S8-61
125794
34 P

N88-19119

Production Automation Project
College of Engineering & Applied Science
The University of Rochester
Rochester, New York 14627

A HIERARCHICAL STRUCTURE FOR AUTOMATIC MESHING AND ADAPTIVE FEM ANALYSIS

by

Ajay Kela, Mukul Saxena and Renato Perucchio

(November 1986)

submitted for publication in
a special issue of
Engineering Computations

The work described in this paper was supported by the National Science Foundation under Grants ECS-8104646 and DMC-8403882 and by companies in the P.A.P.'s Industrial Associates Program. The findings and opinions expressed here are those of the authors and do not necessarily reflect the views of the various sponsors.

A HIERARCHICAL STRUCTURE FOR AUTOMATIC MESHING AND ADAPTIVE FEM ANALYSIS

Ajay Kela*
Corporate Research and Development
General Electric Company
Schenectady, N.Y. 12301

&

Mukul Saxena and Renato Perucchio**
Production Automation Project
and Department of Mechanical Engineering
University of Rochester
Rochester, N.Y. 14627, U.S.A.

SUMMARY

This paper deals initially with a new algorithm for generating automatically, from solid models of mechanical parts, finite element meshes that are organized as spatially addressable quaternary trees (for 2–D work) or octal trees (for 3–D work). Because such meshes are inherently hierarchical as well as spatially addressable, they permit efficient substructuring techniques to be used for both global analysis and incremental re–meshing and re–analysis. The paper summarizes the global and incremental techniques, and presents some results from an experimental closed loop 2–D system in which meshing, analysis, error evaluation, and re–meshing and re–analysis are done automatically and adaptively. The paper concludes with a progress report on a 3-D implementation.

* former Research Assistant, Production Automation Project
** Research Assistant and Director, respectively

1 INTRODUCTION

Interactive computer graphics has reduced the cost of using the Finite Element Method (FEM) to analyze mechanical parts and structures [PERU82]. However, interactive mesh generation still requires the guidance and ingenuity of an expert analyst to produce a valid FEM model, to interpret computed results and to modify the model when results are questionable. Thus analysing a fixed design is usually an iterative process; moreover as design itself is iterative, the current use of the FEM requires continued human guidance within a doubly iterative process. It is obvious that automatic mesh generation, followed by adaptive mesh refinement would dramatically reduce the cost of the design process. Two newly available tools – solid modelling systems [REQU83] and algorithms for a posteriori error analysis [BABU78,PEAN79,KELL83,GAGO83] – make this goal reachable.

< Figure 1 >

Figure 1 illustrates the architecture of an automatic analysis system. The user defines an initial geometrical domain in the Solid Modelling System (SMS) together with such attributes as boundary conditions, loads, material properties, and analysis related parameters. The mesh generator produces a discretized model – the FEM mesh – from the geometric definition and the attribute specification (attributes may determine, for example, the locations of some nodes). The FEM analysis processor computes primary and secondary field variables (in general, the displacements vector at nodal points and the stress tensor within the elements) from the initial FEM model. The error evaluator compares global error estimates derived from the analysis output with pre–specified error–tolerances to either accept the results or request a new analysis based on a modified mesh. In the feed-back loop, the analysis control process indicates regions of refinement in the current model for the next cycle of mesh generation and analysis. In case of reanalysis, mesh generation and mesh analysis proceed through localized mesh refinement and incremental re–analysis, i.e. the use of previous unaltered regions of the mesh as well as intermediate analysis computations to derive new results. This approach to automatic FEM analysis is embodied in an experimental 2–D system whose underlying principles are explained below. All meshes and analytical results that appear in later sections were produced with the experimental system.

The next section opens with a discussion of automatic mesh generation focussed mainly on a particular approach – hierarchical grids – that fosters spatial addressability (an important property explained below). Later sections discuss algorithms for (1) generating hierarchical grid-based meshes, then (2) analyzing such meshes, (3) refining and re-analyzing, and finally (4) extending meshing and analysis to 3-D work. The paper concludes with a short discussion of the strengths and weaknesses of the approach.

2 AUTOMATIC MESH GENERATION

Most "automatic" meshing facilities in contemporary CAD systems operate from wireframe descriptions of objects, via mapping algorithms. The user must partition the domain, which is represented by a collection of edges, into a set of topologically simple subdomains in which meshes can be generated automatically. This approach is unsuitable for a fully automatic meshing procedure because it depends on human judgement both to guide meshing per se and to resolve ambiguities in the wireframe representation.

Genuinely automatic mesh generation must start from an unambiguous representation of the object to be analyzed, and thus some form of solid modelling system (SMS) is a primary utility. Nearly all current SMS's are based internally on either a Constructive Solid Geometry (CSG) Representation or a Boundary Representation, or both [REQU83]. CSG exploits the notion of "adding" and "subtracting" (via set-union and set-difference operations) simple solid building blocks. Boundary schemes describe solids indirectly, via sets of faces which are represented by sets of edges that bound finite regions of surfaces.

The various schemes that have been proposed for automatic mesh generation may be catalogued for present purposes into three families: triangulation, element extraction and recursive spatial subdivision (quadtree and octree) schemes. We shall discuss the first two family briefly and then focus on the third.

Originally limited to 2-D problems, triangulation algorithms require some level of interactive user control to generate irregular assemblies of triangular elements [SUHA72]. Recently, however, Cavendish and co-workers [CAVE85] have developed a two-stage approach to automatic triangulation of solid domains. In the Cavendish method, points are injected into the domain, and then a solid triangulation is induced in which the points become nodes of tetrahedral elements. The main working tool of the second-stage triangulation is a Delaunay algorithm that generates valid meshes of tetrahedral elements within convex hulls of node points. Automatic algorithms are still being sought for (a) inserting points in the procedure's first stage, (b) removing elements that are generated outside the domain, and (c) representing the domain's boundary correctly.

Meshing schemes based on element extraction also result in decomposing the domain into a irregular collection of tetrahedral elements [WOO84, WÕRD84]. Elements are extracted by recursively applying a set of operators that work on the topological and geometrical description of the domain. The tetrahedral meshes that result are coarse and usually contain distorted elements that must be refined for analytical use. Also, existing operators for element extraction are not robust as required for a truly automatic implementation.

In both of these family of approaches, mesh refinement is done by splitting existing elements. Because refinement is driven from a FEM mesh rather than from the original solid model, refinement does not improve the geometric approximation of the original solid. Also, the meshes are not spatially addressable.

The idea underlying recursive spatial subdivision schemes is to approximate the object to be meshed with a union of disjoint, variably sized rectangles (in 2-D) or blocks (in 3-D); these are generated by subdividing recursively a spatial region enclosing the object, rather than the object itself. Figure 2.1 provides a 2-D example. The object – a bracket with a hole – is "boxed" to establish a convenient minimal spatial region, and then the box is decomposed into quadrants. When a quadrant can be classified as wholly inside or outside of the object, subdivision ceases; when a quadrant cannot be so classified, it is subdivided into quadrants and this process continues until some minimal resolution level is reached. (In 3-D, the decomposition proceeds by octants.) Approximations produced in this manner can be represented by logical trees whose nodes have four or eight sons (see Figure 2), hence the popular names <u>quadtree</u> and <u>octree</u> [JACK80].

< Figure 2 >

<u>Inside</u> cells of a spatial decomposition can be converted easily into FEM elements or substructures, but Boundary cells require further processing to produce valid elements that approximate closely the object's boundary.

Recursive spatial decompositions have two intrinsic properties – hierarchical structure and spatial addressability – that are central to the mesh refinement and incremental analysis techniques described later. These intrinsic properties, briefly presented here, are fully discussed in [KELA87].

The tree structure in Figure 2 can be regarded as an organizing or cataloging structure for data describing particular regions of space. At the lowest level of the tree one finds the smallest spatial regions and simplest finite elements. As one ascends the tree, the regions become larger (encompassing multiples of four or eight elemental regions) and the finite elements become super–elements with associated ("assembled") stiffness matrices, collected constraints, and so forth. As we shall see later, such an organization is ideally suited to mesh refinement by subdivision and incremental mesh analysis.

The diagram in Figure 2 suggests the classical approach to accessing the data structure associated with the tree: represent a tree with a linked–list in which nodes are addressed indirectly through downward pointers to sons and perhaps lateral pointers to siblings. Thus one accesses data by following pointers downward from the root of the tree. Alternatively, a recursive spatial decomposition can be viewed as a directly addressable hierarchical grid in which the number of cells in each linear dimension is an integer power of two. The key notion here is a systematic scheme for numbering all possible nodes of the underlying tree. In Figure 2, "1" represents the enclosing box, "2" – "5" represent specific quadrants of "1", "6" – "9" would represent quadrants of "2", and so on. Thus to access the spatial data for a particular node in the underlying tree, one merely calculates an array index through a simple formula and follows the single pointer stored there. This is usually much faster than the pointer–following method noted above, but it carries a storage penalty [KELA87].

Suppose finally that we know the geometric size and spatial position of the "1" cell (the overall box) in Figure 2. We can compute quickly the index of any cell in the hierarchy from its size and position, and conversely from an index we can compute quickly the size and position of the associated spatial cell. We have already seen that cell indices allow access through a single pointer to data associated with the cell, and thus we can associate, without searching, spatial regions with stored data, and stored data with spatial regions. This is what is meant by spatial addressability.

3 AN AUTOMATIC MESHING PROCEDURE

The procedure described below produces a spatially addressable FEM mesh embedded in the lowest level of a hierarchical grid. Higher levels of the grid are used during construction of the mesh and, as explained later, when the mesh is analyzed, refined, and incrementally re–analyzed. The procedure starts with a representation in a Solid Modelling System of the object to be meshed, and operates in two stages. The first stage meshes the interior of the object by spatial subdivision, and the second extends the mesh to the object's boundary. Each stage is described and illustrated below.[1]

We wish to note that the use of quadtree/octree methods for automatic mesh generation was pioneered by Shephard & Yerry [YERR83,YERR84]. Our work is similar to theirs, but the differences are real and important.

STAGE 1: Interior Meshing See Figure 3. The object S is enclosed in a box which is recursively subdivided into a grid whose smallest cell size determines the element size (or element density) of the initial FEM mesh; this minimal size is determined by subdividing cells until no cell contains more than one connected boundary segment of S. As the subdivision proceeds the cells are classified as being In S ("IN"), Out of S ("OUT"), or Neither In nor Out ("NIO"). Cells classified as IN at higher levels in the hierarchy are subdivided to the final grid size without further classification. The collection of IN cells constitutes the interior mesh of S.

< Figure 3 >

The main computational utility used for cell classification is the modified cell classification procedure

$$ModClassCell(cell, solid) = (\text{"IN"}, \text{"OUT"}, \text{"?"}),$$

which is described fully in [LEE82].

STAGE 2: Boundary–Region Meshing

The task here is to fill the region between the boundary of the interior mesh (denoted bIS – see Figure 4a) and the boundary bS of the solid S. Observe that

$$bS \subset (\cup\ NIO\ cells) \cup bIS$$

[1] The discussion here and in the next several sections is cast in 2–D; 3–D extensions are discussed in section 6.

