Engineering with Computers (1996) 12: 144–154

Engineering with Computers

Two and Three-Quarter Dimensional Meshing Facilitators

[1]M. Whiteley, [1]D. White, [1]S. Benzley and [2]T. Blacker

[1] Brigham Young University, Provo, USA; [2] Sandia National Labs, Albuquerque, USA

Abstract. *This paper presents generated enhancements for robust 'two and three-quarter dimensional meshing', including: (1) automated interval assignment by integer programming for submapped surfaces and volumes, (2) surface submapping, and (3) volume submapping. An introduction to the simplex method, an optimization technique of integer programming, is presented. Simplification of complex geometry is required for the formulation of the integer programming problem. A method of 'i–j unfolding' is defined which explains how irregular geometry can be realigned into a simplified form that is suitable for submap interval assignment solutions. Also presented is the processes by which submapping eliminates the decomposition of surface geometry, through a pseudo-decomposition process, producing suitable mapped meshes. The process of submapping involves the creation of 'interpolated virtual edges', user defined 'vertex types' and 'i–j–k space' traversals. The creation of 'interpolated virtual edges' is the method by which submapping automatically subdivides surface geometry. The 'interpolated virtual edge' is formulated according to an interpolation scheme using the node discretization of curves on the surface. User defined 'vertex types' allow direct user control of surface decomposition and interval assignment by modifying 'i–j–k space' traversals. Volume submapping takes the geometry decomposition to a higher level by using 'mapped virtual surfaces' to eliminate decomposition of complex volumes.*

Keywords. Automatic mesh generation; Mesh control; Mapped meshing

1. Introduction

The finite element method, although powerful and versatile, has long been plagued with the problem of effective discretization of geometry. The need for more powerful meshing algorithms continues, but no one technique has appeared which can fill the needs of all meshing tasks. Therefore, various mesh generation research efforts continue in an attempt to automate these meshing processes (Fig. 1). The transformation of a two dimensional (2D) mesh into a three dimensional (3D) volume and shell elements has long been a useful tool in the arsenal of algorithms available for 3D meshing [1]. A 2D mesh simply extruded in some general third dimension produces what is termed a two and one half dimensional (2.5D) volume mesh. However, any number of these 2.5D pieces can be combined to form extremely complex meshes as shown in Fig. 2. As can be seen in this figure, these pieces can be extruded in logically orthogonal directions with respect to each other. We have termed this general meshing method of combining orthogonally independent 2.5D pieces as being two and three quarter dimensional (2.75D).

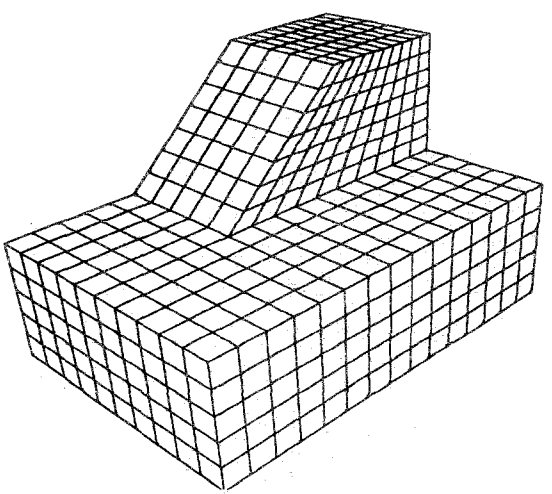

Fig. 1. Mesh generation using automated interval assignment and submapping tools.

Correspondence and offprint requests to: Dr D. R. White, Sandia National Laboratories, PO Box 5800, Albuquerque, NM 87185-0441, USA.

