This is to certify that the

dissertation entitled

Mapping of Finite Element Results
onto Actual Geometries

presented by

Jane Louise Hawkins

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Mechanical Engineering

Erik D. Goodman
Major professor

Date July 12, 1989

MAPPING OF FINITE ELEMENT RESULTS ONTO ACTUAL GEOMETRIES

By

Jane Louise Hawkins

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering

1989

ABSTRACT

MAPPING OF FINITE ELEMENT RESULTS ONTO ACTUAL GEOMETRIES

By

Jane Louise Hawkins

Currently, the most common method for displaying the results of a finite element analysis of an object is to display them on the finite element mesh. This display will "resemble" the actual geometry of the analysis object, to a degree which depends on the fineness of the mesh. A more appropriate and informative display is to see the finite element (FE) results on the actual geometry model. This implies the more complicated need for the curves (representing the FE contours) to be defined on the actual geometry (for example, a sculptured surface model). Other potential complications are an increase in the information needed to display the contours and an increase in the computational load to display (since such entities as sculptured surfaces are displayed, as opposed to the faceted surface of the mesh).

This dissertation presents an effective method for producing the contour information from an FE analysis on objects defined by bounded surface entities based on trimmed Non-Uniform Rational B-Spline (NURBS) surfaces. It also shows the other potential complications stated above to be false. The data needed for display are organized into a compact data file that can be fed to a hardware encoded display mechanism for fast and accurate display of color-shaded contour regions on the actual geometry model.

ACKNOWLEDGMENTS

I would like to extend thanks to Structural Dynamics Research Corporation for funding this work and providing me with the initial idea as well as invaluable insight into the areas of FEA and geometric modeling.

I also wish to thank the members of my committee Drs. Clark Radcliffe, Richard Phillips, Mukesh Gandhi, and Joseph Whitesell for their aid and assistance. In particular, I would like to thank Dr. Richard Phillips for the incredible support he has given me during my entire college career.

Speaking of college careers, I gratefully thank my parents, Robert and Nancy Hawkins, for their loving support over these many years when I'm sure they were wondering what on earth I was doing.

Finally, I would like to extend special thanks to my major advisor, Dr. Erik Goodman, who provided me with interesting research topics, excellent guidance, constant encouragement, and moral and financial support. His sincere belief in me kept me going.

Table of Contents

List of Figures

Chapter 1

Introduction

Finite Element Analysis (FEA), which determines an approximate solution to physical problems that occur in a number of areas of engineering science, is becoming more and more widely used [1]. The finite element method is simply a numerical technique that converts the governing partial differential equation of a problem into a set (generally very large) of linear algebraic equations, which are solved by computer. Due to the constantly increasing computational efficiency of computers, as well as FEA software, the finite element method has become useful in a large number of engineering applications.

The early FE applications were in the area of structural analysis, particularly aircraft structures [2]. Today the method is a valuable tool in fields such as biomechanics, where the irregular geometry of the human anatomy and the impossibility of conducting experiments on human beings prevent the use of other analytical methods [3]. Although FEA is applicable to problems which do not involve geometric objects, the most common use involves determining stresses, deflections and other effects (induced by mechanical loading, heat, magnetic fields, etc.) that result on an object.

Given a geometric object, the finite element method creates an abstraction of the object which consists of a finite set of geometric entities called elements (thus Finite Element analysis). These elements are entities such as planar triangles and quadrilaterals in two-dimensional analysis or tetrahedral and hexahedral elements in three dimensions. [4] The FEA results are calculated with respect to this discrete abstraction (called the finite element mesh or FEM).

For the FEA to be useful, the results must be presented in a manner which allows the analyst (user) to easily interpret them. Initially finite element results were presented in enormous tables of data, which made it nearly impossible for the analyst to interpret the results or determine if the analysis was even reasonable. Today results can be graphically displayed in the form of color contour plots. Contour plots consist of curves (contour curves) having a constant value, for example stress values, which are a specific color signifying an associated stress value, or continuously color-shaded regions between contour curves (contour bands) which represent a range of stress values. These contour curves and bands are typically displayed on the discrete model or mesh. Since the result data is discrete, some form of interpolation is needed to determine where the contour curves are located. This interpolation (for display purposes) is disjoint from the interpolation inherent to the FEA.

The primary objective of the FEA is to analyze an object, that is, predict its behavior given a set of input conditions. If the analysis tool, in this case the FEA, creates an abstraction in order to approximate (hopefully to a high degree of accuracy) a solution, this really doesn't affect the primary objective. Given the results displayed as contour plots on the abstraction or mesh, the user intuitively or subconsciously extrapolates the results onto the object. If software is designed to display contour plots on actual objects, then it will have access to more accurate data than the visual data provided to the user by the mesh contour display. Thus, it can be assumed that the software will extrapolate the data with less loss of accuracy than the user. Moreover, displaying on the actual object provides accurate geometric information about the analysis object, as well as the finite element solution information.

Also, one of the future goals of some FEA software manufacturers is to automate the creation of abstractions (finite element meshes), ultimately hiding them completely from the user. To do this, one needs to display the engineering results as contours on the original geometry model instead of the abstraction. This presumes that the user enters the FEA with a geometry model (mathematical representation) of the analysis object, which is desirable since this facilitates the generation of the mesh. Also in many engineering applications, for example, NC milling operations, it is necessary to have an exact mathematical model of the object in order to manufacture it. [5]

This dissertation describes a unique method (and software) for displaying finite element results (or in practicality other sets of discrete scalar data, for example, finite difference results) on the original geometry model (mathematical model) of an object. This method is based on defining the surface regions that exist between contour curves as geometric entities called trimmed surfaces. These regions are closed and their boundaries consist of contour curves or a combination of contour curves and portions of actual geometric boundaries. The contour curves are defined by a composition of relations, which assure that the range of the composition is a subset of the range of the mathematical model, or in other words, the contour curves lie on the surfaces defining the object.

Thus, the contour regions are represented in terms of their geometric properties as well as their inherent finite element properties. This scheme allows the data needed to define and display the contour plots to be organized in a compact and portable data structure. Data structures for geometric entities have been widely studied and standardization is being undertaken (for example, the Initial Graphics Exchange Specification or IGES [6][7]). More importantly this scheme allows the use of display devices designed to produce fast, high-quality rendered images of

geometry models. These devices allow real-time interaction with the object and its display parameters, for example, rotation and translation of the object and multiple light source effects.

The display device used in this work was a HP9000/350SRX using the Starbase Graphics Library with FEA results obtained from SDRC's Supertab. The objects are defined by a set of trimmed non-linear rational B-spline (NURBS) surfaces (patches) which define a B-rep solid model or simply a surface entity. The NURBS surface is currently being adopted by industries such as aerospace, automotive and shipbuilding as their preferred mathematical model due to its outstanding properties, such as controllable continuity, locality, minimal storage space, efficient evaluation and ease of interactive manipulation [7,8,9,10,11,12,13].

The dissertation is organized such that, Chapter 2 describes the current display methods, presents an overview of the proposed method, and provides a brief comparative study. Chapter 3 describes input requirements, user interaction and a general overview. Chapter 4 details the integration of the FE data with the geometry data. The generation of the trimmed surface representation of contour regions is described in Chapter 5. A pictorial and verbal comparative discussion, extensions and a brief summary are presented in Chapter 6.

Chapter 2

Review of State of the Art

The typical method of displaying finite element results currently consists of displaying these results on the discrete abstraction (FE mesh) of the geometry model of the analysis object. This chapter begins by presenting the topology of the FE mesh. Next the interpolation and display methods used to create contour plots on meshes is overviewed. A general description of the actual geometry model used in this work is presented next. Finally, there is a discussion of the possible techniques for mapping discrete data and/or curves onto geometries.

2.1 Finite Element Mesh Topology

Each element is defined in terms of the topological entities vertex (node), edge (side), and face. A linear side will have 2 nodes; a parabolic, 3 nodes; and a cubic, 4 nodes. Two-dimensional elements have a single face and 3-dimensional elements have multiple faces, each face consisting of a loop of nodes and sides.

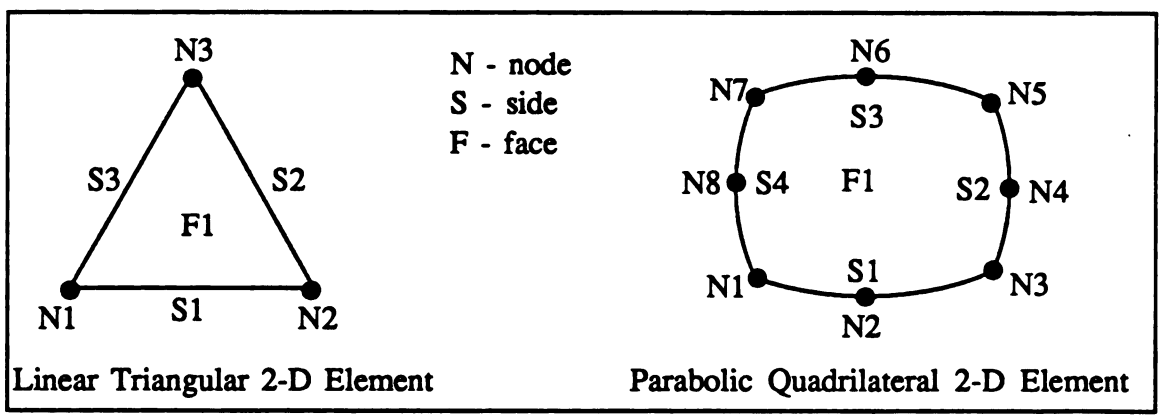


Figure 2.1 Element examples

A node is assumed to be a point on the surface (or in the interior) of the analysis object and is associated with a unique Cartesian coordinate. The results of the finite element analysis are either values computed on the nodes (for example, displacements) or values determined at other points on the element (for example, stresses are determined at integration points). Nodal values are most frequently the "known" discrete data that are used to generate (by interpolation) the contour curves signifying particular constant result values. Thus for some results, such as stresses, a further calculation is required to determine the value at the node.

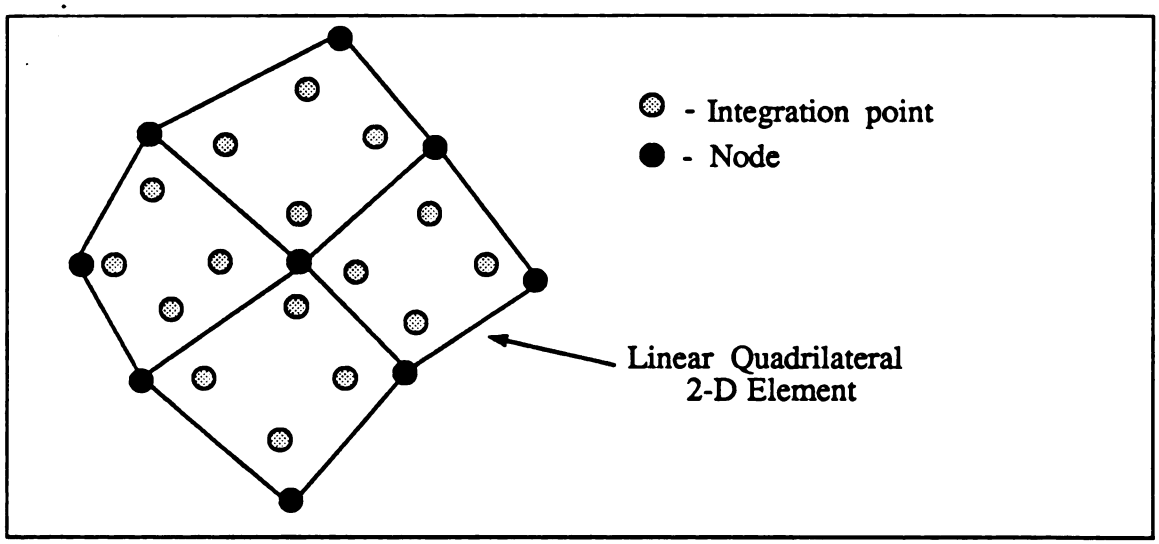


Figure 2.2 FEA result locations

The unique line or curve which exists between 2 nodes which are adjacent (in terms of the graph which is the mesh) will be defined as an edge. Thus an element side will consist of 1 corresponding edge if it is linear, 2 edges if it is parabolic, etc. Also one edge may be on 1 or more elements, that is, coincide with more than 1 element sides or segments of sides.

2.2 Contour Interpolation Methods for Mesh Display

As stated in the introduction there are 2 basic types of contour plots. One displays contour curves (color coded to signify the value they represent) and the other continuous colored contour bands (each color represents a range of values). The contour band plots are a more "expensive" display than the contour curves, but they often provide a better intuition into the results.

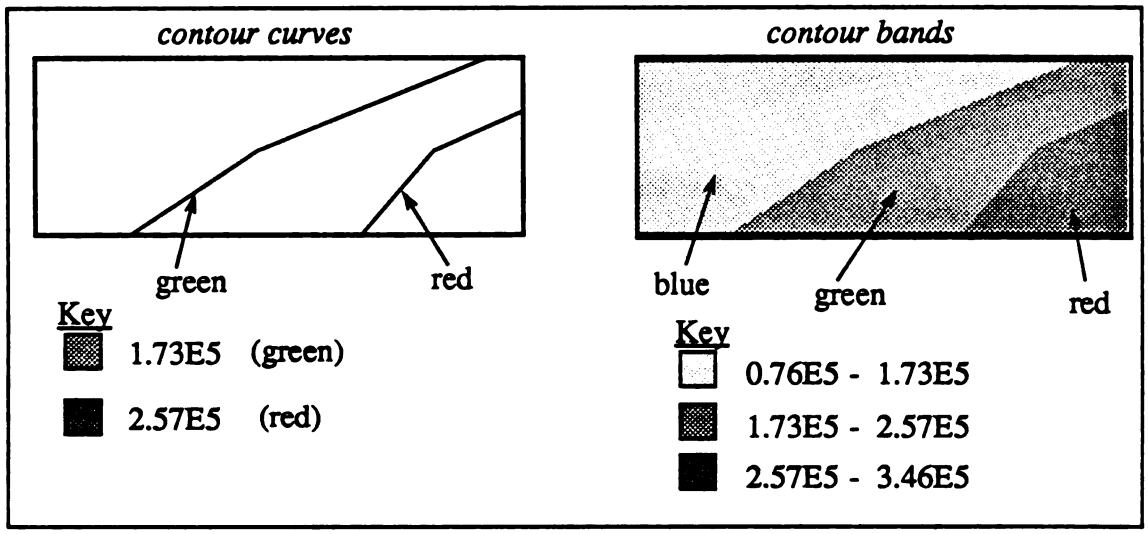


Figure 2.3 Contour plot examples

The problem basic to both display types is the generation of those points with a specific FE property value (called the contour value) which define a contour curve. When displaying the contour information on the finite element mesh (as is the common practice), the problem is reduced to independent examination of each face of an element. The FE property values and screen coordinates (X,Y) of the nodes on a face are known and the contour values that exist on that face must be determined by some interpolation scheme.

Display methods for FEA, such as contouring, are developed primarily by software companies that manufacture FEA packages. Some packages, such as

I-DEAS by Structural Dynamics Reasearch Corporation (SDRC), use a different interpolation method for displaying contour curves, as opposed to contour bands [4]. For the contour curve, the element face is subdivided (based on the location of the centroid) into triangular regions and FE property values are computed (average of nodal values) for the internal locations (see Figure 2.4). Six- and eight-noded faces are subdivided twice. Contour curves are generated by assuming linear interpolation along the edges of the triangles (see Figure 2.5).

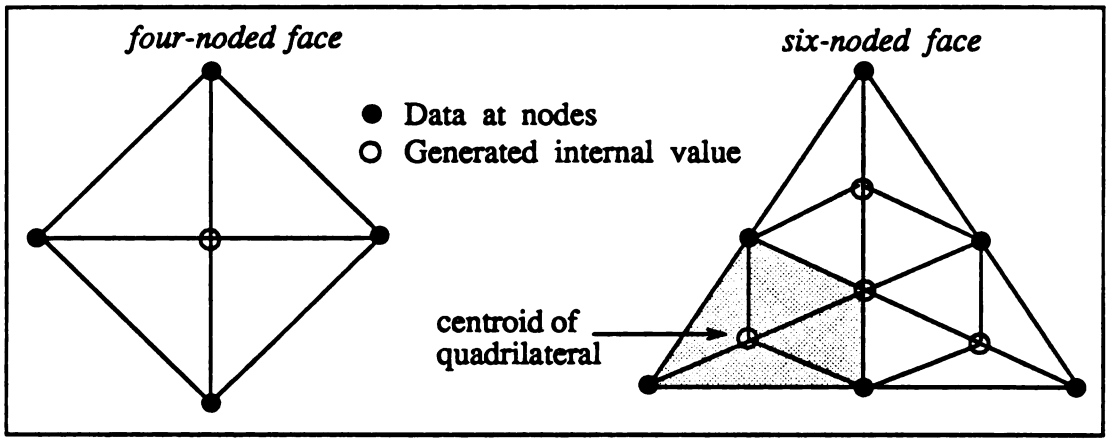


Figure 2.4 Face divisions for contour curve display (I-DEAS)

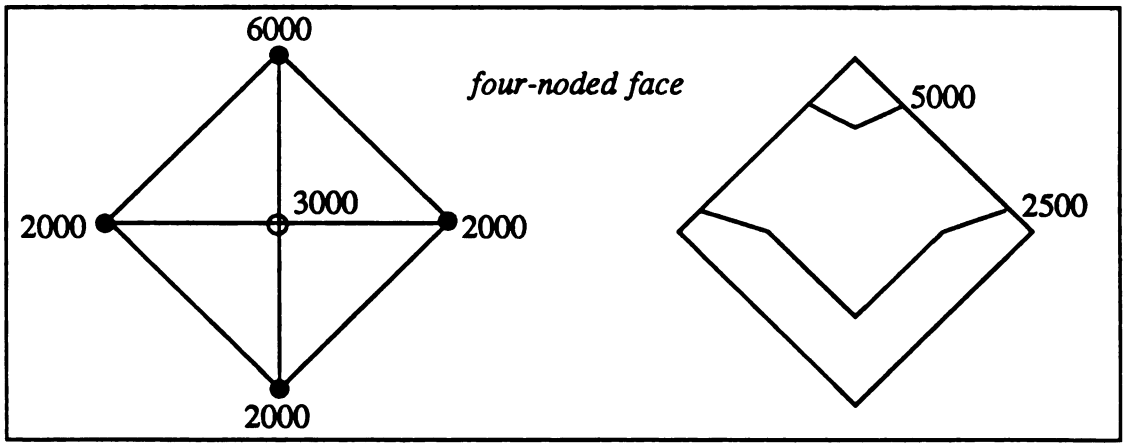


Figure 2.5 Contour curve interpolation scheme (I-DEAS)

For contour band plots, the faces are not subdivided. Continuous color bands are created by displaying one scanline (constant Y screen coordinate) at a time. Contour

value locations are computed by linearly interpolating along the segment of a current scanline that intersects an element face. These contour locations represent points where the color of the scanline switches. The endpoints of the segment are on the sides of the element and their property values are determined by linearly interpolating along the edges. This interpolation scheme is view dependent, that is, different views will create different contour regions.

