Engineering with Computers 8, 179–196 (1992)

Engineering with Computers

An Automatic Coarse and Fine Surface Mesh Generation Scheme Based on Medial Axis Transform: Part II Implementation

H. Nebi Gürsoy
Intergraph Corporation, Analysis Applications-Development, Huntsville, AL, USA

Nicholas M. Patrikalakis
Massachusetts Institute of Technology, Department of Ocean Engineering, Design Laboratory, Cambridge, MA, USA

Abstract. In this paper, we present implementation aspects of a surface finite element (FE) meshing algorithm described in Part I (this volume) [1]. This meshing scheme is based on the medial axis transform (MAT) [2] to interrogate shape and to subdivide it into topologically simple subdomains. The algorithm can be effectively used to create coarse discretization and fine triangular surface meshes. We describe our techniques and methodology used in the implementation of the meshing and MAT algorithms. We also present some running times of our experimental system. We finally report the results we have obtained from several design and analysis applications which include adaptive surface approximations using triangular facets, and adaptive *h*- and *p*-adaptive finite element analysis (FEA) of plane stress problems. These studies demonstrate the potential applicability of our techniques in computer aided design and analysis.

1 Introduction

This work deals with implementations of a finite element (FE) surface mesh generation scheme and our algorithm for medial axis transform (MAT) [2,3] computations on surfaces. More detailed descriptions of these algorithms and discussions of their theoretical aspects and related techniques can be found in Part I (this volume) [1], and also in [4–7].

For the sake of convenience, we briefly review some basic aspects of MAT, which are introduced in [1]. MAT has been proposed as a shape description method by Blum [2,3]. MAT can represent two-dimensional or three-dimensional shapes in terms of its medial axis (MA) (or skeleton or symmetric axis) and associated radius function (RF). The MA of a two-dimensional shape is a set of points each of which has at least two equidistant nearest points on the boundary of the shape. The limit points of this set are normally included in this definition of the MA. Radii of the maximal inscribed circles centered on the MA are defined by the associated radius function (RF) of the MAT. The MA of a two-dimensional shape is composed of continuous curved segments. Those segments are called MA branches. If the boundary of a shape is defined by circular arcs and straight line segments, it can be shown that MA branches are conic sections [3,4]. Points where adjacent MA branches intersect each other are called branch points. In our two-dimensional MAT algorithm [1], we determine an analytic representation of all MA branches, their end points (branch points) and associated RF of each MA branch. These concepts can be analogously extended to three-dimensional shapes [3,7,8]. In three dimensions, MA representations are more complex and MA branches are surface patches (i.e, MA surfaces), curve segments (i.e., MA edges) and discrete vertices.

The FE surface meshing scheme we developed has two stages: shape interrogation and area meshing [1]. First, this scheme uses MAT as an automatic shape interrogation method to extract topologically simple subregions and their length scales from a given complex domain. Our MAT algorithm allows us to efficiently carry out this shape interrogation process. Such an initial shape decomposition can be considered as a coarse FE mesh. Next, those simple subregions are triangulated to generate a fine FE mesh. Thus, a mesh capturing important geometric characteristics of a given domain can be created by our mesh generation scheme in an automated manner.

In this article, we outline our methodology and report efficient techniques used in the development and implementation stages of our prototype system. The second section presents basic implementation aspects and the data structures used in conjunction with our two-step FE surface meshing and MAT algorithms. The third section deals with applica-

Offprint requests: Nicholas M. Patrikalakis, Massachusetts Institute of Technology, Department of Ocean Engineering, Design Laboratory, Cambridge, MA 02139, USA.

tions of our technique to several design and analysis processes. These applications include adaptive faceted approximations of trimmed free-form surfaces and adaptive plane stress analysis using h- and p-convergence FEA methods. Finally, the fourth section summarizes this work and points out related topics for future research.

2 Implementation Aspects of Algorithms

We have implemented our MAT and FE meshing algorithms in C under the UNIX operating system. Our prototype system has been developed on a DEC Vax Station II GPX and a Silicon Graphics IRIS 3030 workstation has been used to interactively run the system. In our implementation, the surface definition and the geometric representation of boundary contours (loops) are the main input of the MAT and subsequent meshing computations. We have established specific data types and the relationships among them using a data abstraction methodology. The next two sections discuss the data structures used and computational aspects of our implementation.

2.1 *Implementation of the Medial Axis Transform*

2.1.1 *Data structures for the medial axis transform*

In general, we have two types of boundary contours, one external boundary contour and contours of internal loops. A typical boundary contour is composed of three different types of boundary elements: straight line segment, circular arc (either concave or convex) and reentrant vertex (i.e., a degenerate case of a concave arc). In our implementation, every internal loop is connected to the external contour along a *virtual cut* using two artifical segments. These artificial segments are not used in the branch point computation and have no contribution to the medial axis of a shape. Their function is to indicate the presence of internal loops during traversal of elements on the boundary in computations of branch points. Thus, typical boundary data of a multiply connected nonconvex shape are composed of straight line, circular arc and artificial segments. An abstract data type segment is defined in terms of structure constructs of the C language. These boundary elements are ordered in a clockwise sequence so that the interior of the region lies to the right. With respect to the clockwise sequence of the boundary segments, we notice that the center of a convex circular arc lies to the right of the contour boundary and the center of a concave circular arc to the left. As a result, when a boundary contour is offset inward, concave arcs expand and convex arcs shrink.

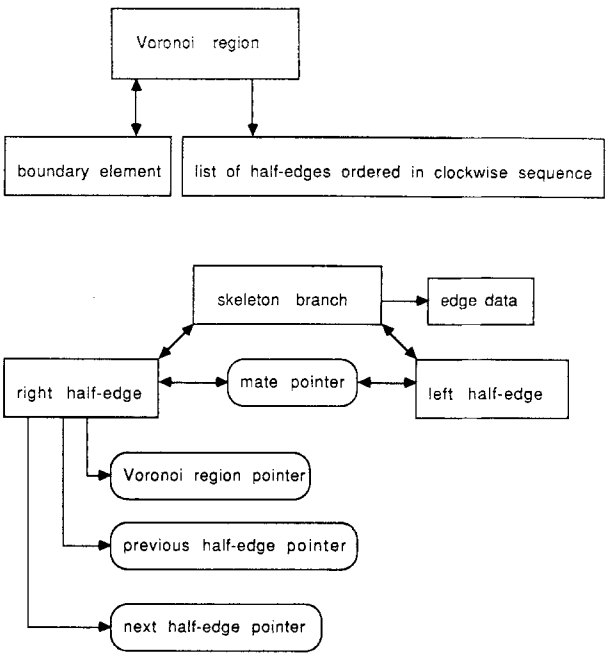
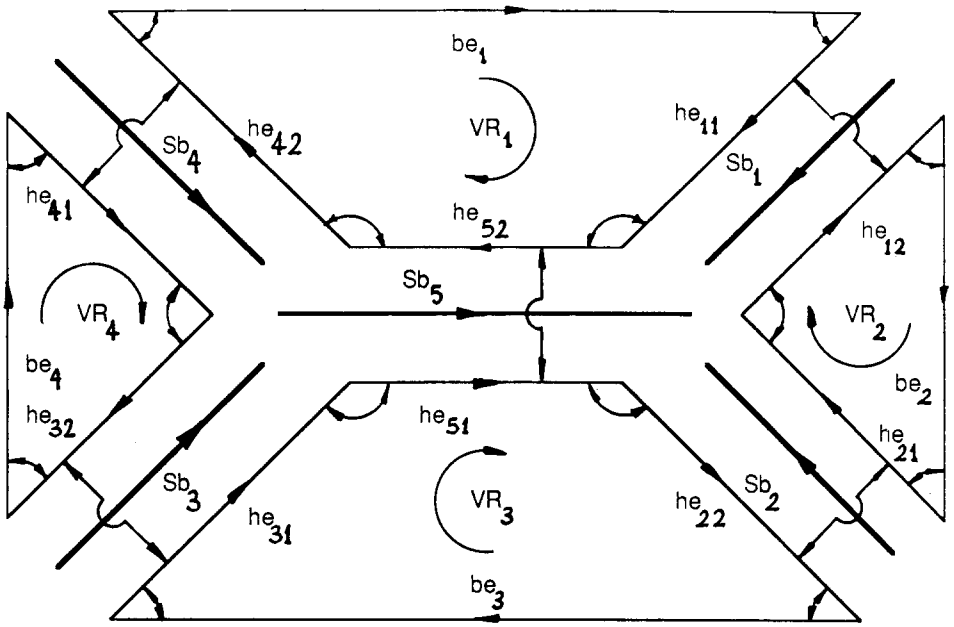


Fig. 1. Graphical representation of the data structures.

Doubly linked *lists* are used as the main data structures for the representation of boundary contours. The boundary elements of a contour are stored as the items of a doubly linked list which also has pointers to store data of MA branch points.