Thus bS usually is contained in the NIO cells and special element–building operations are required, but sometimes segments of bS coincide with bIS (as at the top of Figure 4a) and no special processing is needed. Thus we can mesh the inter–boundary region by visiting each NIO cell and creating elements that link the bS segment passing through it to the interior of the solid.

< Figure 4 >

There are three main technical issues involved in this process: devising a systematic way to insure that all NIO cells are visited, creating nodes on bS, and associating bS–nodes with existing bIS–nodes to form valid elements. We shall discuss each of these issues briefly.

All NIO cells can be visited by an exhaustive scan of the lowest–level grid, or by tree traversal, or by traversing bS. Since no single approach seems to offer substantial advantages over the others, we use grid–scan for generating the initial mesh and, because operations tend to be more localized, tree–traversal for re–meshing and re–analysis.

Figure 4a shows exemplary bS nodes (P1, P2, P3 in Figure 4a) that are created as follows.

- Vertices of bS within each NIO cell (e.g. P2 in Figure 4a) are tagged as such and are always used as finite element nodes.
- Additional bS nodes are created by intersecting bS with the boundaries of the NIO cells (P1 and P3 in Figure 4a).

The generation of valid elements within an NIO cell is straightforward if the cell does not contain bS–vertices (corner-nodes): nodes on bS and bIS belonging to the same NIO cell are simply linked to form quadrilateral and triangular elements (see the lower left portion of Figure 4b). When a corner is present, the corner node is linked to bS and bIS nodes within the cell and templates are used to form a web of triangular elements – see Figure 4b. To avoid generating elements with poor aspect ratios, the distances between nodes are checked by using a node–neighborhood test, and closely spaced nodes are merged into single nodes on bS. Figure 5 provides an example of this process.

< Figure 5 >

The FEM mesh is complete at the end of Stage 2. A regular mesh of quadrilateral elements in the interior results from a direct mapping of IN cells. On the boundary, NIO cells are associated with quadrilateral and triangular elements. It is important to note that, the

FEM mesh inherits the spatial addressability and structure of the hierarchical grid because elements and substructures are associated with the quadrants of the original decomposition. Figure 6 shows an example of a mesh generated by our automatic procedure.

< Figure 6 >

The Shephard-Yerry (S-Y) boundary-region meshing algorithm performs in/out tests on the mid-points and quarter-points of the edges of NIO cells, and then maps each NIO cell into one of a finite number of cut-quadrant forms; each cut-quadrant is then meshed. (We avoid such geometric approximations by computing exact points of intersection on bS.) The final stages of the S-Y algorithm move nodes in NIO cells to the boundary, and then eliminate ill-formed elements by using a Lagrangian relaxation procedure to smooth a triangulated version of the entire mesh. This last operation destroys the uniform quadrilateral interior mesh and also spatial addressability – because elements are not constrained to remain in their original cells.

4 ANALYSIS OF HIERARCHICAL MESHES

This section summarizes a FEM analysis procedure that exploits the properties of the hierarchical, spatially addressable meshes described above. Recall that data specifying the finite elements in the initial mesh are accessed through the lowest level of the hierarchical grid.

One analytical simplification is immediately obvious: because the interior mesh elements are uniform, their stiffness matrices are identical if the material properties are homogeneous and thus only one stiffness matrix need be computed for all of the interior elements. Other, more important analytical simplifications accrue during both assembly and solution of the system of equations, because the hierarchical grid – which has provided spatial substructuring for meshing – can serve also as a multi-level analytical substructuring mechanism.

Assembly Procedure

Most FEM analysis procedures build a single stiffness matrix to cover the whole domain. Our Assembler builds and stores stiffness matrices for every non- OUT cell in the hierarchical grid. This is done bottom-up – see Figure 7 – by assembling son-matrices and condensing-out interior d.o.f.'s to build parent-matrices at each level. The parent nodes of the interior mesh with identical sons (uniform) yield identical substructures, hence need be assembled only once. (The mesh generator tags identical interior-mesh nodes at all levels of the tree to facilitate this.)

< Figure 7 >

Figure 4.2 shows an initial mesh and substructures at various levels in the assembly process. Note in Figure 4.2 a that the initial mesh contains some higher–level substructures; these arise not from assembling lowest–level IN –elements, but from intermediate–level cells that were classified as IN and tagged as substructures during Stage–1 meshing. (The identical stiffness matrices for lowest–level IN –cells are needed in the assembly process only when IN –elements must be assembled with elements in NIO cells.)

< Figure 8 >

Solution Procedure

Figure 9 illustrates various stages in the solution process. After loads and boundary conditions are attached to the root structure, the Solver computes the displacements of all nodal points on the boundary (i.e. the nodal points of the root substructure – see Figure 9a) and then traverses down the tree, recovering displacements of substructure nodes at each level. The displacements at all levels are saved in data records accessed through the hierarchical grid, and the lowest–level displacements are used to compute the stresses in the elements.[2]

< Figure 9 >

Remarks on the Assembly and Solution Procedures

Our experience to date with this substructuring approach to analysis indicates the following.

- The hierarchical grid used for mesh generation has almost all of the data management facilities needed for analytical substructuring.
- The computing time and storage requirements for internal–element assembly are substantially reduced.
- We conjecture that our substructuring technique is asymptotically more effecient than the methods used in standard solvers. Preliminary result that support our conjecture will be reported in [KELA87].

[2] All analysis presented here are linear–static, based on linear isoparametric elements.

- Substructuring based on trees lends itself naturally to parallel (computer) processing.

More broadly our particular approach to substructuring seems promising for non-linear as well as linear analysis. In many practical problems (e.g. contact problems, fracture mechanics, localized plasticity), non-linear behavior occurs in isolated regions, and spatially localized analytical methods should prove to be efficient. (For example: during analysis regions that become non-linear can be tagged in the grid and handled specially.) In other types of problems one may want displacements and stresses only in small critical regions, and here again spatially localized methods seem very appropriate.

5 SELF-ADAPTIVE INCREMENTAL ANALYSIS

In this section we discuss first the techniques used for managing mesh refinement and incremental analysis, and then an error-driven algorithm for closing the feed-back loop in Figure 1.

5.1 Refinement and Re-Analysis

Assume that (1) a mesh has been constructed at the lowest level of the grid, (2) the mesh has been analyzed and the results stored in the grid and (3) evaluation of the results (discussed in the next subsection) has indicated that refinement is needed in a particular spatial region.

Two avenues for refinement are available: h-refinement and p-refinement. In p-refinement successively higher-order shape functions are assigned to the element formulation. To refine a particular element, the old stiffness matrix for the element is invalidated and a new matrix is computed from the new shape function. No new tree-nodes are generated, but the size of the stiffness matrix increases.

In h-refinement, existing elements are subdivided into smaller elements of the same type. To improve the geometric accuracy, localized h-refinement is done on the original geometric model rather than on the current finite element approximation. Thus to refine a particular element, one deletes the element, creates and classifies new vertices and nodes, and inserts the smaller new elements into the grid. Discontinuities of displacements along edges where smaller elements abut on larger elements are avoided by using constraint equations.

< Figure 10 >

Figure 10 shows examples of localized refinement. Note that successive h-refinements improve the geometric approximation of the original solid. A maximum cross-element grading of 2:1 is maintained during refinement.

Storage for the new entities created by h-refinement could be provided by adding a whole new bottom layer to the grid, but this would be wasteful unless very extensive h-refinement is needed. If the h-refinements are sparse, small localized explicit schemes or linked-list methods are more efficient.

Assume now that the original mesh has been refined in a few regions using the methods just described, that the affected elements have been tagged, and that the refined mesh is to be re-analyzed. Clearly one wants to do incremental analysis, i.e. to use partial results from the earlier analysis insofar as possible. These results are available through the hierarchical grid; for example, a tree of K-matrices will exist – see Figure 7.

The incremental Assembler traverses the tree and by examining the sons of each parent node, detects new offspring and computes the appropriate stiffness matrices (Figure 11). Stiffnesses for unmodified elements are recovered from storage, and new and old stiffnesses are combined to form a modified substructure. If a node has no new offspring, the complete old substructure is reused. The incremental Solver works similarly, inspecting tags on data to distinguish valid and invalid old results and reusing the former whenever possible.

5.2 Self-Adaptive Algorithm

Our current algorithm for controlling self-adaptive incremental analysis operates as follows (see Figure 1). After a mesh (either initial or refined) has been analyzed, error indicators are computed for each element together with an estimate of the global error. If the global error exceeds a pre-specified limit, the systems calls for refinement and reanalysis in regions having large local errors. This process continues automatically until the global error estimate falls below the pre-specified limit.

Thus far we have done little research on errors per se, and our current error measures are crude. As in [KELL83], our element error-indicator (ϵ_i) is merely the average of the stress jumps (J_s : normal and tangential) across each of the element's edges with dimension (h) and assuming linear isoparametric elements

$$\epsilon_i^2 = \frac{1-\nu}{E}\frac{h}{24}\int_{\Gamma_k} J_s^2 d\tau$$

normalized by the strain energy of the displaced model. Our global error estimator is simply the sum of the element error indicators. Figure 10c shows the computed values of the element error indicators for a sample problem. Note that, in the vicinity of the hole and around the re-entrant corner the data imply high stress gradients because the error indicators are high. Figure 10d shows an automatic refinement resulting from this set of error indicators.

An obvious improvement to the current algorithm: replace the single global error indicator with a hierarchical series of regional error indicators. These can be computed bottom–up in the tree, and should force selective refinement in cases where the overall (average) error is small but errors in small regions are high.

6 AUTOMATIC MESHING FOR 3-D PROBLEMS

In this section we present the algorithms that we are currently developing to extend to 3-D problems the automatic meshing procedure described in Section 3. Since our work is based on the octree generator built in the PADL-2 solid modelling system [HART83,KELA84], stage 1 of meshing – which includes (i) boxing the domain, (ii) subdividing the box into octal cells, (iii) classifying the cells as IN, OUT and NIO, and (iv) further subdividing and reclassifying NIO cells until a minimal level of subdivision is reached – is virtually completed. Figure 11 shows the interior octree for a PADL defined solid.

< Figure 11 >

Stage 2 involves associating each of the NIO cells (represented by the intersection of the solid with a grid-level octant) to a valid finite element topology. Before being decomposed into elements, NIO cells are classified as Simple (SNIO) or Complex (CNIO). SNIO cells, formed by the intersection of the grid-level octant with a single "cutting" surface, are topologically simple, as shown in Figure 12. CNIO cells, on the other hand, intersect the boundary surface and also contain vertices and edges coming from the solid's boundary. A typical CNIO cell is illustrated in Figure 13. Due to the differences in their geometry and topology, the decomposition of SNIO and CNIO cells proceeds along two different avenues.

< Figure 12 >

< Figure 13 >

Decomposition of SNIO cells

Since the number of possible configurations of SNIO cells is inherently limited, SNIO cells can be decomposed into finite elements by associating the cell to an appropriate template containing a mesh topology. Specifically, the number of possible cases is restricted to seven (the number of vertices of the original octant shaved off by the cutting surface identifies the appropriate template – **Figure 14**).