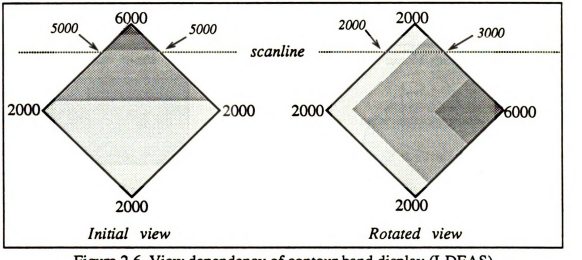


Figure 2.6 View dependency of contour band display (I-DEAS)

Other FEA packages, for example, ANSYS (by Swanson Analysis Systems, Inc.) and NISA (by EMRC), also rely on linear interpolation to produce contour plots [14,15,16,17]. Both, regardless of the type of contour display, linearly interpolate the original edges of the element for contour values. ANSYS does not use a scanline display; the colored regions are based on a straight line contour curve through edge points. Thus, the initial view of the quadrilateral element shown in Figure 2.6 displays the contour regions that ANSYS would present (regardless of view). Some packages, such as NISA, provide a smoothing option for the contour curves. Exact procedures and even basic information are difficult to obtain due to heavy industrial competition in the area of FE analysis.

2.3 Trimmed NURBS Surface

This work proposes that the contours be displayed on the mathematical model of the analysis object. As mentioned in the introduction, the analysis object will be modeled by a set of trimmed NURBS surfaces. These NURBS surfaces are bivariate parametric surfaces. The domain of this mathematical model is called the (surface) parameter space and the range is 3-dimensional Cartesian space (called world space).

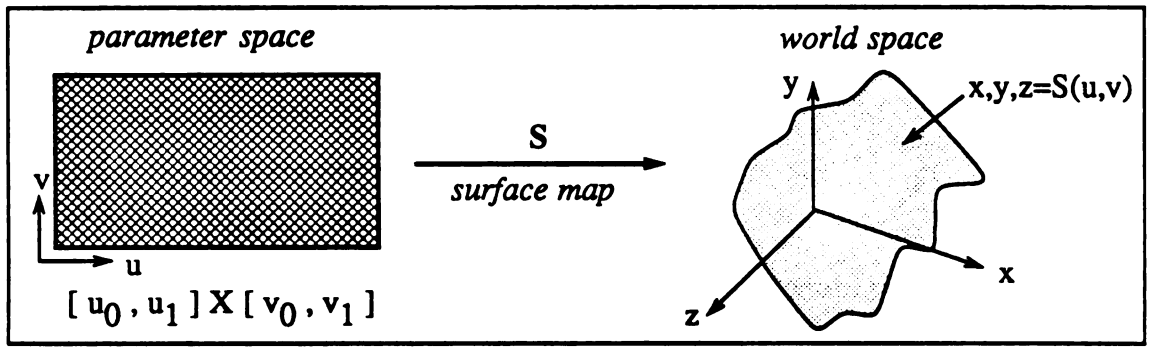


Figure 2.7 Untrimmed bivariate parametric surface

When the rectangular region which is the domain is mapped (surjective) into world space it is possible for certain degeneracies to occur. One type is called a seam and occurs when 2 opposite boundaries of the rectangular region coincide in a single curve in world space. An example of this is a cylinder. Another degeneracy is called a pole and occurs when an entire side of the region is mapped to a single point (the pole). A sphere formed by a single surface exhibits poles and a seam.

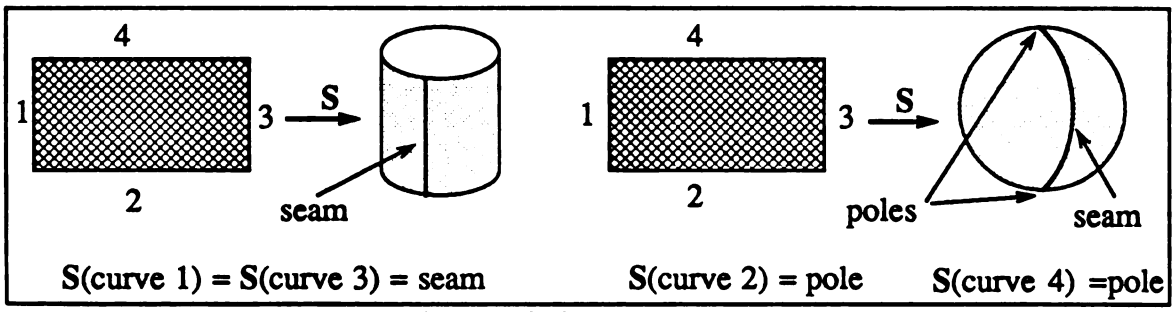


Figure 2.8 Seams and poles

Each of the 4 boundaries of the domain maps into a B-spline curve which, in general, defines the boundaries of the surface in world space. A B-spline curve is a parametric curve defined by a set of control points, a knot vector, and weights. The control points are defined in the range of the curve and are the primary factor in determining the shape of the B-spline curve. The ordered set of control points forms the vertices of a polygon (open or closed) which is called the control point polygon. This polygon mimics the shape of the B-spline.

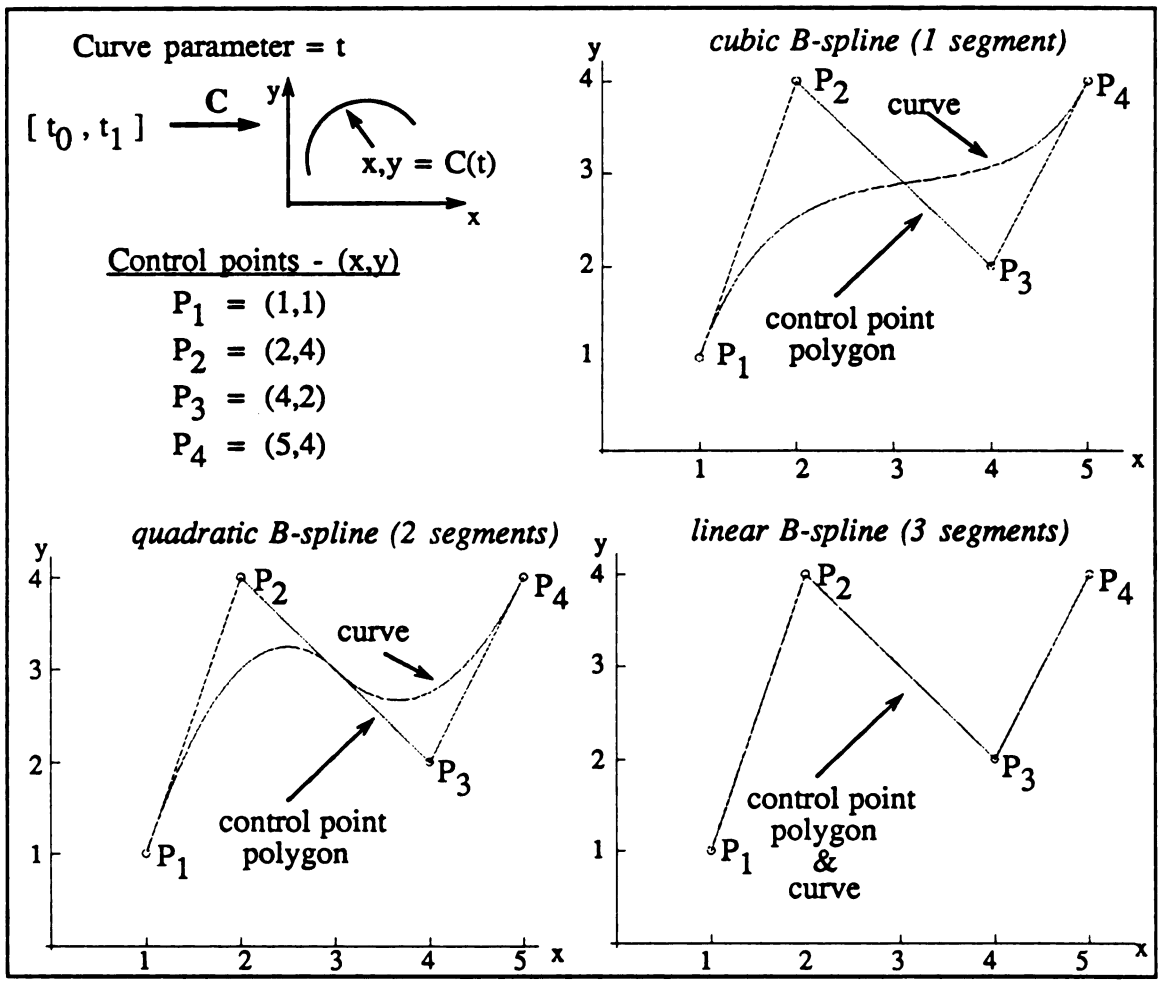


Figure 2.9 B-spline curves

The control point polygon also defines an important entity called the convex hull. Each segment of a B-spline is defined by an ordered subset of the control points. These points are used as vertices to define a polyhedral region. The union of these

regions forms the convex hull. The convex hull has the property that the spline curve is always contained within it. This property also applies to B-spline surfaces, thus providing a bounding entity for the surface. The set of control points that define a B-spline surface form a net of quadrilateral entities, called the control point net.

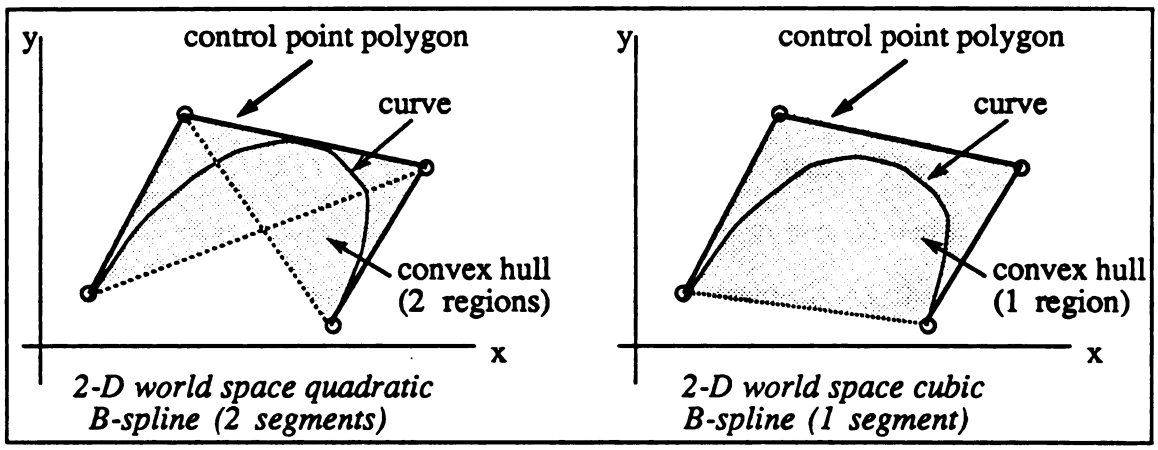


Figure 2.10 Convex hull property

As stated before the B-spline map depends on the control points, knot vector and weights. The knot vector determines the polynomial basis functions for each segment of the spline. Nonuniform knot vectors (thus nonuniform B-splines) allow control over the parametric flow of the spline by controlling the parameter value at which segment endpoints occur. Inherent in uniform B-splines is the property that they are (degree - 1) times continuously differentiable at all points, including knots (segment endpoints). Nonuniformity also allows control over the continuity conditions at knots.

The third degree of freedom in defining B-splines is the weights. A control point is "weighted" by representing it in homogeneous coordinate form. The 3-dimensional Cartesian coordinate (x,y,z) can be represented in homogeneous coordinates by the following quadruplets (x,y,z,1) or (3x,3y,3z,3) or (wx,wy,wz,w). In B-spline

terminology the fourth coordinate is referred to as the weight. When control points are defined with weights they are called rational. Weighting control points provides additional shape control, which is necessary in representing certain curves such as those derived from conic sections.

B-splines can be defined in 3-dimensional as well as 2-dimensional Cartesian or world space. The space in which the control points are defined determines the space of the curve. Curves can also be defined in the parameter space of a surface, that is, B-spline curves defined in the domain of B-spline surfaces. The control points of these curves have coordinates (u,v) (elements of the surface domain) and are called parameter space curves. When these curves are mapped via the B-spline surface map into world space, they are a subset of the range of the surface, that is, they lie on the surface.

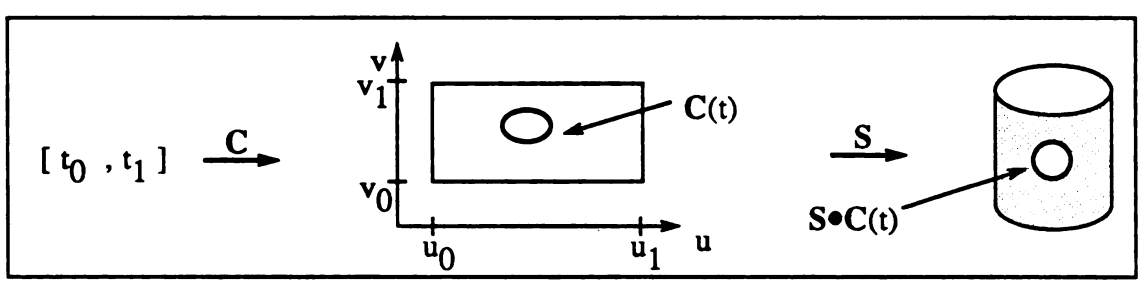


Figure 2.11 Parameter space curve and trimmed surface

Surfaces have untrimmed or natural boundaries (in cartesian or world space) which result from the surface map of the boundaries of the rectangular parameter space domain. Additional curves that lie in the range of the surface (by necessity if they are parameter space curves, or by design if they are world space curves) and satisfy certain other requirements are called trimming curves and produce trimmed surfaces (sometimes referred to as bounded surfaces). One of the aforementioned requirements is that the trimming curve(s) form a loop, by themselves or in

conjunction with untrimmed boundaries (or parts of them), that is closed in world space. The hole curve shown in the figure above is a trimming curve. Another requirement is that these loops are directed; that is, in world space the closed loop is designated as clockwise or counterclockwise. This directed sense of the trimming curve loops must be assigned so that the top of the surface lies on the left of the loop as its directed path is followed (top is inherently defined by the direction of parametric flow on the surface). Each curve in the loop, including segments of untrimmed or natural boundaries, must be defined as a properly directed curve entity (the direction of parametric curves is defined by increasing parameter value).

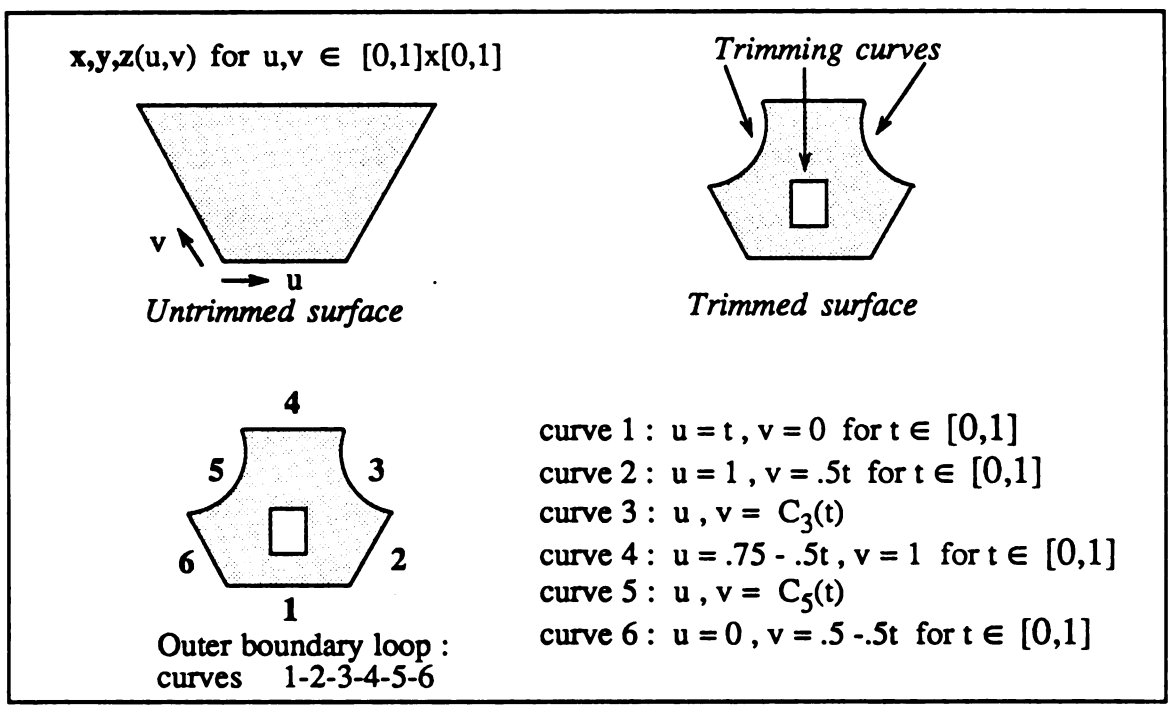


Figure 2.12 Trimmed surface

2.4 Contour Curves on Geometry Models

There are two interpolatory steps in defining a contour curve given an irregular grid (the FE mesh) of data. The initial step is to find the discrete points (referred to

as contour intersection points or just contour points) that represent a contour curve crossing the edge of an element face. The next step is to "connect the dots" or put a curve through these points. To display FE contour information on the geometry model of an analysis object, the contour curves must be defined on the object (just as trimming curves must). Since parameter space curves (versus world space curves) have this property by definition, the natural (and definitively "time and cost" effective) approach is to define the contours as parameter space curves. This implies that at some time during the definition of the contour curve, a parameter space coordinate (u,v) must be determined for each of the contour intersection points (since they are the primary interpolation points). Although information such as surface and curve parameter values for nodes could be stored (since it is known or available when the mesh is created), for this work these parametric coordinates of the nodes are not known. Thus the inverse problem (that is, given an element of the range (x,y,z), find the corresponding element in the domain (u,v) or (t)) must be solved for each node. The following text describes the two basic approaches for determining the necessary coordinates.

One approach involves performing the first interpolatory step in Cartesian space and then finding a "mate" (parameter space coordinate pair) on the analysis object to correspond to each (Cartesian space) contour intersection point. A representative technique would be to determine the point on the surface which is closest to the contour point. For contour points found on element edges which correspond to an actual geometric boundary of a surface (or common boundary curves on multiple surfaces), the closest point on the boundary curve(s) should be found, thus assuring that a contour curve will be "attached" to the geometric boundaries. These operations (belonging to the inverse problem family, nonlinear in nature, and solved via numerical methods) are costly and exhibit the typical numerical convergence problems, as well as the predicament of 2 or more points on a surface equidistant from the contour

point. Moreover, these expensive procedures must be repeated for changes in display specifications, since the "mate" list depends on the number and value of contours desired as well as the data selected (for example, VonMises stress, maximum principal stress, etc.).

