEGEND
ELSE
WRITE (*,1050) LEGEND
1050 FORMAT(/'8. Legend label',/4X,A14)
END IF
20 WRITE (*,1060)
1060 FORMAT('Do you wish to change something on the plot (y/n)?')
READ (*, '(A)') ANS
IF (ANS(1:1) .EQ. 'y' .OR. ANS(1:1) .EQ. 'Y') THEN
   REWIND (LASSCR)
30   WRITE (*,1070)
1070 FORMAT('Choose the number corresponding to the item you wish '
, 'to change')
READ (*,*) INTFLG
IF (DAXIS) THEN
   IF (INTFLG .LT. 1 .OR. INTFLG .GT. 10) THEN
      WRITE (*,1080)
   END IF
GOTO 30
END IF
ELSE
   IF (INTFLG .LT. 1 .OR. INTFLG .GT. 8) THEN
      WRITE (*,1080)
   GOTO 30
END IF
END IF
CALL GINDAT (MAXAXS, MAXCON, MAXELT, MAXLEG, MAXLND, MAXNOD,
1       MAXRNG, MAXTIT, NSPACE, AXIS, CALCCV, DAXIS,
2       DFILE, ICONST, IDFORM, INTFLG, ITERM, LEGEND,
3       LEMCON, LEMTPE, NCONST, NELTS, NGROUP, NODES,
4       NODFLG, PLTMSH, STITLE, TITLE, VALUES, VLEVEL,
5       XYZ)
10   GOTO 10

C
C Go here if the user does not wish to change anything
C
ELSE IF (ANS(1:1) .EQ. 'n' .OR. ANS(1:1) .EQ. 'N') THEN
   RETURN
ELSE
   WRITE (*,1080)
   GOTO 20
END IF
RETURN
END
SUBROUTINE LASWRT (SCRMAX, SCRMIN)

SUBROUTINE LASWRT

PURPOSE: This subroutine transforms screen coordinates to laser printer coordinates and then writes this data out to a file.

Common IUNIT defines the units numbers of all external files used in ISOCON

COMMON /IUNIT/ LASSCR, LASDMP, IDEDMP

CHARACTER*7 COMMD, PRECMD
CHARACTER*80 ANS

Dimension the external variables

DIMENSION SCRMAX(2), SCRMIN(2)

Data supplying the maximum and minimum laser coordinates

DATA XLASMN, YLASMN, XLASMX, YLASMX / 0.0, 0.0, 8500.0, 11000.0 /

Data supplying the border dimensions

DATA XBDRMN, YBDRMN, XBDRMX, YBDRMX / 1500.0, 1094.0, 1060.0, 687.5 /

Determine the plottable range in laser coordinates

XLASRG = (XLASMX - XBDRMX) - (XLASMN + XBDRMN)
YLASRG = (YLASMX - YBDRMX) - (YLASMN + YBDRMN)

Determine the maximum and minimum screen coordinates in the rotated system

XSCRMX = -SCRMIN(2)
YSCRMX = SCRMAX(1)
XSCRMN = -SCRMAX(2)
YSCRMN = SCRMIN(1)