< Figure 14 >

The topolgies embedded in the templates are not unique and include hexahedral, wedge, pyramid and tetrahedral isoparametric linear elements. However, as explained further on in this section – care has been taken in producing mesh topologies that, whenever possible, associate each uncut octant faces to a quadrilateral face of a hexahedral, wedge or pyramid element. We note, finally, that (a) in general, most of the NIO cells are classified as SNIO, and (b) SNIO decomposition is computationally inexpensive.

Decomposition of CNIO cells

The topological description of CNIO cells is not confined to a limited number of possible configurations. Hence, in this case mapping is of little use and the automatic decomposition of the cells can be done only by recursive element extraction. We are currently implementing a family of operators – based on the approach in [WOO84], Figure 15 – that works on the boundary representation (Brep) of the polyhedron associated with the CNIO cell. Because of the complexity of the operations involved – (i) scan the topological information contained in the Brep to identify a candidate element, (ii) verify the validity of the element, and (iii) extract the element and update the Brep – CNIO decomposition is considerably more expensive than template matching.

< Figure 15 >

Elements for 3-D analysis

The family of linear isoparametric elements used in the above decomposition schemes can be generated by collapsing a standard 8-node isoparametric brick element. Note that the use of pyramids is mandated by the necessity of preserving a regular interior mesh of hexahedral elements, whenever tetrahedral elements are introduced in the proximity of the boundary. Pyramids allow interfacing triangular sides belonging to tetrahedral or wedge elements with quadrilateral faces of hexahedral elements without introducing discontinuities in the displacement field.

7 DISCUSSION

Advantages

The main advantage we see is that mesh generation and mesh analysis are integrated and, in effect, collaborate under the control of the error evaluator. Thus the mesher only refines regions where refinement is needed, and the analyzer only computes "what's new" about a refined mesh. This type of efficient adaptive behavior is, in our opinion, the key to efficient automatic finite–element analysis.

Hierarchical substructuring is the driving principle in both the mesh generator and mesh analyzer.[3] It seems to be a very powerful principle of divide–and–conquer genre, in that it enables hard problems (object decomposition, equation–set solution) to be decomposed into smaller, tractable problems via spatial partitioning.

Open Issues

We cite four sets of issues that will require extensive theoretical work.

1. Error measures and indicators: measures better than the ones we use currently are needed, especially for 3-D work.

2. Adaptive convergence: the convergence behavior of the self-adaptive process must be investigated (strong convergence properties are required for a truly automatic system).

3. Computational complexity: preliminary results let us conjecture that hierarchical substructuring techniques are asymptotically more efficient than the methods used

[3] The hierarchical tree might be viewed as a generalization of the structure described in [RHEI80]. However, the latter is applied in subdomains that are mapped to regular figures (squares and triangles), and Rheinboldt's tree addresses the element partitioning induced in the regular figures. By avoiding mapping we are able to use the same structure for both meshing and analysis; further, the regularity of our structure permits systematic cell numbering and, hence, data access through calculated addresses rather than through searching or table lookup.

in standard solvers, but an in-depth study is needed to prove/disprove our conjecture.

4. Non-linear analysis: our approach to substructuring appears promising for non-linear analysis.

While the issues above are certainly important in the long term, in the immediate future one other issue – completing the extension of our meshing and analysis system to 3-D problems – is more pressing. The current status of 3-D work is as follows :

- The 2-D spatial substructuring techniques for managing analysis and adaptive re-meshing and re-analysis extend gracefully to 3-D, and indeed most of the 2-D control code is directly usable in 3-D.
- The major open issues lie in Stage 2 of the automatic meshing procedure, specifically in decomposition of CNIO cells. A promising approach, based on a family of element extractors, is currently being implemented.

In summary , we believe that hierarchical substructuring as embedded in the experimental system described here represents an important contribution on the road to genuinely automatic finite element analysis.

ACKNOWLEDGEMENTS

Herb Voelcker, former director of the Production Automation Project and currently at Cornell University, contributed extensively to this research. Also we acknowledge the contribution and the encouragement of John Goldak, of Carleton University, and of Vic Genberg, of the Eastman Kodak Company. The computer-output displays were produced on equipment donated by Tektronix, Inc., and other Industrial Associate companies of the Production Automation Project provided both equipment and funds for the work. Sustaining support was provided by the National Science Foundation under Grant(s) ECS-8104646 & DMC-8403882. The findings and opinions expressed here those of the authors and do not necessarily reflect the views of the various sponsors.

REFERENCES

[BABU78] I. Babuska and W. C. Rheinboldt, "A-posteriori error estimates for the finite element method", INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, vol. 112, pp. 1597-1615, 1978.

[CAVE85] J. C. Cavendish, D. A. Field and W. H. Frey, "An approach to automatic three-dimensional finite element mesh generation", INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, vol. 21, pp. 329-347.

[GAGO83] J. P. De S. R. Gago, D. W. Kelly, O. C. Zienkiewicz and I. Babuska, "A posteriori error analysis and adaptive processes in the finite element method: Part II – Adaptive mesh refinement", INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, vol. 19, pp. 1621–1656, 1983.

[HART83] E. E. Hartquist, "Public PADL–2", IEEE COMPUTER GRAPHICS & APPLICATIONS, vol. 3, no. 7, pp. 30-31, October 1983.

[JACK80] C. L. Jackins and S. L. Tanimoto, "Oct–trees and their use in representing three-dimensional objects", COMPUTER GRAPHICS & IMAGE PROCESSING, vol. 4, no. 3, pp. 249–270, November 1980.

[KELA84] A. Kela, "Programmers guide to the PADL–2 octree processor output system", INPUT/OUTPUT GROUP MEMO. No. 15; Production Automation Project, University of Rochester; January 1984.

[KELA87] A. Kela, "Automatic finite element mesh generation and self–adaptive incremental analysis through solid modeling", Ph. D. Dissertation, Production Automation Project, University of Rochester, 1987.

[KELL83] D. W. Kelly, J. P. De S. R. Gago, O. C. Zienkiewicz and I. Babuska, "A posteriori error analysis and adaptive processes in the finite element method: Part I – Error analysis", INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, vol. 19, pp. 1593–1619, 1983.

[LEE82] Y. T. Lee and A. A. G. Requicha, "Algorithms for computing the volume and other integral properties of solids: Part II – A family of algorithms based on representation conversion and cellular approximation", COMMUNICATIONS OF THE ACM, vol. 25, no. 9, pp. 642–650, September 1982.

[PEAN79] A. G. Peano, A. Pasini, R. Riccioni and L. Sardella, "Adaptive approximation in finite element structural analysis", COMPUTER & STRUCTURES, vol. 10, pp. 332–342, 1979.

[PERU82] R. Perucchio, A. R. Ingraffea and J. F. Abel, "Interactive comuter graphic preprocessing for three-dimensional finite element analysis", INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, vol. 18, pp. 909–926, 1982.

[REQU83] A. A. G. Requicha and H. B. Voelcker, "Solid modelling: Current status and research directions", IEEE COMPUTER GRAPHICS & APPLICATIONS, vol. 3, no. 7, pp. 25–37, October 1983.

[RHEI80] W. O. Rheinboldt and C. K. Mesztenyi, "On a data structure for adaptive finite element mesh refinements", ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE, vol. 6, no. 2, pp. 166–187, June 1980.

[SUHA74] J. Suhara and J. Fukuda, "Automatic mesh generation for finite element analysis", in ADVANCES IN COMPUTATIONAL METHODS IN STRUCTURAL MECHANICS AND DESIGN, J. T. Oden, R. W. Clough and Y. Yamadoto eds., Univ. of Alabama Press, pp. 607-624, 1974.

[WOO84] T. C. Woo and T. Thomasma, "An algorithm for generating solid elements in objects with holes", COMPUTERS & STRUCTURES, vol. 18. no. 2, pp. 333-342, 1984.

[WŌRD84] B. Wordenweber, "Finite element mesh generation", COMPUTER-AIDED DESIGN, vol. 16. no. 5, pp. 285-291, September 1984.

[YERR83] M. A. Yerry and M. S. Shephard, "A modified quadtree approach to finite element mesh generation", IEEE COMPUTER GRAPHICS & APPLICATIONS, vol. 3, no. 1, pp. 39–46, January/February 1983.

[YERR84] M. A. Yerry and M. S. Shephard, "Automatic three–dimensional mesh generation by the modified–octree technique", INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, vol. 20, pp. 1965–1990, 1984.

LIST OF FIGURES

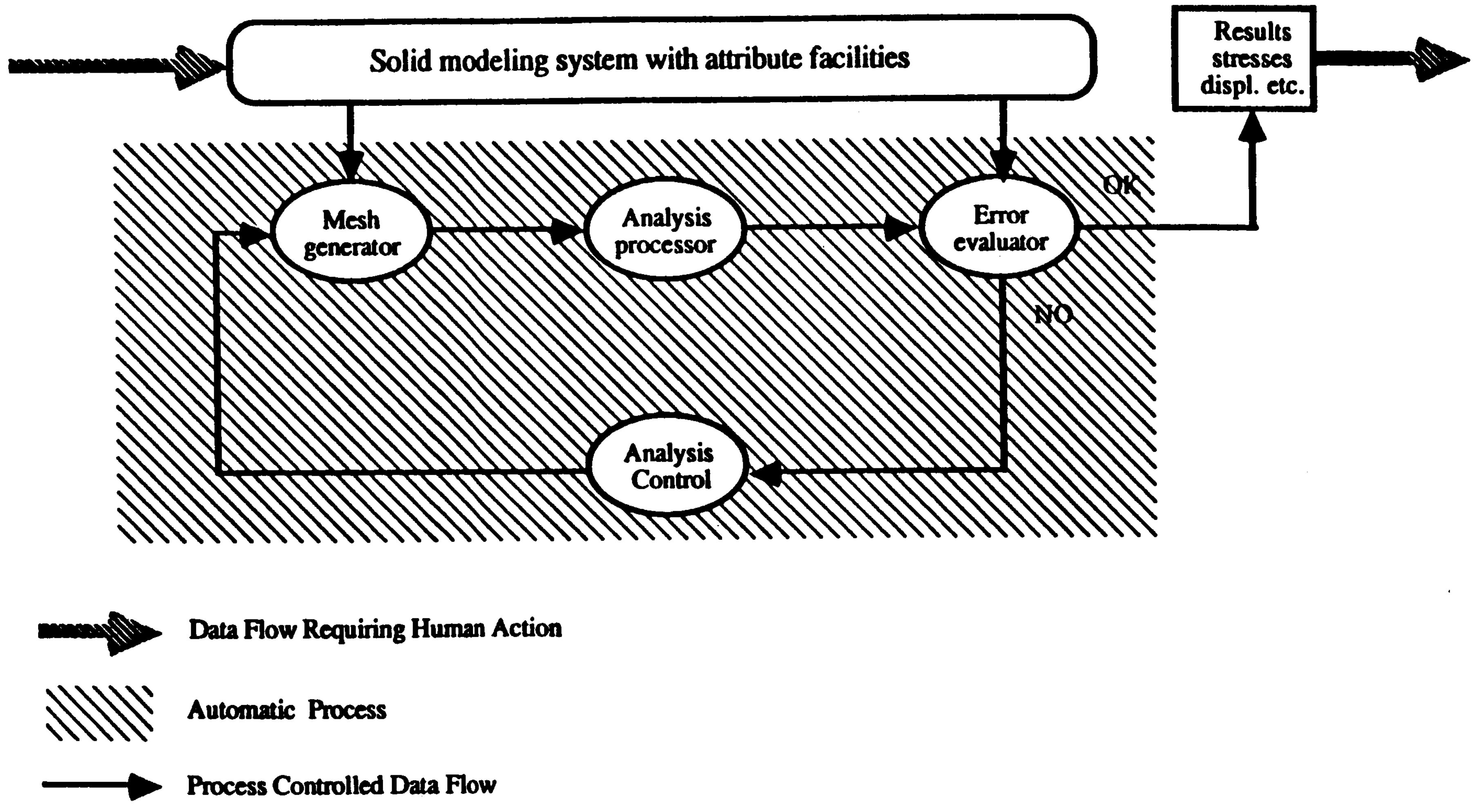

Solid modeling system with attribute facilities
Results stresses displ. etc.
Mesh generator
Analysis processor
Error evaluator
OK
NO
Analysis Control
Data Flow Requiring Human Action
Automatic Process
Process Controlled Data Flow