The other approach involves working entirely in parameter space, that is, using the surface parameter coordinates of the two nodes of an edge to interpolate the contour point's surface parameter coordinate; or for boundary edges, using the boundary curve parameter of the two nodes. This, of course, requires that the surface parameter coordinate (u,v) (or curve parameter (t), if applicable) is known for each node. This brings up one the disadvantages of this approach — the non-uniqueness of the nodal coordinates. In Cartesian space a node has a unique (x,y,z) coordinate, but not necessarily in parameter space; for example, a node on a seam has 2 coordinate (u,v) pairs and a node on a pole has infinitely many. Additionally, a node on a common boundary between 2 (or more) surfaces has different parametric coordinate information for each surface. Thus, additional data storage and bookkeeping is required for this approach.

Since a node is assumed to be on a surface, certain numerical problems are avoided in the second approach as opposed to the first. More importantly, for the second approach, changes in display specifications (for example, number of contour levels or the data selected) do not require that the inverse problem be re-solved. However, the second approach does require a special procedure (to assure contour curve continuity) when dealing with contour points on edges representing multiple surface intersection boundaries. The procedure involves solving the inverse problem, in particular, finding the unique point in Cartesian space (that exists on each surface's coincident boundary curves) that is closest to some point that is not necessarily on any of the surfaces involved, but is by design very close. This closeness, and the

additional fact that the procedure implemented provides an excellent initial guess to the numerical method, causes this particular case of the inverse problem to be "nicer" than the corresponding case of the first approach. This procedure must be applied whenever the contour points change.

Thus, the display dependence and lack of numerical "niceness" of the first approach make it the less desirable one. Additionally, the disparity between the two points from the two approaches, respectively, will depend on the fineness of the mesh and the curvature of the surface (or boundaries), so it would seem fair to assume that, in most cases, the difference would be imperceptible to the human eye. Figure 2.13 shows the two approaches for determining a contour intersection point on a single surface boundary (2-dimensional, as opposed to 3-dimensional, Cartesian space is used simply for brevity of example).

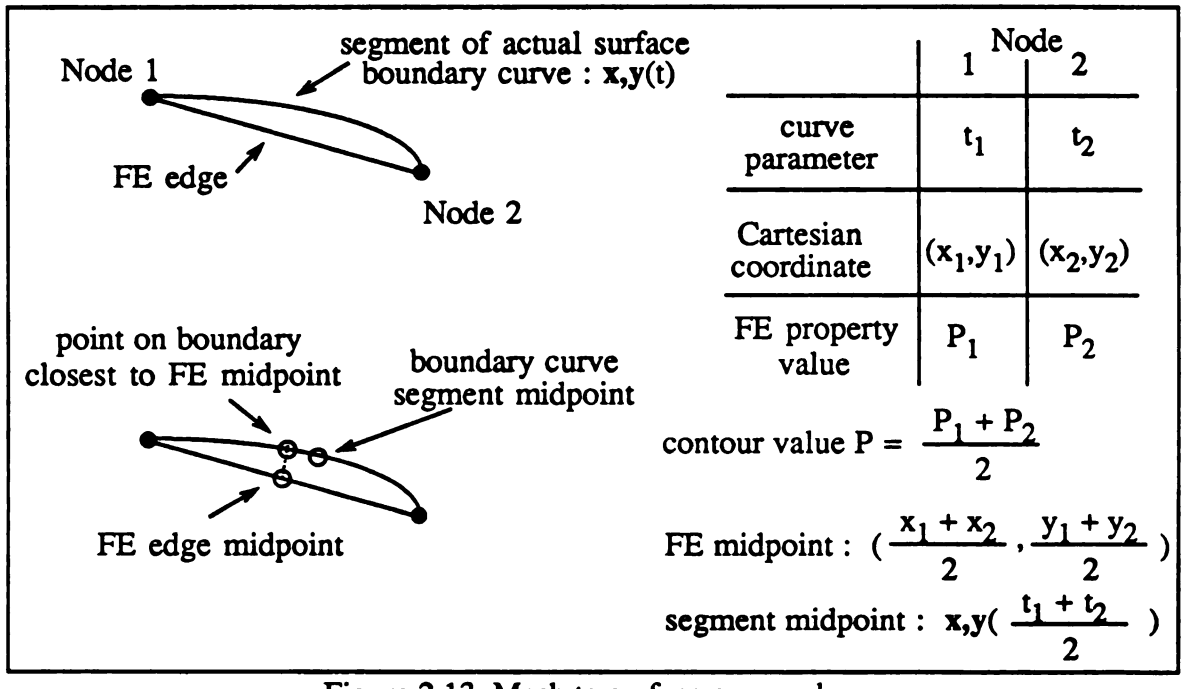


Figure 2.13 Mesh to surface approaches

The above discussion also suggests that projecting Cartesian space contour information into parameter space should be avoided if possible (for example, using

existing routines which convert/project known Cartesian space curves into parameter space). Thus, the second interpolatory step of creating the curves between the contour intersection points should be performed in parameter space. This work does this by using the contour points as the control points for a linear B-spline curve in parameter space (they could also be used as interpolatory points for a smoothing algorithm to produce a smooth curve). It should be noted that although these contour curves are line segments in parameter space, they will not necessarily be straight lines when they are mapped (via the NURBS surface map) into world space, since they will inherit the curvature and parametric flow of the surfaces they are on.

The primary negative aspect of creating contour curves by interpolating entirely in parameter space is that the basis of the FE solution is the mesh which is defined in Cartesian space. Thus interpolating contour information on the mesh and projecting it onto the surfaces would more represent the nature of the solution. This is discounted by the previous suggestion that in most cases the difference will be imperceptible and also by the fact that the solution is approximate to begin with. One other negative aspect is the fact that the nodal coordinates are not unique in parameter space if seams and/or poles exist. This requires some extra bookkeeping as well as the expense of determining if seams/poles exist.

Chapter 3

Trimmed Surface Representation of Contour Regions

As mentioned in the introduction, the main scheme of this work is to define the regions that exist between contours as trimmed surfaces (partitions) of an original surface of the analysis object. This essentially converts the display operation into displaying geometric entities as opposed to a contouring operation. Chapter Two provided background information and outlined approaches and several things to be considered when addressing this task. This chapter will present the approach that is used, describe the specific input and output for the process, and present a general overview of the procedure of defining contour regions as trimmed surfaces.

3.1 Boundary Curves

As discussed in Chapter two, trimmed surfaces are defined by a set of boundary curves which form closed directed loops in world space. These curves are required to be subsets of the range of the surface, that is, lie on the object. Since displaying contours curves on objects also requires that the contour curves lie on the object, they could be used as part of a boundary loop, provided they are properly directed. If a contour curve is closed, then it can be used by itself as a trim boundary (directed in one sense as an outer boundary and in the other sense as an inner boundary to two adjacent contour regions). On the other hand, if a contour curve crosses an actual geometric boundary curve of a surface, then a segment of this boundary curve will have to be used in order to form a closed loop of curves. Thus, there are two types of curves to be created, contour curves and boundary segments, in order to represent contour regions as trimmed surfaces.

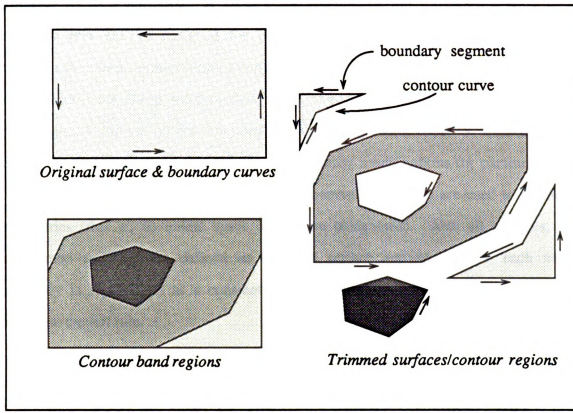


Figure 3.1 Trimmed surface representation of contour regions

3.2 Input Files

There are two types of input information that are provided in data file form. The first contains the mathematical model of the analysis object and the second has the FEA results of an analysis performed on the object. For this work two specific types of files are used to provide this information, but different file types could be incorporated as long as they provide the essential information. The mathematical model, a set of trimmed NURBS surfaces, is stored in an IGES-like file. The FE results are stored in an I-DEAS universal file.

The Initial Graphics Exchange Specification (IGES) files are an accepted attempt at standardizing the data files involved in computer graphics. The current version number is 4.0 and includes specifications for NURBS curves and untrimmed and

trimmed NURBS surfaces. A request for change for the forthcoming version 5.0 proposes new entities designed for better describing trimmed surfaces, a bounded surface entity and corresponding boundary entities (see Appendix F) [32], as well as specifications for B-rep solid models. The software associated with this work uses the proposed format of the bounded surface entity with NURBS surfaces as the underlying untrimmed surface. The boundary entity specifications for parameter space curves are followed, but only parameter space curves (NURBS) are used to define the boundaries (that is, no model space curves are designated). Also all surfaces, even untrimmed surfaces, are defined as bounded surface entities, that is, each natural boundary is represented as a constant surface parameter curve and directed properly to follow the left rule.

The universal file which contains the FE results is a file peculiar to the I-DEAS software. Much of the data needed for this work is not explicitly provided in these universal files. A simple example concerns the geometry to mesh information. I-DEAS defines an FE entity called a mesh area. One dataset in the universal file tells which nodes and elements are in a mesh area and another provides which surface a mesh area is on. The desired information is what surface an element is on. Another example concerns the solution location of the FE results. Properties, such as stress, are determined at locations in the interior of the element (that is, not at a node) called integration points. For the contouring operation, the FE results are needed at the nodes and thus must be estimated by some averaging or interpolation scheme. Since this work uses (as primary input) data provided at nodes to display contour information, the process of calculating the data is not central to this work. Thus, a simple averaging (as opposed to a weighted average based on some weighting value such as distance) of the FE data at the integration points immediately surrounding a node is used, as noted in Figure 3.1. However, this is a "plug in-plug out" type of procedure and can be easily replaced with a more complex scheme.

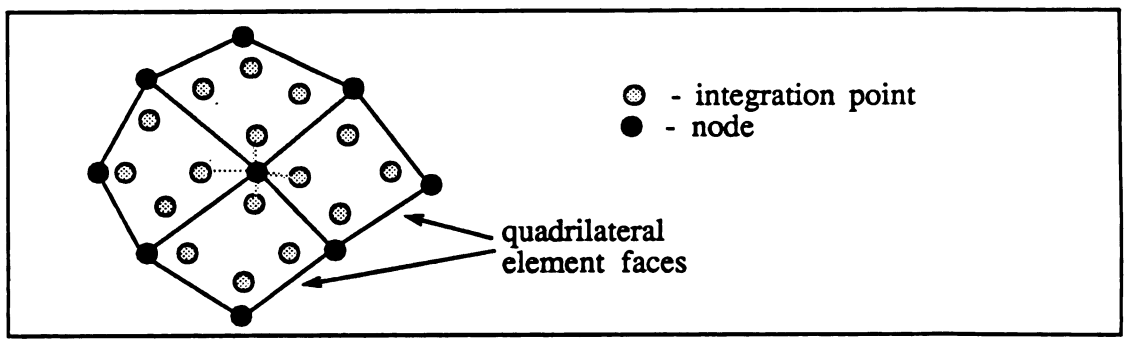


Figure 3.2 FE nodal data

3.3 User Contour Input

Once the geometry model and FEA data are available, the final information needed consists of the particular FE data that is to be displayed (for example, Von Mises stress) and the number and range of contour values to display. The software associated with this work provides a limited menu of display options, shown in Figure 3.3. The default color band contour display consists of 6 regions representing equal ranges of the data from the minimum to maximum data value. The software gives the user the option of changing the minimum and/or maximum values to display and the number of regions.

Contour Data	Contour Options
Maximum principal stress	Number of contours
Minimum principal stress	Show min/max
Maximum shear	Override minimum
Von Mises stress	Override maximum

Figure 3.3 Contour display options

3.4 Procedure for Defining Geometric Contour Regions

The procedure for representing the contour regions as trimmed surfaces is separated into two main processes; data integration and trimmed surface generation. The data integration stage divides into two parts; mesh/geometry integration and integration of contour information. The most intensive part of the entire integration stage is performing the inverse operation of recovering the geometric information that was used to generate the FE mesh. In an understated sense this stage does preprocessing, that is, the data needed to perform the second or generation stage are calculated and organized. Some of these data are edge entities, surface parameter (u,v) or curve parameter (t) values for nodes, directed loops of FE entities (nodes, edges, and elements) that are on or attached to geometric boundary loops, and contour intersection points for each edge.

The trimmed surface generation portion of the procedure also consists of two steps. The first step is to examine the boundary curves of a surface and divide them into segments based on contour intersection points as endpoints. Each segment must have its own mathematical representation. The next step is to walk each of the contour curves, generating/interpolating contour intersections on interior edges while walking, and to place them, correctly oriented, with the boundary segments already determined to form properly defined trimmed surfaces.

Since these procedures become quite enigmatic, an outline is provided in Appendix E. Additionally, at the beginning of appropriate sections of Chapter 4 and 5, there is a flow diagram (see Figure 3.4) presenting the location of the current section's topic in regards to the overall process. The three major areas (heavy outlined boxes at left) are: integrate mesh & geometry data, determine contour intersection points, and generate contour regions. These major areas are expanded into subareas as encountered. The subarea associated with the current section is

highlighted (as read data is below). The abbreviation bnd. stands for boundary; calc., calculate; int., interior; deter., determine; gener., generate; and pts, points.

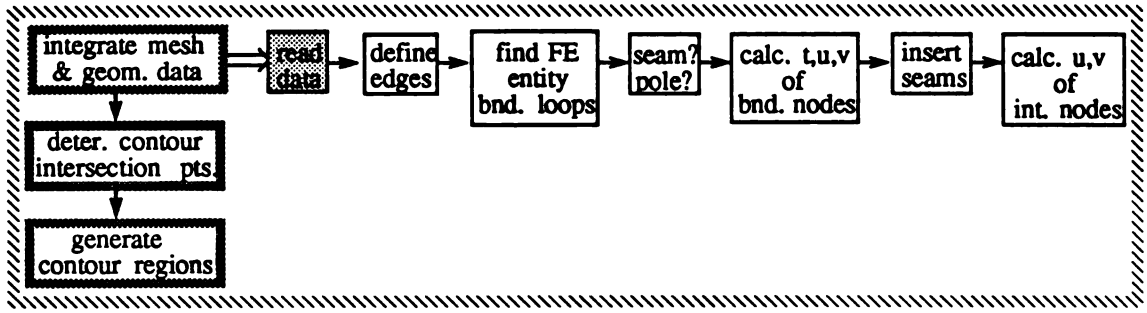


Figure 3.4 Procedural flow diagram

3.5 Output

This work provides a two-stage solution. The first stage consists of the data file containing the trimmed surfaces which represent the contour band regions. The file is an IGES-like file, the same as the geometry model input file. IGES file descriptions provide the capability of specifying a display color. This capability is used to tag each trimmed surface with the appropriate data range it represents. This use of IGES as an output format is intended only for demonstration purposes; in any commercial implementation, a more easily (and quickly) processed "internal" format would be used.

The second stage of the output is the actual display of the data. The geometry file created in stage one is inputted into software designed to make use of the display capabilities of a specific workstation. As mentioned in the introduction, this work uses the excellent display abilities of the HP9000/350 with SRX graphics accelerator and Starbase Graphics Library. Starbase supports the display of trimmed NURBS surfaces of the type developed, thus making it a good candidate for a display mechanism. It does have one major drawback, which is the inability to use hardware acceleration to outline trimmed surfaces. Thus the display option on this device will consist only of contour bands, not curves.

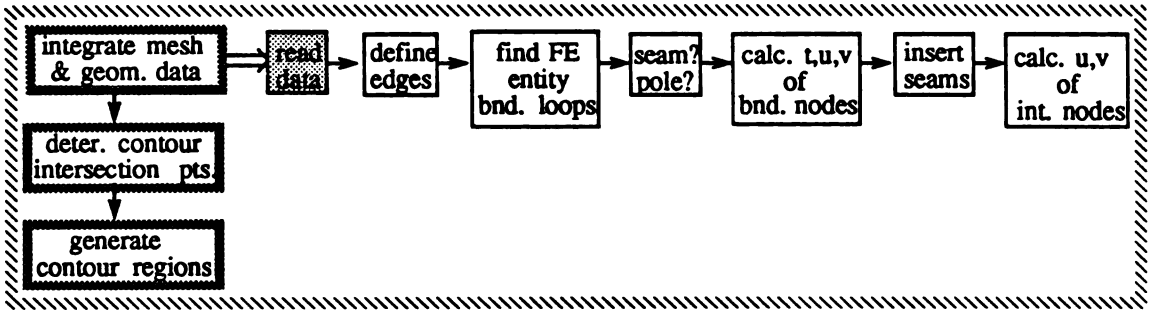
Chapter 4

Integration of FE Data with Geometry Data

As mentioned in Chapter 3, the purpose of the data intergration or preprocessing stage is to make ready the data needed for generating the two types of curves which define the trim boundaries of the contour regions. One type are the individual curves which come from subdividing the actual geometric boundary curves into segments. These segments are based on the contour intersection points that occur on FE edges that represent surface boundaries. Thus, the information needed is a properly directed loop of boundary contour points with the information of which curve they are on and the curve parameter (t) for each point. The other curves are the contour curves; the primary data needed to generate these are the surface parameter values (u,v) for all nodes, the FE data value at each node, a center value (FE data) for each element face, and contour intersection information for each edge in the mesh. An explanation of the flow diagrams provided at the beginning of each section is found in Section 3.4 and an outline of the processes described in this chapter is in Appendix E .

As alluded to in Sections 2.4 and 3.4, the majority of work done in this stage is recovering the information that was thrown away when the FE mesh was generated. Examples of these data are surface parameter values for nodes and curve information for boundary nodes. In Chapter 6 there is a discussion of the information that could be stored during mesh generation to eliminate many of the compute intensive steps described in the following sections.

4.1 Initial Manipulation of Input Data



The input data can be put into 3 general categories — geometry data, FE mesh data, and contour data. The geometry data (found in the geometry input file) consists of NURBS surface data for all surfaces of the analysis object and ordered lists of parameter space NURBS curves that define the geometric boundaries of the surfaces. The FE data (found in the FE results file) includes element-to-node data (that is, given an element label, the list of nodes that comprise this element are available), element-to-mesh area and node-to-mesh area data, nodal coordinates (Cartesian), and FE data at nodes (for example, displacements) and FE data on elements (for example, stress). The FEA results file also provide the bridge information between geometry and the FE mesh, namely mesh area-to-surface data. The final information is the user-provided display data which specifies the particular type of FE data to display and the number and range of contour values desired.

The "givens" described above are used to provide the following additional data. First, in terms of general mesh information, node-to-element data and element-to-surface data are constructed. Next, the specific FE data to be displayed are computed for nodal locations (see section 3.2). Also a "center" value for each element face is computed by averaging the specific FE data on that face. These data are the original data which are provided in the FE results file — for example, stress values at integration points. Finally, a list of scalars which are the FE data values for each contour is constructed.