Determine the scale between the laser coordinates and screen coordinates

IF (XSCALE .LT. YSCALE) THEN
   SCALE = XSCALE
ELSE
   SCALE = YSCALE
END IF

Determine the offset in laser coordinates

OFFSTX = (XLASRG - (XSCRMX - XSCRMN)*SCALE)*0.5 + XBDRMN
OFFSTY = (YLASRG - (YSCRMX - YSCRMN)*SCALE)*0.5 + YBDRMN

.
C Read in screen coordinates from the laser scratch file,
C transform to laser coordinates, and write out to the
C laser dump file for plotting
C
ENDFILE (LASSCR)
REWIND (LASSCR)
PRECMD = '
C
C ** Start loop to read in data **
C
10 READ (LASSCR, END = 20) COMMAND, X, Y
   PRECMD = COMMAND
   IXLASR = INT((-Y - XSCRMN)*SCALE + OFFSTX)
   JYLASR = INT((X - YSCRMN)*SCALE + OFFSTY)
   IF (COMMAND .EQ. 'linemod') THEN
     IXLASR = INT(X + 0.5)
     JYLASR = INT(Y + 0.5)
   END IF
   WRITE (LASDMP,1000) COMMAND, IXLASR, JYLASR
1000 FORMAT(A7, 2I5)
   GOTO 10
C
C ** End loop **
C
C Send the plot to the laser printer if desired
C
C Determine if user wishes to send last plot to laser printer
C
20 WRITE (*,1010)
1010 FORMAT('Do you wish to send the last plot to the laser ',
     1 'printer (y/n)?)'
     READ (*, '(A)') ANS
     IF (ANS(1:1) .EQ. 'y' .OR. ANS(1:1) .EQ. 'Y') THEN
       WRITE (*,1020)
       1020 FORMAT('Sending to laser printer, don’t touch keyboard ',
     1 'until completed ...........
       CALL SYSTEM ('claser < lasout > grfile.temp')
       CALL SYSTEM ('lpr -Papple -g -r ~/suniso/grfile.temp')
       WRITE (*,1030)
       1030 FORMAT('/Completed ...........'/)
     ELSE IF (ANS(1:1) .EQ. 'n' .OR. ANS(1:1) .EQ. 'N') THEN
       RETURN
     ELSE
       WRITE (*,1040)
       1040 FORMAT('Incorrect response - please try again')
       GOTO 20
     END IF
     END
SUBROUTINE PCONTR (AFIRST, COORMN, COORMX, CSIZE, DEBUG, H,
1       ITERM, LEMCON, LEMTE, MAXBUF, MAXCON,
2       MAXELT, MAXLND, MAXNOD, MAXPNT, MAXRNG,
3       MAXVAL, NCONTR, NDIV, NELTS, NSPACE,
4       PLTMAX, PLTMIN, PScale, VALBUF, VALUES,
5       VERBUF, VLEVEL, VMID, VNODE, VOMAX,
6       VOMIN, XYZ, XYZELT)

SUBROUTINE PCONTR

PURPOSE: To paint the CONtour levels with color or shading
using the technique developed by J.F. Stelzer.
This method subdivides the elements into regions
and determines the color in each region from the
interpolation functions at the centroid of the regions.

LOGICAL DEBUG, ELNFLG

Dimension the external variables

DIMENSION AFIRST(NSPACE), COORMN(NSPACE), COORMX(NSPACE),
1       CSIZE(3), H(MAXLND), LEMCON(MAXLND,MAXELT),
2       LEMTE(MAXELT), NDIV(MAXELT), PLTMAX(2),
3       PLTMIN(2), PScale(NSPACE), VALBUF(MAXVAL),
4       VALUES(MAXNOD), VERBUF(MAXPNT,NSPACE),
5       VLEVEL(MAXCON), VMID(MAXRNG), VNODE(MAXLND),
6       XYZ(MAXNOD,NSPACE), XYZELT(NSPACE,MAXLND)

Determine the midpoint of the contour ranges

IF (NCONTR .NE. 0) THEN
   VMID(1) = (VOMIN + VLEVEL(1))*0.5
   DO 10 I = 2,NCONTR
       VMID(I) = (VLEVEL(I - 1) + VLEVEL(I))*0.5
       CONTINUE
   END IF

Loop over the number of elements

DO 140 IELT = 1,NELTS
   ELNFLG = .TRUE.

Determine the Number of DIVisions in each element

NDIVSN = NDIV(IELT)

Determine the number of nodes (NODEXT) and the distance
between the centroids of each subdivision (SUBDIV)

LEMTYP = LEMTE(IELT)
IF (LEMTYP .LE. 3) THEN
   NODEXT = LEMTYMP*4
   NLNOD = 1.5*LEMTYP**2 + 16.5*LEMTYP - 14
   SUBDIV = 2.0/FLOAT(NDIVSN)
ELSE IF (LEMTYP .GT. 3) .AND. LEMTYMP .LE. 6) THEN
   NODEXT = (LEMTYP - 3)*3
NLNOD = (LEMTYP**2 + 15*LEMTYP - 70)/2
SUBDIV = 1.0/FLOAT(NDIVSN)
ELSE
  STOP 'PCONTR - Incorrect element type'
END IF

C  Fill up the element matrices
  DO 20 I = 1,NODEXT
    NOD = LEMCON(I,IELT)
    VNOD(I) = VALUES(NOD)
    DO 30 J = 1,NSPACE
      XZYTE(J,I) = XYZ(NOD,J)
    30  CONTINUE
  20  CONTINUE