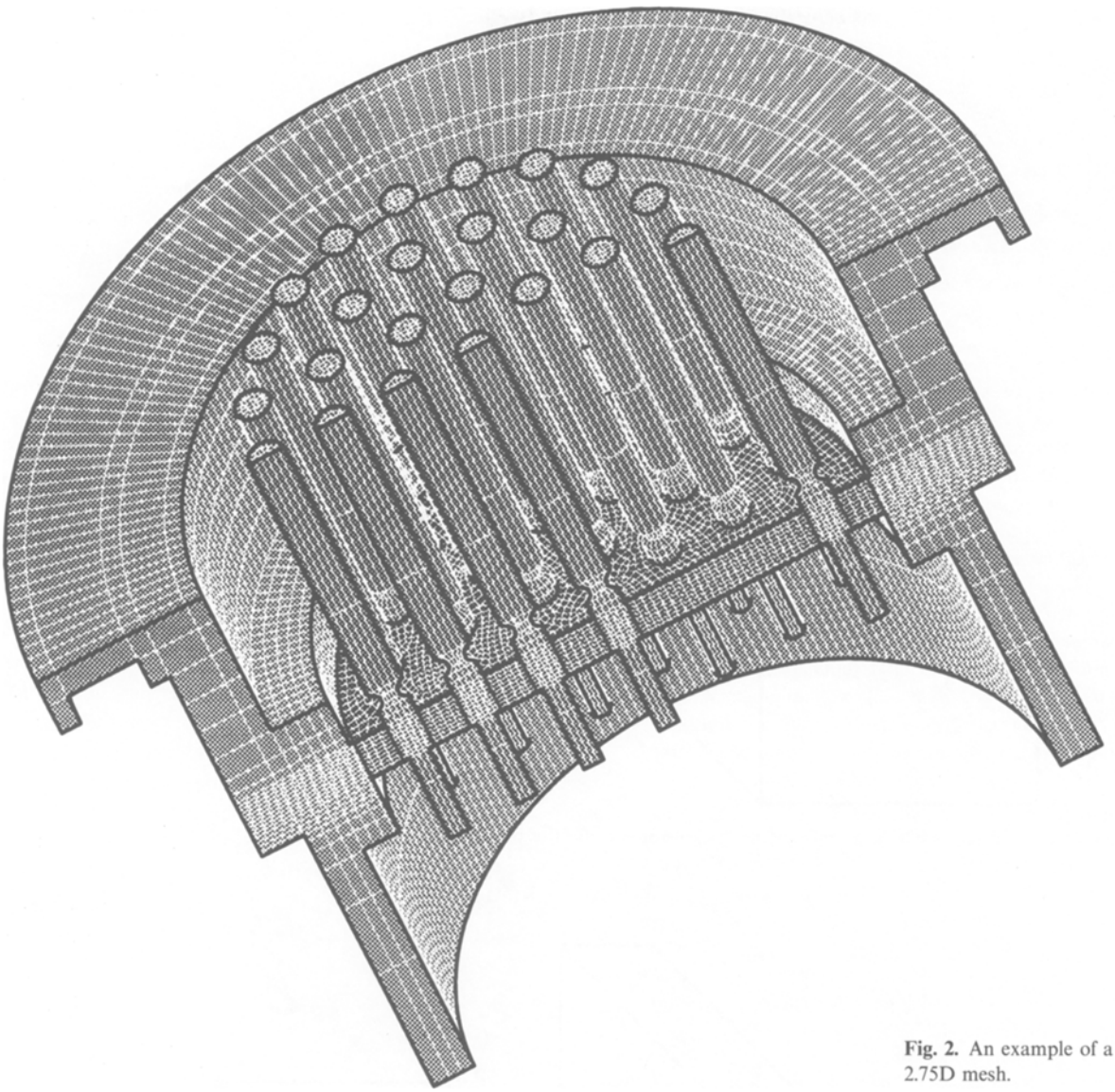

Fig. 2. An example of a complex 2.75D mesh.

In their simplest form, 2.5D tools depend only on a 2D mesh as input. Thus they have often been developed independent of any formal geometric description of the 3D geometry. This disconnection from the actual geometry presents problems in generalizing the algorithms to general sweeps, in accurately combining nodes shared between merged pieces, and in applying and verifying boundary conditions. It also prevents the effective use of general 3D CAD data when available.

Another limitation that comes from the disconnection from actual geometry is that any decomposition into a 2.5D piece must propagate through the entire geometry as shown in Fig. 3. The example shown in this figure is simplistic; however, it is easy to envision the difficulties caused by propagating decompositions in more complex geometries. With the proper tools, including those described in this paper, the propagation can be largely eliminated. Figure 4 shows how these tools would be effective at reducing the decomposition of this object to only one cut. Implicit for this type of decomposition to work effectively would be the use of the following techniques which are presented in this paper.

- *Automated interval assignment.* Automated assignment of edge discretization levels (interval assignment) is necessary for any type of efficient use. With the submapping tool this is especially important to automate correctly, as interval propagation is nontrivial.
- *N-edged surface mapping and submapping.* Enhanced mapping and submapping techniques allow the necessary regular grid to be established on surfaces

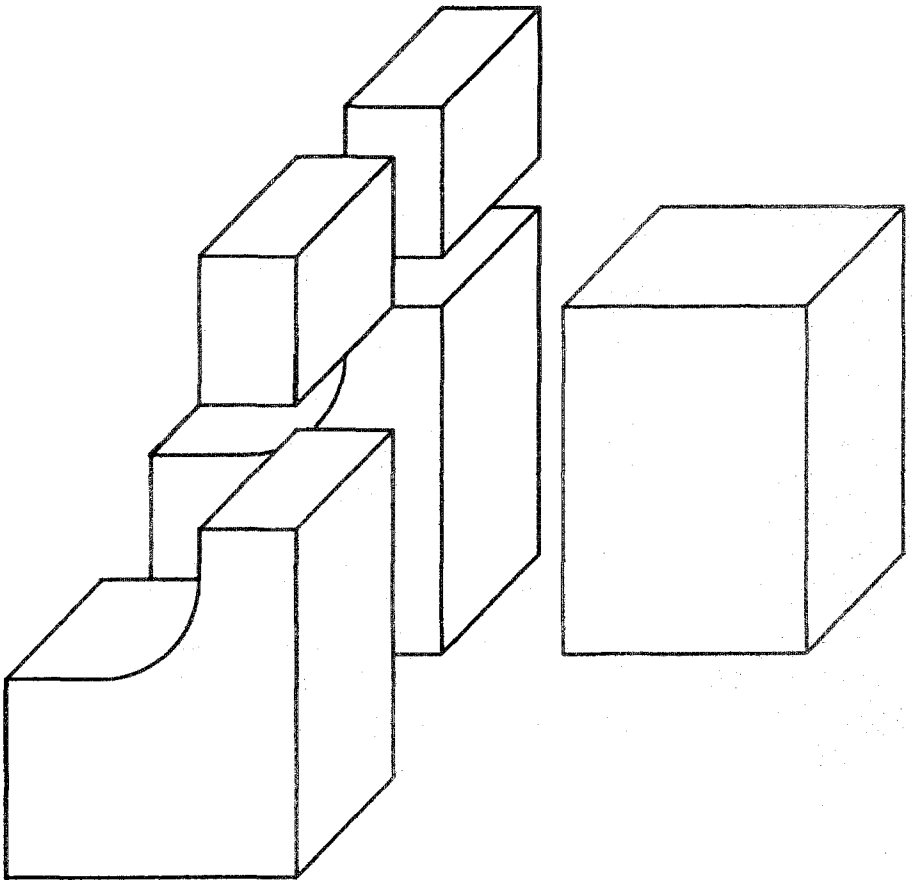

Fig. 3. Simple geometry showing the effect of propagating decompositions.

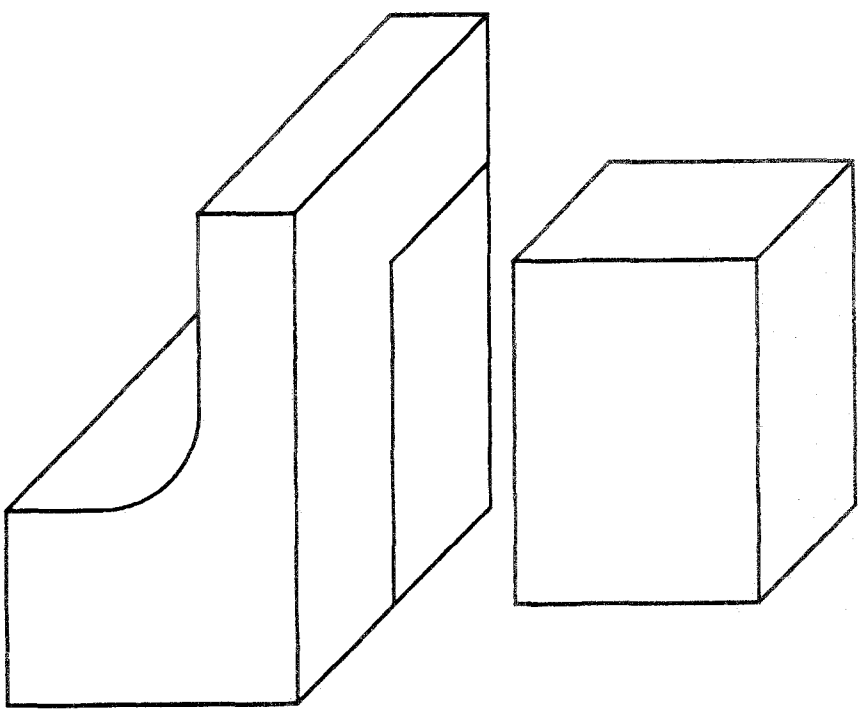

Fig. 4. Non-propagating decomposition allowed with the proper 2.75D meshing tools.

containing any number of edges (n-edged) but with block-like characteristics. The surface between the two blocks in Fig. 4 is such a surface. This tool eliminates the need to break this surface into only four-sided divisions.