The MA branches and Voronoi edges decompose a shape into a set of subregions, generally called Voronoi regions [1]. The regions are bounded by boundary elements of the initial contours and associated MA branches and Voronoi edges. Here we recall that a boundary element is associated with a distinct subregion. In the computation of the MAT, we determine not only descriptions of the regions but also the *adjacency relationships* among themselves. For this purpose, we have developed a variant of the *face-edge* data structure proposed by Weiler [9] for two-manifold geometric models. In this representation, the boundary of every region is composed of a boundary element and a chain of MA branches. Every MA branch has two half edges associated with the two adjacent Voronoi regions. MA branches are also represented using structure constructs. Figure 1 illustrates the relationships among these objects.



Sb : skeleton branch

VR : Voronoi region

be_i : boundary element of Voronoi region VR_i

he_{i1} : right half-edge of skeleton branch Sb_i

he_{i2} : left half-edge of skeleton branch Sb_i

Fig. 2. Objects used in boundary representation.

Several dynamically allocated data structures are also employed to carry out the computation. A *queue* is used to store the contours (loops) to be processed. From this queue, a contour is removed and processed to compute "effective MA branch points" and the associated offset distance as discussed in [1]. The computed branch points of a contour are stored using a list. Then this list is processed to determine "effective offset distance" and branch point(s). If final branch points are found, the computation of the MAT of the contour stops. If branch points are not final or the interior area of the contour is not nil, the contour is offset. In that case, if the contour gives rise to only intermediate branch points, there will be one resulting offset contour. But if it involves initial branch points as well, the offset of the contour may be split at the initial branch points. For example, the boundary contour of a simply connected polygon with n initial branch points, will be split into $n + 1$ offset contours. In the case of a multiply connected polygon, initial branch points may be generated by the interactions among internal loops and/or internal loops and the external contour. Then splitting or merging of contours occurs depending on the geometry of the polygon. This situation is resolved by traversing the boundary, and discovering the tangent adjacencies created at initial branch points. To represent such adjacency relationships, boundary elements have pointers to another boundary element which is the tangent element at an initial branch point.

Once a contour is offset, the newly generated offset contours are put into the queue to be processed. The branch points have pointers to the boundary segments that generate these points. The branch points also denote the end points of the MA branches. In Fig. 2, the disassembled MA of a simple polygon illustrates the data objects used and adjacency relationships among them.

2.1.2 *Computational aspects*

There are several numerical computation stages involved in our MAT processes. First, a two-dimensional shape bounded by a set of B-spline curves

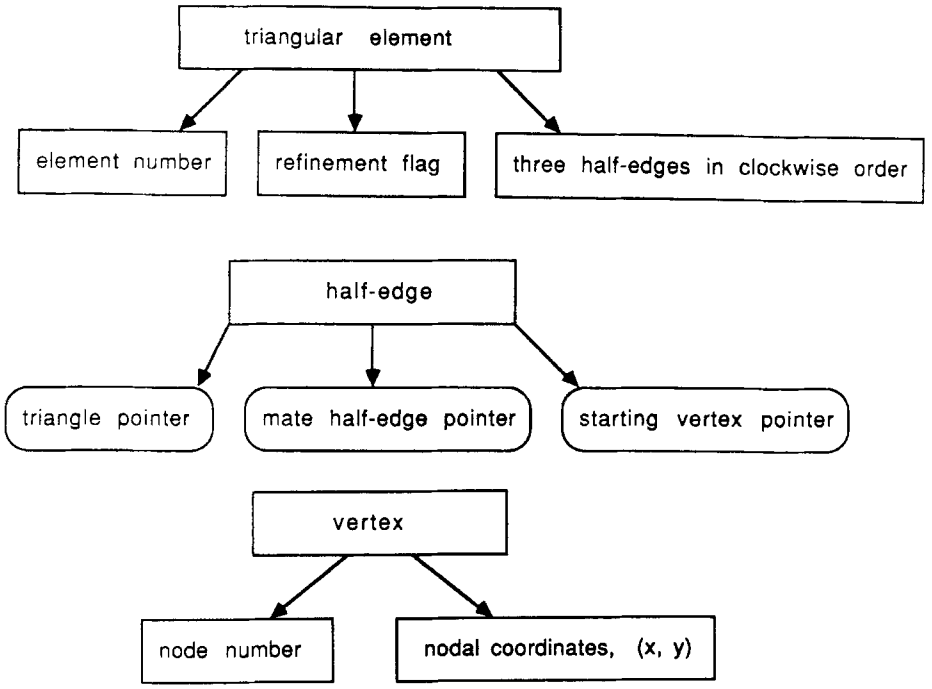


Fig. 3. Relationships between objects used in the mesh generator.

needs to be approximated. A boundary curve is approximated in terms of vertices, straight lines, and circular arc segments using the algorithm based on [10] and summarized in [7]. Our implementation is numerically stable and efficient and can handle a wide variety of parametric curves including high order nonuniform rational B-splines (NURBS).

In a MAT process, we can formulate the computation of MA branch points as an intersection problem between two conic sections [7]. This problem can be reduced to the solution of a polynomial of at most degree four using elimination methods [11]. We use a standard root solver [12] to determine the four potential roots of this polynomial.

In our implementation, the MAT computation is carried out in double precision involving 16 decimal digit arithmetic. Computations of MA branch points, values of associated RF and effective offset distances involve a tolerance for numerical comparisons. We use a relative tolerance of 0.5×10^{-6} with respect to the maximum dimension of the object as a default value in our prototype system.

2.2 *Implementation of the Meshing Algorithm Based on the Medial Axis Transform*

2.2.1 *Data structures for surface triangulation*

Our mesh generation scheme allows mesh smoothing and adaptive local mesh refinement. Efficient implementation of operations requires that adjacency information among triangular elements be available explicitly in the data structures. In our implementation, we have adopted a B-Rep scheme to represent the resulting FE mesh. In this representation, an individual triangle is treated as a topological element "face." We use a variant of the face-edge data structure [9]. Figure 3 illustrates the structures used in our implementation. Adjacency relationships between adjacent triangles are explicitly represented. This information allows us to implement very efficient mesh smoothing and local refinement processes. We use a dynamically allocated doubly linked list structure to store triangular elements.

2.2.2 *Mesh smoothing*

After the mesh generation process, the resulting FE mesh is smoothed (relaxed) to improve the shape characteristics of individual triangular elements, if this is required. A simple coordinate averaging process has been found to lead to efficient smoothing. Our mesh smoothing process is similar to the methods of Cavendish [13] and Chae [14]. In our implementation, coordinates of interior nodes are modified by averaging coordinates of the node and those of all surrounding nodes. In this iterative process, before a new iteration starts, coordinates of all interior nodes are updated using new coordinates com-

puted in the previous step. This iterative process stops either after a certain number of iterations or if the maximum value of displacements of all interior nodal points at an iteration becomes less than a prescribed percentage of the maximum value of displacements of the previous iteration. For a particular point P, the coordinates at iteration $i + 1$ can be determined from the following relation

$$P^{(i+1)} = \tfrac{1}{2}[P^{(i)} + \frac{1}{m}\sum_{j=1}^{m} Q_j^{(i)}] \quad (1)$$

where i denotes the iteration step and Q_j are coordinate vectors of m points adjacent to the point P. This adjacency information is denoted as the vertex–vertex adjacency relationship $P\{Q_j\}$ in B-Rep schema [9].

Although this smoothing process is efficient it has one disadvantage. If a mesh contains an extreme nonconvex boundary (such as reentrant corners), this smoothing technique may destroy the topology of the mesh by moving edges of elements in such areas to the outside of the actual region. Therefore, a robust implementation of this smoothing process requires that elements adjacent to re-entrant corners be identified and not smoothed, or processed separately. An example of mesh smoothing is illustrated in Fig. 6.

2.2.3 *Local mesh refinement*

One of the objectives of our implementation is to create a compatible mesh when a local mesh refinement process is carried out. In a compatible mesh, degrees of freedom associated with finite element nodes are common to all adjacent elements. For local adaptive mesh refinement, we use a bisection technique. In this approach, a triangle is split into two halves across its longest edge. Our refinement approach is similar to the mesh refinement method proposed by Rivara [15]. However, our mesh representation methodology and refinement process, which are based on a B-Rep scheme, are different from Rivara's method.

When a triangle is split into two halves across its longest edge and if a second triangular element adjacent to the split edge exists, the second element must also be processed since the middle node inserted on the split edge is not a common vertex. Therefore, this refinement process has, to some extent, a propagative nature. Numerous examples, chosen for their complexity and diversity, led us to the conclusion that this mesh refinement approach, in general, exhibits a local refinement character and does not effect the whole domain being meshed. Figure 4 illustrates two possible cases and operators involved in this mesh refinement scheme.

Case A: This case is either the starting or terminal point of the refinement process. When the refinement process starts, it first splits a triangle along its longest edge by introducing an incompatible node. In the next step, the triangular element, which is adjacent to the edge with the incompatible node, is determined and refined. If this edge is the longest of the next triangle to be processed, the refinement terminates after the next triangle is split into two triangles. If the edge containing the incompatible node is on the boundary of the mesh, the refinement also terminates. If the edge of the next triangle is a shorter one, the triangle is split into three triangles as described in the following case.