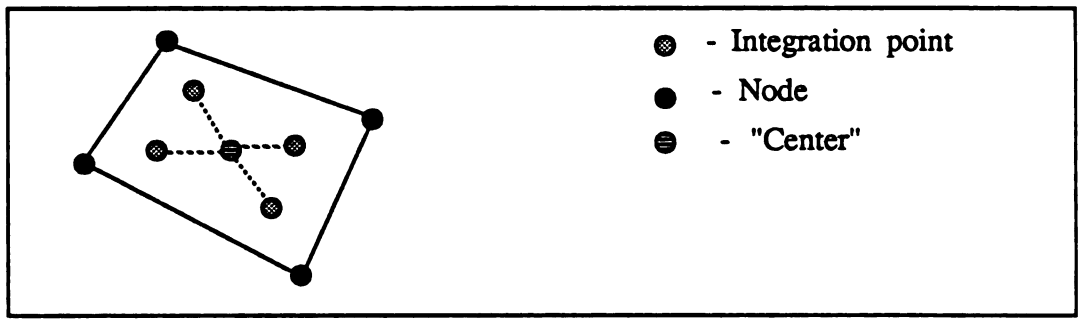
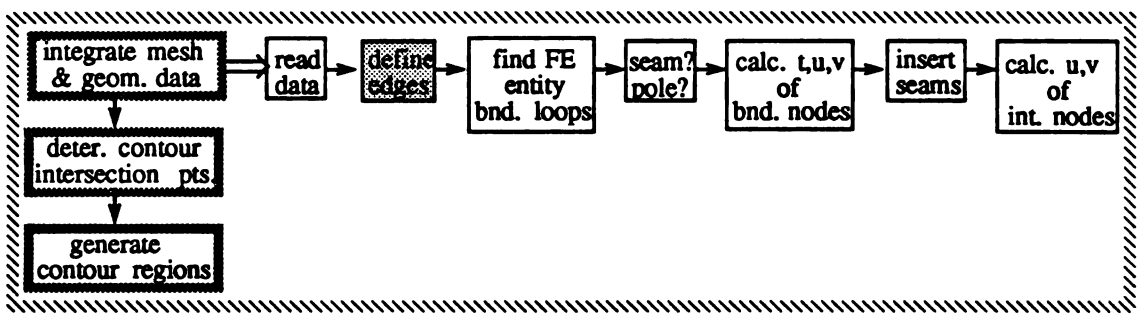


Figure 4.1 Element data values

4.2 Edges



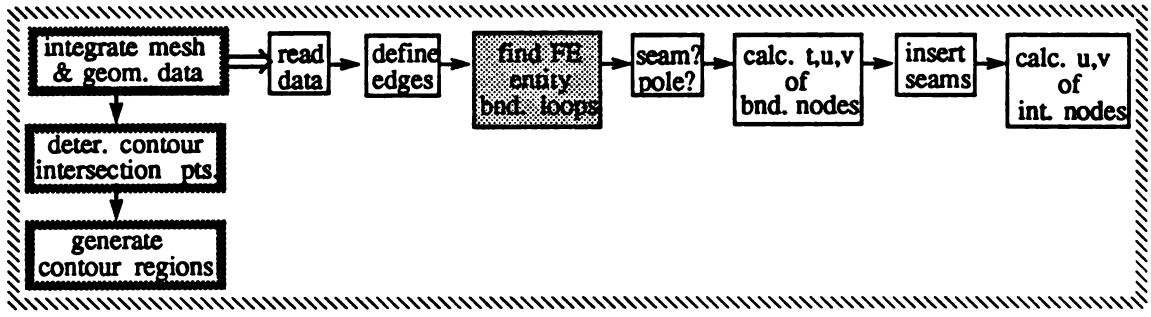
The primary entity used in creating the boundary segment and contour curves is the edge, that is, the unique entity joining 2 nodes and coinciding with a section of an element side. Since this is not a standard FE entity (see section 2.1), the edges must be discovered, defined, and labeled. Since this work is concerned with displaying results on the surfaces of an object, only those element faces which represent part of a surface will be assigned edge entities. The software associated with this work assumes that the FE mesh consists of surface or 2-dimensional elements, but it could easily be extended to handle 3-dimensional elements with the inclusion of a surface face entity (see section 6.1). Consequently, in the subsequent text of this chapter, the term element implies a 2-dimensional element or the face of a 3-dimensional element.

Since an edge is the unique entity joining 2 nodes (see Section 2.1), these nodes define the edge. The process of discovering edges is achieved by examining every

element and listing the node pairs that define its edges. The desired final state of this list is a lexicographic ordering of the pairs. Thus, as each pair is added to the list (element by element), if the order isn't correct (within the pair), the nodes are switched and the pair is flagged to record that their order was switched. The final lexicographically ordered list has pairs that define the same edge, but are on different elements, adjacent. Each unique pair is given an edge label and edge-to-node data are created, as well as element-to-edge data (that is, given an element label, the ordered loop of edges which define it is available) and edge-to-element data (that is, given an edge label, a list of elements it is on is available).

Due to the nature of the geometry model, certain geometric characteristics of each edge must be determined. These characteristics will be called the edge type and there are 5 different types: (0) interior, (1) single boundary, (2) multiple boundary, (3) seam, and (4) pole. Type 1 or single boundary edges are those which have nodes on the geometric boundary of only one surface. Type 2 or multiple surface edges are those which have nodes that are on more than one surface (that is, the nodes are on the common boundary). When the edge labels are assigned, the information of how many elements an edge is on and if these elements are on different surfaces is available. Therefore, edges which are type 1 (only on one element) or type 2 (on more than one element and these elements are on different surfaces) can be determined. Interior edges (type 0) are the default type, that is, all edges are typed as interior unless proven otherwise. There are 2 cases classified as seam edges (type 3): one case is where both of the edges' nodes are on the seam of a surface and the other is where the edge straddles a seam. A pole edge (type 4) is an edge which has one of its nodes situated on the pole of a surface. Seam and pole edges, if they exist, are examined (discovered) later in the preprocessing or integration stage.

4.3 FE Boundary Loops



The first major step in integrating the FEM with the actual geometry is to create a correctly oriented list of FE entities that are "attached" to each geometric boundary loop. This is done as preparation for determining the boundary curve parameter (t) of the nodes on a geometric boundary, as a stepping stone for the calculation of the interior node (u,v) coordinates, and later for producing a correctly oriented list of contour intersection points on a geometric boundary loop. The entities needed are the edges (which have both nodes on a boundary), nodes (on a boundary), and elements (which have at least one edge on a boundary).

This procedure is started during the edge labeling process by creating lists of boundary edge labels (during labeling) for each surface. Each edge entry in the lists also points to the appropriate element it is attached to and the node label of its first (based on proper ordering as described below) node. This node is known based on the node pair tagged to each edge and the switch flag described in section 4.2. The next step is to examine each of these lists and order the edges (and elements) so that they follow the geometric boundary loop in the correct direction. An important assumption is made at this point in regards to the meshing done on the analysis object. When a list of nodes is presented that defines an element, the list is ordered and directed in a counterclockwise direction. The assumption here is that this definition of counterclockwise is consistent with the geometry model's definition of counterclockwise. Thus, given an element's list of nodes, the correct direction on the geometric boundary is the same as in the list (see figure 4.2).

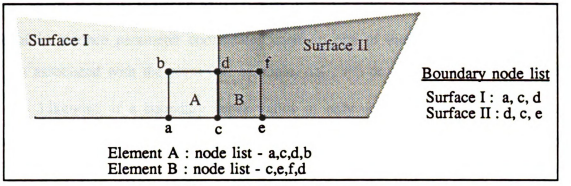
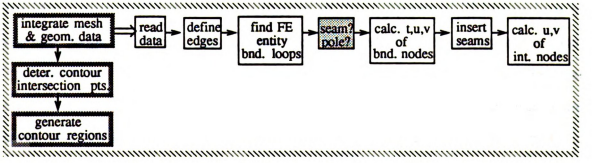


Figure 4.2 FE boundary loops

The procedure used to create a boundary loop starts with an arbitrary selection of the first edge on a boundary list. This edge is deleted from the list and then the list is searched for the second node of this initial edge, which is the first node of some edge still in the list. This deletion and search is continued until the initial node (first node of the first edge) is returned to. The final data should present a correctly oriented loop of nodes, edges, and elements. These loops are generated until all surface boundary edge lists are empty.

4.4 Seams and Poles - Part 1



Untrimmed parametric surfaces exhibit seams if they are closed in one of the surface parameters (this information is provided in the geometry file of the analysis object). Closed implies that for the untrimmed surface there will be 2 natural boundary curves (constant parameter curves) (for example, $u=u_0$, $v=[v_0, v_1]$ and

$u=u_1$, $v=[v_0, v_1]$) that coincide to form a seam (see Section 2.3). Thus, if a point has one of its surface parameter coordinates equal to one of the two constant parameter values associated with the seam (for example, (u_0 , v*) or (u_1 , v*)), then it is on the seam. Likewise, if a boundary curve varies in only one surface parameter and the other is equal to one of the seam parameter values, then this curve lies along the seam. It is, of course, possible to have a trimmed surface which does not have a seam, even though its underlying untrimmed surface does (in other words, the seam was trimmed away).

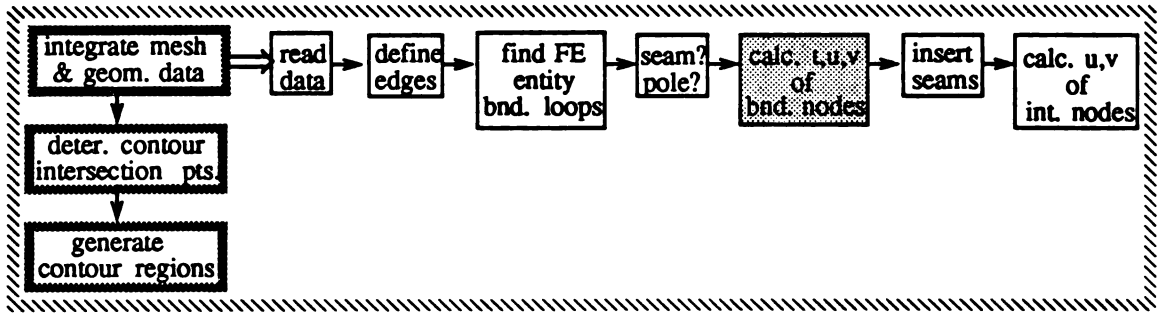
One difficulty concerning seams is how they fit into the boundary curve loops which define the bounded entity trimmed surface. Curves which coincide with a seam do not need to be included in a boundary loop since closed loops in world space (as is required) can be formed without them. Inclusion of seam curves in boundary loops will still produce closed world space loops, but more curves are provided than are actually needed to define the surface. If seam curves are included, then the number of FE loops found in the process of Section 4.3 will be more than the number of geometric boundary loops. This is because the edges that lie on a seam are not typed as boundary edges and thus are not included in the creation of the FE loops. Thus if a surface is closed and the number of FE loops doesn't match the number of geometric loops, then seam curves have been included in the boundary curve loop. The display mechanism (Starbase graphics library) used in this work requires that seams be included in the boundary loops. Therefore, it is necessary to insert seam curves into the FE loops in their appropriate positions. This is done during the computation of the boundary curve parameter value for the loop nodes (see Section 4.5).

This brings up the question of the relationship between a seam curve and the standard FE entities (node, edge, and element). If nodes (and the edges they define)

lie along the seam, then the seam curves can be handled just as the boundary curves. Otherwise, if elements straddle the seam, then some "fake" FE entities and extra interpolation of the FE data are needed. However, it is highly unlikely that this straddling will occur, given that the usual method of meshing a parametric surface model involves generating the nodes in parameter space. Therefore, in this work it is assumed that no elements straddle seams.

Surfaces which have seams can also have poles (for example, a sphere) or surfaces can have no seam and one pole (a bottle) or 2 poles (a hemisphere). Poles are generated by an untrimmed boundary curve for which every control point of this curve is the same, causing the degenerate curve consisting of a single point — the pole. Again it is possible that a pole (or poles) exist on the untrimmed surface but have been trimmed away on the object. If a pole does exist, it is important to locate and flag it, since one set of partial derivatives degenerates at (and near) a pole, causing any numerical method being used in that area to fail to converge. Poles on the untrimmed surface are found by examining the natural boundaries for repeated control points. Poles on the trimmed surface are found by examining each node on the surface for a matching Cartesian coordinate (of the multiple control point that defines the pole). Given that a node match has been made, all edges attached to this node are labeled type 4 or pole edges and the unique surface parameter for the node is recorded.

4.5 Boundary Node Curve Parameters



The next step in the preprocessing is to determine the appropriate boundary curve and curve parameter (t) value associated with the endpoints of each edge on a boundary. There are several situations (shown in Figure 4.3) that can occur in regards to edges and the boundary curves. Thus a single node can have different curve parameter values on different edges (the case of a node coinciding with a curve endpoint) and different curve and surface parameters on different edges (the case of a node on a seam). Additionally, an edge may lie on two different curves, thus making it necessary to "split" the edge into two pieces. These "split" edges will then point to the parameter values of coincident points which represent the endpoints of the two curves involved. This is necessary so that interpolation (to determine contour intersection points) using the curve parameter values of the nodes on this type of edge can be done. These "split" edges are one of the special cases that is taken into consideration, but would most likely not occur so long as the FE "mesher" used the actual boundary curves to generate the mesh.

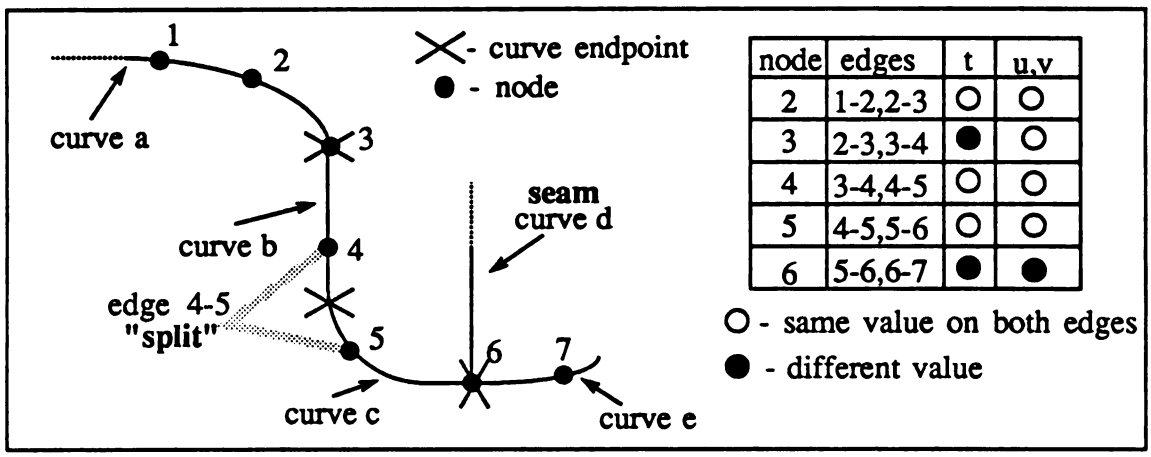


Figure 4.3 Boundary edge cases

This observation (that is, the unlikelihood of "split" edges) implies that nodes lie on boundary curve endpoints. Thus, it is assumed (to start) that the initial endpoint of the first curve in the geometric boundary loop is a node. If the first curve in a geometric boundary loop is a seam curve, then the next curve in the loop is chosen as the starter and current curve. The procedure, essentially, places a small box (in Cartesian space) around the initial endpoint of this curve. All the nodes in the FE boundary loops for this surface are examined for fitting inside the box. If one fits, then this position in this FE loop is marked as the starting position, the endpoint parameter for this curve is stored as the parameter value for this node, and the corresponding surface parameter value (u,v) is computed. If there is no fit, then the box is enlarged and the nodes checked again. This procedure continues until at least one node fits in the box. If more than one node fits in the box, then the one closest to the endpoint is chosen. The next step is to determine if this node is on the initial curve. This is accomplished by computing a point on the curve a short distance away from the endpoint. If this point is closer to the node than the endpoint, then the node is assumed to be on this curve and its curve parameter value (and surface parameter coordinate) is computed (see Appendix A - Point/Curve Calculations). If the test

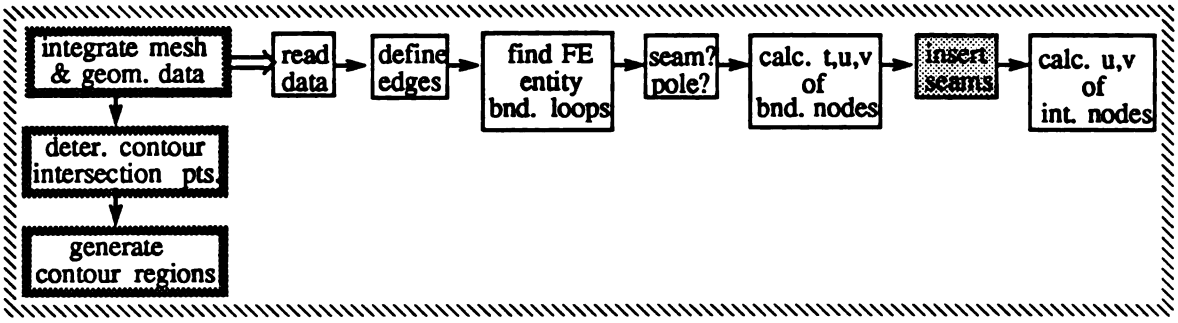
fails, then the next node in the ordered FE loop is chosen as the first node on the current curve.

After this initial node has been determined, each node (and edge) in the FE boundary loop is processed (in order) until the initial node has been looped back to. As each node is processed the different edge cases mentioned earlier are watched for. For example, as each new node is brought up, it is determined if the end of the current geometric boundary curve has been reached or passed over. If seams or poles are possible, then the curve endpoints are watched for the surface parameter values which signal their appearance. The seam curve FE data are inserted (that is, if they are not already included in the boundary loop) as the seam curves are encountered (this process is described in the next section). The procedure described above for determining the desired characteristics of boundary nodes and edges (outlined below) is repeated for all FE loops on all surfaces.

```
if curve is a seam
    go to next curve in loop
if curve endpoint does not match a node in FE loop
    find closest node
    if node is not on current curve, get next node in FE loop
    compute and record node's curve data
else
    record node's curve data
while nodes exist on FE loop, get next node
    if node is past current curve's endpoint
        get next curve
        if not a seam curve
            if current edge spans 2 curves, flag as split
            compute and record node's curve data
        else
            process seam curve
```

Figure 4.3 Boundary node evaluation

4.6 Seams and Poles - Part 2

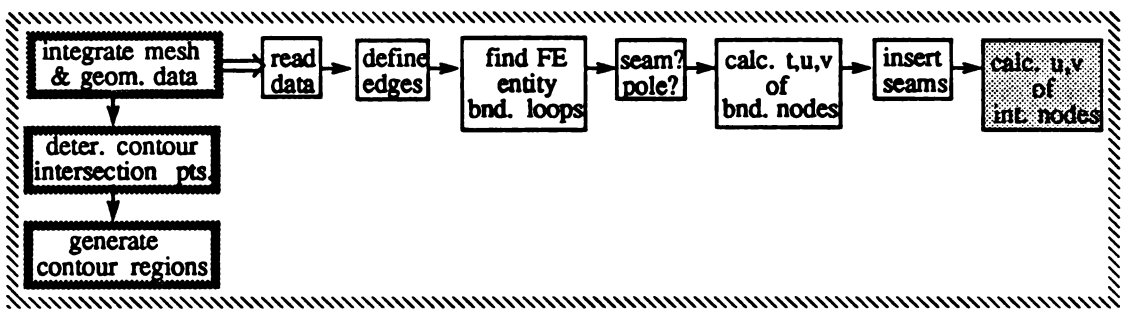


During the following and matching up of the geometric and FE boundary loops, described in the previous section, seam curves are processed when encountered. This processing consists of generating the nodes, edges and elements that exist along the seam curve — that is, one of two coincident curves that lie along a seam or a portion of one. The current node in the FE loop process, is the initial node (endpoint) of the seam curve. The general procedure for walking along the seam is to take the current node and find all the elements that contain this node and are on the current surface. Two of these elements will have another common node adjacent to the current node; these elements are "buddies" along the seam. One of these elements belongs to the current seam curve and the other to the coincident, opposite directed curve that fits in to the loop later. The information for this curve is stored for future use. Thus, when a seam curve is encountered, "storage" is checked for a previously discovered curve with the correct initial node.