- *User defined surface vertex types.* To insure that the required mesh can always be obtained from the mapping and submapping tools, the user may need to specifically identify the type of mapping desired at certain vertices. That is true where vertex angles alone are insufficient to determine the desired behavior.
- *N-faceted volume mapping and submapping.* When a propagation limiting decomposition is undertaken, many of the bodies resulting from the decomposition are block-like in nature, but are multifaceted (more than just 6 faces/block). The volume mapping tool must be robust enough to handle these geometries. A volume submapping algorithm is likewise useful in eliminating the need for many decompositions.
- *N-faceted sweeping.* As with mapping, to limit decompositions, the 2.5D sweeping tools must also handle multifaceted/multi-segmented sweeps such as the lower volume shown in Fig. 4. The sweep from the front surface to the back will require the tool to traverse two surfaces along the top, but only one surface on the left, right and bottom.

This paper discusses these tools and their interaction. First, the idea of 'surface vertex types', 'i–j–k' traversals, and 'hard and soft interval settings' will be defined. These definitions are needed for proper discussion of both the automated interval assignment process and submapping techniques. Finally, the n-faceted volume mapping, submapping and general sweeping tools are discussed.

2. Definitions

The definitions of 'surface vertex types', 'i–j–k traversals' and 'hard/soft' interval settings are introduced in this section. They are essential for the proper presentation of automated interval assignment and submapping tools.

2.1. Surface Vertex Type

The 'surface vertex type' provides the absolute classification of the interior surface angles (*end*, *side*, *corner*, *reversal*) at specified vertices. The 'surface vertex type' allows the user to control the mapping and submapping algorithms. The following are possible 'surface vertex types' (see Fig. 5):

- *End* (*normally* $\pi/2$): An *end* vertex is a vertex at which only one element is inserted. A row of quadrilateral elements terminates at an *end* vertex.
- *Side* (*normally* π): A *side* vertex is a vertex which is attached to two quadrilateral elements. A row of elements continues past a *side* vertex.
- *Corner* (*normally* $3\pi/2$): A *corner* vertex is a vertex which is attached to three quadrilateral elements. A row of elements turns a logical corner around the *corner* vertex.
- *Reversal* (*normally* 2π): A *reversal* vertex is a vertex which is attached to four quadrilateral elements. A row of elements reverses direction at a *reversal* vertex.

The 'surface vertex type' classifications control the 'i–j–k' traversals of the automated interval assignment and submapping algorithms (defined in Section 2.2). It also controls which vertices will be selected as 'logical corners' for the mapping algorithm.

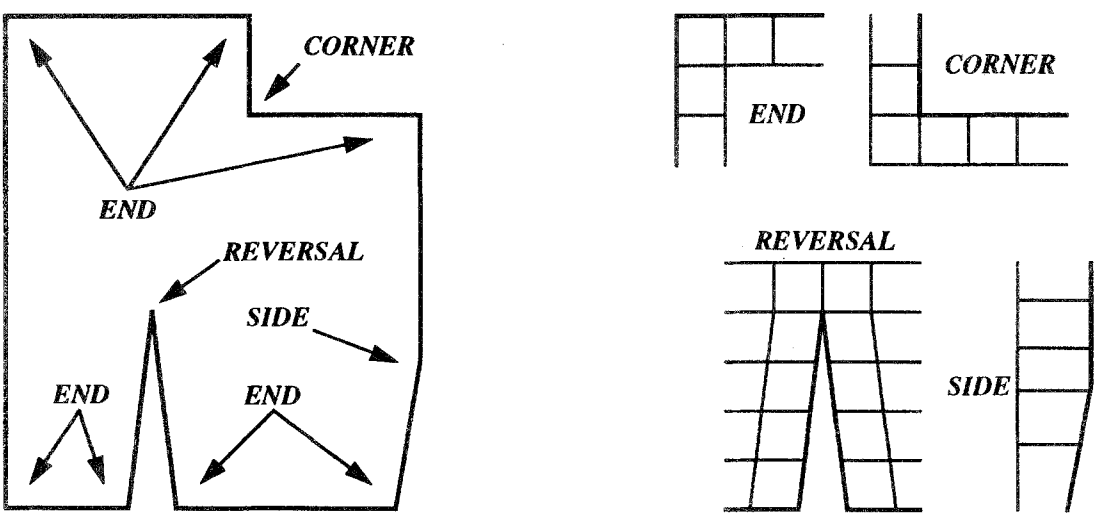

Fig. 5. Surface vertex type examples.

The interior surface angle at a vertex may not fall into a single specific classification. Several categories are possible depending on the topology and geometry of the model. For instance, depending on the geometry, an angle of 270° may be classified as either a *corner*, *side* or *reversal*. The 'surface vertex type' is necessary to allow explicitly defining interior surface angles that may fall into a 'fuzzy' classification. The user can specify the 'surface vertex type' to classify these 'fuzzy' angles, thus providing the control to produce the desired meshing results. In all cases the algorithms will classify all vertices based on their interior surface angle measure unless the 'surface vertex type' has been specified to explicitly classify the vertex.

2.2. *i–j–k* Traversals

The '*i–j–k*' traversals are based on a local '*i–j–k*' cartesian coordinate system. The coordinate system is local to the surface or volume geometry that is being evaluated. The traversal classification definition is the same for both automated interval assignment and submapping algorithms, although they are implemented for different purposes. For surface mapping/submapping and for automated interval assignment only a 2D subset (i.e. '*i–j*' traversals) are necessary, while the 3D submapping requires full 3D '*i–j–k*' traversals.