Case B: This case is, in general, an intermediate refinement step. It can be a terminal step only if the longest edge of the triangle is on the boundary of the mesh. Given a triangle with an incompatible node on one of its shorter edges, first the triangle is split into two triangles by introducing a new incompatible node on its longest edge. Then identifying the new triangle which contains the first incompatible node, this new triangle is split into two triangles by connecting the two incompatible nodes. Thus three new triangles are generated in total. In the next step, the triangle adjacent to the edge containing the new incompatible node is determined and processed similarly.

Since our representation is based on a B-Rep scheme, we can make an analogy between the refinement operators and standard Euler operators modifying a manifold topology in a boundary model. The process which splits a triangle is similar to a "make-vertex-edge-face" type Euler operator which involves creation of a new vertex, edge, and face [16]. Thus an advantage of our mesh refinement procedure is that it can easily be implemented in a boundary based solid modeling system if appropriate "low level" Euler operators are provided.

During the mesh refinement process, the doubly linked list structure is manipulated in such a way that when a triangle is refined it is removed from the list and the newly generated triangles are inserted at the end of the list. Figure 5 illustrates our list management approach in the mesh refinement process. This approach is computationally efficient and involves linear computation time with respect to the number of newly created elements.

Basic aspects of our list manipulation process are discussed here. In our implementation, the list has pointers to its *head* and *tail* elements and also a pointer is used to indicate current position on the

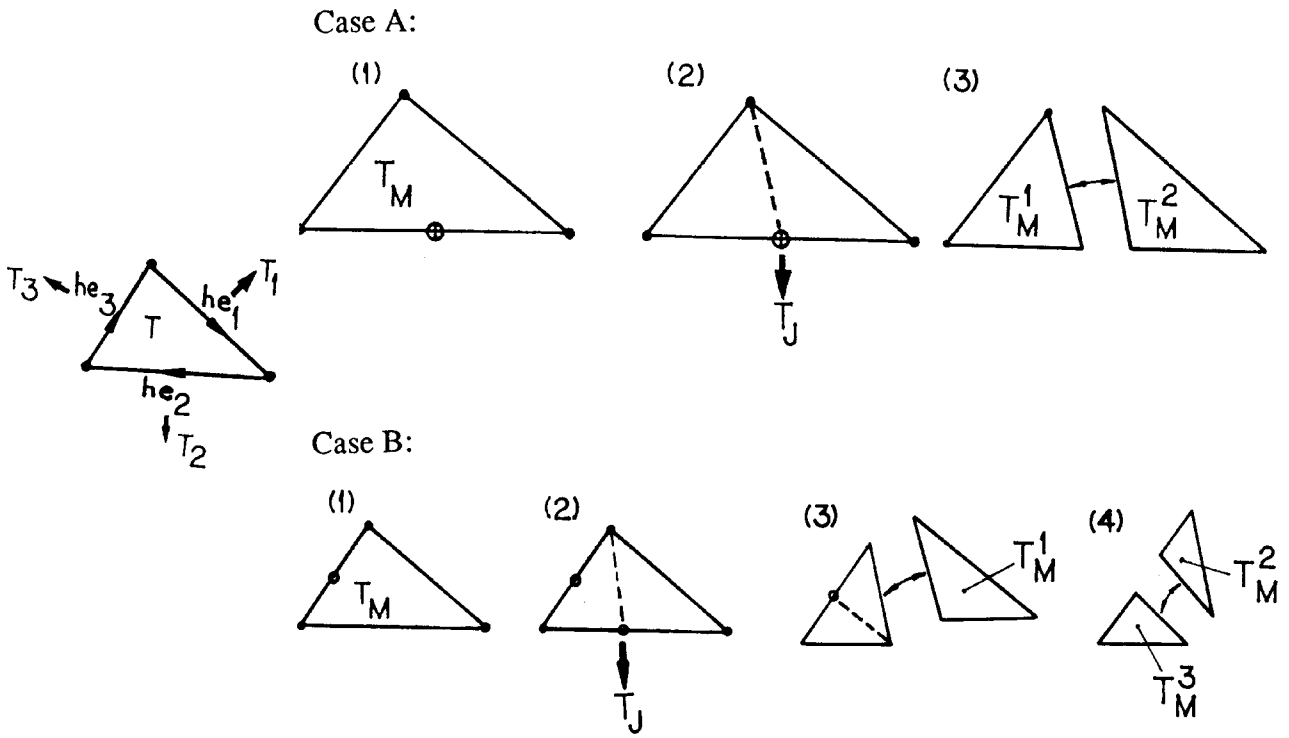


Fig. 4. Triangle splitting operators.

list. The current element pointer (CEP) indicates the element to be refined. We need one additional pointer which points to the previous element on the list and is called the previous element pointer (PEP), (see Fig. 5). Suppose we want to refine the element with offset m from the head of the list. Thus CEP points to triangle T_m and PEP points to triangle T_{m-1}. During the mesh refinement, two different operators are used to handle the two cases (i.e., cases A and B shown in Fig. 4).

- *Double Triangle Splitting* (*Case A*): T_m is removed from the list and two newly created triangles are added to the end of the list. Using the adjacency information of the edge containing the incompatible node, CEP is set. If it is a null pointer, this indicates the terminal case, namely that the edge with the incompatible node is on the boundary. In this case, the refinement propagation terminates, and CEP is reset to point to a triangle which is the element next to triangle pointed by PEP (i.e., triangle T_{m+1} as shown in Fig. 5). Otherwise, CEP points to the triangle which was topologically adjacent to the T_m along its longest edge. This triangle can be at any position in the list.
- *Triple Triangle Splitting* (*Case B*): Suppose that at the beginning triangle T_m has an incompatible node on one of its shorter edges. In this case, T_m is split and three new triangles are added into the end of the list. The adjacency relationship of the longest edge of T_m is employed to identify the next triangle. CEP is set to point to this element. If CEP is a null pointer, this indicates that the longest edge is on the boundary and, therefore, the refinement process terminates for triangle T_m. In that case, CEP now points to a triangle which is the element next to the triangle pointed by PEP (i.e., triangle T_{m+1} as shown in Fig. 5). Otherwise, the new triangle indicated by CEP is identified. This triangle can be at any position in the list. The triangle indicated by CEP is split in a similar manner.

In this list manipulation strategy, we always keep track of an initial part of the list which is already processed. This initial part of the list is indicated by the head and PEP of the list. In this way, we do not have to repeatedly traverse the list structure in order to determine the next element to be processed during the course of a mesh refinement.

2.3 *A Test Case for Running Time Results and Meshing Examples*

For a numerical experiment, we have used a simply connected region and generated four meshes with increasing element densities. Figure 6 illustrates the

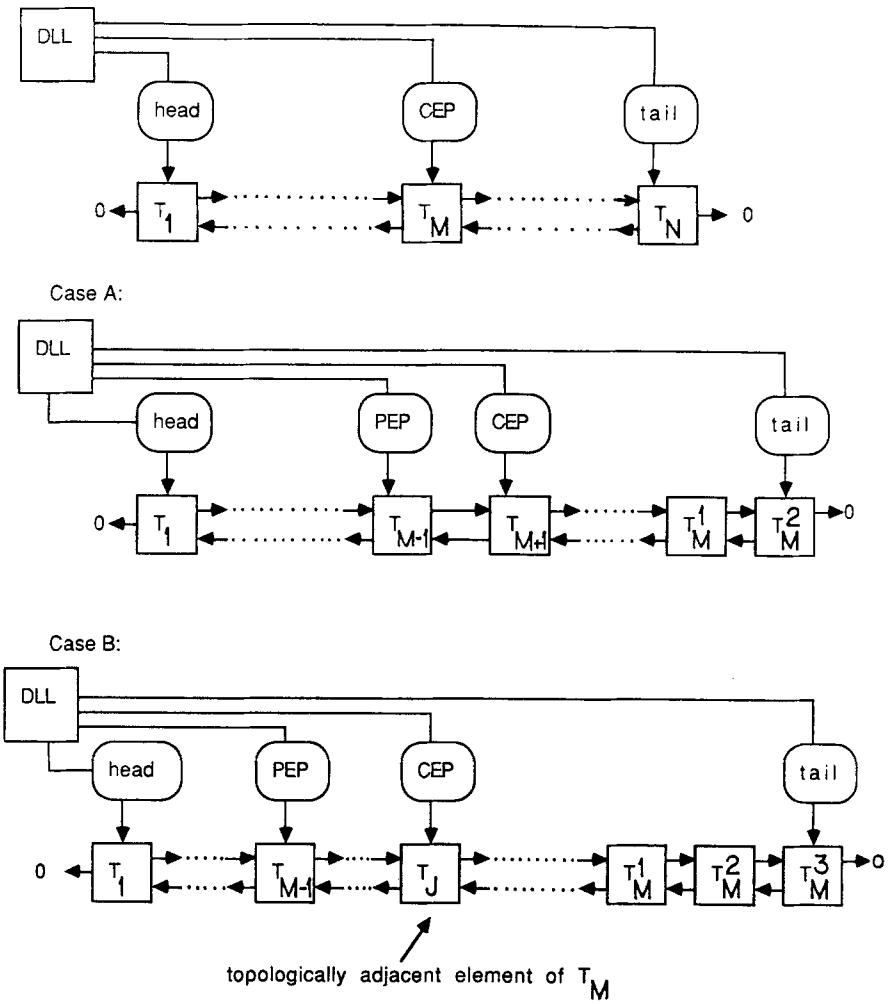


Fig. 5. Doubly linked list manipulation during the mesh refinement process.