C  Determine if this element is inside the window
  NODCNT = 0
  DO 40 I = 1,NODEXT
    IF (XZYTE(2,I) .LT. COORMN(2)) NODCNT = NODCNT + 1
    40  CONTINUE
  IF (NODCNT .EQ. NODEXT) ELNGL = .FALSE.
  NODCNT = 0
  DO 50 I = 1,NODEXT
    IF (XZYTE(2,I) .GT. COORMX(2)) NODCNT = NODCNT + 1
    50  CONTINUE
  IF (NODCNT .EQ. NODEXT) ELNGL = .FALSE.
  NODCNT = 0
  DO 60 I = 1,NODEXT
    IF (XZYTE(1,I) .LT. COORMN(1)) NODCNT = NODCNT + 1
    60  CONTINUE
  IF (NODCNT .EQ. NODEXT) ELNGL = .FALSE.
  NODCNT = 0
  DO 70 I = 1,NODEXT
    IF (XZYTE(1,I) .GT. COORMX(1)) NODCNT = NODCNT + 1
    70  CONTINUE
  IF (NODCNT .EQ. NODEXT) ELNGL = .FALSE.
  IF (.NOT. ELNGL) GOTO 140

C  Start filling up the buffers with the values of the quantity to
C  be contoured at the centroid of the regions and the vertices
C  of the polygons to be filled with color

C  Reset the buffer counters
  IVALUE = 0
  IPOINT = 0

C  Start the loops for the quadrilateral elements
  IF (LEMTYP .LE. 3) THEN
    DO 100 I = 1,NDIVSN
      R = -1.0 + SUBDIV*0.5
      S = -1.0 + SUBDIV*(FLOAT(I - 1) + 0.5)
      DO 90 J = 1,NDIVSN
        C  Compute the value to be contoured at the element centroid
        C

        90    CONTINUE
    100  CONTINUE
C
IVALE = IVALE + 1
CALL ESHPAE (LEMTP, NODEXT, R, S, H)
VCNTRD = DOT (H, VNOD, NODEXT)
IF (NCONTR .GE. 1) THEN
  IF (VCNTRD .LT. VLEVEL(1)) THEN
    VCNTRD = VMID(1)
  ELSE IF (VCNTRD .GE. VLEVEL(NCONTR)) THEN
    VCNTRD = VMID(NCONTR + 1)
  ELSE
    DO 80 L = 2, NCONTR
      IF (VCNTRD .GE. VLEVEL(L - 1) .AND.
         .LT. VLEVEL(L)) THEN
        VCNTRD = VMID(L)
      END IF
    CONTINUE
  END IF
END IF
VALBUF(IVALE) = VCNTRD

C
C Compute the vertices of this region
C
RPOLY = R - SUBDIV*0.5
SPOLY = S - SUBDIV*0.5
CALL ESHPAE (LEMTP, NODEXT, RPOLY, SPOLY, H)
CALL CVLTOG (H, IPOINT, MAXPNT, MAXLND, NODEXT,
           NSPACE, XYZELT, VERBUF)
RPOLY = R + SUBDIV*0.5
CALL ESHPAE (LEMTP, NODEXT, RPOLY, SPOLY, H)
CALL CVLTOG (H, IPOINT, MAXPNT, MAXLND, NODEXT,
           NSPACE, XYZELT, VERBUF)
SPOLY = S + SUBDIV*0.5
CALL ESHPAE (LEMTP, NODEXT, RPOLY, SPOLY, H)
CALL CVLTOG (H, IPOINT, MAXPNT, MAXLND, NODEXT,
           NSPACE, XYZELT, VERBUF)
RPOLY = R - SUBDIV*0.5
CALL ESHPAE (LEMTP, NODEXT, RPOLY, SPOLY, H)
CALL CVLTOG (H, IPOINT, MAXPNT, MAXLND, NODEXT,
           NSPACE, XYZELT, VERBUF)

C
C Update the "r" coordinate
C
R = R + SUBDIV
C
C Check to see if the buffer is full. If so, go plot it.
C
IF (IVALE .EQ. MAXVAL) THEN
  CALL PLTPBF (AFIRST, ITERM, IPOINT, IVALE, LEMTP,
       MAXPNT, MAXVAL, NSPACE, PLTMAX, PLTMIN,
       PSIZE, VALBUF, VERBUF, VOMAX, VOMIN)
END IF
CONTINUE
90
100