Figure 1

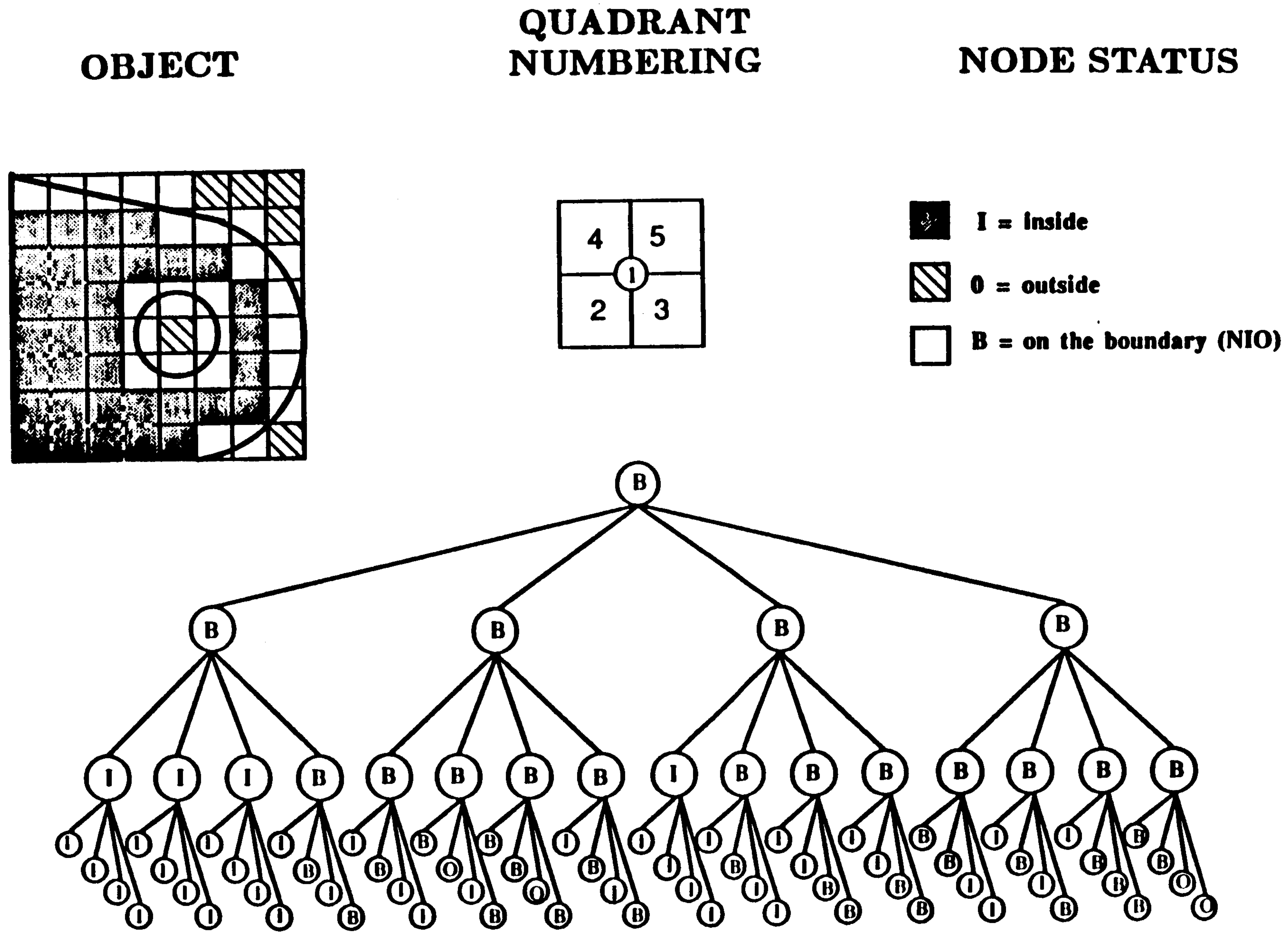

OBJECT
QUADRANT NUMBERING
4
5
1
2
3
NODE STATUS
I = inside
0 = outside
B = on the boundary (NIO)

Figure 2

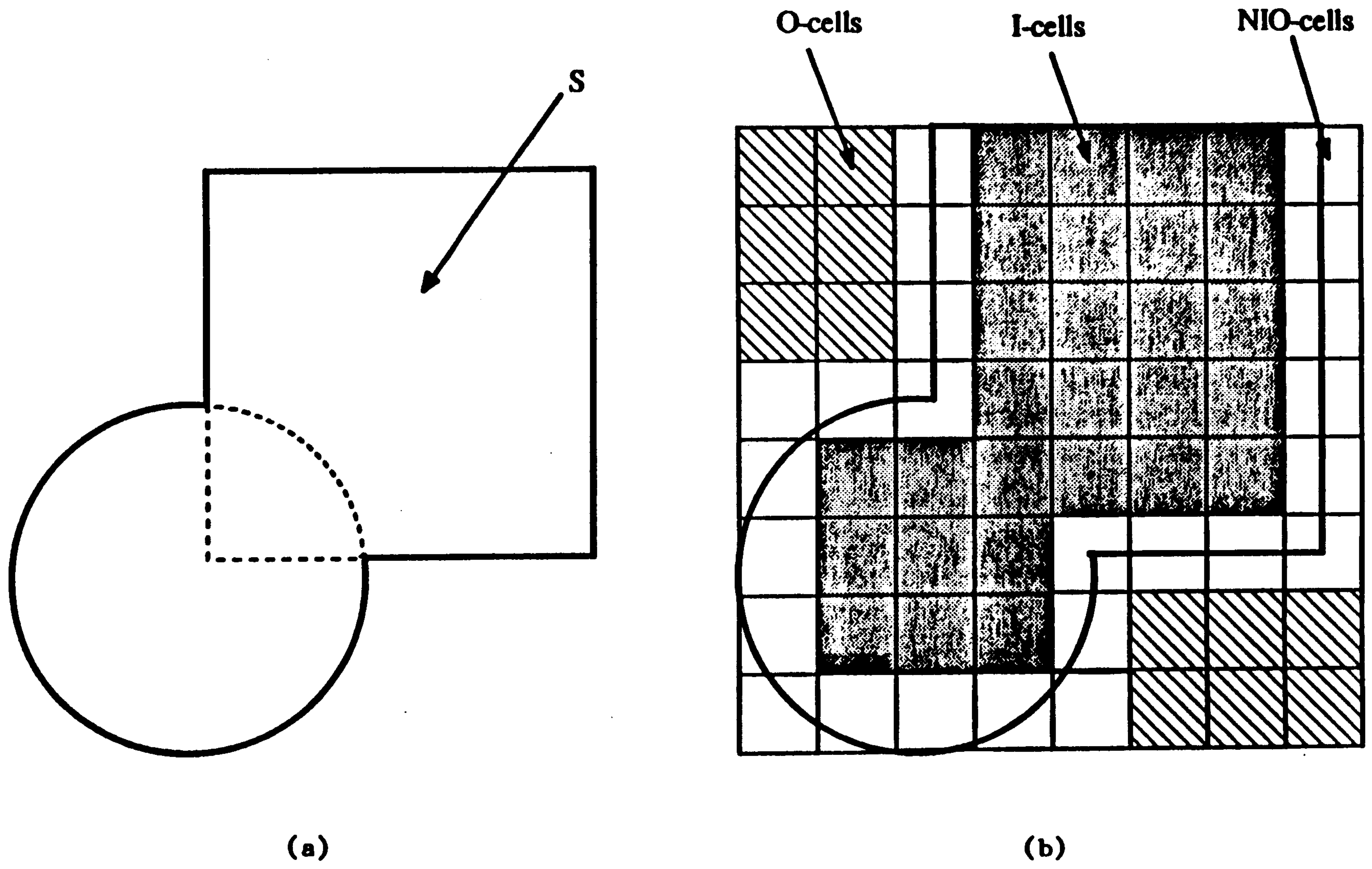
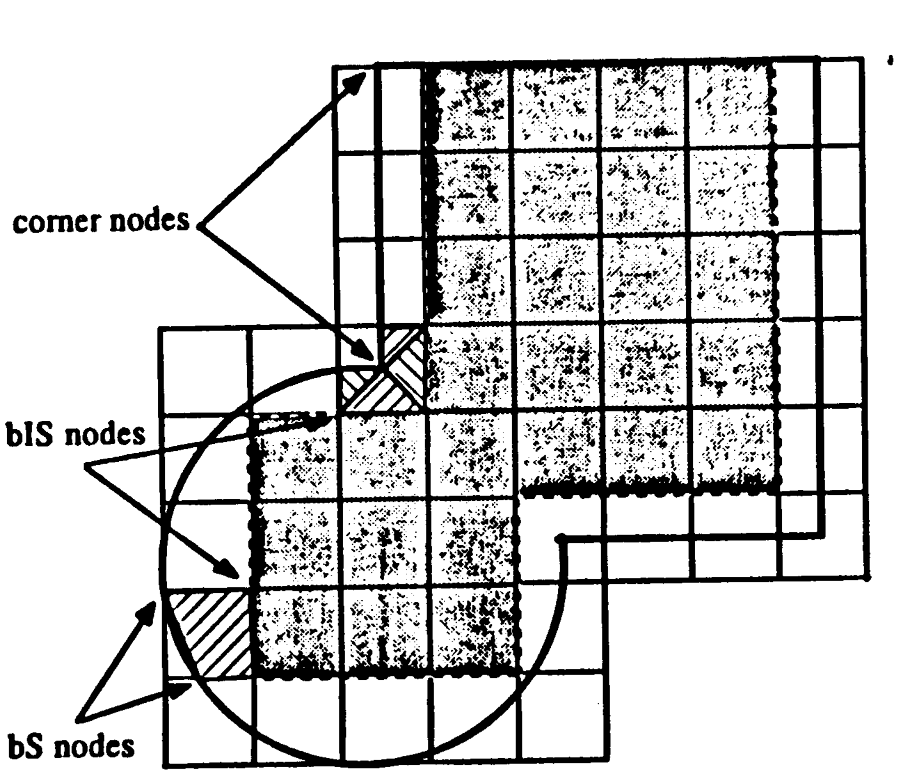