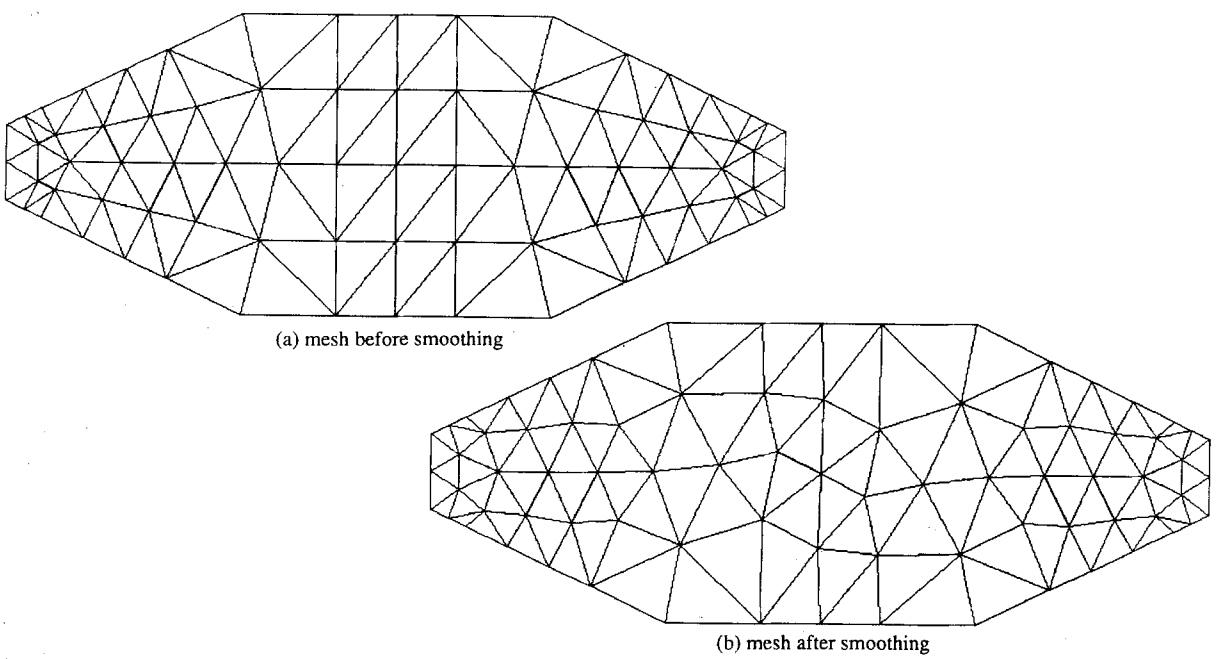


Fig. 6. Triangular mesh of a convex region with 120 elements.

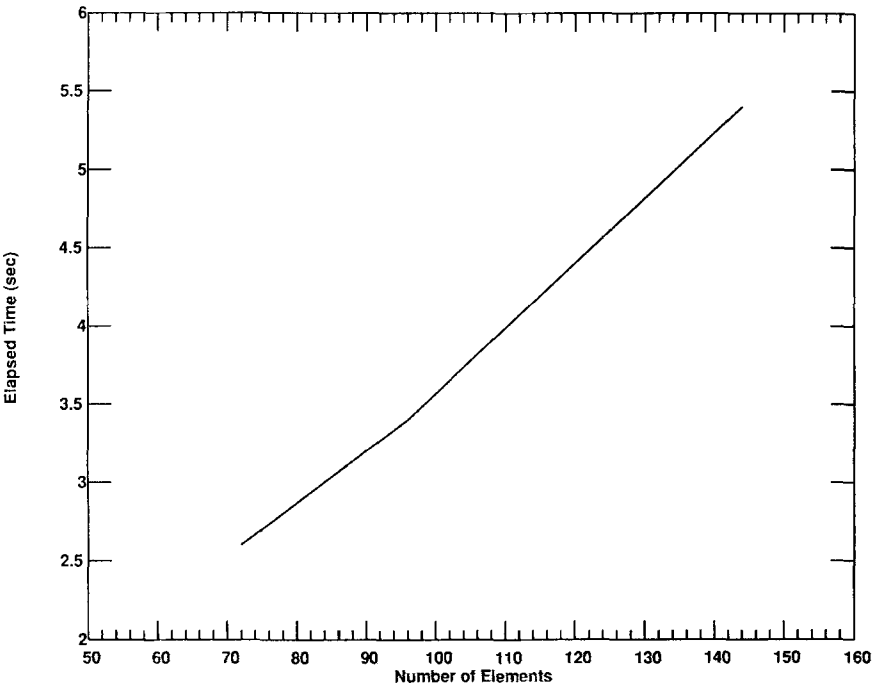


Fig. 7. Computation time results of four meshes of simple region.

initial and smoothed meshes generated by our algorithm for one particular mesh density. This particular mesh is made up of 120 elements and 79 nodes. The four FE meshes generated in this numerical experiment contain 72, 96, 120, and 144 elements and are associated with total running time 2.6, 3.4, 4.4, and 5.4 seconds, respectively. We note that the time for MAT computations is the same in these four cases. These running times, shown in Fig. 7, and the other results have been obtained using a DEC Vax Station II GPX. The linear character of the running time with respect to number of elements supports our claim regarding the time complexity of our FE mesh generator [1].

The six representative examples of Table 1 (see Fig. 8) demonstrate the capabilities of our mesh generation algorithm. Specifications and computational times of these examples are given in Table 1.

Table 1.

Mesh	Subdomains	Elements	Nodes	Time (sec)
Fig. 8(a)	14	88	59	3
Fig. 8(b)	34	192	123	9
Fig. 8(c)	36	208	136	11
Fig. 8(d)	36	320	195	21
Fig. 8(e)	40	324	200	22
Fig. 8(f)	56	416	270	35

The computation times represent total time of the mesh generation process including the MAT computation. These small computation times indicate the high potential of our method for real-time design and analysis applications.

3 Design and Analysis Applications

In this section we present various applications drawn from engineering design and analysis to illustrate the usefulness of our interrogation and meshing techniques based on the MAT.

3.1 Adaptive Faceted Approximation of Trimmed Curved Surface Patches

In the first application, we use our triangulation scheme to discretize and approximate trimmed curved surface patches in terms of a set of planar facets with a prescribed precision.

Such a triangulation of a trimmed surface patch may be used for numerous purposes. It may be employed as a FE discretization of untrimmed and trimmed plate and shell structures. Such a discretization can, in turn, be analyzed using triangular plate and shell finite elements, which are commonly available in existing FEA systems. It could also be used to convert B-Rep models with curved surfaces into faceted representations for polyhedral solid

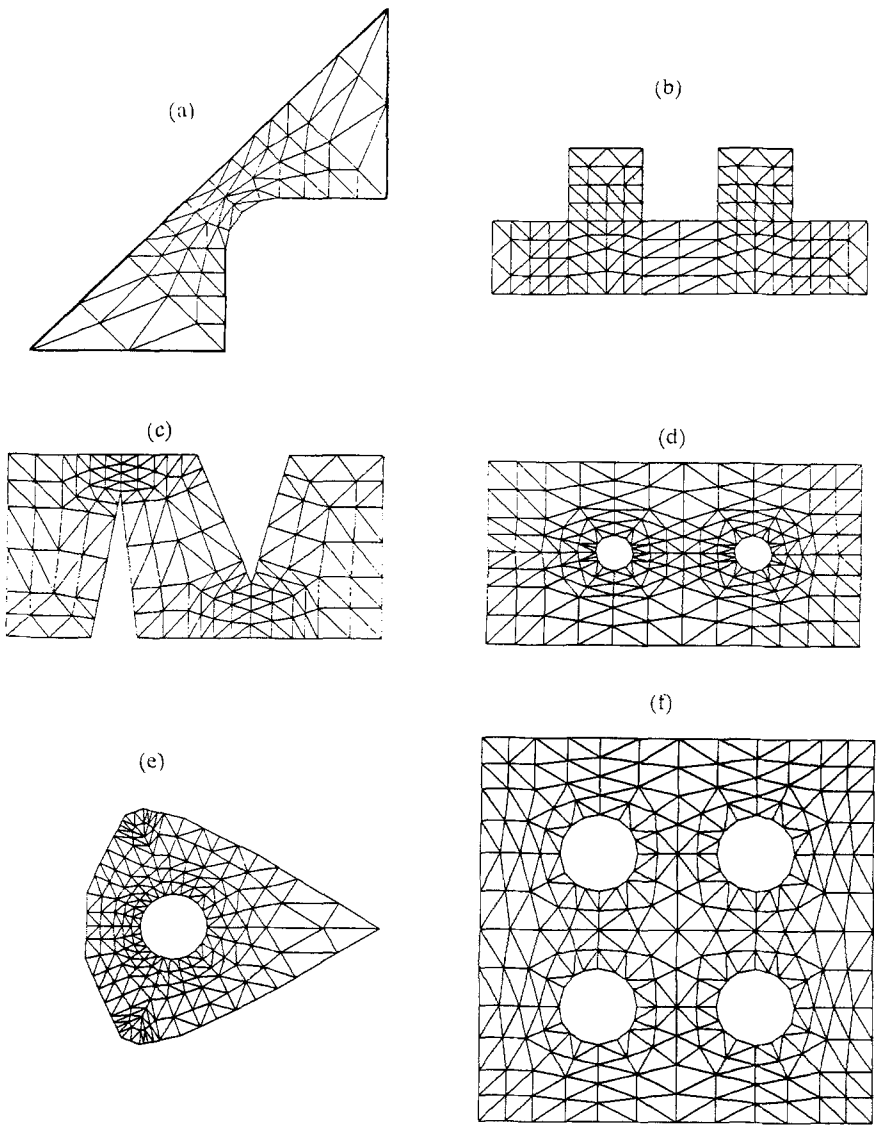


Fig. 8. Finite element mesh examples.

modelers. Other potential applications of this method include integral properties computation, point-set classification and ray tracing of trimmed curved surface patches.