C
C Start the loops for the triangular elements
C
ELSE
  DO 130 I = 1, NDIVSN
R = SUBDIV*0.25
S = SUBDIV*(FLOAT(I - 1) + 0.25)
JLAST = NDIVSN*2 - 1 - ((I - 1)*2)
DO 120 J = 1,JLAST

C Compute the value to be contoured at the element centroid

IVALUE = IVALUE + 1
CALL ESHAPE (LEMTYP, NODEXT, R, S, H)
VCNTRD = DOT (H, VNODE, NODEXT)
IF (NCONTR .GE. 1) THEN
   IF (VCNTRD .LT. VLEVEL(1)) THEN
      VCNTRD = VMID(1)
   ELSE IF (VCNTRD .GE. VLEVEL(NCONTR)) THEN
      VCNTRD = VMID(NCONTR + 1)
   ELSE
      DO 110 L = 2,NCONTR
         IF (VCNTRD .GE. VLEVEL(L - 1)) .AND.
            VCNTRD .LT. VLEVEL(L)) THEN
            VCNTRD = VMID(L)
         END IF
      CONTINUE
110  END IF
END IF
END IF
VALUE(IVALUE) = VCNTRD

C Compute the vertices of this region

IF (MOD(J,2) .NE. 0) THEN
  RPOLY = R - SUBDIV*0.25
  SPOLY = S - SUBDIV*0.25
  CALL CVLTG (LEMTYP, NODEXT, RPOLY, SPOLY, H)
  CALL ESHAPE (LEMTYP, NODEXT, RPOLY, SPOLY, H)
  CALL CVLTG (H, IPOINT, MAXPNT, MAXLND, NODEXT, NSPACE, XYZELT, VERBUF)
1
  RPOLY = R + SUBDIV*0.75
  SPOLY = S - SUBDIV*0.25
  CALL ESHAPE (LEMTYP, NODEXT, RPOLY, SPOLY, H)
  CALL CVLTG (H, IPOINT, MAXPNT, MAXLND, NODEXT, NSPACE, XYZELT, VERBUF)
  CALL ESHAPE (LEMTYP, NODEXT, RPOLY, SPOLY, H)
  CALL CVLTG (H, IPOINT, MAXPNT, MAXLND, NODEXT, NSPACE, XYZELT, VERBUF)
  S = S + SUBDIV*0.5
ELSE
  RPOLY = R + SUBDIV*0.25
  SPOLY = S - SUBDIV*0.75
  CALL ESHAPE (LEMTYP, NODEXT, RPOLY, SPOLY, H)
  CALL CVLTG (H, IPOINT, MAXPNT, MAXLND, NODEXT, NSPACE, XYZELT, VERBUF)
1
  RPOLY = R + SUBDIV*0.25
  SPOLY = S + SUBDIV*0.25
  CALL ESHAPE (LEMTYP, NODEXT, RPOLY, SPOLY, H)
  CALL CVLTG (H, IPOINT, MAXPNT, MAXLND, NODEXT, NSPACE, XYZELT, VERBUF)
1
  RPOLY = R - SUBDIV*0.75
  SPOLY = S + SUBDIV*0.25
END IF
CALL ESHAPE (LEMTYP, NODEXT, RPOLY, SPOLY, H)
CALL CVLTOG (H, IPOINT, MAXPNT, MAXIND, NODEXT, NSPACE, XYZELT, VERBUF)

1
S = S - SUBDIV*0.5
END IF

Update the "r" coordinate

R = R + SUBDIV*0.5

Check to see if the buffer is full. If so, go plot it.

IF (IVALUE .EQ. MAXVAL) THEN
CALL PLTPBF (AFIRST, ITERM, IPOINT, IVALUE, LEMTYP, MAXPNT, MAXVAL, NSPACE, PLTMAX, PLTMIN, PSCALE, VALBUF, VERBUF, VOMAX, VOMIN)
120
CONTINUE
130
CONTINUE
END IF

Plot the buffer since the buffer has been loaded

IF (IPOINT .GT. 0) THEN
CALL PLTPBF (AFIRST, ITERM, IPOINT, IVALUE, LEMTYP, MAXPNT, MAXVAL, NSPACE, PLTMAX, PLTMIN, PSCALE, VALBUF, VERBUF, VOMAX, VOMIN)
END IF
140
CONTINUE

Plot the legend for the color regions

CALL PLTCLG (CSIZE, ITERM, MAXRNG, NCONTR, PLTMAX, VLEVEL, VMID, VOMAX, VOMIN)

RETURN
END
SUBROUTINE PLTFIN (ITERM)

SUBROUTINE PLTFIN

PURPOSE: This subroutine closes the plotting device.