Thus, this seam walking process produces a list of nodes, edges, and elements. After each new edge is discovered and flagged as that type, the edge after it on the same element is checked for being a boundary edge. The seam walking process is complete when this occurs. The next problem is to determine where this seam ended up, that is, the FE loop and the boundary curve. For the FE loop, the FE node loops for the current surface are searched to find the node that was the final one found on the seam curve. For the boundary curve, there are two cases. If the seam curves

have been included in the geometric boundary loops, then the required curve is given in the loop definition. Otherwise, the boundary curves for the surface are scanned for one which has its first control point equal to the final node coordinate (parameter space) of the seam curve.

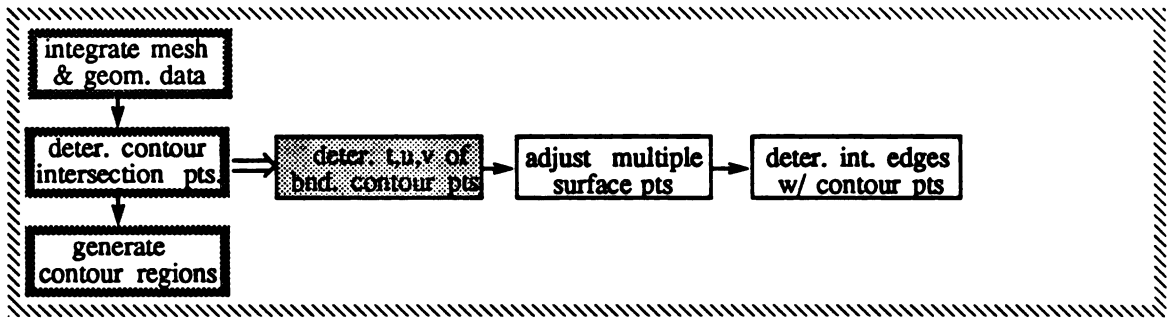
4.7 Interior Node Values



The next step is determine the surface parameter pair for all nodes on the interior of a surface. To begin, each element on a surface is flagged as "unsolved" (additionally, the structure set up to store the surface parameter values for the nodes provides information that tells which nodes have been solved for) and the first element in a boundary loop is "examined". Element "examination" consists of finding a node that has been solved for and walking around the node list using adjacent known node values as initial guesses for the unknown nodes (see Appendix B-Surface/Point Calculations). Once an element is completely solved (and flagged as such), the last edge (that is, the edge which has the last node solved for as its initial node) is looked at to determine the other element it is attached to. This element is then "examined" and the procedure continues until an element is reached which has all of its nodes known. At this point the next (unsolved) element in the boundary loop is used as a starting element and the process above is continued. After all elements in the FE loops are used as starters, the element list is examined for

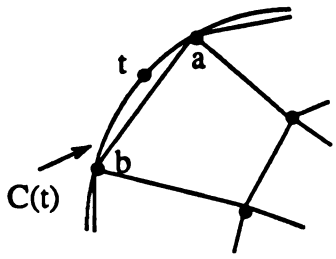
"unsolved" elements. An element in this "unsolved" list that has at least one known node is used as a starter and then the list updated. This process is repeated until all node coordinates are known.

4.8 Contour Intersection Points on Boundary Edges



The final steps in the data integration stage bring in the display data, that is, the FE data values for the contour curves. These values are examined with respect to their existence (contour intersection points) on edges. The desired locational information for a contour intersection point of a boundary edge is the curve and a curve parameter value for the point. This parameter value is calculated by linearly interpolating between the endpoints of the edge (see figure 4.3). If the edge is "split", as described in section 4.5, then the contour point is on one part of the edge or the other. To determine which, a quasi-contour point is calculated/interpolated based on the surface parameter coordinate (u,v) of the nodes. The relationship (in parameter space) between this point, the nodes and the curve endpoint is used to determine the best candidate for the edge part and the interpolation factor for calculating the curve parameter value (t). The criterion used is outlined in Figure 4.4. All boundary edges are processed by following the FE boundary loops that were generated earlier. The final data are a correctly oriented loop of contour points on an existing FE loop where each contour point points to — the edge, element, and boundary curve it is on; its curve and surface parameter coordinates; its FE data

value and the contour range associated with the boundary of the surface following this point; and a "used" flag (used in the generation stage) initialized to zero.

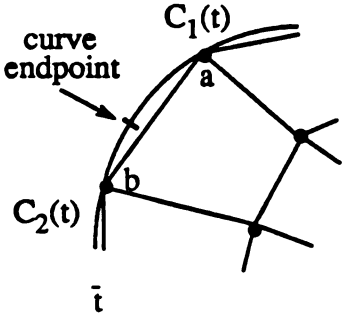


	Node a	Node b
curve parameter	t_1	t_2
FE property value	P_1	P_2

FE data contour value : P where $P_1 < P < P_2$ or $P_2 < P < P_1$

contour intersection point : $t = t_1 + (P - P_1) \bullet (t_2 - t_1) / (P_2 - P_1)$

Figure 4.4 Boundary edge contour points



	Node a	Node b
curve parameter	t_1	t_2
FE property value	P_1	P_2

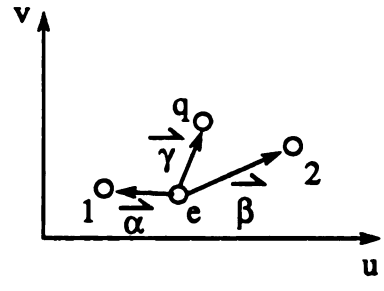
curve endpoint on C_1 : t_{e_1}
curve endpoint on C_2 : t_{e_2}

FE data contour value : P

$u,v(t_1) = u_1, v_1$; $u,v(t_2) = u_2, v_2$; $u,v(t_{e_1}) = u,v(t_{e_2}) = u_e, v_e$

quasi-contour point (q) : $u = u_1 + (P - P_1) \bullet (u_2 - u_1) / (P_2 - P_1)$

$v = v_1 + (P - P_1) \bullet (v_2 - v_1) / (P_2 - P_1)$



if $\dfrac{\vec{\alpha} \bullet \vec{\gamma}}{|\alpha|\,|\gamma|} > \dfrac{\vec{\gamma} \bullet \vec{\beta}}{|\gamma|\,|\beta|}$

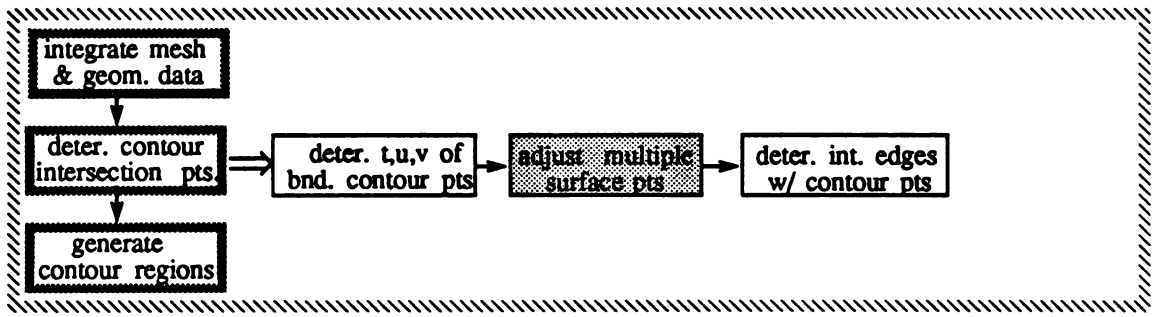
$t = t_1 + (t_{e_1} - t_1) \bullet |\gamma - \alpha| / (|\gamma - \alpha| + |\gamma|)$

else

$t = t_{e_2} + (t_2 - t_{e_2}) \bullet |\gamma| / (|\gamma - \beta| + |\gamma|)$

Figure 4.5 Split boundary edges

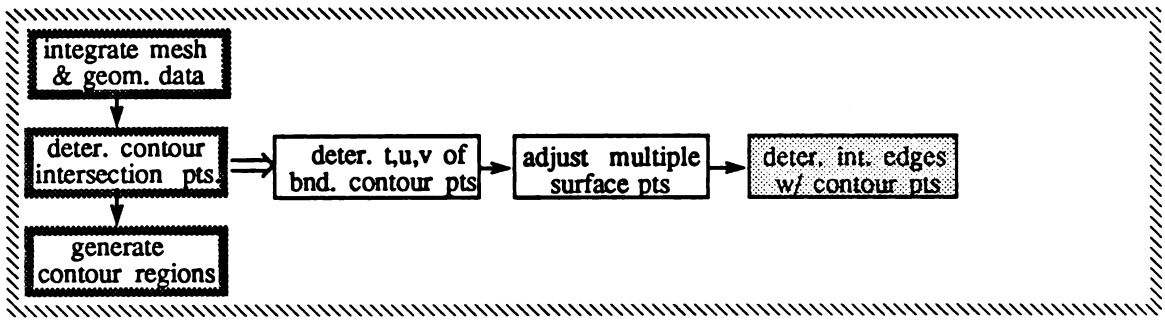
4.9 Contour Intersection Points on Multiple Surface Edges



For contour intersection points on boundary edges, the curve parameter values of the nodes are used to determine/interpolate the location (see previous section). For a boundary edge which lies on more than one surface, a contour point is determined for each surface. When the parameter value of each contour intersection is evaluated (that is, x,y,z(u,v(t))), the resulting points may not coincide. This would cause a break in the contour curve as it crosses the boundary.

To "smooth out" the boundary portion of a contour curve, a specific x,y,z point is determined for the contour intersection point. This point is chosen to be the average of the Cartesian coordinates of the interpolated contour points found for each surface. This average point is not necessarily on the object. Using this point, each surface's boundary curve is examined to determine the point on it closest to the averaged point. Since all of these boundary curves are coincident, with regard to Cartesian space and the current edge region, the Cartesian coordinate of the closest point for each curve will be the same. The new curve parameter values for the contour intersection point are determined in the closest point calculation.

4.10 Contour Intersection Points on Interior Edges



For interior edges, the surface parameter coordinate (u,v) for the nodes is used to calculate/interpolate the location of the contour intersection point. It is a linear interpolation and is the same as the computation shown in Figure 4.4 for the quasi-contour point. However, this calculation is not performed in the preprocessing stage of the work, but instead during the contour walking stage of generation. The preprocessing of interior edges consists of determining if contour intersection points exist on an edge and determining the FE data values associated with them.

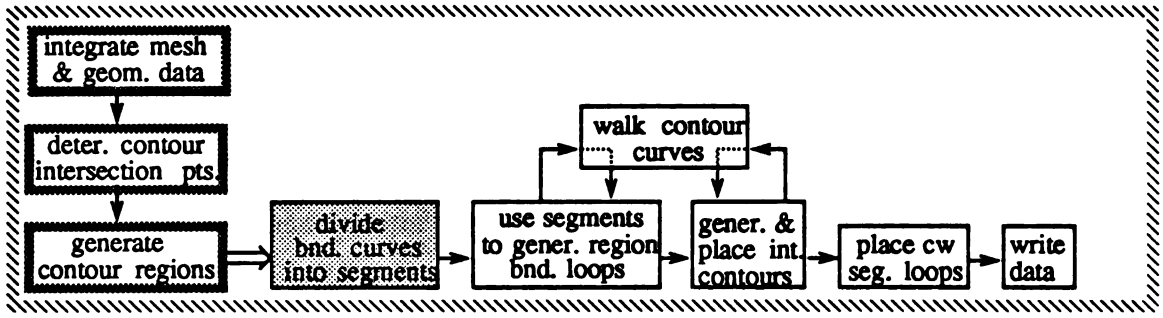
Chapter 5

Generation of Contour/Trim Regions

As mentioned, the purpose of this work is to define the contour regions of a surface as bounded-entity-trimmed surfaces within the surface. This requires a list of parameter space NURBS curves that form a closed loop in world space and represent the boundaries of the contour region. The curves will come in two forms — segments of the original parameter space boundary curves and parameter space linear B-spline curves that represent the contour curves. The boundary segment curves are determined first and then used to initiate the loop of curves that define the boundaries of the contour regions that contact the geometric boundary of a surface. The contour curves are walked/determined as each loop is formed. During this walking process the usual contouring idiosyncrasies are encountered and dealt with, as are the problems that are inherited from working in parameter space (for example, poles). The final step consists of generating the interior contour curves (those curves which do not contact an original geometric boundary) and associating them with the proper regions already generated. The entire generation procedure is applied to each surface independently.

An outline of the procedures in this chapter is in Appendix E. An explanation of the flow diagrams found at the beginning of some sections is in Section 3.4.

5.1 Boundary Segment Curves



In Chapter 3 the methods for determining a properly oriented loop (FE loop) of edges and elements associated with the geometric boundaries (with seams included) were presented. Also each of these edges was examined for contour intersection points and the curve label and curve parameter value for each of these contour points was calculated. Thus, a boundary curve is made up of segments that have contour points (and/or curve endpoints) as endpoints. Each of these segments needs to be defined as a B-spline curve. The final product of this process is a properly directed loop of these segment curves that correspond to the geometric boundaries of a surface.

How these segment curves are defined depends on whether the device that uses them as input accepts a type of B-spline that has parameter bounds as part of its definition. This type of curve is similar to a trimmed surface, that is, there is an underlying curve (like the untrimmed surface), but only a portion of it is the actual curve being defined. This portion is defined by a range of parameter values $[t_a , t_b]$ (the bounds). If this type of curve is acceptable, then the segments are defined using the original boundary curve definition but with the appropriate bounds. These bounds, of course, are the curve parameter values that were found for each contour intersection point.

If this type of curve is not acceptable, as is the case for the display mechanism used in this work, then the segment's curve definition becomes much more complex.

Each segment must be defined as a B-spline curve that has the segment endpoints as its endpoints (that is, its first and last control points). This is achieved by adding control points to the original set of control points that define the boundary curve in a manner so that the shape of the curve is unchanged. The final set of control points can be sliced (maintaining the original order of the points) into subsets that define each segment (see Figure 5.1). The method used is called the Boehm knot insertion algorithm [18,19]. Normally this algorithm would have to be applied k (the degree of the curve) times to each contour intersection point in order to separate each segment's control points. This is computationally equivalent to evaluating the curve at this point (that is, computing u,v(t) using the DeBoor B-spline evaluation algorithm). However, the final application involves computing/adding one new control point which is the contour point and relabeling others. Since the coordinate (u,v) of this point is known, only relabeling is done (that is, no computation). Thus, the algorithm "proper" is applied k-1 times for each contour point. This fact, as well as the simplicity of the algorithm, make it the best choice for this job [18,19,20,21,22,23].

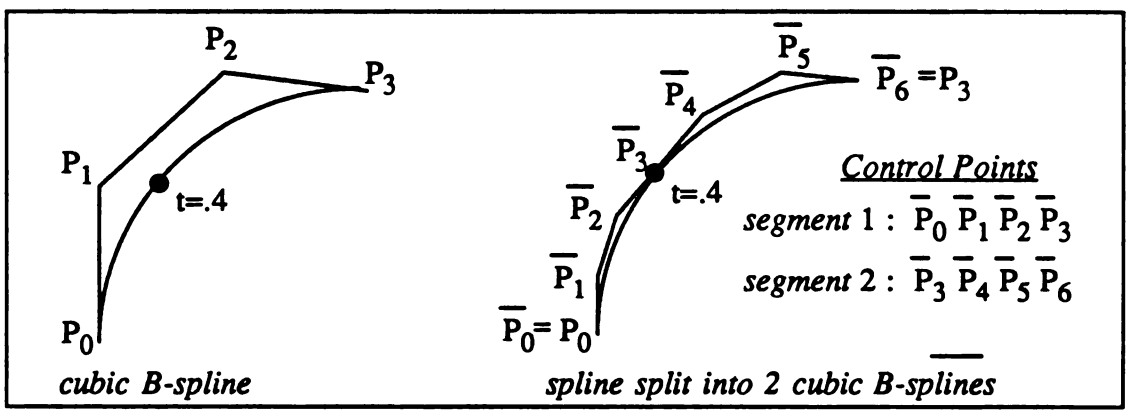
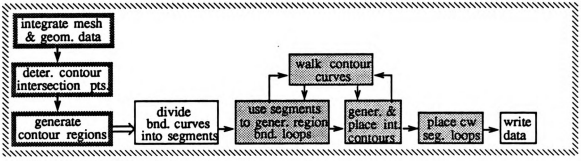


Figure 5.1 Boehm knot insertion

This section again brings up the differences between aspects of this work and the methods that would be used if control over all of the input/output were possible. Naturally, the bounded curves described first would be the method of choice for defining the segments, since this is a computation-free procedure.

5.2 Generation of Boundary Loops for Contour/Trim Regions



In the previous section, the creation of an oriented set of curves representing segments of the geometric boundary curves that exist between contour intersection points (or a curve endpoint and a contour intersection point) on a boundary was described. These boundary contour points are starting and stopping points for the contour curves. When the appropriate boundary segment curves (hence called segments) and contour curves are put together to form a closed loop, they define the outer boundary of a trimmed surface/contour region. This type of boundary (that is, one that includes segments) and the associated region will be referred to as exterior. The contour regions that are formed solely by contour curves (see Figure 5.2, region A) will be called interior regions (also these contour curves will be designated as interior). Additionally, if a surface is associated with more than one FE boundary loop, then one loop will be counterclockwise (outer boundary) and the others clockwise (inner boundaries or "holes"). The direction of a loop is determined via the method described in Appendix C.