3.1.1 Extension of the triangulation technique to trimmed curved surface patches

We present a methodology to extend the MAT computation from planar regions to trimmed curved surface patches. Using the B-Rep method, we describe a trimmed patch as a face using a set of bounding loops on a surface patch. The face of interest on the surface patch is bounded by one exterior loop and also, possibly, by a set of disjoint interior loops, all defined on the surface patch. An untrimmed parametric surface patch of NURBS form, $S_{k,l}(u, v)$ of maximum orders k and l in the parametric variables u and v is represented in homogeneous coordinates as

$$S_{k,l}(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,k}(u) N_{j,l}(v) P_{ij} \tag{2}$$

where $P_{i,j} = [w_{ij}x_i, w_{ij}y_{ij}, w_{ij}z_{ij}, w_{ij}]$ are the control points in a $(n + 1) \times (m + 1)$ grid and $N_{i,k}(u)$, $N_{j,l}(v)$ are B-spline basis functions defined over nonuniform knot vectors in the u and v directions given by $\{u_0, \ldots, u_{k+n}\}$ and $\{v_0, \ldots, v_{l+m}\}$, respectively. The bounding loops are defined in terms of NURBS curves defined on the parameter space of the surface patch. Such a parametric curve, $C_r(s)$ of order

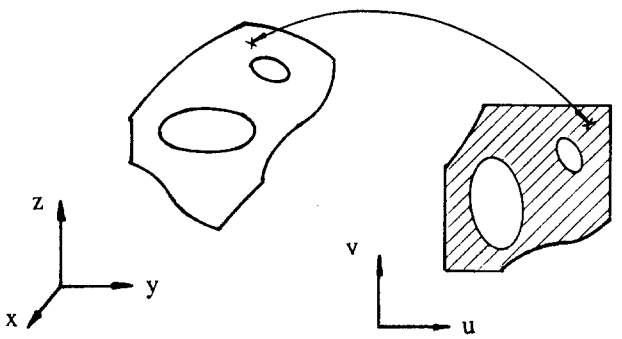


Fig. 9. Mapping between parameter space and three-dimensional space.

r is represented in homogeneous parametric space as

$$C_r(s) = \sum_{k=0}^{q} N_{k,r}(s) P_k \tag{3}$$

where $P_k = [w_k u_k, w_k v_k, w_k]$ are the $(q + 1)$ control points and, similarly, $N_{k,r}(s)$ are B-spline basis functions defined over a nonuniform knot vector $\{s_0, \ldots, s_{r+q}\}$. The concept of MAT on a curved surface patch can be generalized using the grass-fire analogy and the resulting generalized offset curves on surfaces. Such offset curves on surfaces are defined via minimal paths (geodesics) emanating from an initial curve in a direction orthogonal to the curve. Formal definition of such offsets of curves on NURBS surfaces and their approximation in terms of spline curves using an adaptive technique can be found in [17].

In this work, however, we do not deal with MAT based on the concept of minimal paths (geodesics) on a curved surface. Instead we perform the MAT computation in the parameter space exactly and we map the resulting MA branches and Voronoi diagram to three-dimensional space using the patch equation (2) (see Fig. 9). This approach is a straightforward extension of our MAT algorithm; however it cannot, in general, represent actual *isometry* information of a curved surface. Therefore, it is an approximate method.

3.1.2 Faceted surface approximation algorithm

Given a B-Rep model of a trimmed curved parametric patch, we can readily compute the MAT of the shape on the parameter space [1]. The MAT computation results in the MA, associated RF, and a set of subregions decomposing the parameter space of the trimmed patch. Using our triangulation technique [1], we can next generate a mesh on the parameter space of the trimmed surface patch. If required by our application, smoothing can be invoked to improve the shape characteristics of the triangles. Once individual triangles are generated on the parameter space, then they are readily mapped into the three-dimensional space using equation (2).

This triangulation is the initial discretization of the trimmed patch. In this process, the trimmed surface patch is approximated using a set of triangular planar facets. Since surface curvature information of the patch is not used by MAT, we may need to refine the faceted approximation so that it satisfies certain accuracy requirements. For this purpose, we define two error norms. An error norm is used to account for the Euclidean distance between the centroid of a triangular facet and a corresponding point on the surface obtained from the surface equation using the parameter values of the centroid. Another error norm can be introduced to identify the error involved in the direction of the unit normal of the surface. In this case, the unit normal vector of a triangular face is determined and it is again compared with the unit normal vector of the surface at a point which corresponds to the centroid of the triangle. If a triangle has distance error and normal vector angular discrepancies greater than prescribed error bounds, then the triangle is further subdivided to obtain a better approximation for the curved surface patch. The local mesh refinement technique described in previous sections is used for this purpose.

To evaluate the level of accuracy attained in this approximation, we use a norm based on Euclidean distance. Given a triangular mesh, we compute the distance from the centroid of an individual triangle to the corresponding point on the trimmed surface patch. If this distance value is greater than a prescribed tolerance, a refinement flag associated with the triangle is set. In our implementation, the value of the refinement flag is determined by the ratio of the distance value to the prescribed tolerance. The floor value of the refinement flag indicates the level of subdivision to be carried out for the triangle. By traversing the doubly linked list structure which holds the triangles of the mesh, every triangle is similarly processed to determine the value of the refinement flag.

Once refinement flags of individual triangles are set, the list structure is traversed again to carry out the actual refinement process. If a triangle has a refinement flag greater than or equal to unity, it is split. The resulting descendant triangles assume refinement parameter values one less than the parent

Algorithm: Adaptive Surface Faceting

```
input:  Boundary representation of trimmed curved surface patch and distance
          tolerance (TOLERANCE).
output: List of triangular facets.
begin
        Compute Voronoi decomposition in parameter space using MAT
          Algorithm [1];
        Construct list of triangles as a coarse triangulation in parameter space
          using MAT Meshing Algorithm [1];
        for every triangle {
          Compute distance error norm (DISTANCE_ERR);
          if DISTANCE_ERR > TOLERANCE
          [comment: set refinement flag of triangle]
            FLAG ← [DISTANCE_ERR / TOLERANCE];
        }
        while there exists a triangle with FLAG ≥ 1 in triangle list
          Refine triangle;
end
```

triangle's refinement flag value. The refinement process continues until the refinement flag values of all triangles are reduced to zero. The pseudocode above summarizes the steps of our technique to triangular trimmed curved surface patches with complex curved boundaries.

3.1.3 Examples

Our scheme to triangulate trimmed curved surface patches has been implemented and tested with a number of complex and diverse examples. Figure 10 illustrates triangulation of two such trimmed patches. In these two cases, the surfaces are defined as integral bicubic B-spline patches.

3.2 Adaptive Finite Element Analysis

The second application deals with adaptive FEA of linear elasticity problems. Our coarse and fine meshing scheme based on the MAT provides an initial mesh appropriate for an adaptive refinement solution method. The automatic mesh generation capabilities of our meshing system have been used for both *h*- and *p*-adaptive analysis processes.

In recent years, an important research aim of the FEA community has been to develop more automatic analysis procedures by integrating FEA with CAD systems [14,18–23]. Our automatic shape interrogation and mesh generation method can be effectively used to achieve this objective. In this section, we report results of two adaptive FEA applications using our prototype system.

3.2.1 Adaptive finite element analysis

In an adaptive FEA scheme, the objective is to automatically produce results satisfying a prescribed degree of accuracy [24]. The basic steps involved in a typical adaptive process may be summarized as follows.

1. Start with definition of the problem and an initial FE mesh.
2. Perform the required FEA.
3. Using an error estimator, predict the discretization errors to determine those portions of the analysis model that are not yielding the required degree of accuracy.
4. Improve the portions of the mesh that are not satisfactory, return to the second step, and continue this process until the desired level of accuracy is achieved.

This is an iterative process in which distinct steps of analysis, error estimation, and mesh improvement are carried out in sequence. For mesh improvement, there are four methods [24].