INCLUDE "/usr/include/f77/usercore77.h"
INTEGER VSURF(VWSURFSIZE)
DATA VSURF / VWSURFSIZE*0 /
CALL PLTMOV (ITERM, 0., 0.)

Execute appropriate code for the plotting device

GOTO (10, 20, 30), ITERM

Plot code for Selenar terminals

10 CALL FINITT (0, 0)
RETURN

Plot code for Celerity 7180 terminals

20 CALL G1$QUIT
CALL G1$ENTGRA
CALL G1$MODDIS (1)
CALL G1$VAL8 (0)
CALL G1$CLEAR
CALL G1$QUIT
CALL G1$STOP (0)
RETURN

Plot code for SUN terminals

30 CALL GETVIEWSURFACE(VSURF)
CALL TERMINATEDDEVICE (LOCATOR, 1)
CALL TERMINATEDDEVICE (BUTTON, 1)
CALL TERMINATEDDEVICE (BUTTON, 2)
CALL TERMINATEDDEVICE (BUTTON, 3)
CALL CLOSETEMPSEG ()
CALL TERMINATECORE ()
RETURN
END
SUBROUTINE PLTINI (AXIS, COORMX, COORMX, DAXIS, ICONTR, ITERM,
1       LEGEND, MAXNOD, NODES, NODFILG, NSPACE, PLTMXH,
2       STITLE, TITLE, WFLAG, XYZ, AFIRST, CSIZE,
3       ITERM, PLTMAX, PLTMIN, PSSCALE, SCRMAX, SCRMIN,
4       UNPIN)

SUBROUTINE PLTINI

PURPOSE: This subroutine initializes the plotting device and draws an axis if one is desired.

LOGICAL DAXIS, NODFILG, PLTMXH, WFLAG
CHARACTER*(*) AXIS, LEGEND, STITLE, TITLE

Dimension the external variables

DIMENSION AFIRST(NSPACE), AXIS(NSPACE), COORMX(NSPACE),
1       COORMX(NSPACE), PLTMXH(NSPACE), PLTMXH(NSPACE),
2       PSSCALE(NSPACE), SCRMAX(NSPACE), SCRMIN(NSPACE),
3       UNPIN(NSPACE), XYZ(MAXNOD,NSPACE)

Dimension the internal variables

DIMENSION ADELT(3), ALAST(3), CSIZE(3), PLTRNG(2), SSIZE(2)

Initialize the plot parameters

CALL PLTSRT (ITERM, CSIZE, PLTMXH, PLTMXH, SCRMAX, SCRMIN, SSIZE)

Initialize the plot device

CALL PLTDIPD (ICONTR, ITERM)

Determine the plottable coordinate range (i.e. PLTMXH - PLTMXH)

DO 10 I = 1,2
    PLTRNG(I) = PLTMXH(I) - PLTMXH(I)
    CONTINUE
10

Determine the minimum and maximum coordinates (in physical units) to be plotted

IF (.NOT. WFLAG) THEN
    DO 20 I = 1,NSPACE
        COORMX(I) = XYZ(1,I)
        COORMX(I) = COORMX(I)
        CONTINUE
20
    DO 40 J = 1,NSPACE
        DO 30 I = 1,NODES
            IF (XYZ(I,J) .GT. COORMX(J)) COORMX(J) = XYZ(I,J)
            IF (XYZ(I,J) .LT. COORMX(J)) COORMX(J) = XYZ(I,J)
            CONTINUE
30
    CONTINUE
40
END IF

Scale the x-axis to fill the screen

CALL SCALE (COORMX(1), COORMX(1), ADELT(1), AFIRST(1), ALAST(1))
C Scale the y-axis to fill the screen

    CALL SCALE (COORDX(2), COORMN(2), ADELT(2), AFIRST(2), ALAST(2))

C If the window flag is set, let the first and last values on an axis be what was computed in subroutine WINDOW

    IF (WFLAG) THEN
        DO 45 I = 1,2
            AFIRST(I) = COORMN(I)
            ALAST(I) = COORMX(I)
        END DO
        CONTINUE
    END IF