The '*i–j*' traversals classify geometric curves on each surface as '$+i$', '$+j$', '$-i$' or '$-j$' in the local '*i–j*' coordinate system. The classification of the surface curves, in the local '*i–j*' coordinate system, is based on the 'surface vertex type' at each geometric vertex. The classification process starts at an arbitrary vertex and traverses the surface vertices in a counterclockwise (CCW) direction. The first curve is arbitrarily assigned to be in the '$+i$' direction. As the geometric boundary curves are traversed, the next curve is classified based on the 'surface vertex type' of the vertex between it and the previous curve. For example, if a curve was classified as '$+i$' and if the next CCW curve was attached to the previous curve with a 'surface vertex type' of type *end*, then the forward curve would be classified as '$+j$'. The traversals in '*i–j*' space are shown in Table 1. Using this table in the example just given, the vertex is an *end* case, the input state was '$+i$', so the new direction can be determined from the table as '$+j$'. Note: for *side* case (normally 180°), the '*i–j*' classification does not change from curve to curve.

During submapping, not only is it important to know a direction, but an actual (i, j) coordinate value. This assignment is straightforward, assuming an edge mesh has been generated using appropriate interval assignments. The node at the first vertex is assigned a coordinate of (0, 0) in '*i–j*' space, and nodes along the first curve (assumed to be in the '$+i$' direction) are incremented in '*i*' (e.g. (1, 0), (2, 0), etc.). As vertices are reached which change the '*i–j*' direction, as described above, the appropriate coordinate is incremented or decremented. Extending the previous example, if an end vertex is reached and the node at that vertex was assigned (9, 0) as its (i, j) coordinate, the next node would increment in the $+j$ direction (i.e. its '*i–j*' coordinate would be (9, 1)) (see Table 1). Surface vertex classifications and '*i–j*' traversals provide a method to classify surface boundary curves and to assign all meshed nodes on the surface boundary an '*i–j*' coordinate in the local '*i–j*' coordinate system.

Table 1. Traversal tables

Input state	End	Side	Corner	Reversal
$+i$	$+j$	$+i$	$-j$	$-i$
$+j$	$-i$	$+j$	$+i$	$-j$
$-i$	$-j$	$-i$	$+j$	$+i$
$-j$	$+i$	$-j$	$-i$	$+j$

3.3. Hard and Soft Interval Settings

In the meshing procedure, the user often desires varying levels of control. For interval assignment, this translates to allowing the user to specify an exact number ('hard') or an approximate number ('soft') of intervals.

'Hard' set intervals are defined as an absolute number of intervals that are assigned to a curve. Intervals that are designated as 'hard' are not adjusted by the automated interval assignment algorithm.

'Soft' set intervals are defined as an approximate number of intervals that are assigned to a curve, surface or volume rather than an absolute number. Intervals that are designated as 'soft' are taken as a lower bound by the automated interval assignment algorithm. If a curve is not defined, the interval assignment defaults to a 'soft' that produces one interval on the curve. Adjustments to 'soft' curves are minimized by the automated interval assignment algorithm while satisfying dependencies and compatibility constraints for all geometric surfaces.

3. Automated Interval Assignment

Appropriate interval assignment is critical to producing high quality meshes for mapped and submapped

surfaces or volumes. It can be a tedious and time consuming process to manually designate interval settings on all curves of complex geometry, especially since satisfying compatibility for meshing primitives is required to produce acceptable mapped meshes for a number of existing algorithms [2]. The automated interval assignment process insures that surface and volume compatibility are automatically maintained, based on a minimal number of initially defined interval settings. The automated interval assignment tool is based on an optimization method known as linear programming [3]. The linear programming procedure produces a solution set of intervals for each curve that satisfies multiple geometric dependencies and subgeometric compatibility constraints.

The term linear programming (LP) describes a particular class of extremization problems in which the objective function and the constraint relations are linear functions. The general form of the (single objective) linear programming model can be stated mathematically as: find $x = (x_1, x_2, \ldots, x_n)^t$ so as to *optimize* (either maximize or minimize) the objective function *subject to* the specified constraints. Note that each constraint may be of *type* I (inequality, $\leq$), *type* II (inequality, $\geq$), or *type* III (equality, $=$).

$$\textit{optimize:} \quad z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$\begin{aligned}\textit{subject to:}\quad & a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n (\leq, =, \geq) b_1\\ & a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n (\leq, =, \geq) b_2\\ & \vdots \qquad\qquad \vdots \qquad\qquad \vdots\\ & a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n (\leq, =, \geq) b_m\\ & x_1, x_2, \ldots, x_n \geq 0\end{aligned}$$

Because the necessary conditions for an interior minimum is the annihilation of the first derivative of the function with respect to the design variables, linear programming problems have a special feature. That is, the derivatives of the objective function with respect to the design variables are constants which are not necessarily zeros. This implies that the extremum of the linear programming problem cannot be located in the interior of the feasible design space and, therefore, must lie on the boundary of the design space described by the constraint relations. Since the constraint relations are also linear functions of the variables, the optimum design must lie at the intersection of two or more constraint functions, unless the binding constraint is parallel to the contours of the objective function. The class of linear programming problems, involving large numbers of variables and constraints are usually solved by an extremely efficient and reliable method known as the simplex method [4].

The development of appropriate constraint equations is the foremost concern when formulating linear programming problems for general meshing algorithms. The use of integer programming has been explored by Tam and Armstrong [2], and is extended here for use by the pseudo-decomposition, submapping algorithm. The method of '*i–j* unfolding' is presented as a novel technique for reorganizing surface geometry for the purpose of formulating constraint equations that are conducive to producing submapping solution sets. As described in Section 2.2, the purpose of the '*i–j*' traversals is to classify geometric curves of each surface ('$+i$', '$+j$', '$-i$', or '$-j$') as they relate to the local '*i–j*' coordinate system.

The classification process starts at an arbitrary vertex and traverses the surface vertices in a counterclockwise direction. The first curve is arbitrarily assigned to be in the '$+i$' direction. As the geometric boundary curves are traversed, the next curve is classified based on the 'surface vertex type' of the vertex between it and the previous curve. Figure 6 (left) shows an '*i–j*' classification of a surface in its local '*i–j*' coordinate system. After the classification of curves is complete, the unfolded surface geometry can be formulated. The unfolded surface geometry is built by taking all curves classified as $[+i]$ and grouping them into one side of the unfolded geometry while all curves classified as $[-i]$ are grouped into another side that is opposite the $[+i]$ curves. The curves classified as $[+j]$ and $[-j]$ are grouped into the unfolded geometry in a similar manner. Figure 6 (right) shows the placement of the curves in the resulting 'unfolded' geometric surface (left).