Figure 3



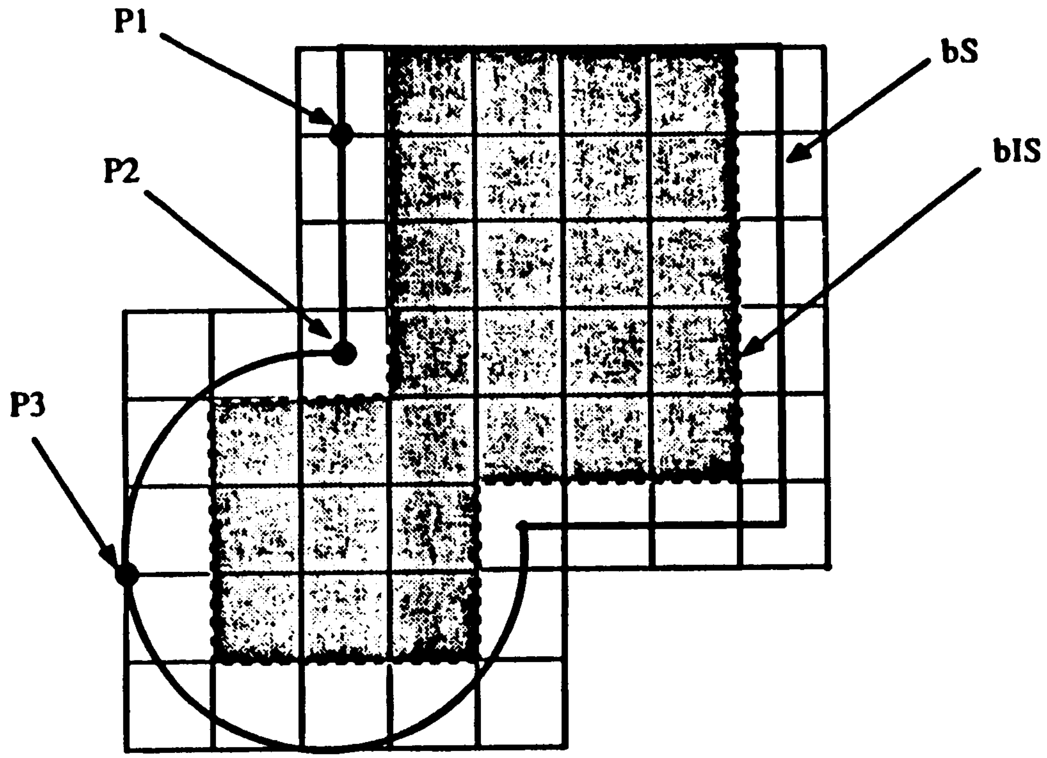


Figure 4

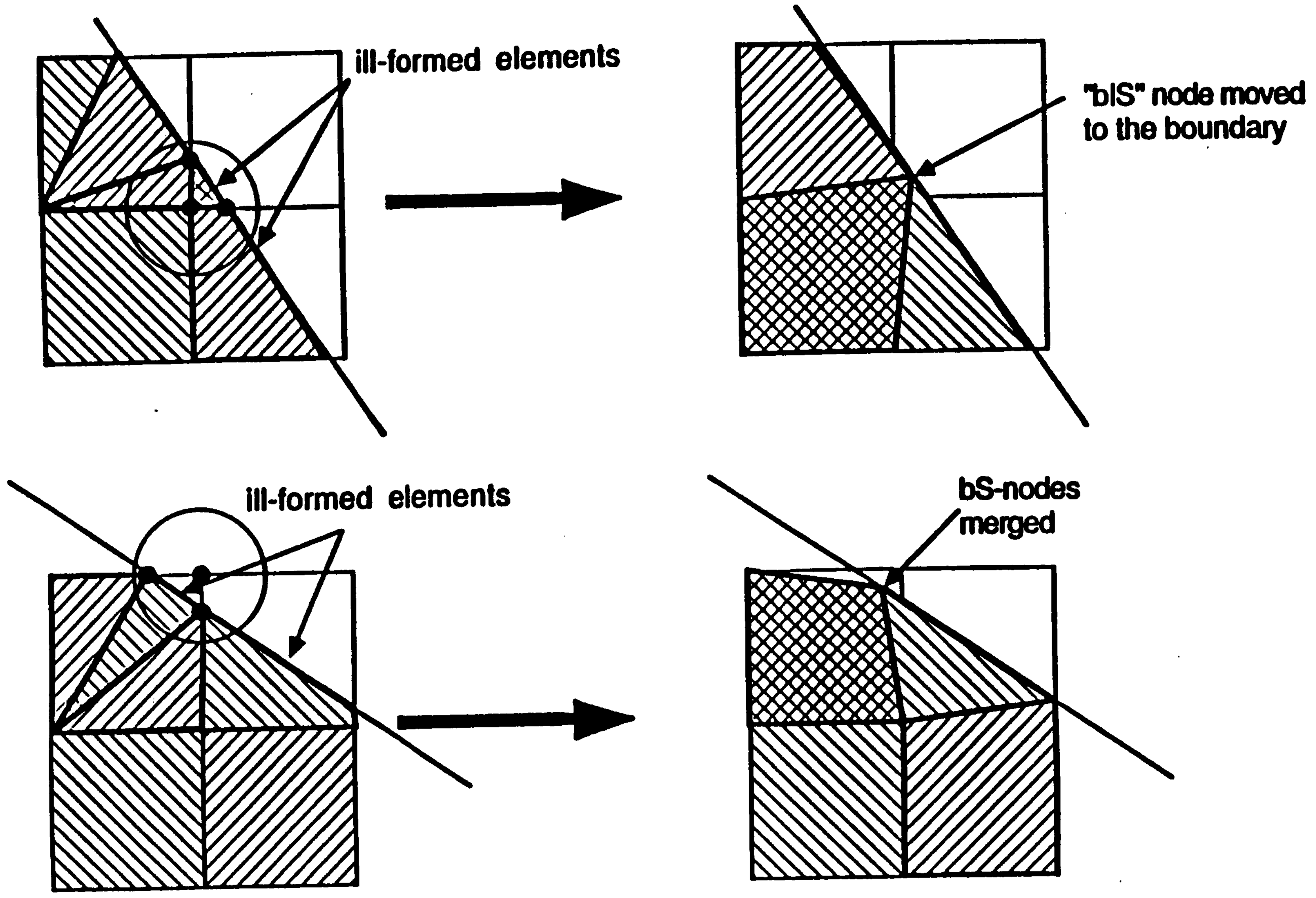

ill-formed elements
"bIS" node moved
to the boundary
ill-formed elements
bS-nodes
merged