C Determine the plotting scale factors and the coordinate units per inch

    DO 50 I = 1,2
        PSCALE(I) =ABS (PLTRNG(I)/(ALAST(I) - AFIRST(I)))
        UNPIN(I) = (SCRMAX(I) - SCRMIN(I))/(PSCALE(I)*SSIZE(I))
    CONTINUE

C Use subroutine SCALEP to proportionally scale the plot

    IF (.NOT. WFLAG) THEN
        CALL SCALEP (ADELT, AFIRST, ALAST, COORMN, COORMX, NSPACE, 1, PLTRNG, PScale, UNPIN)
    END IF

C Plot a border around the entire screen

    CALL PLTMOV (ITEM, SCRMIN(1), SCRMIN(2))
    CALL PLTDRW (ITEM, SCRMIN(1), SCRMAX(2))
    CALL PLTDRW (ITEM, SCRMAX(1), SCRMAX(2))
    CALL PLTDRW (ITEM, SCRMAX(1), SCRMIN(2))
    CALL PLTDRW (ITEM, SCRMIN(1), SCRMIN(2))

C Plot the title

    CALL STRCNT (TITLE, NCHARS)
    RANGE = SCRMAX(1) - SCRMIN(1)
    WIDTH = CSIZE(3)*FLOAT(NCHARS)
    X = (RANGE - WIDTH)*0.5 + SCRMIN(1)
    Y = SCRMAX(2) - 1.165*CSIZE(3)
    CALL PLTTXT (0.0, CSIZE(3), ITEM, NCHARS, TITLE, X, Y)

C Plot the sub-title

    CALL STRCNT (STITLE, NCHARS)
    RANGE = SCRMAX(1) - SCRMIN(1)
    WIDTH = CSIZE(2)*FLOAT(NCHARS)
    X = (RANGE - WIDTH)*0.5 + SCRMIN(1)
    Y = SCRMAX(2) - 1.165*CSIZE(3) - 2.0*CSIZE(2)
    CALL PLTTXT (0.0, CSIZE(2), ITEM, NCHARS, STITLE, X, Y)

C Draw the axes if desired
IF (DAXIS) THEN
    CALL PLTAXS (AFIRST, ALAST, ADELT, AXIS, CSIZE, ITERM,
      NSPACE, PLTMIN, FSSCALE, SCRMIN)
END IF

C Plot the legend title

C IF (LEGEND .NE. ' ') THEN
    X = PLTMAX(1) + CSIZE(2)*2.0
    Y = PLTMAX(2) - CSIZE(2)*2.0
    CALL STRCNT (LEGEND, NCHARS)
    CALL PLTTEXT (0.0, CSIZE(2), ITERM, NCHARS, LEGEND, X, Y)
    Y = Y - CSIZE(2)*0.5
    CALL PLTMOV (ITERM, X, Y)
    X = X + (FLOAT(NCHARS) - 0.5)*CSIZE(2)
    CALL PLTDRAW (ITERM, X, Y)
END IF
RETURN
END
SUBROUTINE WINDOW (AFIRST, CSIZE, ITERM, NSPACE, PLTMAX, PLTMIN,
   PScale, COORMN, COORMX, WFLAG)

SUBROUTINE WINDOW

PURPOSE: This subroutine uses terminal specific drivers to use
a mouse or other locator device to window-in on a
rectangular area of the screen.

CHARACTER*80 ANS
LOGICAL WFLAG

Dimension the external variables

DIMENSION AFIRST(NSPACE), COORMN(NSPACE), COORMX(NSPACE),
   CSIZE(3), PLTMAX(2), PLTMIN(2), PScale(NSPACE)

Dimension the internal variables

DIMENSION IXSCRN(2), JYSCRN(2), PMIN(2), PMAX(2)

