INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2000; **49**:127–144

Automatic scheme selection for toolkit hex meshing[§]

David R. White[†] and Timothy J. Tautges[*,‡]

Sandia National Laboratories, Albuquerque, NM, U.S.A.

SUMMARY

Current hexahedral mesh generation techniques rely on a set of meshing tools, which when combined with geometry decomposition leads to an adequate mesh generation process. Of these tools, sweeping tends to be the workhorse algorithm, accounting for at least 50 per cent of most meshing applications. Constraints which must be met for a volume to be sweepable are derived, and it is proven that these constraints are necessary but not sufficient conditions for sweepability. This paper also describes a new algorithm for detecting extruded or sweepable geometries. This algorithm, based on these constraints, uses topological and local geometric information, and is more robust than feature recognition-based algorithms. A method for computing sweep dependencies in volume assemblies is also given. The auto sweep detect and sweep grouping algorithms have been used to reduce interactive user time required to generate all-hexahedral meshes by filtering out non-sweepable volumes needing further decomposition and by allowing concurrent meshing of independent sweep groups. Parts of the auto sweep detect algorithm have also been used to identify independent sweep paths, for use in volume-based interval assignment. Published in 2000 by John Wiley & Sons, Ltd.

KEY WORDS: cubit; mapping; submapping; sweeping; hexahedra; toolkit

1. INTRODUCTION

The finite element method is used to simulate a wide variety of physical phenomena, for example heat transfer, structural mechanics, and computational fluid dynamics. In recent years, mesh generation has emerged as one of the major bottlenecks in the simulation process. Although a high degree of automation is available in tetrahedral mesh generators, hexahedral mesh generators still require a great deal of user intervention. Since it is widely believed that hexahedral meshes are more accurate and more robust for some types of finite element analysis, especially in the non-linear regime, these types of meshes are more commonly used.

There has been a great deal of research into automated, all-hexahedral meshing algorithms [1–3], but as yet no algorithm has been found with the key characteristics of high robustness, high mesh quality and low element count. Therefore, current hexahedral mesh generation techniques rely on

[*]Correspondence to: Timothy J. Tautges, 1500 Johnson Drive, Madison, WI 53706, U.S.A.
[†]E-mail: drwhite@sandia.gov
[‡]E-mail: tjtautg@sandia.gov
[§]This article is a U.S. Government work and is in the public domain in the U.S.A.

Contract/grant sponsor: United States Department of Energy; contract/grant number: DE-ACOU-94AL85000

Received 15 April 1999
Revised 4 August 1999

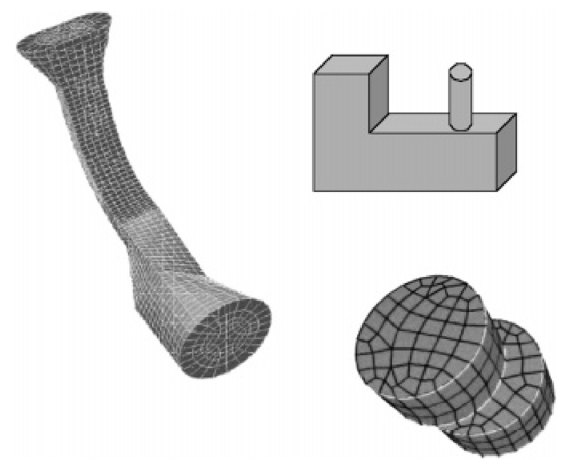

Figure 1. Sweepable geometries.

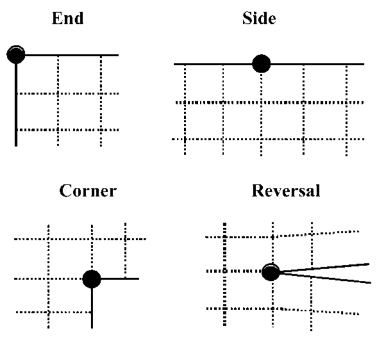

Figure 2. Mesh topology in neighbourhood of surface vertex types; geometry (vertices and edges) are represented by solid points and lines, mesh edges represented by dashed lines.

a set of simpler tools, which when combined with geometry decomposition leads to an adequate mesh generation process. The meshing algorithms in these tools include mapping/submapping [4], primitive templates [5], and sweeping or extrusion [6]. Of these, sweeping tends to be the workhorse algorithm, usually accounting for at least 50 per cent of most meshing applications.

The sweeping algorithm involves extruding a set of quadrilaterals into a third dimension, producing a hexahedral mesh. The cross-section of the geometry being meshed can vary along the sweep direction, and the number of quadrilaterals in the set being swept can vary as well. Figure 1 shows several sweepable geometries. Many of the commercial mesh generation software packages currently include some form of sweeping algorithm [7, 8], and varieties of this algorithm are reported elsewhere in the literature [6, 9, 10].