Figure 5

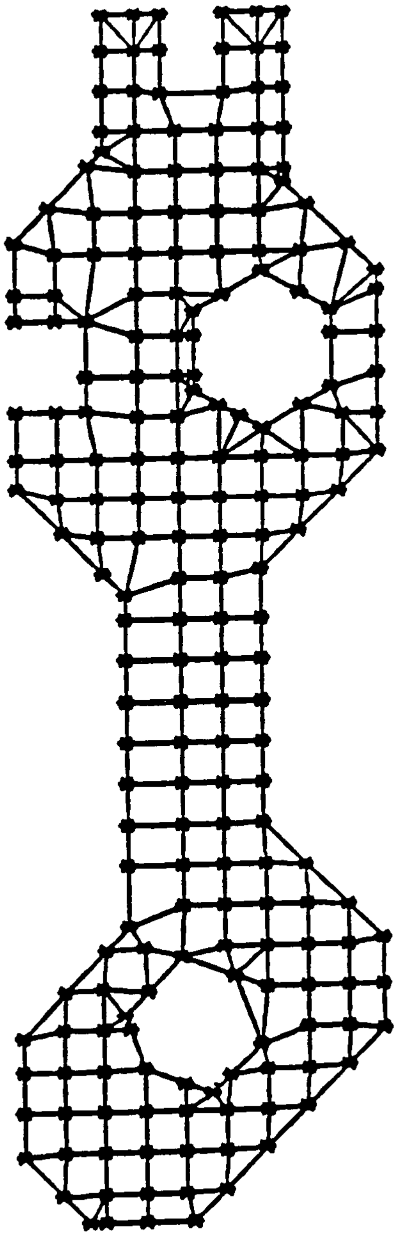


Figure 6

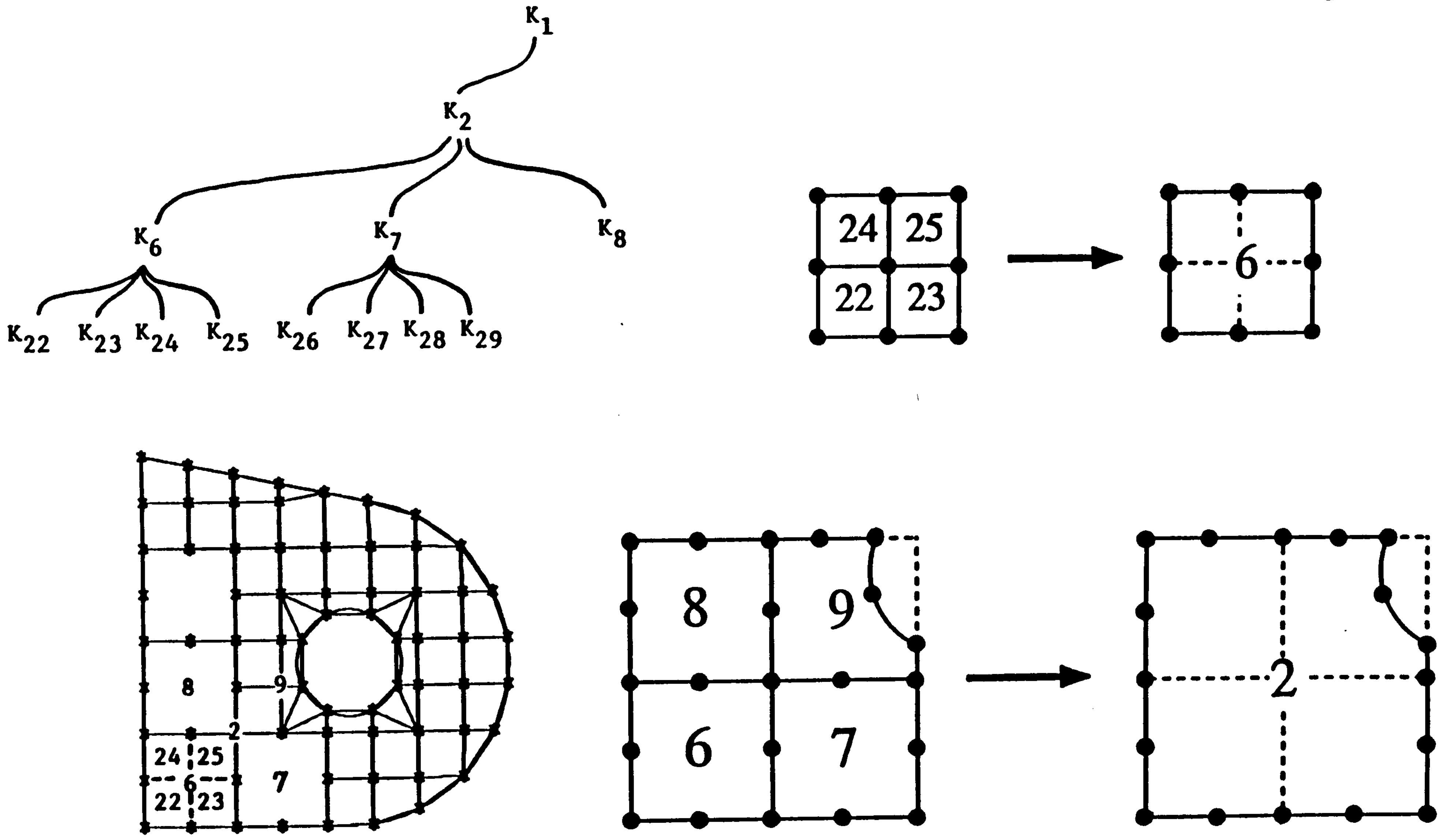

K_1
K_2
K_6
K_7
K_8
K_{22}
K_{23}
K_{24}
K_{25}
K_{26}
K_{27}
K_{28}
K_{29}
24
25
22
23
6
8
9
6
7
2
24
25
6
22
23
7
8
9
2