The method of '*i–j*' traversal classifications eliminates the need for surface or volume decomposition prior to solving for interval division solutions. Linear constraints can be directly formulated from the resulting 'unfolded' surface geometry. The 'unfolded' surface geometry reflects a four sided surface with multiple curves on each side. The constraint equations for the 'unfolded' surface are easily developed (i.e. $[\sum +is] = [\sum -is]$ and $[\sum +js] = [\sum -js]$. The following constraint equations were developed using the 'unfolded' surface geometry in Fig. 6.

$$\textit{maximize:}\quad z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8$$

$$\textit{minimize:}\quad z(x) = -z(x)$$

$$\begin{aligned}\textit{subject to:}\quad & x_1 = x_3 + x_5 + x_7\\ & x_2 + x_6 = x_4 + x_8\\ & \{x_1, \ldots, x_m\} > 1, \textit{ or, } x_i > b_i\\ & \qquad b_i > 1 \textit{ for 'soft set' curves}\\ & \{x_1, \ldots, x_m\} = b_i\\ & \qquad x_i = b_i \textit{ for 'hard set' curves}\end{aligned}$$

x_i is replaced by b_i for 'hard set' curves in all equality contraints

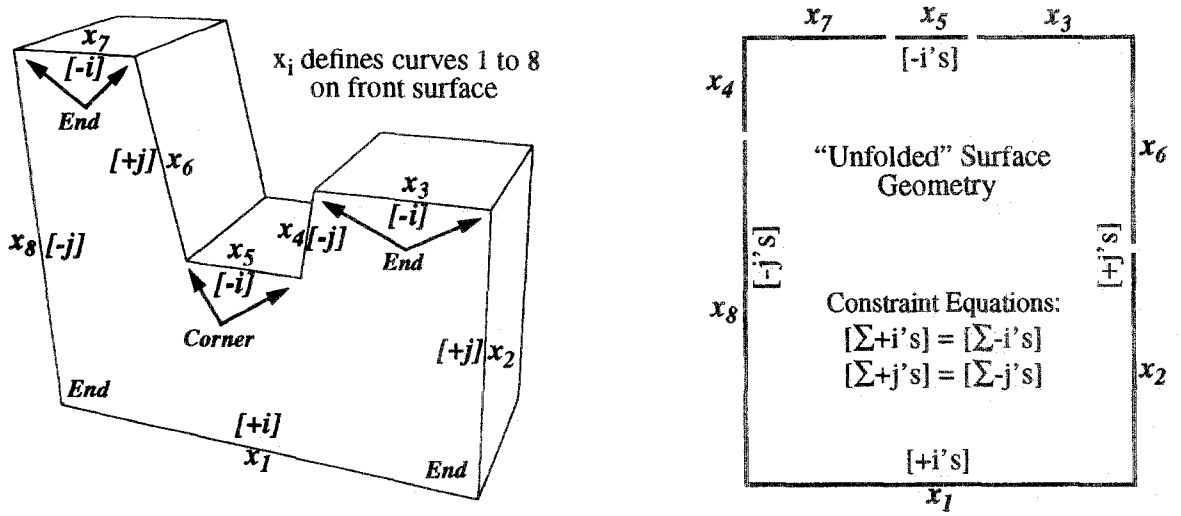

Fig. 6. Local i–j traversal classifications and resulting 'unfolded' surface geometry.

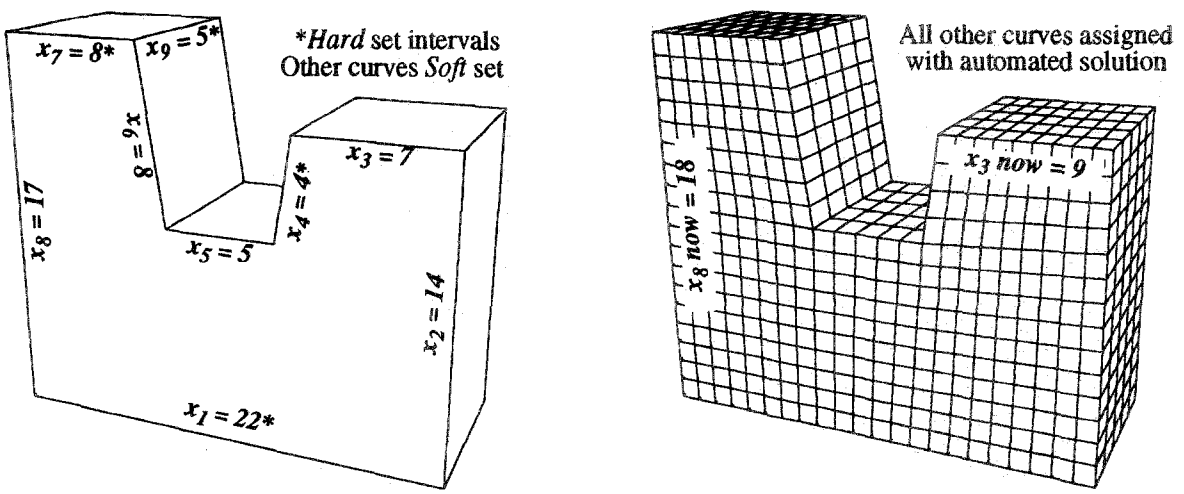

Fig. 7. Initial interval settings and final interval assignment.

The initial interval settings are shown in Fig. 7 (left). The curves indicated with a '*' are '*hard*' set curves, the other curves indicated are '*soft*' set based on curve length. The linear program is formulated assuming that the remaining curves have a lower bound interval setting of one. After the linear programming problem is formulated, it is then solved using the simplex method. The resulting solution set is then applied to the respective curves. The curves now have interval assignments that are conducive to submap meshing. An automatic pseudo-decomposition is formed by the submappings tool based on the interval division solution derived for each curve. The resulting mesh is shown in Fig. 7 (right).