Execute appropriate code for plotting devices

GOTO (10, 20, 30), ITERM

Stop the plot to read the window flag

Plot code for Selanar terminals

10 WRITE (*,1000) CHAR(27), CHAR(59)
1000 FORMAT(2A1,$)
   CALL PLTMOV (ITERM, 850.0, 200.0)
   WRITE (*,1010) CHAR(31)
1010 FORMAT(A1,$)
   WRITE (*,1020)
1020 FORMAT('Press w <CR> to'
   WRITE (*,1010) CHAR(29)
   CALL PLTMOV (ITERM, 850.0, 186.0)
   WRITE (*,1010) CHAR(31)
   WRITE (*,1030)
1030 FORMAT('window followed')
   WRITE (*,1010) CHAR(29)
   CALL PLTMOV (ITERM, 850.0, 172.0)
   WRITE (*,1010) CHAR(31)
   WRITE (*,1040)
1040 FORMAT('by c to define')
   WRITE (*,1010) CHAR(29)
   CALL PLTMOV (ITERM, 850.0, 158.0)
   WRITE (*,1010) CHAR(31)
   WRITE (*,1050)
1050 FORMAT('the window edges')
   WRITE (*,1010) CHAR(29)
   READ (5, '(A)') ANS
   IF (ANS(1:1) .EQ. 'W' .OR. ANS(1:1) .EQ. 'W') THEN
      WFLAG = .TRUE.
      GOTO 50
   ELSE
      WFLAG = .FALSE.
RETURN
END IF

C C Plot code for Celerity 7180 terminals

20 READ (5,'(A)') ANS
WFLAG = .FALSE.
RETURN

C C Plot code for SUN terminals

30 CALL SETCHARPRECISION (1)
CALL SETFONT (2)
CALL SETCHARSIZE (9.0, 9.0)
CALL PLTMOV (ITERM, 850.0, 200.0)
CALL TEXT ('Press right mouse')
CALL PLTMOV (ITERM, 850.0, 186.0)
CALL TEXT ('button to window')
CALL PLTMOV (ITERM, 850.0, 172.0)
CALL TEXT ('or left mouse button')
CALL PLTMOV (ITERM, 850.0, 158.0)
CALL TEXT ('to continue')
CALL AWAITANYBUTTON (100000000, NBUTTN)
IF (NBUTTN .EQ. 3) THEN
  WFLAG = .TRUE.
  CALL SETLINEINDEX (1)
  CALL SETFILLINDEX (1)
  CALL SETTEXTINDEX (1)
ELSE IF (NBUTTN .EQ. 1) THEN
  WFLAG = .FALSE.
ELSE
  GOTO 30
END IF

C C Determine a corner of the box and draw a marker there

50 IF (WFLAG) THEN
  DO 70 I = 1, 2
  CALL INSPOS (ITERM, IXSCRN(I), JYSCRN(I))
    IF (IXSCRN(I) .LT. PLTMIN(1) .OR.
      1     IXSCRN(I) .GT. PLTMAX(1) .OR.
      2     JYSCRN(I) .LT. PLTMIN(2) .OR.
      3     JYSCRN(I) .GT. PLTMAX(2)) GOTO 60
    CALL MARKER (ITERM, IXSCRN(I), JYSCRN(I), CSIZE(1))
  CONTINUE

70 IF (WFLAG) THEN
  IF (IXSCRN(1) .LT. IXSCRN(2)) THEN
    PMIN(1) = FLOAT(IXSCRN(1))
    PMAX(1) = FLOAT(IXSCRN(2))
  ELSE
    PMIN(1) = FLOAT(IXSCRN(2))
    PMAX(1) = FLOAT(IXSCRN(1))
  END IF
  IF (JYSCRN(1) .LT. JYSCRN(2)) THEN
    PMIN(2) = FLOAT(JYSCRN(1))
  END IF
PMAX(2) = FLOAT(JYSCRN(2))
ELSE
  PMIN(2) = FLOAT(JYSCRN(2))
  PMAX(2) = FLOAT(JYSCRN(1))
END IF

C Draw a box around the window area
C
CALL PLTMOV (ITERM, PMIN(1), PMIN(2))
CALL PLTDRW (ITERM, PMAX(1), PMIN(2))
CALL PLTDRW (ITERM, PMAX(1), PMAX(2))
CALL PLTDRW (ITERM, PMIN(1), PMAX(2))
CALL PLTDRW (ITERM, PMIN(1), PMIN(2))
IF (ITERM .EQ. 1) THEN
  CALL WAIT (5000)
ELSE
  CALL WAIT (100000)
END IF

C Determine the minimum and maximum coordinates in plot units
C
DO 80 I = 1, 2
  COORMX(I) = AFIRST(I) + (PMIN(I) - PLTMIN(I))/PScale(I)
  COORMN(I) = AFIRST(I) + (PMAX(I) - PLTMIN(I))/PScale(I)
80  CONTINUE
END IF
RETURN
END