Figure 7

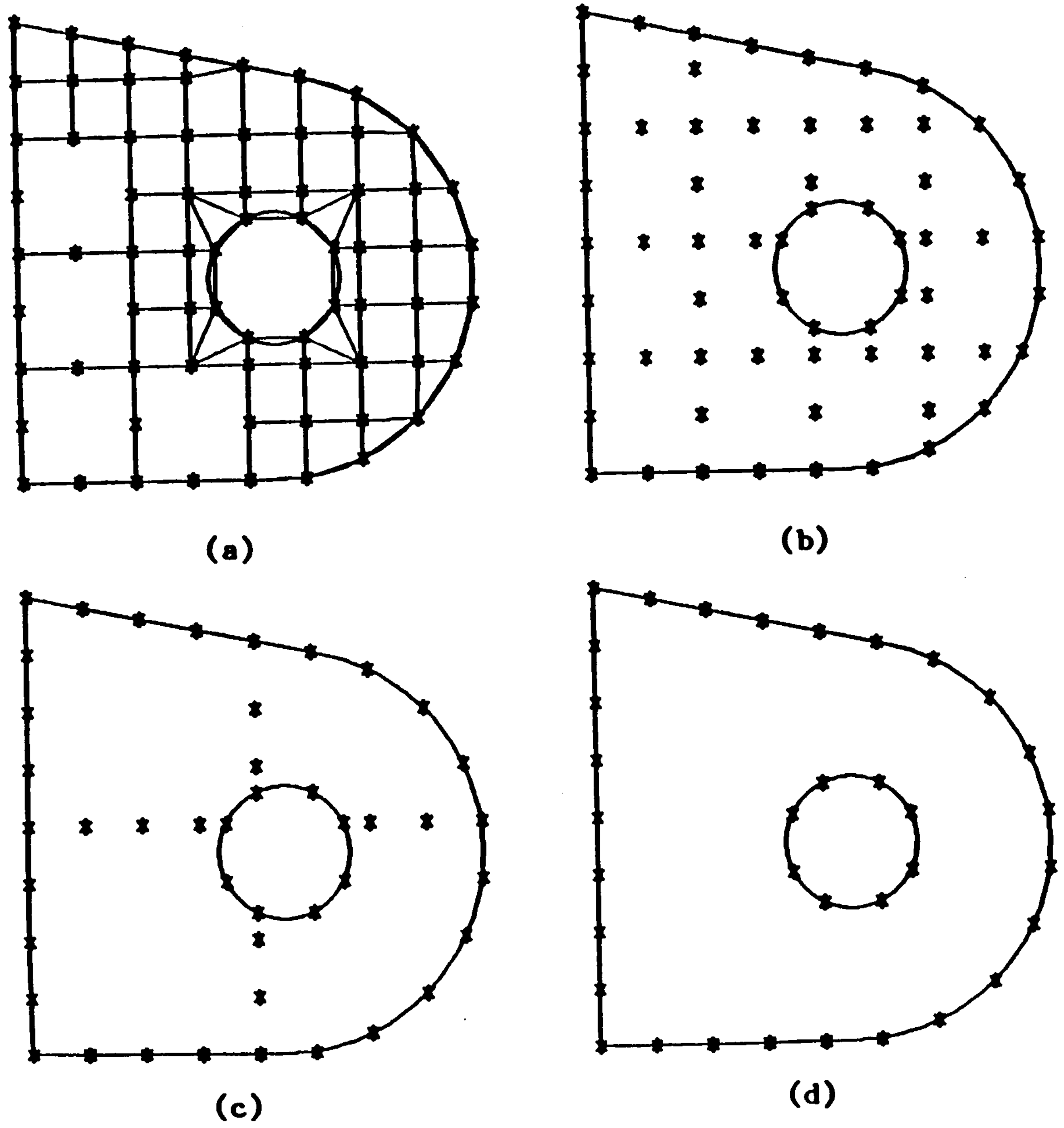


Figure 8

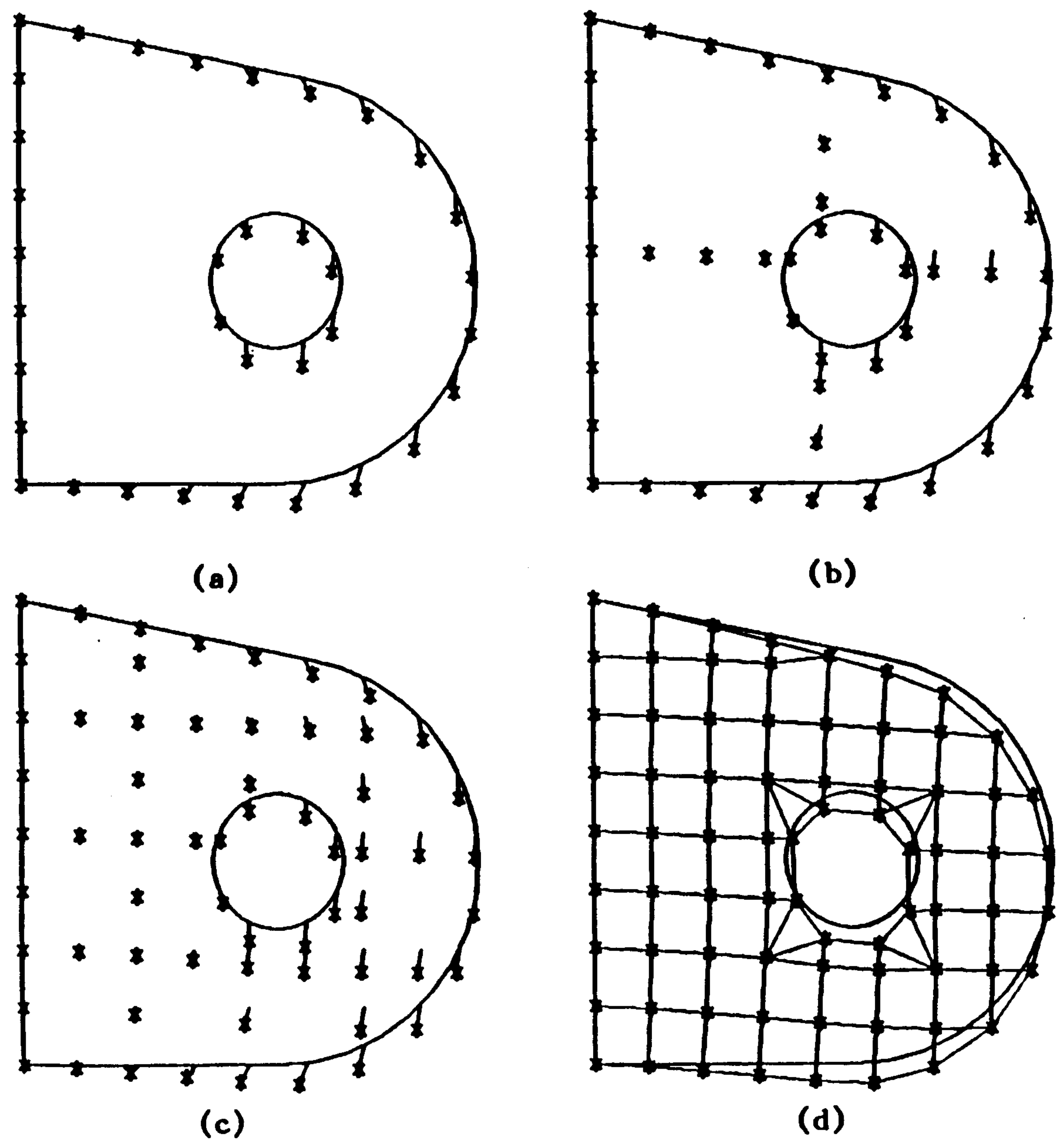


Figure 9

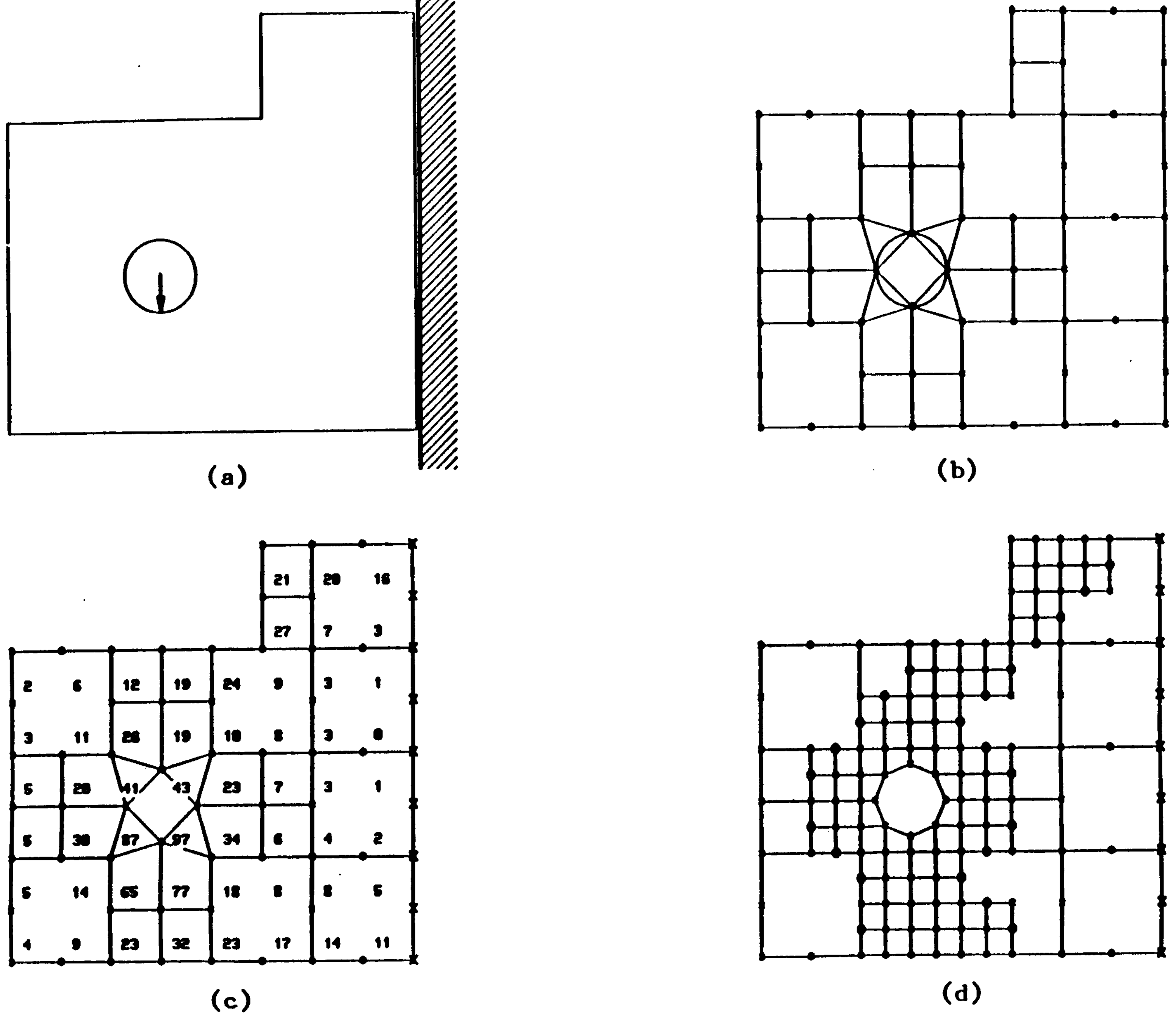

(a)
(b)
21 20 16
27 7 3
2 6 12 19 24 9 3 1
3 11 26 19 10 8 3 8
5 20 41 43 23 7 3 1
5 30 87 97 34 6 4 2
5 14 65 77 18 8 8 5
4 9 23 32 23 17 14 11
(c)
(d)

Figure 10

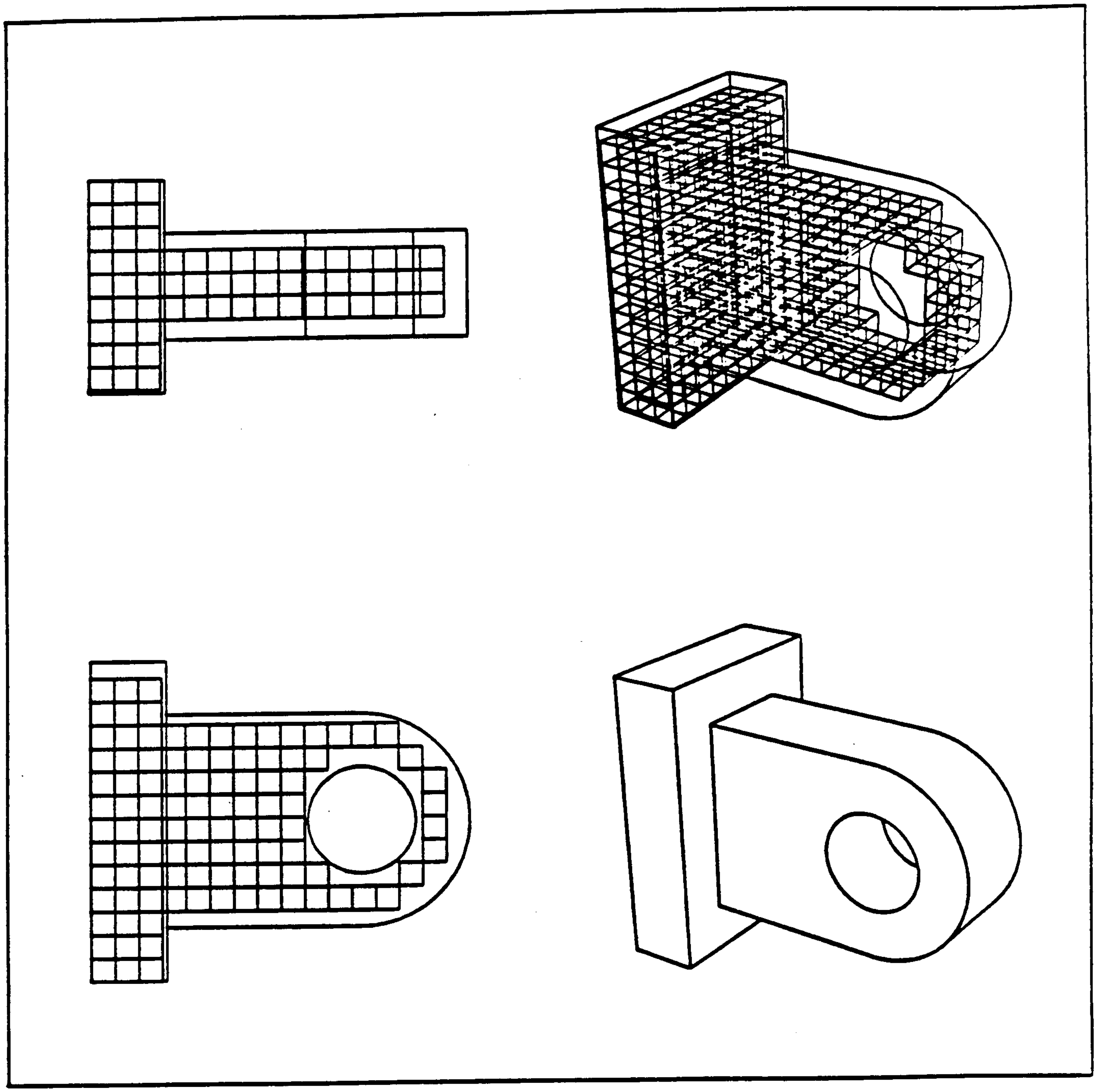


Figure 11

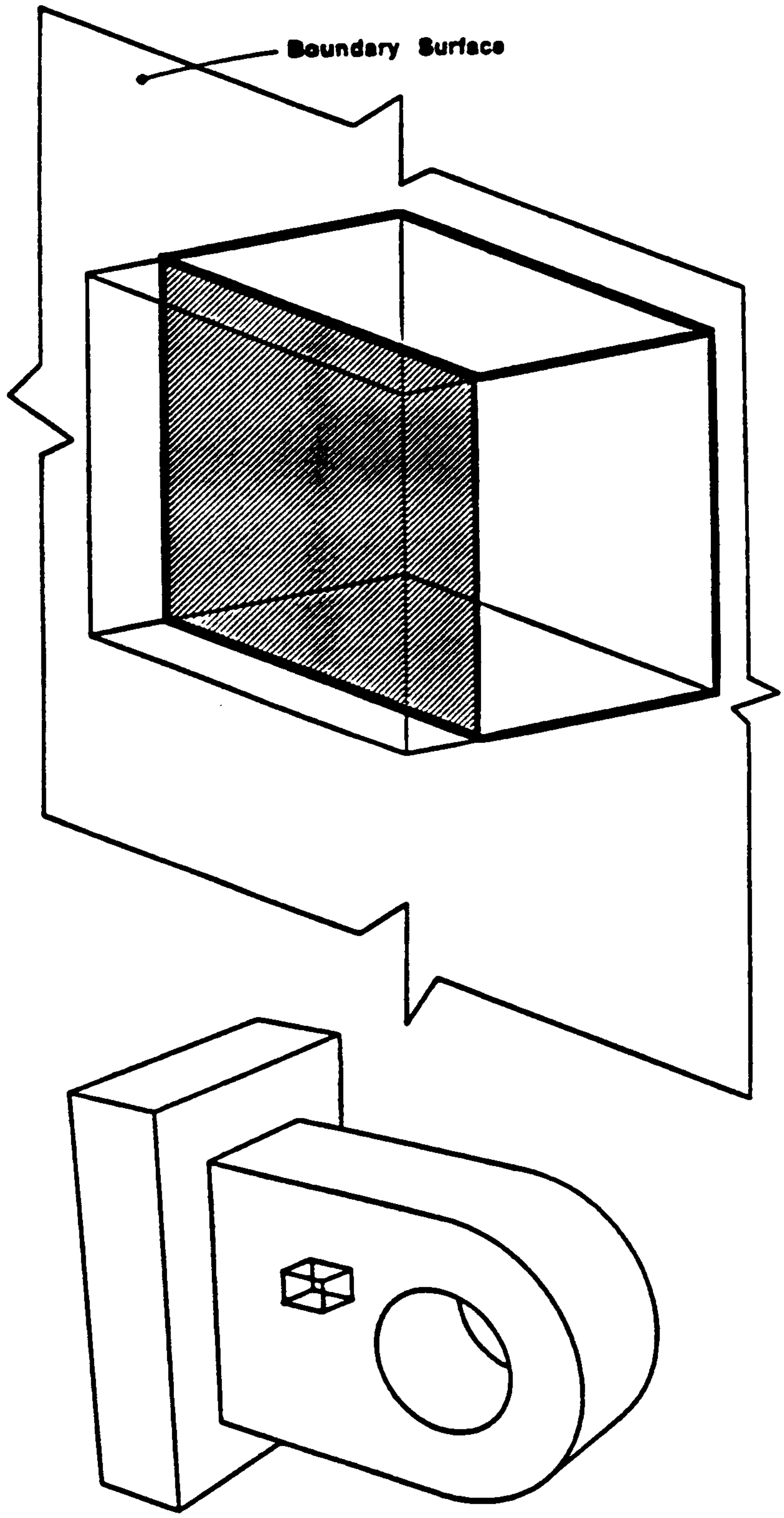
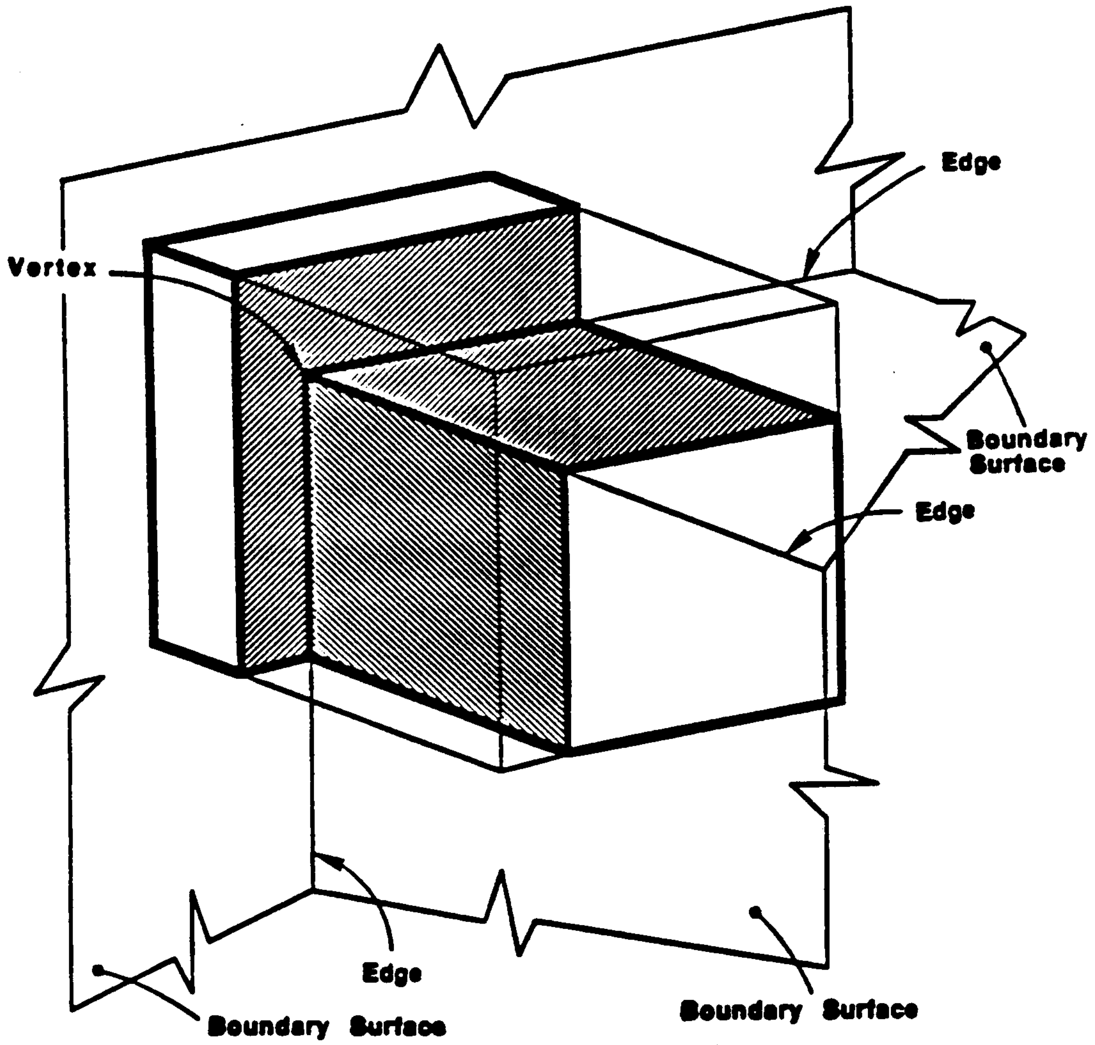