As seen from the resulting mesh Fig. 7 (right), curve x_2 was adjusted to nine intervals and curve x_8 was adjusted to 18 intervals to provide a mesh solution that was compatible for submapping. The remaining curves not initially set were assigned intervals based on the linear programming solution. In all, four curves were '*hard*' set, five curves were '*soft*' set, and 20 curves had interval assignments determined by the linear programming solution. It can be seen from the resulting mesh (right) and initial interval settings (left), that a minimal number of intervals are required to produce a compatible mesh that propagates over the entire volume, these properties of compatibility and propagation can also be extended to several connected and interconnected volumes as well. The resulting mesh also shows that the automated interval assignment algorithm, using the 'unfolded' surface geometry method, is conducive to producing submap meshes.

4. *N*-Edged Surface Mapping and Submapping

To make mapping and submapping more efficient and reduce propagation of dissections, the surfaces involved must not be limited to a certain number of edges. For example, mapping requires four logical sides, but any number of physical edges are allowed to lie on those sides. Submapping is a direct approach to limiting decomposition of geometry but it is dependent on the *n*-edged mapping algorithm to map any surface that does not contain *corners* or *reversals*. These two tools then work together to provide a reduction in the propagation of dissections.

4.1. *N*-Edged Surface Mapping

The designation of a logical quadrilateral, fit to the surface, is required for the mapping of an *n*-edged surface. Four vertices are selected that best translate the surface into a logical quadrilateral. These four vertices are selected as the logical corners and are defined with a surface vertex type of *end*. The surface vertex type functionality, defined in Section 2.1, can be used to simplify the process in cases where the surface geometry could have multiple logical quadrilateral fits. The curves between the logical corners are grouped as a logical side. The logical rectangle that is formed, is meshed using the two-dimensional mapping transformations as discussed by Cook and Oaks [5]. The mapping algorithm works for surfaces where one logical quadrilateral can be formulated, but where there are corners or reversals, *n*-edged submapping is needed.

4.2. *N*-Edged Submapping

Many current meshing techniques require the decomposition of surface geometry into four sided divisions before the mesh generation can be applied. Submapping automates the geometry decomposition of surfaces that contain *corners* or *reversals*; an example of this is shown in Fig. 8 (left). This automation is done by breaking the geometry into pseudo-quadrilateral regions and then mapping these regions by the standard mapping transformations [5]. A regular distribution of the grids is produced by submapping, as is shown in Fig. 8 (right). Using '*i–j*' traversals, submapping locates *corners* or *reversals* where the surfaces should be cut and creates virtual edges to divide the surface into *n*-edged mappable regions.

The submapping algorithm assumes that the edges of the surfaces have already been meshed before it begins. This enables the submapper to be geometry independent and use a nodal loop to define the geometry. A nodal loop is a linked list in which all of the nodes around the edge of the surface are stored in a counter-clockwise order. The loop containing the edge nodes, is then traversed using the '*i–j*' traversals described in Section 2.2. When corner and reversal nodes are found they are stored and become the starting points where pseudo-cuts will be made to divide the surface into simple mappable regions. The end points, or cut nodes, of the virtual edges, are then found by their position in '*i–j*' space (see Fig. 9).

To find the 'cut node' in '*i–j*' space each node in the nodal loop is given an '*i–j*' coordinate according to the '*i–j*' space traversals for submapping as defined

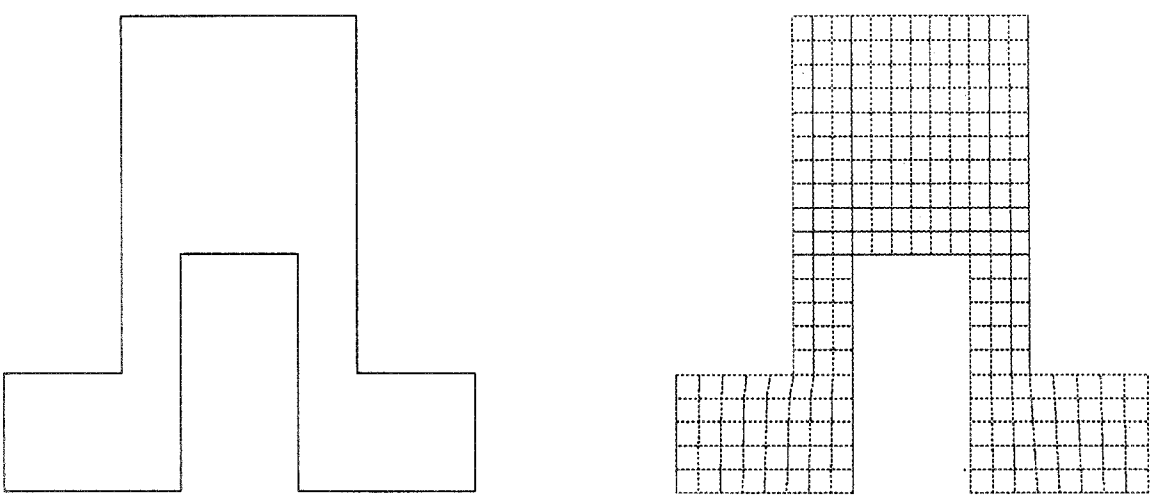

Fig. 8. Example of submapping geometry and resulting mesh.

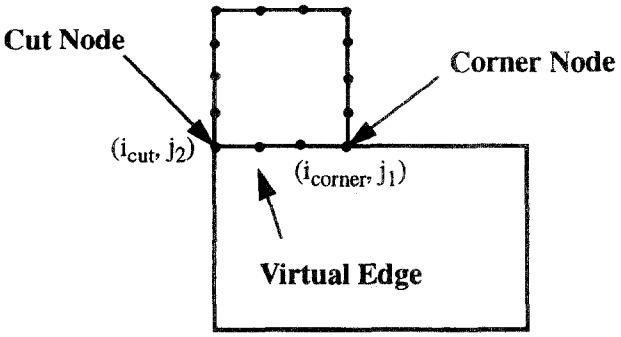

Fig. 9. Corner and cut node.

in Section 2.2. To insure successful surface meshing, the submapping algorithm insures that the traversal finishes with the coordinate (0, 0) at the end of the loop. If incorrect intervals have been assigned, or some of the interior angles are fuzzy, then the submap tool will stop. The fuzzy angles can be set by assigning vertex types to the fuzzy vertices as defined in Section 2.1. Vertices need to be defined properly so that the '*i–j*' traversal ends at zero and the submap tool can successfully mesh the surface. The '*i–j*' coordinates of the nodes are then stored in an ordered 2D array. The ordering is simultaneous with the nodes in the list.