While sweeping is a widely used algorithm, it is not very automated. Before a volume can be 'swept', the algorithm must be provided with input about which surface meshes are being swept along which side surfaces, and for how far. In practice, the process of determining and specifying these source/target surfaces is user-intensive and error-prone. In order to increase the level of automation in all-hexahedral meshing, an automatic method for determining sweep directions and source/target surfaces is needed.

The detection of swept features has been studied in the feature recognition community for some time [11, 12]. However, the resulting algorithms are usually geometry-based, relying on arrangements such as parallel surfaces for detecting the features. These arrangements represent geometric constraints placed on extruded volumes that are determined by the application, for example solids to be manufactured by machining.

This paper describes a new algorithm for detecting extruded or sweepable geometries. This algorithm is based on topological and local geometric criteria, and is more robust than feature recognition-based algorithms. This algorithm has been implemented in the CUBIT mesh generation toolkit [13], and has been shown to work for most sweepable geometries. This algorithm could also be used to detect extruded features for the purposes of feature extraction and dissembly planning.

This paper is arranged as follows. Definitions used in describing this algorithm are given in Section 2; the characteristics that make a geometry sweepable (also referred to as 2.5D) are discussed in Section 3; the auto sweep detection algorithm is described in Section 4, while an algorithm for grouping volumes with sweep dependencies is described in Section 5; examples are given in Section 6. Conclusions are given in Section 7.

2. BACKGROUND AND DEFINITIONS

We begin with a geometric solid defined by a BREP or boundary representation model, for example as described in Mortensen [14]. We further constrain this solid such that no surface contains a bounding vertex twice without also containing a connected bounding edge twice (by containing an entity twice, we mean that entity occurs on the bounding loop(s) of edges twice). We also assume that all bounding surfaces are oriented with respect to the volume under consideration. That is, we do not consider ‘non-manifold’ volumes sharing surfaces with other volumes (non-manifold volumes can be handled by modifying surface normals and loop directions for one of the sharing volumes). Before describing the automatic sweep detect algorithm, we must define several geometric and topological characteristics used in the algorithm. These characteristics either determine local (e.g. surface) meshability, or they combine local information across collections of surfaces to form meta-information about the volume being meshed. These characteristics will be used in the next section to define the auto sweep detect algorithm.

(*Surface*) *vertex type*: The classification of the topology of a vertex bounding a quadrilateral surface mesh by the number of quadrilaterals sharing the vertex. Four vertex types are used: End, Side, Corner, and Reversal. The number of quadrilaterals sharing a vertex of these types is one, two, three and four, respectively. The mesh topology represented by each vertex type is shown in Figure 2. For convenience, an integer is assigned to each of these vertex types; this integer is used to evaluate meshability (this will be described shortly). The integers assigned to the vertex types End, Side, Corner and Reversal are $+1$, 0, -1, and -2, respectively.

(*Volume*) *edge type*: Analogous to surface vertex types, the classification of the topology of geometric edges bounding a meshed volume. Edge types End, Side, Corner and Reversal are defined as being shared by one, two, three and four 3D elements in the volume, respectively. Note that the 3D elements can be of any type (hexahedra, prisms, etc.).

Structured mesh: In 2D, an all-quadrilateral mesh, where interior nodes (i.e. non-boundary nodes) are shared by exactly four quadrilaterals. In 3D, an all-hexahedral mesh whose bounding surface meshes are structured and whose interior nodes are shared by exactly eight hexahedra.

Mapped mesh: In 2D, a structured mesh whose boundary consists of exactly four End-type vertices or nodes, and any remaining boundary vertices and/or nodes are of type Side, i.e. are shared by exactly two quadrilaterals.

Discussion: In a 2D mapped mesh, the boundary nodes and edges between a pair of non-Side vertices make up a ‘side’ of the map; since mapped meshes contain exactly four End vertices, they also contain exactly four sides. End vertices typically fall on geometric vertices, although this is not required.[¶] Sides are made up of one or more geometric edges, since in a BREP model edges bound faces.

Submapped mesh: In 2D, a structured mesh whose boundary vertices and nodes are each type End, Side, Corner or Reversal.

Discussion: The submapping algorithm produces structured meshes [4]; it differs from the standard 'mapping' or TFI algorithm in that it admits surfaces with more than four sides. It has been shown in Reference [4] that a surface can be meshed with the submapping algorithm if all its vertices can be classified with a distinct vertex type, and the sum of vertex types is four. Thus, if the vertex types bounding a surface sum to four, the surface can be submapped. Note that determination of vertex types is a geometric operation local to the surface.

In many cases, the angle used to determine vertex type is not an integer multiple of 90°. Therefore, a 'fuzzy' range around those integer multiples is used. If the angle is within a tolerance value (epsilon) of $90n°$, the vertex is assigned the appropriate vertex type. The more this angle deviates from $90n°$, the more distortion will be present at the quadrilateral(s) sharing that vertex or node. Note that the vertex type uniquely determines the topology of the mesh sharing that vertex, and therefore can be used to determine the topology of the overall mesh.