Figure 12


Edge
Vertex
Boundary Surface
Edge
Edge
Boundary Surface
Boundary Surface

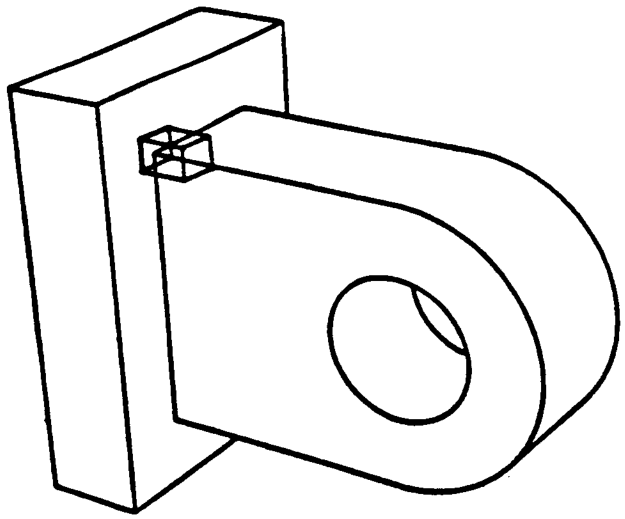
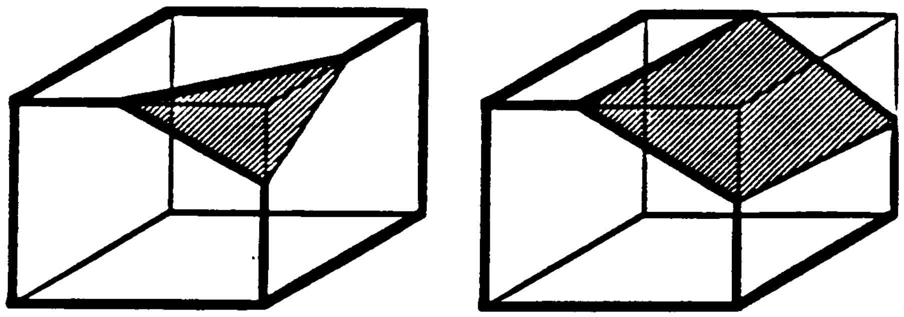


Figure 13

SNIO Cells



Template-derived F.E. Topologies

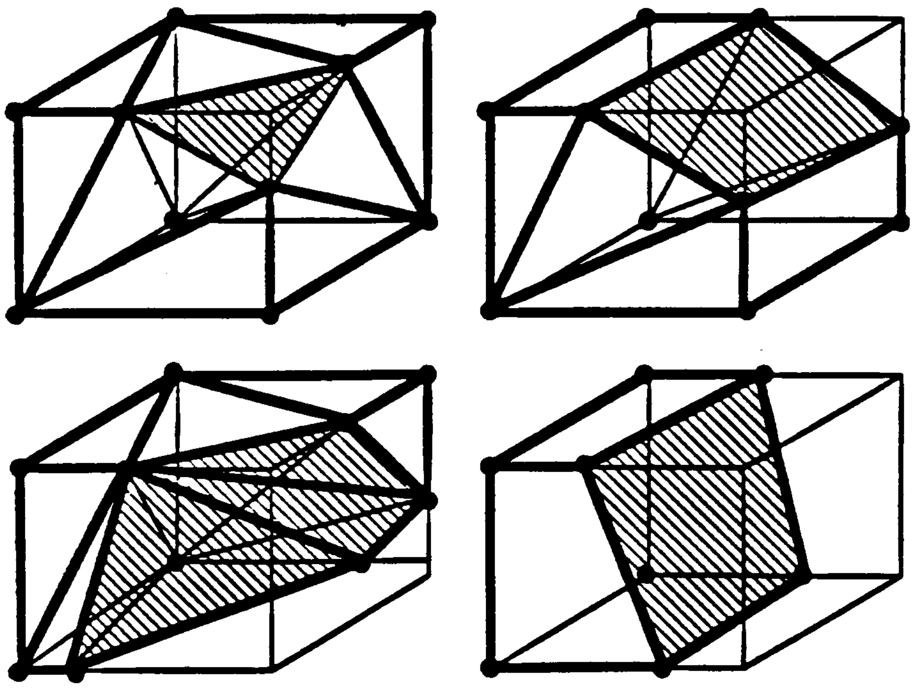
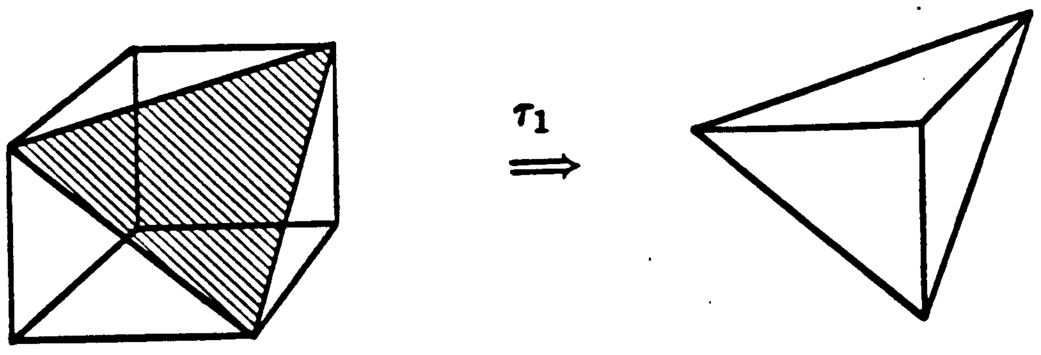
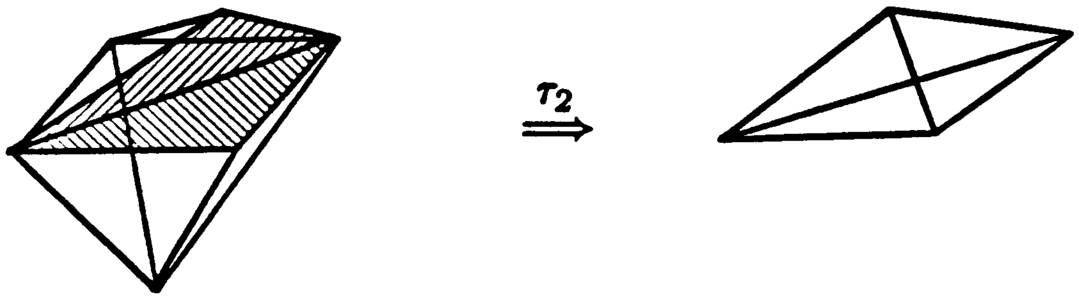


Figure 14

- Extraction of tetrahedra :
  Operators τ_1 and τ_2 (Woo & Thomasma, 1983).





- Extraction of pyramids : operator τ_3.

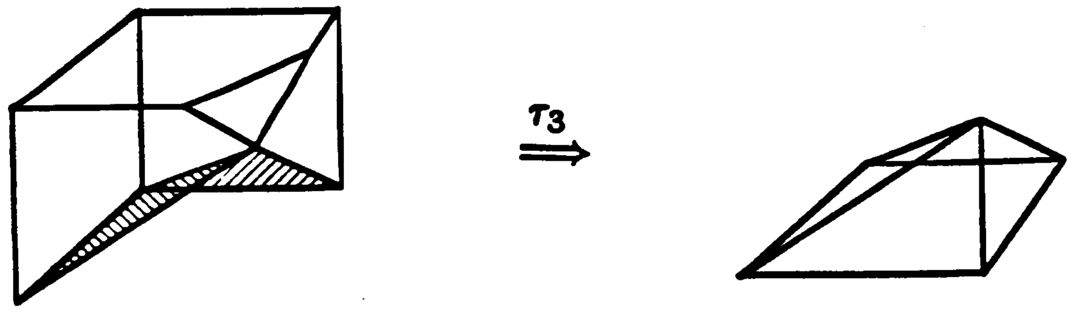


Figure 15