1. The finite element size is reduced in each subsequent solution step of *h*-convergence FEA.
2. The degree of interpolation functions is increased in each subsequent step of *p*-convergence FEA.
3. Nodal points in the domain are repositioned in *r*-convergence FEA.
4. The previous mesh is discarded and a new analysis cycle is started again with a new mesh layout in a remeshing approach.

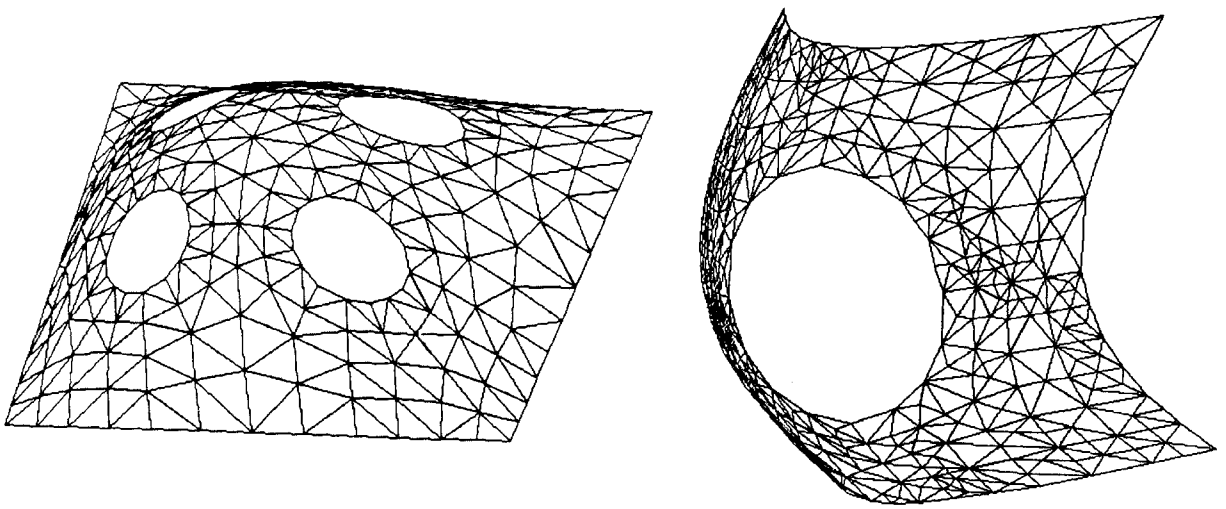


Fig. 10. Triangulation of trimmed curved surface patches.

The single most important feature of an adaptive FE code is the prediction of errors in the current solution. Errors involved in the numerical solution of a FE formulation can be separated into three main groups [25]:

- *Discretization errors:* These are caused by representing a continuum using a finite number of degrees of freedom in the discretized system.
- *Round-off and truncation errors:* These are caused by the limitation of digital computers in representing and processing real numbers.
- *Solution errors:* There are various causes for these errors. For example, solution errors in constitutive modeling are due to linearization and integration of the constitutive relations. Solution errors in the calculation of the dynamic response arise in the numerical integration of the equations of motion with respect to time. Also, solution errors arise from iterative solutions because convergence is measured on increments in the solution variables that are small but not zero.

A priori FE error analysis can only indicate the convergence rate of the numerical solution [26]. Therefore, quantitative error estimation is based on *a posteriori* error analysis which makes use of actual FE results to estimate the discretization error. Depending on the degree and location of discretization errors, the FE mesh is refined locally to obtain more accurate results in the next analysis cycle. In constructing an error estimator in the *displacement* based FE approach, several criteria have been developed during the last decade [24]: predicted strain energy density variations; stress discontinuities along element boundaries; and, magnitude of violation of internal element equilibrium.

To some extent, all these criteria have been shown to work well in practice. In existing adaptive FEA schemes, the *h*-version and *p*-version refinement techniques are the most widely used. It has been shown that the *p*-version adaptive analysis has better convergence characteristics in comparison to the *h*-version [24]. One major disadvantage of the *h*-adaptive approach is that in the vicinity of singularities in the problem domain, (such as re-entrant vertices, constrained corners, and concentrated loads), the rate of convergence is slow. Therefore, problems with severe singularities require very fine FE meshes and this, in turn, reduces the efficiency of the numerical solution. On the other hand, the *p*-adaptivity involves a smaller number of finite elements and high order interpolation functions. In an experimental two-dimensional adaptive system, these two techniques are used in tandem so that their favorable properties are better exploited [23].

3.2.2 *An adaptive finite element analysis scheme for plane elasticity problems*

In this section, we discuss the technique we implemented to determine discretization errors in a FE solution of linear elasticity problems. Given a Cartesian coordinate system. (x, y, z), we deal with the solution of elasticity problems whose governing differential equations describing equilibrium of a volume element are of the following form [27]

$$\sigma_{ij,j} + f_i = 0 \tag{4}$$

where $\sigma_{ij}(\sigma_{ij} = \sigma_{ji})$ are components of the symmetric stress tensor, and f_i are components of applied body force per unit volume, a subscript after a comma denotes partial derivative with respect to j, and a repeated index denotes summation over x, y and z.

Equation (4) is solved in a domain Ω, subject to prescribed conditions on its boundary Γ.

$$u_i = \bar{u}_i \text{ on } \Gamma_u \quad \sigma_{ij} n = t_i = \bar{t}_i \text{ on } \Gamma_t \quad \Gamma = \Gamma_u \cup \Gamma_t \quad \Gamma_u \cap \Gamma_t = \emptyset \tag{5}$$

where u_i are components of displacement, t_i components of boundary traction (i.e., force per unit area), n_j components of the unit normal vector of the boundary, and Γ_u and Γ_t portions of the boundary with displacement and stress boundary conditions.

In this formulation, components of strain are defined as follows

$$\varepsilon_{ij} = \tfrac{1}{2}(u_{i,j} + u_{j,i}) \tag{6}$$

and stress is related to strain by the following constitutive equation

$$\sigma = D\,\varepsilon \tag{7}$$

where D is the material matrix of the domain, $\sigma = [\sigma_{ji}]$, $\varepsilon = [\varepsilon_{ij}]$ are the stress and strain tensors. We ignore initial strains and initial stresses in this analysis.

In the FE discretization, we approximate the solution function $u(x, y, z)$ using a trial function, $u^*(x, y, z)$

$$u(x, y, z) \approx u^*(x, y, z) = N(x, y, z)\, U \tag{8}$$

where $N(x, y, z)$ are shape functions and U is a vector of unknown nodal displacements.

We can obtain the global system of equations of the FE discretization using the above trial function either by a weighted residual method or by a variational approach in the following form [28]

$$KU = F \tag{9}$$

where K is the stiffness matrix and F is the force vector which are defined as follows.

$$K = \int_\Omega B^T D B \, d\Omega \tag{10}$$

$$F = \int_\Omega N^T f \, d\Omega + \int_{\Gamma t} N^T \bar{t} \, d\Gamma \tag{11}$$

where superscript T denotes matrix transpose, and matrix B contains derivatives of shape functions such that we can write strain displacement relationship in the following form.

$$\varepsilon^* = BU \tag{12}$$

Using the FE solution, we can readily determine the stress.

$$\sigma^* = D\varepsilon^* \tag{13}$$

The approximate solutions u^*, σ^* and ε^* are different from the exact values u, σ and ε, and the differences are the discretization errors in the FE solution. We note that the numerical solution also includes round-off and truncation errors which are assumed negligible in this analysis. Following [25], we can define absolute solution errors for displacements and stresses by

$$e = u - u^* \tag{14}$$

$$e_\sigma = \sigma - \sigma^* \tag{15}$$

We note that in the above equations u and σ are actual values, which are not available. Therefore, it is necessary to *predict* these error values by other means.

These forms of the error are pointwise definitions. The *energy norm* of the solution is:

$$\|u\| = \left[\int_\Omega (Bu)^T D (Bu) \, d\Omega\right]^{1/2} \tag{16}$$

$$\|u^*\| = \left[\int_\Omega (Bu^*)^T D (Bu^*) \, d\Omega\right]^{1/2} \tag{17}$$

Similarly we can define a more useful form of the error in a global manner as a norm [23], [28].

$$\|e\| = \left[\int_\Omega (Be)^T D (Be) \, d\Omega\right]^{1/2} \tag{18}$$

$$\|e\| = \left[\int_\Omega e_\sigma^T D^{-1} e_\sigma \, d\Omega\right]^{1/2} \tag{19}$$

The *error norm* in the whole domain can be calculated by adding the contributions of error norms of all elements used in the discretization as follows [24].

$$\|e\| = \left[\sum_{i=1}^{m} \|e_i\|^2\right]^{1/2} \tag{20}$$

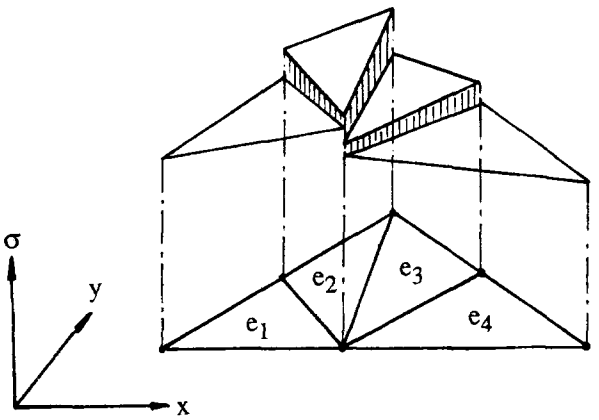


Fig. 11. Stress jumps involved in finite element solution.

where i spans all elements in the FE discretization and m is the number of elements in the mesh.