The nodal loop defining the surface is then traversed to find a matching cut node for the *corner* node or *reversal* node. The cut node is generally defined as the next closest node that has a matching '*i*' or '*j*' coordinate with the *corner* or *reversal*. In Fig. 9 the *corner* node is matched up with the next closest node that has the same '*j*' coordinate as the cut node. Once a good cut node has been found, a virtual edge is formed to divide the surface. New nodal loops are then formed which define the new regions. When a virtual edge is made, two regions are formed. If both of the regions still contains *corners* or *reversals*, both regions will be stored and the submapping algorithm will process one of them until no *corner* or *reversals* exist. Submapping will then return to work on the second loop until it is finished. If a formed region has no *reversals* or *corners*, the mapping transformations [5] are applied. If the region still has *corners* or *reversals*, the new nodal loop is stored and the process continues recursively. Submapping is finished when there are no regions with *corner* or *reversal* nodes and all pseudo-regions have been meshed.

The virtual edges are interpolated curves, rather than being straight lines between the cut and *corner* nodes. The interpolation occurs in two-dimensions using the relative position of the cut and *corner* nodes and the bounding nodes along the edges that are between the cut and *corner* nodes. The virtual edge is made by interpolating each node in the edge by its bounding nodes and the *corner* and cut nodes. Figure 10 shows that the node located at $(i_{\text{now}}, j_{\text{now}})$ is interpolated from the bounding nodes and the cut and *corner* nodes. The other nodes between the corner and cut nodes are created similarly and are then used to create the virtual edge.

The equations to calculate the interpolated nodes depend on the '*i–j*' direction of the cut and the sign of the direction. For the case where the cut is in the '$+i$' direction (see Fig. 10), the nodes for the interpolated edge are found by the following equation:

$$
\begin{aligned}
&(x_{\text{now}}, y_{\text{now}}, z_{\text{now}}) \\
&\quad = \frac{1}{2} \times \left((x, y, z)(i_{\text{boun1}}, j_1) \times \frac{(j_2 - j_{\text{now}})}{(j_2 - j_1)} \right) \\
&\qquad + \frac{1}{2} \times \left((x, y, z)(i_{\text{boun2}}, j_2) \times \frac{(j_{\text{now}} - j_1)}{(j_2 - j_1)} \right) \\
&\qquad + \frac{1}{2} \times \left((x, y, z)(i_1, j_{\text{corner}}) \times \frac{(i_2 - i_{\text{now}})}{(i_2 - i_1)} \right) \\
&\qquad + \frac{1}{2} \times \left((x, y, z)(i_2, i_{\text{cut}}) \times \frac{(i_{\text{now}} - i_1)}{(i_2 - i_1)} \right)
\end{aligned}
$$

For each new node, the boundary node's '*i–j*' coordinates change along with its projected $(i_{\text{now}}, j_{\text{now}})$ position. The equation then determines the real space position of the nodes. With this, the new node is created and then moved onto the surface. After the nodes are created, they are linked together to define the virtual edge. Interpolated virtual edges are useful to help the mesh follow the geometry by mimicking the node positions around the cuts.

Finally, submapping can be performed for *n*-numbers of *corners* or *reversals*. As shown in Fig. 11, the geometry can have many different features if the submapper can determine the vertex types successfully, either by interior angles or surface vertex types. Submapping uses interpolated virtual edges to perform pseudo-decomposition of the surface geometry; it then maps the decomposed surfaces without physically altering the geometry. This tool is very powerful in preventing propagation of geometry decompositions and leads to volume submapping, which can eliminate the need for time consuming geometry decomposition.

5. *N*-Faceted Volume Mapping and Submapping

To eliminate the need for geometry decomposition, true 3D mapping is needed. *N*-faceted volume mapping

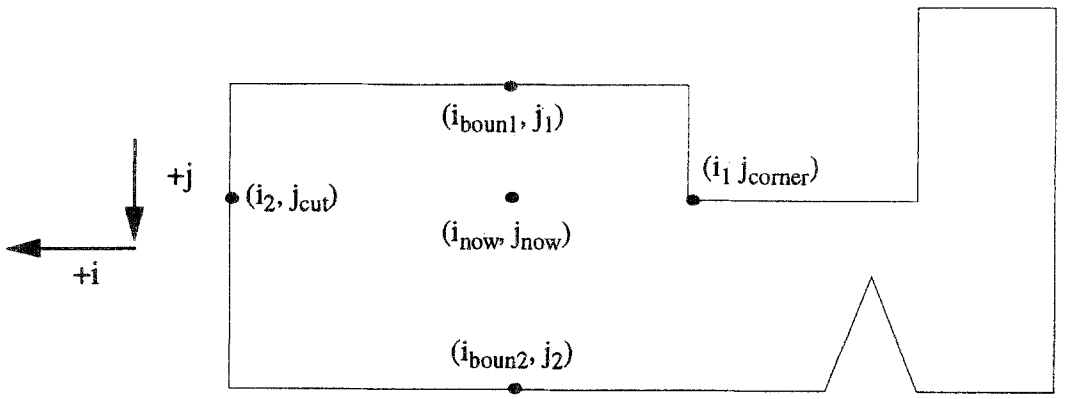

Fig. 10. Node interpolation for virtual edges.

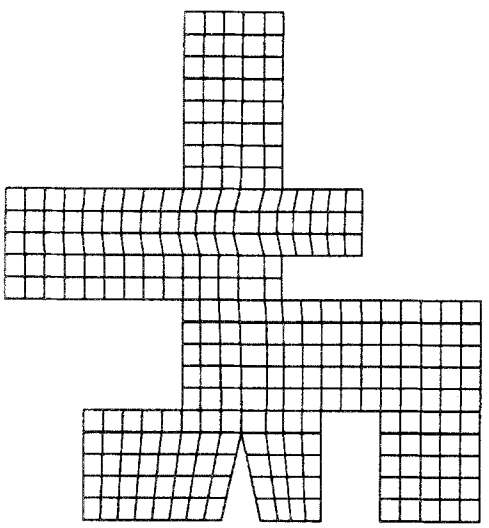

Fig. 11. Example of N-number of concavities in geometry.

is a usable tool that automates 3D meshing for objects that have n-number of faces. Where surface mapping requires a logical quadrilateral, volume mapping requires a logical hexahedral. Volume submapping is another step to further improve 3D meshing. Volume submapping will automatically use pseudo decomposition, like submapping (see Section 4.2), to break volumes down into logical hexahedrons.