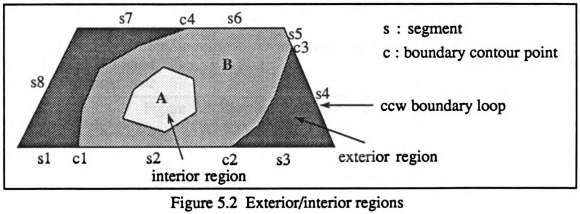
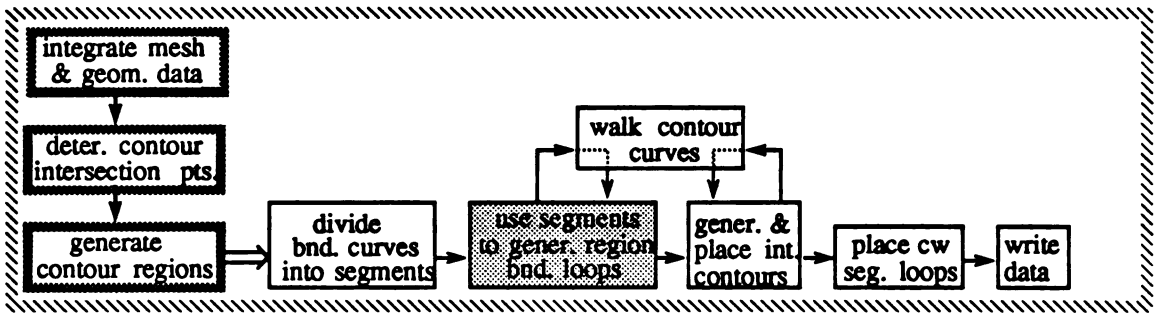


Figure 5.2 Exterior/interior regions

5.2.1 Exterior Regions via the Outer Boundary



The exterior regions associated with segments on the counterclockwise (or outer boundary) loop are generated first. A segment is employed as a starter for a loop of curves that will define the outer boundary of an exterior region. For each boundary contour point there are contour point-to-segment data, where the contour point is the segment's head. For each segment there are segment-to-contour point data, where the contour point is the segment's tail. Note that not all segments end with a contour point (for example — s5, s7, and s8 of Figure 5.2). These segments have a tail contour point labeled zero, which is interpreted as "add on the next segment". As an example, Figure 5.3 gives a shorthand version of a path through the logic used to generate the region of Figure 5.2 labeled B (generating/walking contour curves is described in the next section).

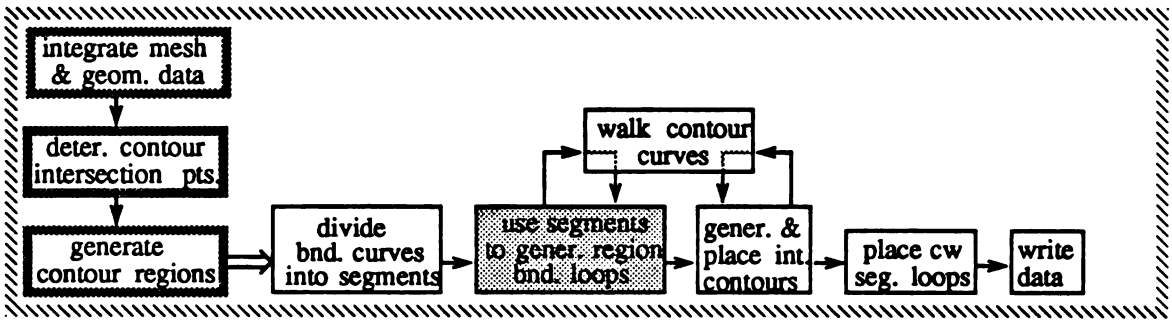
pull unused s2 from the segment list
s2 has c2 as its tail
generate contour which starts with c2 and ends up at c3
c3 is s5's head
s5 has no tail, so add on s6
s6 has c4 as its tail
c4 has been used, so retrieve contour data, which includes end at c1
c1 is s2's head
s2 is used, quit

Figure 5.3 Exterior generation logic example

The segments are processed in order until all have been used. For example, in Figure 5.2 the first region generated would start with s1, generate the contour curve

that goes from c1 to c4, then add on s7 and s8. The next region generated would be region B, etc. Note that each segment belongs to one contour region, but each contour curve belongs to two (for example, the contour curve from c1 to c4 is a curve defining one region and the same curve but directed from c4 to c1 is part of another region). To keep track of what has been done, the segments and all contour intersection points have a flag that is initialized to zero to signify that it is unused. Once a segment has been used to define a region, the segment's flag is changed to that region's label (that is, a nonzero integer). Once a contour point has been used to generate a contour curve, its flag is changed to the curve label assigned to the contour curve. This allows the contour curve to be easily included in its second region. Also, the labels of the two regions for each contour curve are stored, as well as the FE data values associated with each contour.

5.2.2 Exterior Regions via Inner Boundaries



The next step is to generate the exterior regions attached to the clockwise boundary loops ("holes" in the original surface). There are two cases; either some of the segments in the loop were used already (Figure 5.4a) or all are unused (Figure 5.4b). If the first case is true, then an unused segment that is adjacent to a used segment is picked as a starting segment for the region generation procedure described earlier. The loop is also processed as before, that is, examining segments in order along the FE loop. Thus, the set of curves generated for a region will be directed in the counterclockwise direction and be the outer border of the region.

If the second case is present (all of the segments are unused), then one of the regions generated while processing the clockwise (geometric) boundary loop will actually be an inner boundary for some (as yet undetermined) contour region. This group of contour curves and segments is the only loop attached to the geometric "hole" that will be clockwise. Thus, each loop is checked for its direction until the clockwise one is found. The problem of finding out which region (and outer boundary) is associated with this inner boundary is deferred until after the interior contour curves are examined.

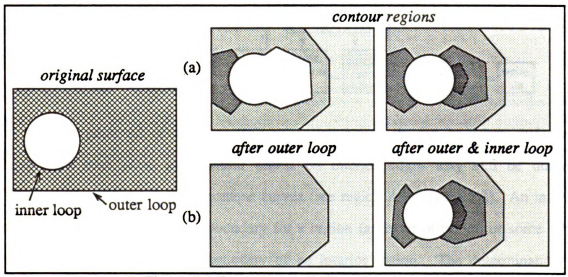
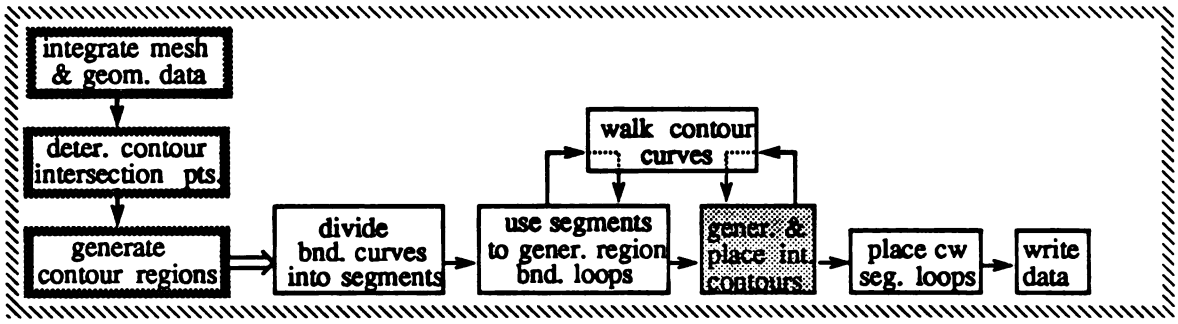


Figure 5.4 Inner geometric boundary loops

The discussion so far has been limited to defining the geometric properties (that is, the boundary curves) of the contour regions. There is one additional piece of information needed to properly define these contour regions, that is, the range of FE data values that the region represents. This range, of course, is signified by a color. Thus, each region must have a color label attached to it. For regions which contain segments of the original geometric boundary, the color is determined based on a segment's head-position contour intersection point. In particular, as the exterior region is generated a situation will occur where a contour point on a geometric

boundary has been reached and this point is the head of the oncoming segment. Each boundary edge contour point has a color label attached to it which signifies the appropriate data range on the area of the boundary that follows. Thus, the color of exterior regions is determined during their generation.

5.2.3 Interior Regions



At this point all of the contour intersection points on the boundary and seam edges have been used, but contour points on interior edges may still be unused. These points will form interior contour curves (see region A of figure 5.2). An interior contour curve defines an inner boundary for a region (an exterior region or some other interior region) and the outer boundary of an interior region. The determination of what region needs to have an inner boundary "added to" it is the main difficulty in dealing with interior contour curves/regions. The term "added to" refers to the requirement that the region being "added to" already exists (that is, the outer boundary has already been generated). This assures that the current interior contour curve will be assigned to its correct region, that is, a contour curve will not be discovered (later in the processing) that lies between the current contour curve/inner boundary and the outer boundary it was "added to". There are a first-attack approach and a backup approach for processing the interior contour curves; each reflects the need for an already generated region to operate on.

The first-attack approach looks for useful information during the search for unused contour points. This search examines the surface's edge list (which points to the contour point list) instead of the contour point list directly. This is prompted not only by the need to know the starting edge, but also because the (useful) information of whether another contour point exists on an edge is readily available. A desirable edge to choose as a starter for an interior contour curve generation is one with both a used and an unused contour point on it, since the used point "knows" what curve it is on and the curve "knows" what regions it belongs to. Since the FE data values of the contour points in question make only one region feasible, the correct region to "add" the contour to has been found. Additionally, the data range (or color) of the interior region is deducible from these points (for example, if the current (unused) value is 10 kpsi and the used value is 15 kpsi, then the interior region's range has 10 kpsi as its upper bound). Thus, the first approach to generating interior contour curves is to make use of all edges which have a used/unused contour point combination.

If the situation is that all of the "unused" edges have only unused contour points on them, then the backup (a more unwieldy) approach is used. First, all the remaining contour curves are generated along with the FE range of their interior regions (method for determining range is described below). During the generation process the minimum and maximum values of the contour intersection point coordinates (u,v) are calculated. These values are used to form a min/max or bounding box around each contour curve. The curves are examined in order of decreasing box area — this assures that the "add only to existing regions" requirement is satisfied. Given a specific contour curve and associated FE data value, the next step is to sort through the existing regions and find those which are acceptable candidates. Acceptability means the region has the correct FE data value range (the current contour value and the range values of the associated interior region determine the correct range for the "add to" region). Next, bounding boxes (based on

all control points for all boundary curves — segments and/or contour curves) are formed for those regions which are acceptable candidates. Those regions' boxes that do not contain the curve box are discarded. The remaining regions are examined more closely by determining if a point on the contour curve is inside the exact boundary polygon (series of control point polygons) of the region (see Appendix D). Of course, anytime in the above procedure when the acceptable list of regions is a single region the procedure is stopped..

```
examine edges
    create list of unused contour points (with edge data)
    flag points :  0 if point is only contour point on edge
                   1 if point shares edge with other unused points
                   2 if points shares edge with at least 1 used point
while a 2-flag point exists
    determine region of used neighbor point
    grab an element attached to this edge
    walk contour curve
        flag points as used and delete from unused list
        if encounter a 1-flag point, change other points on edge to 2-flag
    add curve to neighbor region (curve=inner boundary)
    create new region (curve=outer boundary)
if unused list is not empty
    if 1-flag point exists
        generate all contour curves on associated edge
        create bounding box for each curve
        determine color of region inside curve
        find curve with biggest box
        place curve : find acceptable existing regions
                      if more than 1 region
                          determine bounding box for each
                          check if curve box is inside any region boxes
                      if more than 1 region box, do polygon/polygon test
                      add curve to region (curve=inner boundary)
                      create new region (curve=outer boundary)
        do other curves found and create regions in appropriate order
    else  (i.e., only 0-flag points exist)
        generate all contour curves
        create bounding box for each
        order from largest box to smallest
        place curves - processing in order
```

Figure 5.5 Procedure for interior contour curves

Another difficulty that is encountered with interior contour curves is that when one is walked/generated, the direction is not evident. If it is walked in the counterclockwise direction, then the curve control points are properly oriented to define the outer boundary of the interior region; if clockwise, then the inner boundary of the "add to" region. Therefore each interior curve must be checked for its generation direction (see Appendix C). The direction is also used to determine the FE data range of the interior region that is being processed via the second approach (described above). If the curve was generated in the clockwise direction, then the tail node of the starter edge (with respect to the starter element) and the current contour value determine the range of the interior region (see Figure 5.5). On the other hand, if counterclockwise is the case, then the head node is examined The nodes examined are inside the interior region..

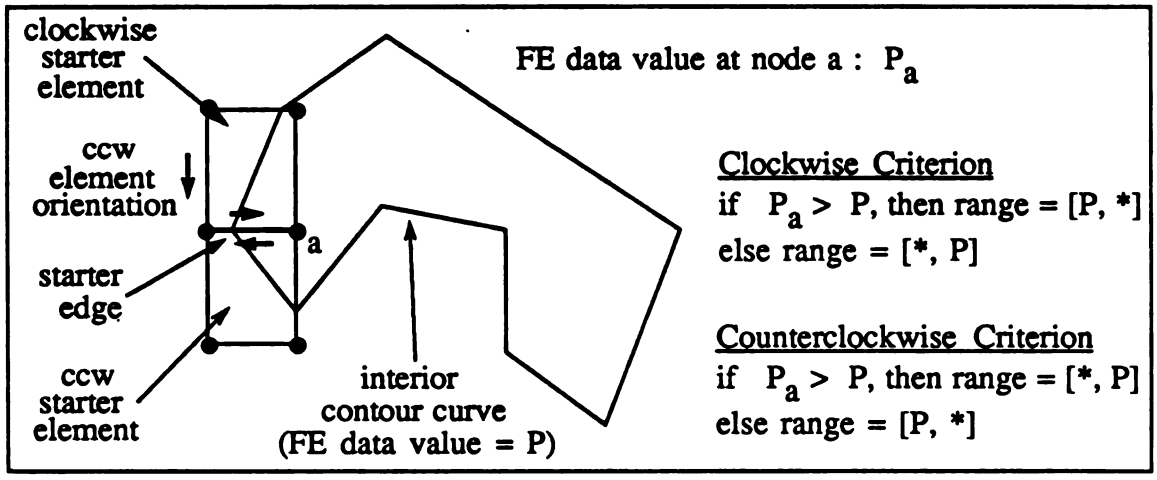
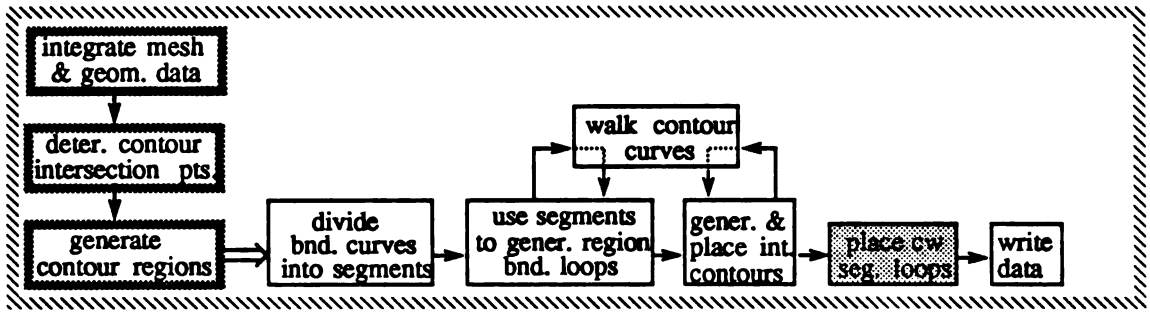


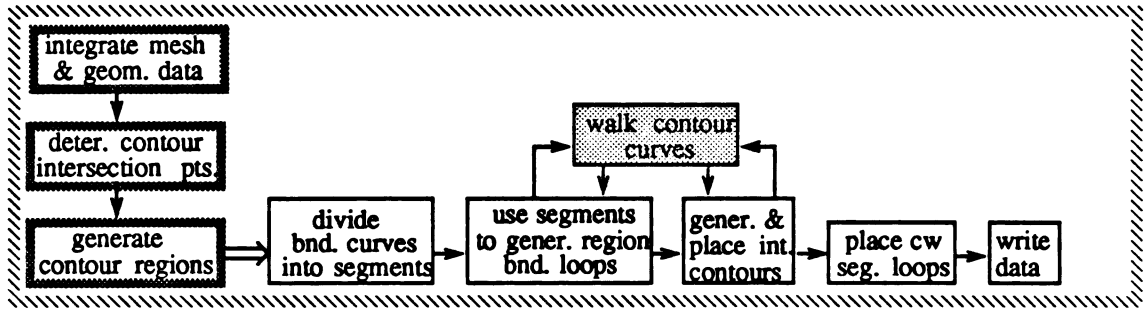
Figure 5.6 Determination of FE data range (color) for interior regions

5.2.4 Exterior, Inner Boundaries



The final operation in the region creation is to take care of (place with the appropriate region) any inner boundaries that were generated in the inner loop/exterior region part of the generation process. The procedure is similar to that outlined above for interior contour regions. Again, the first step is to compute a bounding box for the loop of curves forming an inner boundary. The next step is to sort through the existing regions and determine those which are acceptable candidates. Just as with other exterior generated boundary loops, the color (FE data range) for the associated region was determined during generation. Thus this range defines the acceptable regions. The procedure that follows is the same placing procedure as for the interior contour curves — bounding boxes are formed for all of the acceptable regions, those that don't contain the inner boundary loop are discarded, and those regions that remain are put to the more detailed and final test.

5.3 Generation/Walking of Contour Curves



Contour curves are defined as linear uniform non-rational B-splines in parameter space. The control points for the spline are the contour intersection points that are encountered as the contour curve is generated/walked. The curves will consist of straight line segments in parameter space and curved (depending on the underlying surface) segments in world space. These segmented curves will have point continuity but not necessarily slope continuity. Linear splines are, of course, the "simplest" curve to fit to the contour points; other options are feasible (some of which are discussed in Section 6.6) and could easily be substituted since the generation of contours is a "plug in-plug out" type of operation [24,25,26].

In the previous section, the boundaries for the trim/contour regions were described. These boundaries consist of geometric boundary segments and contour curves or just contour curves. Thus there are two types of contour curves — those that attach to a boundary segment and interior contour curves. The start and end of contour curves are defined by a contour intersection point. Although the general walking procedure is the same for both types of contour curves, the starting and stopping procedures are different.

For the first type of contour curve (that is, a curve which starts from a boundary contour intersection point), the following information in known about the starting contour point : the edge and element of the contour point and the FE data value (for example, Von Mises stress) of the point. This is the basic information needed for the

walking procedure. To register when the contour curve has reached its endpoint, the edge type of each edge encountered is checked. If the edge is one of the boundary types, then the curve is complete. For the second type, only the edge and FE datum are known. In this case, an arbitrary element which is attached to the given edge is chosen for a starter. The curve is complete when the starting edge is returned to.

As mentioned above, the walking procedure needs an element, an edge, and a FE datum value for a particular contour intersection point. This walking procedure processes elements one at a time, moving across the surface via adjacent elements. The input edge to the processing of an element will be called the entry edge (in keeping with the metaphor of walking). For each element there is an ordered list of the edges that form it. The first step in the element process is to examine each edge in this list (skipping over the entry edge) for unused contour intersection points of the same value (that is, the same FE data value). If only one appropriate edge is found in this search, then this is the exit edge.

If there are multiple edges (this is often called a degenerate element in contouring lingo [25]), then a temporary pair of edges is created that go between the "center" of the element and the two nodes which define the entry edge. The entry edge and two temporary edges form a triangle (see Figure 5.5). Every element has a FE data value associated with its "center", thus a quasi-contour intersection point can be determined for a temporary edge of the triangle. One of these edges (and only one) will have a quasi-contour point with the appropriate data value. This quasi-point directs the contour toward the correct exit edge. If the temporary edge associated with the head node of the entry edge has the quasi-point, then the edge list is processed in reverse order, starting with the entry edge, until an edge with an appropriate contour point (the exit edge) is found. If the tail node is involved, then

the list is processed in forward order, beginning with the entry edge, until the exit edge is found.