Using the energy and error norms, we also define the *relative error,* which is a measure of the relative error involved in the solution [23,25]

$$\eta = \|e\|/\|u\| \tag{21}$$

In the *displacement* based FE formulation of elasticity problems. C^0 (displacement) continuity is assumed in the trial functions of the FE approximation. Therefore, numerical results, in general, give rise to discontinuous approximations of σ, (see Fig. 11).

This character of the solution is well known since the early days of FEA. Therefore, stress smoothing techniques have been commonly used in practice to provide improved quantitative results from the discontinuous stress functions [29,30]. The smoothed stress values turn out to be much better approximations of the actual values of stress than the FEA results before smoothing [29]. Based on this observation, we assume that such a smoothed stress distribution can be used to *predict* the solution error. If we substitute the smoothed stress value, s, for the exact stress value, σ, in Eq. (15), we obtain a prediction for the error in stress as follows.

$$e_\sigma \approx s - \sigma^* \tag{22}$$

Rank and Zienkiewicz [31] have shown that such an error prediction method is equivalent to the other *a posteriori* error estimators used for error predictions in adaptive FEA [24].

Now we can determine the energy and error norms using Eqs. (17) and (19) and predict the relative error involved in the solution [23,24]. Note that the error is orthogonal to the solution function in the FE approximation [28].

$$\eta = \|e\|/(\|u^*\|^2 + \|e\|^2)^{1/2} \tag{23}$$

Once we have a FE solution, the smoothed stress distribution can be readily obtained. Using the above expressions, we can predict the error in the solution. In an adaptive FEA based on h-convergence, after determining the error norm, we carry out mesh refinement to reduce the size of elements associated with an unacceptable error value before we start the next solution step. Suppose we try to achieve a relative error tolerance, η_{tol}, in the FE solution. Then we require that the adaptive FEA process ends when the following condition is satisfied by the resulting mesh:

$$\eta \leq \eta_{\text{tol}} \tag{24}$$

It is assumed that the solution error is uniformly distributed throughout the mesh in a converged adaptive FEA. Based on this assumption, the following condition specifies the acceptable average value of error norm of individual finite elements [23]

$$\|e_i\| \leq \bar{e} = \eta_{\text{tol}}[(\|u^*\|^2 + \|e\|^2)/m]^{1/2} \tag{25}$$

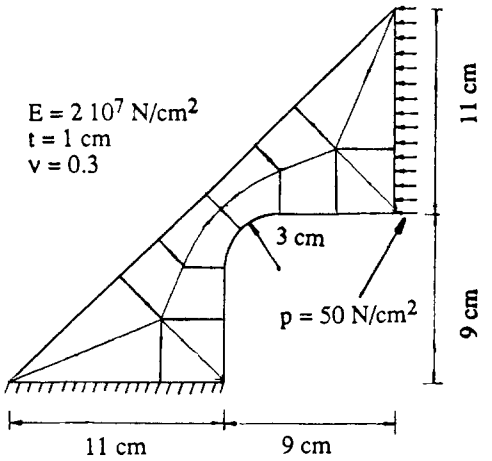
where right hand side of the inequality represents the allowable mean error norm associated with the current mesh. We can check whether an element needs to be refined using a ratio of its error norm to this mean value:

$$\rho_i = \|e_i\|/\bar{e} \tag{26}$$

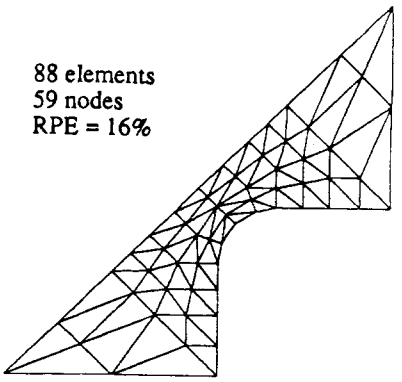
If the ratio is greater than unity for a given element, the error associated with that element is higher than the prescribed tolerance, η_{tol}. Therefore, the element should be refined to a certain subdivision level whose degree is determined by the ratio, ρ. For h-version mesh refinement purposes, we can use the local refinement technique discussed in section 2.

3.2.3 Examples

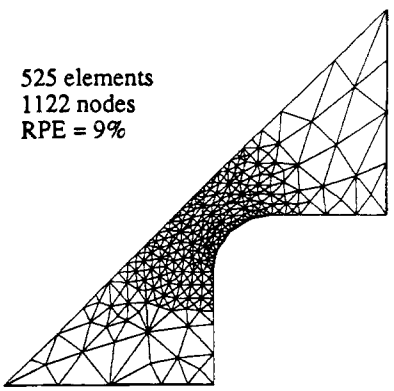
In plane stress formulation of elasticity problems the only nonzero stress components are σ_{xx}, σ_{yy}, and σ_{xy}. In our implementation, the FE solver uses six-node triangular isoparametric elements. Input data of the FEA are preprocessed and nodes are renumbered so that the bandwidth of the global stiffness matrix becomes minimum. Our automatic node renumbering scheme is based on the method



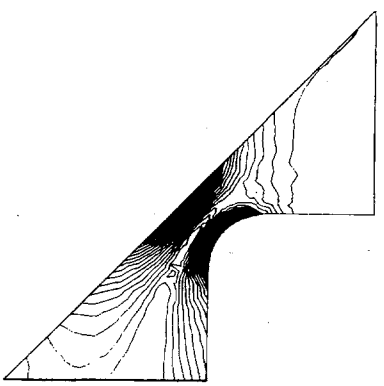
model and p-version mesh



the first h-version mesh



the last h-version mesh



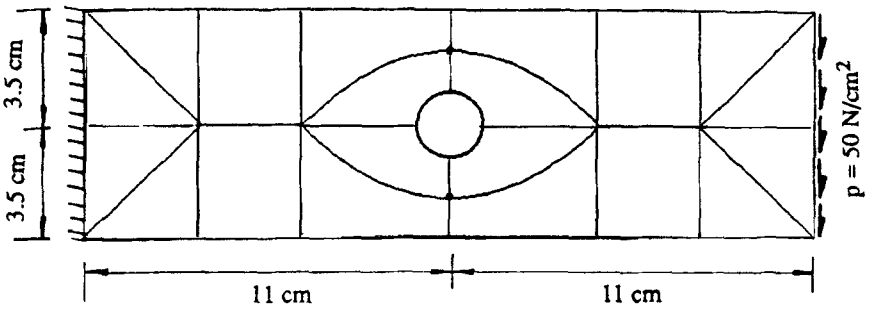
Von Mises stress contours

Fig. 12. Adaptive analysis of a bracket.

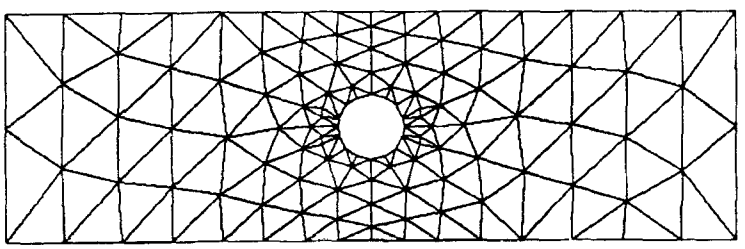
presented in [32]. Element stiffness matrices are numerically evaluated using a three-point Gauss quadrature rule. The resulting system of linear equations is solved using the Gauss elimination method. The error indicator used in our prototype system is based on the techniques presented in the previous section. In our implementation, the smoothed stress distribution is obtained using a simple process. First, stress components computed at discrete points in the element (i.e., at three Gauss quadrature points) are extrapolated to the nodes of the element. Then for a given node, smoothed stress values are calculated by taking the average of *weighted* stress contributions of all adjacent elements. Simple averaging of stress at nodal points works correctly only if a mesh is composed of elements with the same size. We assume that the problem domain has uniform thickness and use the area of an element as its weight factor to account for the size of adjacent elements in this smoothing process. This smoothing approach is similar to the "lumped mass least-squares" approximation in one dimension presented in [30].

The following two representative model problems have been adaptively solved by our prototype *h*-adaptive FEA system. We have also solved these two model problems using the *p*-adaptive solution functionality provided by an integrated modeling and analysis system [33]. Results of these model problems obtained from the two analysis systems are in good agreement. In the solution strategy based on *p*-convergence, coarse subdivisions generated by our MAT technique have been employed as initial FE meshes. We observe that this approach gives rise to significant time reduction in computation cycles of *p*-adaptive analysis as compared to the same computation cycles using meshes generated for *h*-adaptive analysis.