5.1. N-Faceted Volume Mapping

The volume mapping algorithm has been derived from the three dimensional mapping transformations [5]. This algorithm picks eight vertices of a volume based on the number of face elements attached to each vertex. The 'eight vertices' must only have three faces attached. This translates the n-faceted volume into a logical hexahedral. Volume mapping depends on all of the surfaces of the volume being mappable. Figure 12 (left), displays a volume that is mapped by the volume mapping tool. This figure shows that a volume can have many different defining surfaces, yet one logical cube can be derived that defines the body.

5.2. N-Faceted Volume Submapping

The ultimate goal in automated mesh generation is to minimize the manual operations required to produce quality mesh. Volume submapping will mesh volumes that contain more than eight 'logical corners' by using virtual surfaces analogous to submapping using virtual edges. Unlike submapping however, the virtual surfaces will be true maps rather than interpolations. It will also introduce the need for 'i–j–k' traversals to locate areas where the virtual surfaces should be placed. Volume submapping is currently under development. The goal for volume submapping is to improve usability and reliability in 3D meshing where uniform grid-like characteristics are desired. An example of the type of geometry that will be volume submapped is shown in Fig. 12 (right). This volume is currently just surface meshed using automated interval assignments and submapping. The implementation of volume submapping will eliminate the need for geometry decomposition on such bodies. This will be performed by surface submapping, and virtual surfaces that automatically dissect the volume into n-faceted volume mapping regions.

6. Conclusions

New tools have been created to enhance finite element meshing. These tools are automated interval assignment, n-edged mapping and submapping, n-faceted sweeping, volume mapping and volume submapping. These tools simplify the mesh generation by automating many tasks previously done manually.

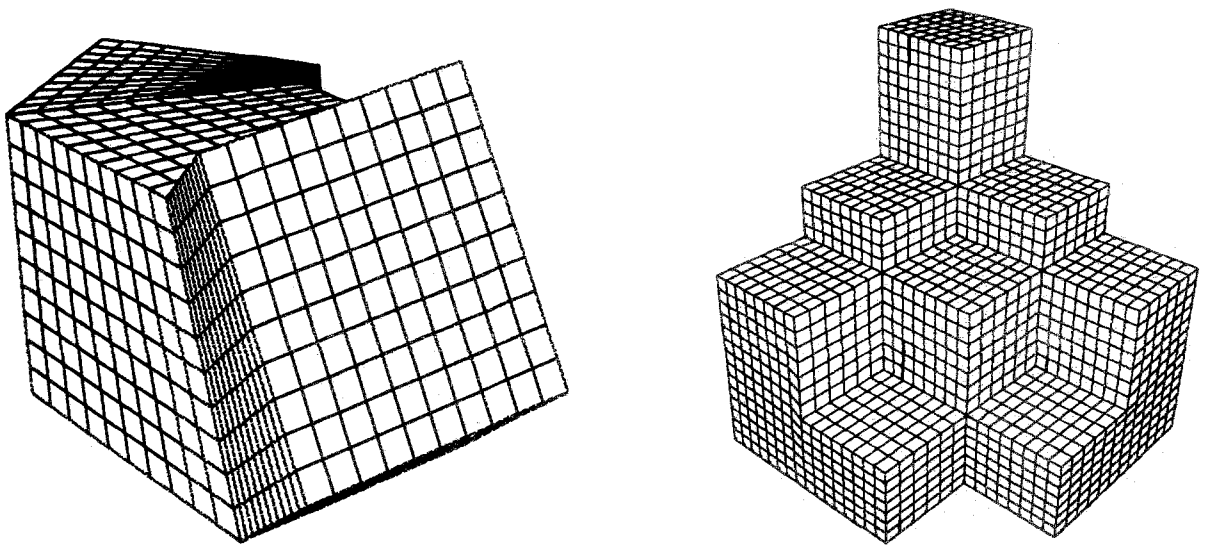

Fig. 12. Volume mapping and volume submapping.

Automated interval assignment provides a number of benefits to 2.5D, 2.75D and 3D meshing. Automated interval assignment eliminates the tedious and time consuming process of manually designating interval settings on all curves of complex geometry. It also minimizes the number of intervals that are required to produce a compatible mesh that propagates over an entire volume. These properties of compatibility and propagation extend beyond a single volume and make it possible to produce interval assignment solutions for several connected and interconnected volumes. Finally the unfolded geometry method based on '*i–j*' traversals provides an extremely efficient method for formulating linear programming solutions that are conducive to submap meshing.

N-edged mapping and submapping reduce the need for costly manual surface geometry decompositions. *N*-edged mapping requires no decomposition when one logical quadrilateral can be formed on the surface. Submapping uses *n*-edged mapping to reduce manual surface decomposition by automating the process, using pseudo-cuts called interpolated virtual edges. The '*i–j*' traversal method is an effective means to identify the cut nodes relative to the *corner* or *reversal* nodes. In cases where fuzzy angles exist around vertices, surface vertex settings can be used to improve the mesh. Submapping produces regular quadrilateral element distribution with very little use interaction.

True 3D mesh generation has often been limited to geometry that has been dissected into hexahedral volumes. *N*-faceted volume mapping and submapping are algorithms that automate this process when block-like elements are desired. *N*-faceted volume mapping is an application of the 3D mapping transformations derived by Cook and Oaks [5], in which volumes are translated into a logical hexahedron. Volume submapping is the next step in the automation of the volume mapping of volumes that contain more than one logical hexahedral subvolume. Volume submapping takes the pseudo-decomposition method to a higher level by using mapped virtual surfaces to reduce decomposition of volumes. It uses '*i–j–k*' traversals to locate the regions for cutting. Volume submapping is currently in the developmental stage and is not currently functional.

References

1. Gilkey, A.P.; Sjaardema, G.D. (1989) GEN3D: A GENESIS Database 2D to 3D transformation program. Technical Report SAND89-0485, Sandia National Laboratories, Albuquerque, New Mexico, March
2. Tam, T.K.H.; Armstrong, C.G. (1993) Finite element mesh control by integer programming, International Journal for Numerical Methods in Engineering, 36, 2581–2605
3. Wolfe, C.S. (1985) Linear Programming Algorithms, Prentice-Hall, Virginia
4. Haftka, R.T.; Gurdal, Z. (1992) Elements of Structural Optimization, Kluwer Academic Publishers, Massachusetts
5. Cook, W.A.; Oaks, W.R. (1983) Mapping methods for generating three-dimensional meshing, Computers in Mechanical Engineering, 1, 67–72.