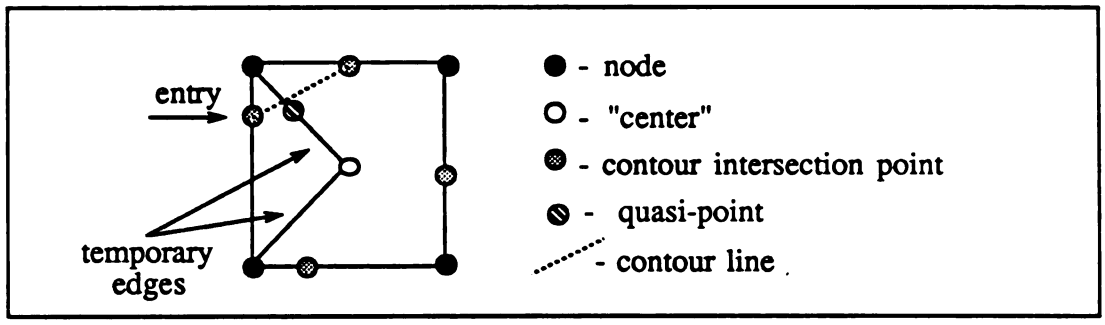


Figure 5.7 Contouring element degeneracy

Now that the exit edge has been determined, then the surface parameter coordinate for the exit edge's contour point needs to be computed (since only existence and FE data values were determined for non-boundary edge contour points in the data integration/preprocessing stage). As mentioned in Section 4.10, the interpolation scheme to determine the surface parameter coordinate of the contour point on an edge is linear and uses the surface parameter coordinates of the node endpoints (see Figure 5.7a). If a surface is being processed that has poles, then each edge's type is checked before interpolation. To handle poles, the multiple value coordinate of the pole node is replaced by the coordinate of the edge's other node (see Figure 5.7b). Once known, the contour point coordinate is added to the control point list for the current contour curve. The next step in the element process is to examine the exit edge's element list to determine the element adjacent to the current element (that is, where to walk to next). Finally, the exit edge (and contour point) becomes the entry edge (and contour point) and the adjacent element becomes the current element. Thus, the element process can repeat itself until the curve is complete.

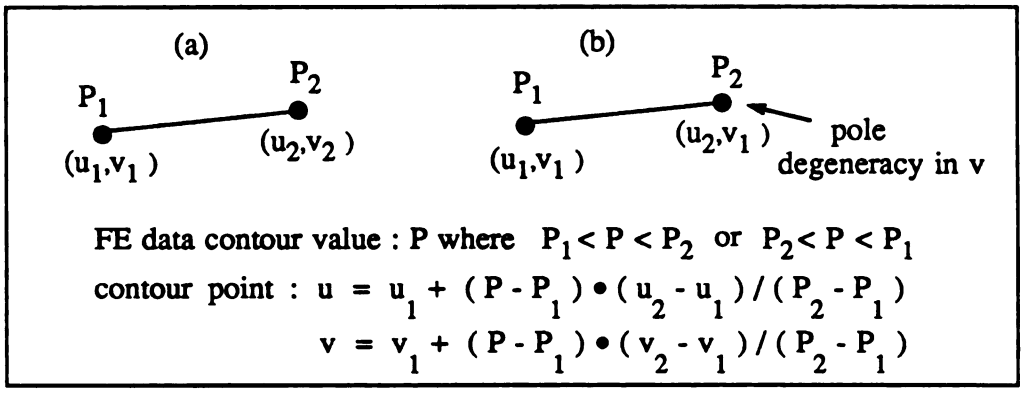


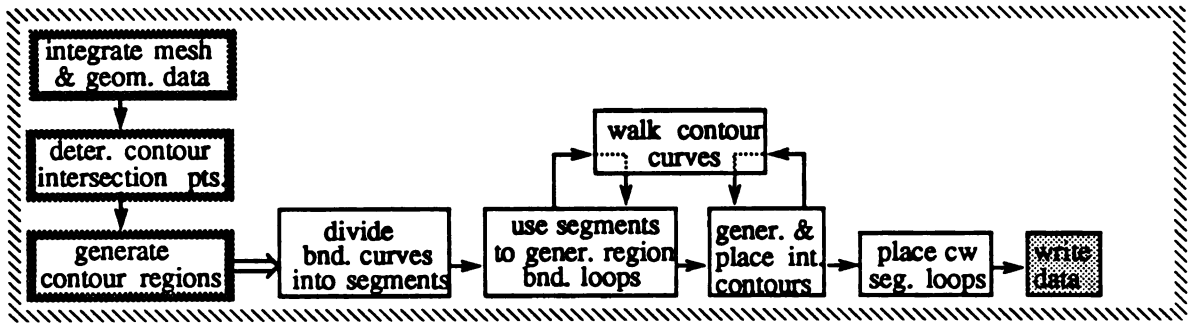
Figure 5.8 Interior and pole type edge interpolation

Based on what type of curve is been generated there are two methods to determine if the end of the walk has been reached. If an exterior contour curve is being generated, then the edge's type is checked — if it is a boundary edge, then the contour curve is complete. If an interior contour curve is being generated, then the exit edge is checked for being the initial or starter edge used for the walk. In both of these cases the surface parameter coordinate is already known and is just "looked up" for placement as the final control point for the curve.

```
if walking from boundary segment
    retrieve edge and element of contour point
    flag as exterior walk
elseif walking from interior edge and contour point
    choose an element that this edge is on (i.e., pick first in list)
    record label of initial contour point
    flag as interior walk
while curve in not finished
    examine element edges & make a list of edges with proper contour points
    if more than 1 edge, perform degenerate element test
    if exterior-flag and edge is a boundary type
        get u,v of contour point and finish curve
        retrieve segment started by edge's contour point
    elseif interior-flag and contour point is initial one
        get u,v of initial point and finish curve
    else
        compute u,v of contour point
        determine next element
```

Figure 5.9 Contour walk procedure

5.4 Final Geometry Information



After the generation process, all of the information has been determined that is necessary to define the contour regions as trimmed NURBS surfaces (color coded). Each region points to its color label, its underlying untrimmed surface, and a set of boundary loops. Each loop points to an ordered set of parameter space curves (either segments or contour curves). Each curve is properly directed so that they attach head to tail, based on the parameter flow of each curve, and form the desired closed loop (in world and parameter space since seams are included in boundary definitions).

Chapter 6

Results and Discussion

In Chapter 5, the processes used to determine the information needed to produce a geometry file containing the contour/trim regions was described. Once this geometry file has been outputted, the contour regions can be displayed. Some displays produced from this work (as well as one from a typical FE package) are presented in this chapter. The chapter continues with comparisons and a discussion of some extensions and implications of this work. A brief summary ends the chapter and the text.

6.1 Pictorial Results

Once all of the curves that define the trim boundaries of the contour/trim regions have been generated and attached to their appropriate regions (see Section 5.4), the contour regions are ready for display via a mechanism which handles trimmed NURBS surfaces. This work uses (for display purposes) the Starbase Graphics Library on the HP9000 SRX family of workstations. This Library has been used to create SURFMAN, software developed at the Case Center for CAEM which displays trimmed NURBS surfaces and activates certain key/knob operations. There is complete positional and rotational control of the object, two variable-color light sources, perspective view, wireframe display, Gouraud shading, quick faceted shading, specular reflection and other rendering operations (such as control of clipping planes, backface cull, and depth cueing). All operations are real time.

The following photos show a contour-band display of the maximum shear values determined from an FE analysis on a three-surface object (the left and right surfaces are planar and the center surface is part of a cylinder). The first three figures show the contour bands displayed on the FE mesh. The display is produced by the post-processor of the Engineering Analysis module of I-DEAS on a SUN 3/60 workstation. Figure 6.1 and 6.2 show continuous-colored contour bands; Figure 6.1 shows the outline of the element borders along with the contour information. The third figure displays continuous-shaded contour bands, that is, a simple light source and shading model is employed to produce a shaded image. The normal vectors at locations on the object are one of the inputs to the shading model — in particular, they determine the color's intensity (along with light source data). Since an element is planar, the intensity is constant across each element. This produces the discontinuity in color intensity across elements on the cylindrical surface. Also note the jagged appearance of the cylindrical surface's curved boundary since this is a display of the mesh, not the actual object. In these displays, the perspective transformation was turned on , which aids in conveying shape information.

The remaining photos (Figures 6.4-6) are SURFMAN displays of this work's contour/trim regions for the given analysis object. Note the smooth boundaries and changes in intensity which result from displaying the actual geometric entity. The mathematical definition of the contour/trim regions provide continuous normal vector information to the shading model. There are two light sources located in the upper corners of the screen (in front of the object). Figures 6.4 and 6.5 show two different views of the object; note the reflection of the light sources on the object. Figure 6.6 is another view and also shows the main control panel (buttons and knobs) for SURFMAN. Perspective was turned off for these figures. The display of the color key and FE data range values is provided through a diagnostic option of SURFMAN which allows the inclusion of an external subroutine.

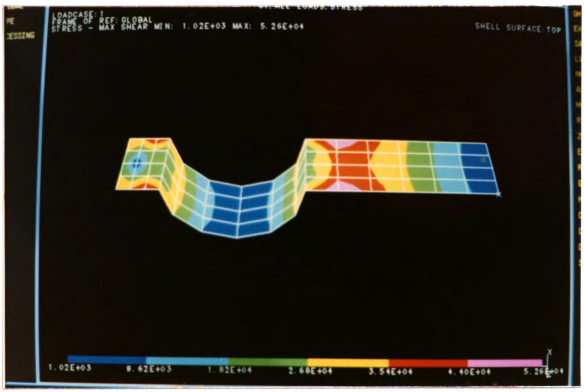


Figure 6.1 Contour band mesh display (I-DEAS) with element borders

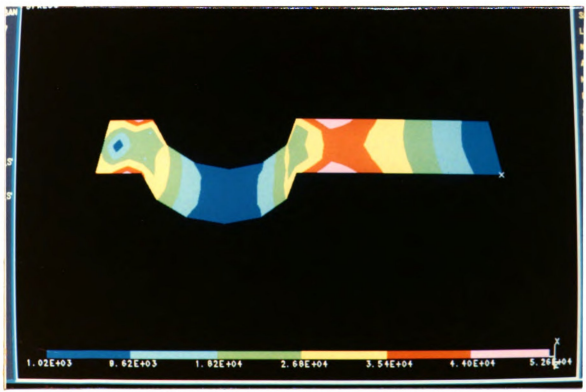


Figure 6.2 Continuous-colored contour bands (I-DEAS)

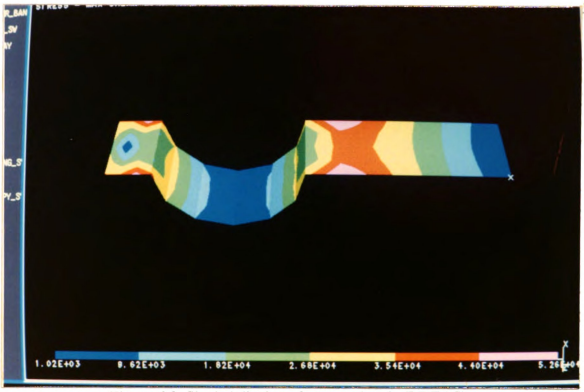


Figure 6.3 Shaded display of contour bands (I-DEAS)

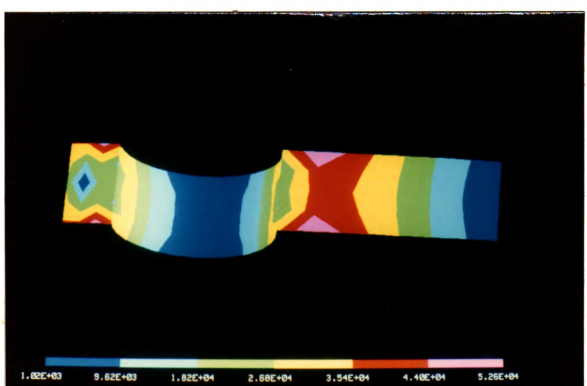


Figure 6.4 Shaded display of contour/trim regions - view 1

Figure 6.5 Contour/trim regions - view 2

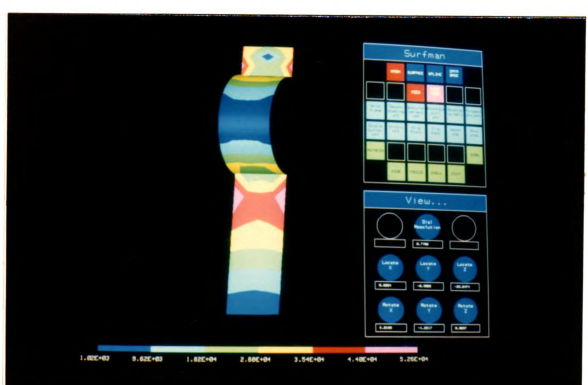


Figure 6.6 Contour/trim regions with SURFMAN control panel

6.2 Pros, Cons, and Comparisons

One of the major advantages of defining contour regions as geometric entities (that is, trimmed surfaces) is that the results are view independent. What this means is that once the contour regions are appropriately defined, the display input is set. The object can be rotated and viewed from all angles without any further calculations (with respect to calculating/interpolating contour information). In contrast, mesh displays interpolate contours in screen space coordinates, thus requiring that all contour information be recalculated if the view parameters are changed.

Of course, the most basic benefit of defining contour regions as trim surfaces is that the actual object is displayed along with the FE data. Thus, all visual geometric information (for example, exact boundaries or curvature) is provided as well as the desired analysis information. This brings up a very important and desirable feature of this work — the fact that the display mechanism for the object design phase is the same as that which is required for the display of FE results. In other words, only one display routine is required regardless of what stage the user/designer is at — object modeling or FE analysis. Thus the specialized hardware and firmware available for speedy object display is available for FE post-processing needs.

Another positive aspect of this particular work is its scope of applicability. Although it has been tailored to work with IGES geometry files and I-DEAS finite element result files, it could be easily altered to work with some other formats. Additionally, it is not restricted to FEA. Any numerical method, such as finite differences, that creates a discretized model of the analysis object is appropriate for displaying the results via this work's methods. These aspects make it appealing to work the localized input problem presented in this work, as opposed to the global problem.

The discussion in Section 6.3 relates which processes were forced upon this work due to the input information limitations and which are inherent to the solution. The solution thus far has been presented as a two stage process, but it can also profitably be viewed as three divisions. The first division concerns the integration of the FE mesh data and the geometry data (those processes described in Sections 4.1-4.7). The second division concerns integrating the user-specified contour display information (Sections 4.8-4.10) and the third division is the generation stage (Chapter 5). The processes in division one are the most compute-intensive of the work, but as just discussed, they could be easily simplified if incorporated into the meshing operation in a global approach. Important to the approach of this work is the fact that for a given object and FE results these division one operations are only done once. Therefore, in regards to a comparative study between displays of FE data on the geometry versus on the mesh, the more important operations are those which belong to divisions two and three.

In the processes that integrate the contour information with current FE/geometry data, the main difference between a geometry display and a mesh display is the handling of the geometric boundaries of the surfaces of the object. In a mesh, one edge is indistinguishable from another. But edges that are associated with geometric boundaries must be special-handled for contour displays on the object. All boundary elements are interpolated based on curve parameter values (t). Although this requires interpolation in only one variable, the contour point must be evaluated (u,v(t)) on its associated boundary curve. Additionally, there is the problem of "matching up" contour points that occur on multiple surfaces. To start the process, the full evaluation (x,y,z(u,v(t))) of the contour points on each surface is needed. However, there is a good chance that the points will match up naturally (for example, if the boundary is linear or if the curves involved have equivalent parametric flow). Also, as mentioned in Section 4.9, the iterative process used to perform the inverse

problem involved in "matching up", has excellent conditions. This suggests that the process would be a one step iteration and thus equivalent to evaluating the point and its derivatives once. Therefore, one of the major expenses (not present in mesh display) is the handling of boundary edges. This cost will depend on such things as the number of surfaces and the number of boundaries (which do not contribute to the workload for a mesh display) in addition to the number of contour points on the boundary (which does contribute but not to the same degree).

For the generation process, if the curve entities defined with parameter bounds (as discussed in Section 5.1) are used, then this process will impose some extra (that is, not involved in mesh display) bookkeeping time, but no extra computational time. The generation of contour curves will be equivalent computationally whether the curves are being generated in screen space (on the mesh) or parameter space (on the object). Thus the principal additional costs are the extra information needed to properly line up the curves to form the contour region boundary loops and the possible difficulties involved with interior contour curves (see Section 5.2.3).

These loops and the curves that form them are the only additional information needed to define the trim/contour regions. The data for the underlying untrimmed surface of each region is unchanged. Thus no new surface data is required, only new trimming curves. A tailored data format for the "bounded" segment curves just discussed could consist only of a pointer to the original trimming curve and the 2 parameter bound values. Also since each contour curve is used twice (in different directions), the curve data can be stored once but with a direction flag attached. However, the two data format suggestions made would also make it necessary to flag curves as either segments or contour curves (since the information stored for each is different). Overall though, the storage requirements for the data defining the trim/contour regions is relatively small.

6.3 Global Approach

It was mentioned in several places in this work, that certain operations would not be part of the procedure if the general problem discussed in this work was approached globally. Globally implies that one has control over all aspects of the process — for example, the information that is known and stored at mesh generation time. Although the calculation of the surface and curve parameters for the nodes is a one-time process, it is a compute-intensive process and could easily be avoided if the nodal information concerning parameter values were stored at mesh generation time. For all of the interior nodes (excluding nodes on seams or poles) there is a unique (u,v) known at mesh generation. It is also reasonable to assume that a general purpose mesher designed to create FE meshes on arbitrary NURBS surfaces would have to have some knowledge of seams to avoid creating redundant nodes. Thus, information such as which nodes are on seams should be readily available, as well as one of the two surface parameter pairs for these seam nodes. With respect to a boundary node, the curve parameter value must be known for at least one surface that it is on, and the knowledge of what nodes are on what curves and their order along these curves is definitely available. Therefore, with the basic information detailed above, the inverse problem is reduced to finding some of the curve parameters of boundary nodes. Additionally, the creation of the FE boundary loops can utilize important, facilitating information.

It would be feasible to compute all parametric information for nodes at mesh time. Since the parametric information for a node is not necessarily unique, storing this information is more intricate and requires more structure than the storing of the nodal Cartesian coordinates, but definitely outweighs the alternative (that is, the "backwards" solution). It is also feasible to completely determine the FE loop entity lists for nodes and elements during the meshing operation. This eliminates the

intensive search operation done on the boundary edge lists in order to determine the aforementioned loop entities. This leaves the following integration stage processes : a much simplified procedure for creating the FE boundary loops, determining contour points, and making sure multiple surface contour points match up.

In terms of the generation stage, the main process to be considered is the generation of the boundary segments. As mentioned in Section 5.1, there is an acceptable definition format for parametric spline curves that sets a parameter range for the actual portion of the curve that is desired. This provides a computation-free method for generating segments. Thus the display mechanism should be able to understand this type of format. Therefore, the remaining processes in the generation stage are the generation of the contour curves and determining their proper location in terms of boundary loops and contour regions.