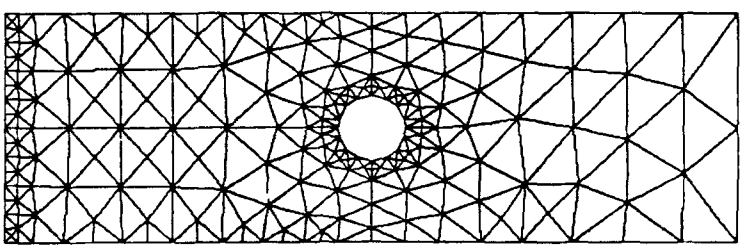
In each of these cases, the problem domain has a Young's modulus $E = 2 \times 10^5$ MPa, Poisson's ratio



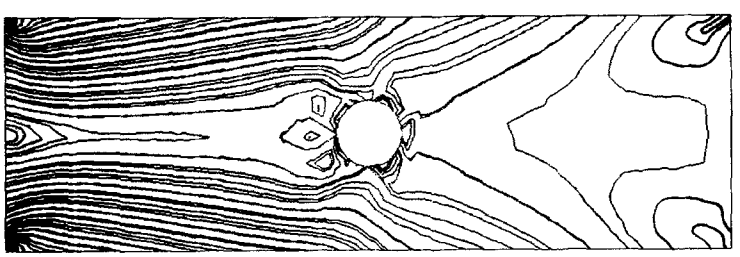
model and p-version mesh



the first h-version mesh: 204 elements, 460 nodes, RPE = 12%



the last h-version mesh: 364 elements, 810 nodes, RPE = 7%



Von Mises stress contours

Fig. 13. Adaptive analysis of a plate with a hole.

$\gamma = 0.3$ and a thickness $t = 1$ cm. The adaptive scheme has provided a solution with relative error less than 10% [relative percentage error (RPE)] in each case. Domain descriptions, boundary conditions, and applied loads of these two model problems are shown in Figs. 12 and 13.

In the first example (see Fig. 12), a bracket is analyzed. The error indicator has identified a stress concentration in the narrow part, and local refinement is performed in that region. In the second example (see Fig. 13), a plate with a circular hole at its center is analyzed. A clamped boundary condition is specified at one end and a uniformly distributed shear force is applied at the other end. The clamped end boundary condition gives rise to two singularities and a boundary layer which are identified by the error indicator. The adaptive solution also identifies some critical elements around the circular hole in the problem domain. Plots of von Mises stress contours [27] of these two model problems clearly indi-

cate those critical areas. All these test cases demonstrate the efficiency and usefulness of our automatic mesh generator and the prototype system.

4 Summary and Conclusions

We have reported the implementation of a prototype system based on our MAT and surface meshing algorithms [1], and also results of several design and analysis applications obtained by using this system. Our adaptive surface approximation technique is efficient and can be used for solving numerous design problems. Our experience with the prototype system indicates that our meshing and *h*-adaptive analysis techniques can provide an automated solution scheme for a wide variety of analysis problems. Our coarse shape decomposition method can be effectively used in an adaptive FEA system based on the *p*-convergence method. We observe that this approach can give rise to substantial reduction in computation time. As a result, our technique can assist a *p*-adaptive FEA scheme to become an interactive analysis method in integrated design and analysis systems. For future work, we propose to investigate potential capabilities and applications of our interrogation and meshing techniques as automated *idealization* and *model creation* methods for *h*- and *p*-adaptive FEA processes.

Acknowledgments

Funding for this research was obtained in part from MIT Sea Grant College Program (grant number NA90AA-D-SG424) and the Office of Naval Research in the United States (grant numbers N00014-87-K-0462 and N000-91-J-1014). Mr. E. C. Sherbrooke contributed to the computer implementation.

References

1. Gursoy, H.N.; Patrikalakis, N.M. (1992) A coarse and fine finite element mesh generation scheme based on medial axis transform: Part I algorithms, Engineering with Computers, Springer-Verlag, New York
2. Blum, H. (1967) A transformation for extracting new descriptors of shape. Models for the Perception of Speech and Visual Form, Weinant Wathen-Dunn (Editor), MIT Press, Cambridge, MA, 362–381
3. Blum, H. (1973) Biologcal shape and visual science (Part I), Journal of Theoretical Biology, 38, 205–287
4. Patrikalakis, N.M.; Gursoy, H.N. (1988) Skeletons in shape feature recognition for automated analysis, MIT Ocean Engineering Design Laboratory Memorandum 88-4
5. Patrikalakis, N.M.; Gursoy, H.N. (1989) Shape feature recognition by medial axis transform, MIT Ocean Engineering Design Laboratory Memorandum 89-1
6. Patrikalakis, N.M.; Gursoy, H.N. (1990) Shape interrogation by medial axis transform, Advances in Design Automation 1990, Volume One, Computer Aided and Computational Design, B. Ravani (Editor), ASME, NY, 77–88
7. Gursoy, H.N. (1989) Shape interrogation by medial axis transform for automated analysis, PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA
8. Hoffmann, C.M. (1990) How to construct the skeleton of CSG objects, Technical Report, CSD-TR-1014, Purdue University
9. Weiler, K.J. (1985) Edge-based data structures for solid modeling in curved-surface environments, IEEE Computer Graphics and Applications, 5, 1, 21–40
10. Patrikalakis, N.M. (1989) Approximate conversion of rational splines, Computer Aided Geometric Design, 2, 6, 155–165
11. Sederberg, T.W.; Anderson, D.C.; Goldman, R.N. (1984) Implicit representation of parametric curves and surfaces, Computer Vision Graphics and Image Processing, 28, 1, 72–84
12. NAG (1988) Numerical Algorithms Group FORTRAN Library, NAG, Oxford, England
13. Cavendish, J.C. (1974) Automatic triangulation of arbitrary planar domains for finite element method, International Journal for Numerical Methods in Engineering, 8, 679–697
14. Chae, S.W. (1988) On the automatic generation of near-optimal meshes for three-dimensional linear elastic finite element analysis, PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA
15. Rivara, M.C. (1984) Algorithm for refining triangular grids suitable for adaptive and multigrid techniques, International Journal for Numerical Methods in Engineering, 20, 745–756
16. Mantyla, M. (1988) An introduction to Solid Modeling, Computer Science Press, Rockville, Maryland
17. Patrikalakis, N.M.; Bardis, L. (1989) Offsets of curves on rational B-Spline surfaces. Engineering with Computers, 5, 39–49
18. Wordenweber, B. (1984) Finite element analysis for the naive user, Solid Modeling by Computers, From Theory to Applications, Plenum Press, NY., M.S. Pickett, J.W. Boyse (Editors), 81–101
19. Yerry, M.A.; Shephard, M.S. (1984) Automatic three-dimensional mesh generation by the modified octree technique, International Journal for Numerical Methods in Engineering, 20, 1965–1990
20. Kela A.; Perucchio R.; Voelcker H. (1986) Towards automatic finite element analysis, ASME Computers in Mechanical Engineering, 5, 1, 57–71
21. Fenves, S.J. (1986) A framework for cooperative development of finite element modeling assistant, Proceedings of the International Conference for Engineering Analysis, University College, Swansea, England, K.J., Bathe, D.R.J. Owen (Editors), 475–485
22. Schroeder, W.J.; Shephard, M.S. (1988) Geometry-based fully automatic mesh generation and the delaunay triangulation, International Journal for Numerical Methods in Engineering, 26, 2503–2515
23. Zienkiewicz, O.C.; Zhu, J.Z.; Gong, N.G. (1989) Effective and practical *h-p* version adaptive analysis procedures for the finite element method, International Journal for Numerical Methods in Engineering, 28, 879–891
24. Babuska, I.; Zienkiewicz, O.C.; Gago, J.; Oliveira, E.R.A. (1986) Accuracy Estimates and Adaptive Refinements in Fi-

nite Element Computations, John Wiley, Chichester, England
25. Utku, S.; Melosh, R.J. (1983) Errors in finite element analysis, State-Of-The-Art Surveys on Finite Element Technology, A.K., Noor, W.D. Pilkey (Editors), ASME, New York, 297–324
26. Strang, G.; Fix, G. (1973) An Analysis of the Finite Element Method, Prentice-Hall, Englewood Cliffs, NJ
27. Timoshenko, S.P.; Goodier, J.N. (1970) Theory of Elasticity, McGraw-Hill, New York
28. Zienkiewicz, O.C.; Morgan, K. (1983) Finite Elements and Approximations, John Wiley & Sons, New York
29. Hinton, E.; Owen, D.R.J. (1979) An Introduction to Finite Element Computations, Pineridge Press Ltd., Swansea, England
30. Langtangen, H.P. (1989) A method for smoothing derivatives of multilinear finite element fields, Communications in Applied Numerical Methods, 5, 4, 275–281
31. Rank, E.; Zienkiewicz, O.C. (1987) A simple error estimator in the finite element method, Communications in Applied Numerical Methods, 3, 3, 243–249
32. Collins R.J. (1973) Bandwith reduction by automatic renumbering, International Journal for Numerical Methods in Engineering, 8, 27–43
33. IFEMp (1990) Intergraph finite element modeling/solver *p*-adaptive module reference manual, Intergraph Corporation, Huntsville, AL