6.4 Display of Contour Curves

The contour display type in this work was limited to continuous color-shaded contour regions as opposed to contour curves. This was due to the inability of the display mechanism (Starbase) to outline surfaces. Outlining is all that is needed since the contour curves are simply portions of outlines of the trimmed surfaces that represent the contour regions. If the contour/trim regions are ordered according to their data range (for example, from lowest to highest values) and outlined in this order, then the contour curves will be the color of the region with this contour value as its lower bound (the curves are drawn twice, but the "higher" color always overlays the lower). The original surfaces should be outlined in a non-contour color to cover up the misleading and incorrect colored segments that will exist on the actual surface boundaries. An excellent display for contour curves (that is totally feasible,

but not with the particular display library used for this work) would be to shade all the contour/trim regions in the color gray, outline the contour/trim regions in the appropriate color (as discussed earlier), and then outline the actual surfaces in gray. This would show a shaded image of the object along with the colored contour curves.

6.5 Examining FE Data in the Object's Interior

This work dealt with an object that was modeled/defined by mathematical surfaces that represent the object's exterior. As was mentioned in Chapter 3, the FE model of the object could consist of 3-dimensional elements whose faces lie on the exterior of the object. The methods described in this work provide a display of the FE data for these exterior faces. To look into the interior of the object, the following procedure could be attached to what has been developed for the surface display. The first step is to define a viewing plane, that is, a plane cutting through the object. The object's data that coincides with this plane (that is, a planar cross section) will be the additional data to be displayed. Effectively, the data plane is the intersection of the viewing plane and the object (most object modelers have the capabilities of determining this intersection object). This cross section can be defined as a trimmed NURBS surface. Alternatively, a Boolean difference with the half-space defined by the cutting plane can be calculated, yielding the boundary representation surface and boundary curves directly. The surface is planar and the trim curves come from the intersection of the plane with the surfaces of the object. Thus, the appropriate geometric input is available. The next problem is to create the FE data on the cross section. This requires that the intersection of the mesh and the plane be computed and the FE data values for these point be interpolated. The next step is to create elements for the just-determined FE data points by joining together those points which came from the same 3-D element. Thus, an FE mesh is known for the surface.

Therefore, all the input required for the surface display developed in this work is available.

6.6 Smooth Contour Curves

The contour curves generated in this work are linear B-splines and produce curves with slope discontinuities. One way to improve the jagged appearance of the contour curves is to generate more points along the contour curve, but still producing a linear segmented curve (that is, the segments will be smaller and thus the curve will look smoother). The best approach to this problem is to subdivide the elements as discussed in Section 2.2 (see Figure 2.4). This would involve dropping the new points on the element onto the surface, which is an expensive process.

Another method is to actually produce smooth contour curves in parameter space which will also be smooth on a surface of the object (given that the surface is smooth). There are many methods for producing smooth curves through a set of discrete points (in this case the contour points found for a contour curve). The principal problem that is encountered in this process is controlling the curves between the points to prevent them from intersecting each other. Any smoothing routine developed for display on the FE mesh (such as the one used by NISA) will be applicable for contouring in the parameter space of a surface, as would be needed by this work.

6.7 Summary

This work has presented a new approach to the display of FE contour data. This "newness" can be broken down into two aspects. First, the display of contour

information on the actual geometry is currently not a common practice. Secondly and more importantly, the definition of the contour bands (regions) as trimmed surface entities is a unique approach to the given display problem. Overall it seems that the trimmed surface representation of contour information is a viable, exciting, and attractive proposition, especially when incorporated into the global design and analysis process.

APPENDICES

Appendix A

Point/Curve Calculations

Given : Surface definition : $x,y,z(u,v)$

Curve definition : $u,v(t)$

Curve derivatives : $dx/dt = \delta x/\delta u \bullet du/dt + \delta x/\delta v \bullet dv/dt$

$dy/dt = \delta y/\delta u \bullet du/dt + \delta y/\delta v \bullet dv/dt$

$dz/dt = \delta z/\delta u \bullet du/dt + \delta z/\delta v \bullet dv/dt$

Problem : Determine the curve parameter value t such that $x,y,z(u,v(t)) = x_d,y_d,z_d$.

Solution : Iterative, numerical technique (extension of point/surface calculation [13]).

Given a computed or initial guess value for t --

Taylor series - first order (Newton's method) : $x_d - x(t) = delt \bullet dx/dt$

$y_d - y(t) = delt \bullet dy/dt$

$z_d - z(t) = delt \bullet dz/dt$

Least squares solution - 3 equations, 1 unknown (delt) :

$$delt = \frac{dx/dt\ (x_d - x) + dy/dt\ (y_d - y) + dz/dt\ (z_d - z)}{dx/dt\ dx/dt + dy/dt\ dy/dt + dz/dt\ dz/dt}$$

Compute new value : $t = t + delt$

Convergence check : Is $\left| x,y,z(u,v(t)) - x_d,y_d,z_d \right| \leq \varepsilon$?

Appendix B

Point/Surface Calculations

Given : Surface definition : x,y,z(u,v)

Problem : Determine the surface parameters (u,v) such that $x,y,z(u,v) = x_d,y_d,z_d$.

Solution : Iterative, numerical technique [13].

Given a computed or initial guess value for (u,v) --

Taylor series - first order (Newton's method) :

$$x_d - x(u,v) = delu \bullet \delta x/\delta u + delv \bullet \delta x/\delta v$$
$$y_d - y(u,v) = delu \bullet \delta y/\delta u + delv \bullet \delta y/\delta v$$
$$z_d - z(u,v) = delu \bullet \delta z/\delta u + delv \bullet \delta z/\delta v$$

Least squares solution - 3 equations, 2 unknowns (delu, delv) :

$$A = \begin{pmatrix} \delta x/\delta u & \delta x/\delta v \\ \delta y/\delta u & \delta y/\delta v \\ \delta z/\delta u & \delta z/\delta v \end{pmatrix}, b = \begin{pmatrix} x_d - x(u,v) \\ y_d - y(u,v) \\ z_d - z(u,v) \end{pmatrix} \qquad \begin{pmatrix} delu \\ delv \end{pmatrix} = (A A^T) A^{-1} b^T$$

Compute new values : $u = u + delu$, $v = v + delv$

Convergence check : Is $\left| x,y,z(u,v) - x_d,y_d,z_d \right| \leq \varepsilon$?

Appendix C

Polygon Direction Calculations

Given : Ordered set of vertices that define a closed polygon in 2-D space.

Problem : Determine if the prescribed order of the vertices is clockwise or counterclockwise.

Solution : Find the vertex **m** with the minimum v coordinate. Label the vertex before **m** (in terms of the prescribed orientation) as **a** and the vertex after **m** as **b**.

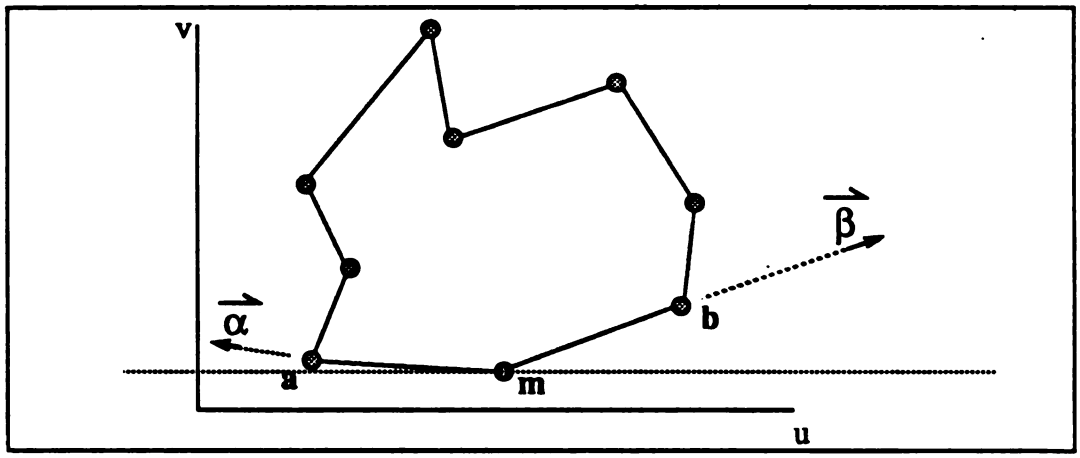


Figure C.1 Polygon orientation - ccw example

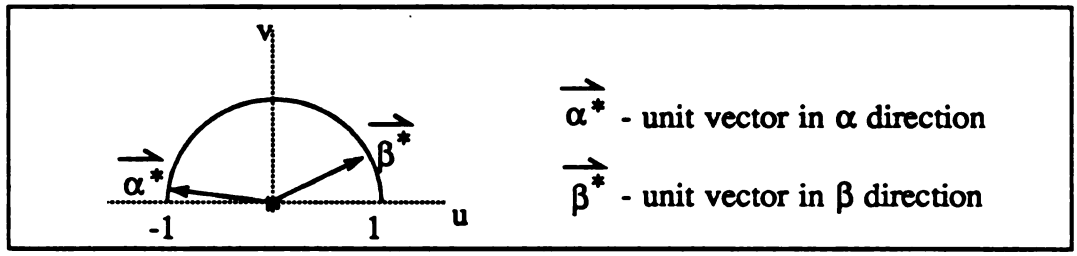


Figure C.2 Unit vector test

Thus, the following relationship between the u components of the unit vectors above implies a counterclockwise orientation.

$$\frac{u_a - u_m}{\sqrt{(u_a - u_m)^2 + (v_a - v_m)^2}} < \frac{u_b - u_m}{\sqrt{(u_b - u_m)^2 + (v_b - v_m)^2}}$$

Appendix D

Polygon/Polygon Calculations

Given : Two closed polygons (defined by an ordered set of vertices) in 2-D space. One is called the current polygon and the other the outer region polygon.

Problem : Determine if the current polygon is inside the outer region polygon.

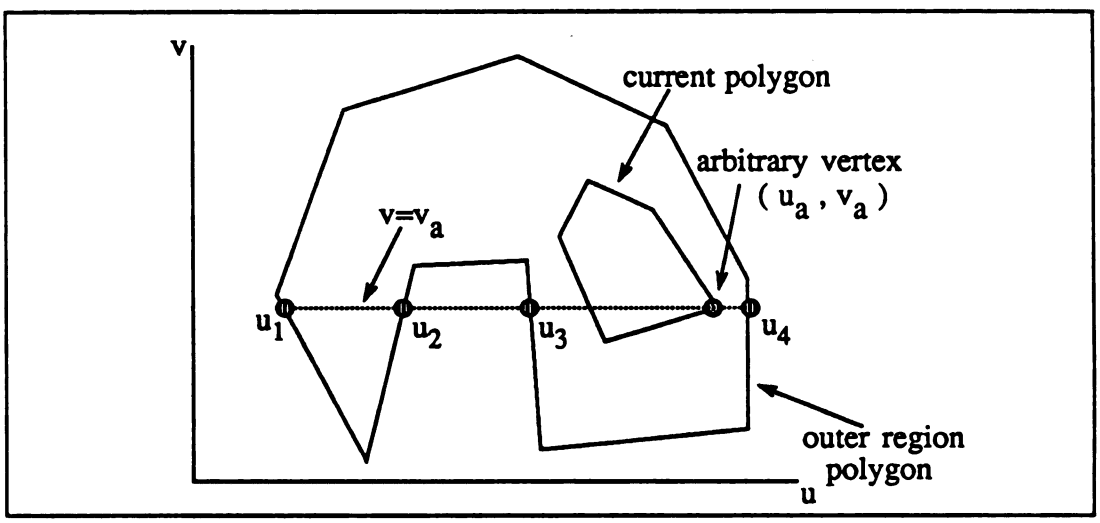


Figure D.1 Inside polygon test

Solution [31] :

1) Choose an arbitrary vertex on the current polygon - (u_a , v_a).

2) Determine the intersections between the line $v=v_a$ and the outer region polygon (if an intersection occurs on a vertex which is a local extremum, ignore it.)

3) Order the intersection points with regards to the u coordinate.

4) Pair the points (for example, (u_1 , u_2) and (u_3 , u_4)).

5) If u_a falls within one of the pairs (for example, $u_a \in (u_3 , u_4)$ in Figure D.1), then the current polygon is inside the outer region polygon.

Appendix E

Procedural Outline

I. Integrate FE mesh and results data with geometry data
   A. Read data
      1. Geometry file
         {curve-to-boundary loop-to-surface}
      2. FE results file
         {element-to-node, node-to-Cartesian coordinate, element-to-mesh area, mesh area-to-surface, FE data on nodes and elements}
      3. User-provided contour data
         {number of contours, total range}
   B. Organize and manipulate data
      1. Calculate nodal and element center FE data
      2. Determine element-to-surface data
      3. Determine list of scalars that are contour values
   C. Create edges
      1. Sort node pairs
      2. Assign edge labels
      3. Set up edge data
         {edge-to-node, element-to-edge, edge-to-element, type}
      4. Determine boundary edges
   D. Create FE boundary loops
      1. Arrange each surface's boundary edges via nodes, i.e., line up head to tail
      2. Make oriented list of nodes, edges, and elements
   E. Correlate geometric boundary loops with FE loops
      1. Determine curve parameter value(s) of boundary nodes
      2. Determine geometric curve labels of edge endpoints
      3. Insert seams in FE loops as encountered
         a. Type seam edges
         b. Determine FE entities along seam
         c. Create seam curve if seam not included in geometric loop
   F. Calculate surface parameter values for interior nodes

II. Integrate contour information with FE/geometry data
   A. Determine boundary edge contour points
   B. Match up multiple surface points
   C. Determine occurrence of contour points on interior edges

III. Generate contour/trim regions (one surface at a time)
  A. Create boundary segment curves for each boundary loop
    1. Insert knots at contour points
    2. Grab control points and other info to describe each segment
  B. Process segment loops to generate regions attached
    1. Generate trim region (outer) boundaries that contain segments on original surface outer boundary and seams
      a. Determine FE data range color
      b. Generate contour curves encountered
        i. Get starting edge and element
        ii. Find exit edge (check type for boundary)
        iii. Determine adjacent element
        iv. Go to ii
      c. Store curve data and curve order for loop
    2. Generate trim region boundaries that contain segments on original surface inner boundaries (if exist)
      a. Determine FE data range color
      b. Generate contour curves encountered
      c. Store curve data and curve order for loop
      d. Determine cw/ccw if no segments have been used (set aside cw data)
  C. Generate and place interior contour curves/regions
    1. Look for edges with used/unused contour point combinations
      a. Determine region to add contour to as an inner boundary
      b. Determine color of interior region
      c. Walk contour watching for beginning edge to reappear
    2. Generate all remaining contour curves and place
      a. Determine cw/ccw
      b. Determine color of interior region
      c. Determine color of "add to" region
      d. Determine bounding box (& its area) for each curve
      e. Process curves (one at a time) from largest box area to smallest
        i. Sort existing regions for those with proper color
        ii. Determining bounding box for these regions
        iii. Throw out those with boxes that do not contain curve box
        iv. Use polygon/polygon calculation to find correct region (if needed)
  D. Place any cw loop generated during segment loop processing (i-iv above)

IV. Write trim region data to a data file

Appendix F

Excerpt from Bounded Surface Proposal for IGES 5.0

The following is from an IGES Request for Change which details the proposed entities of bounded surface and boundary for defining trimmed surfaces [32].

Parametric surface - S(u,v) - eligibility requirements

1) The untrimmed domain of S(u,v) is a rectangle, D, consisting of those points (u,v) such that $a \leq u \leq b$ and $c \leq v \leq d$ for given constants a,b,c and d with $a<b$ and $c<d$. The curves u=a, u=b, v=c, and v=d are the boundary curves of the parameter space.
2) The mapping S = S(u,v) = (x(u,v) , y(u,v) , z(u,v)) is defined for each ordered pair (u,v) in D.
3) The mapping S is one-to-one in the interior, but necessarily on the boundary, of D.
4) The mapping S has continuous normal vectors (except at poles).

Bounded Surface Entity

1) TYPE - boundary curves are either represented only by model space curves or by model and parameter space curves
2) SPTR - pointer to untrimmed surface directory
3) BDCNT - number of boundaries
4) BDPT - pointer to boundary directory

Parameter space boundary requirements

1) Model space image of curves in boundary line up head to tail & form a closed path.
2) Curve must be oriented such that the cross product of the positive surface normal and the tangent vector to the curve's image is a vector pointing toward the trimmed surface region.

Boundary Entity

1) TYPE
2) SPTR
3) N - number of curves in the boundary

    CRVPT - pointer to curve directory of model curve

    SENSE - correct orientation of tangent for model space curve

    PSCNT - number of parameter space curves that make up model curve

        PSCPT - pointer to parameter space curve directory

LIST OF REFERENCES

List of References

1. P. Kinnucan, Boom Times Come to Finite-Element Analysis, *The S. Klein Computer Graphics Review*, Spring 1987, 29-38.

2. H. C. Martin and G. F. Carey, *Introduction to Finite Element Analysis*, McGraw-Hill, New York, 1973.

3. D. Dietrich, FE Analysis For the Body, *Mechanical Engineering*, March 1987, 48-52..

4. I-DEAS, Supertab Engineering Analysis, Pre- and Post- Processing, User Guide, I-DEAS Level 3.

5. R. B. Jerard, R. L. Drysdale, K. Hauck, B. Schaudt, and J. Magewick, Methods for Detecting Errors in Numerically Controlled Machining of Sculptured Surfaces, *IEEE Computer Graphics and Applications*, Vol. 9, No. 1, Jan. 1989, 26-39.

6. *Initial Graphics Exchange Specification (IGES), Version 4.0*, U.S. Department of Commerce, National Bureau of Standards, June, 1988.

7. L. Piegl and W. Tiller, Curve and Surface Constructions using Rational B-splines, *Computer-aided Design*, Vol. 19, No. 9, Nov. 1987, 485-498.

8. E. Cohen, T. Lyche and R. Riesenfeld, Discrete B-Splines and Subdivision Techiques in Computer-Aided Geometric Design and Computer Graphics, *Computer Graphics and Image Processing* **14**, 1980, 87-111.

9. D. S. Papageorgiou and K. Beier, Interactive Computer Graphics Program for Generating B-Spline Curves, in *Proceedings, SAE/ESD International Computer Graphics Conference, April 1987*, 21 -26.

10. J. G. Fiasconaro and D. S. Maitland, Non-Uniform Rational B-Splines, in *Proceedings, SAE/ESD International Computer Graphics Conference, April 1987*, 47-52.

11. B. A. Barsky, A Description and Evaluation of Various 3-D Models, *IEEE Computer Graphics and Applications*, Jan. 1984, 38-51.

12. H. Huitric and M. Nahas, B-Spline Surfaces: A Tool for Computer Painting, *IEEE Computer Graphics and Applications*, March 1985, 39-47.

13. I. D. Faux and M. J. Pratt, *Computational Geometry for Design and Manufacture*, John Wiley & Sons, 1985.

14. *ANSYS Manual*, Pre- and Post Processing, Version 4.4, Swanson Analysis Systems, Inc.