Link: A datastructure used to store sets of edges of a submapped surface according to their parameter directions.

Discussion: A submappable surface can be assigned a local parameter space, with two parametric directions, referred to as i and j; each edge on the surface can be assigned to one of these directions. For each parameter, there is a set of bounding edges in each of the positive and negative parametric directions (e.g. $+i$ and $-i$); the classification of edges in these sets can be inferred from the choice of a reference vertex and the vertex types by traversing around boundary loop(s) of the surface. This is depicted in Figure 3. When an edge is assigned a parameter, that parameter varies in value along the edge, while the other parameter is constant along that edge.

Chain: Consider a traversal across a mappable surface, from one side to the opposite side. If the surface sharing the side traversed to is mappable, it too can be traversed in the same way. Traversal stops either when a non-mappable surface is encounted or when a side is encountered which has already been traversed. If all paths of the traversal end on a side that has already been traversed, the group of surfaces forms a loop of surfaces sharing edges of a given parameter; we define this loop as a chain. In the simplest case, a chain can be a single periodic surface (see Figure 4, Chain A) or a sequence of 4-sided mappable surfaces.

Submappable surfaces can be traversed in almost the same manner. However, since submappable surfaces can have more than four sides, the traversal path can split; for example, a traversal which begins on a $-i$ side can traverse to two $+i$ sides. Traversal continues in this manner until all traversal paths end, either by encountering a non-mapped and non-submapped surface, an edge which has already been traversed, or by joining with another traversal path. We designate the group of surfaces a chain if all traversal paths end on edges which have already been traversed. An example of a branched chain is shown in Figure 4, Chain B.

Since traversal begins with edges of a given parameter, e.g. $+i$, and proceeds only to edges of the same parameter but opposite sign, e.g. $-i$, it follows that surfaces in a chain are traversed in one parameter and not the other, and we can assign a consistent parameterization across the

[¶]In the absence of a mesh, the vertex type is assigned using the angle of the edges meeting at that vertex; once a mesh exists, the mesh topology overrides any angle-based classification. This allows the placement of end-type vertices in places not normally assigned End-type, e.g. on the boundary of a circular surface.

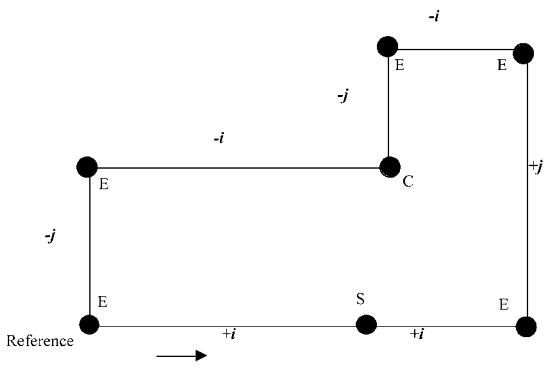

Figure 3. Submappable surface, showing vertex types (E=end, C=corner, S=side) and parametric directions $(+/-i, +/-j)$.

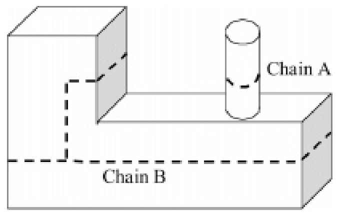

Figure 4. Chains on a solid; Chain A is on a periodic surface; Chain B is branched.

chain such that all surfaces are traversed in the same parameter. The parameter assigned to the edges being traversed is referred to as the traversed parameter, while the other parameter is the non-traversed parameter.

Sweeping: In 2D, the generation of a 2D mesh by extruding a set of connected 1D edges into a second dimension, in single or multiple steps or layers, such that the 1D topology of each layer is identical to that of the original 1D sequence of edges. In 3D, the generation of a 3D mesh by extruding a topologically 2D mesh into a third dimension, in a single or multiple steps or layers, such that the 2D topology of each layer is identical to that of the original 2D mesh. In either case, straight-line segments join the corresponding vertices between adjacent layers.

Many-to-one sweeping: A variant of 3D Sweeping, where instead of each layer having identical 2D topology, the topology of a given layer can be modified by adding surface elements sharing edges with the elements in the layer. The added elements are swept along with the original elements into the third dimension, so that subsequent layers have topology identical to the combination of the original and the added elements.

Many-to-many sweeping: Similar to many-to-one Sweeping, except that element removal as well as addition from/to the layer of elements being swept is allowed.

2.5D volume mesh: a volume mesh constructed using one of the three types of sweeping defined above.

3. WHAT MAKES A VOLUME 2.5D?

We seek a precise definition of the geometric and topological conditions under which a volume can be meshed using Sweeping. These conditions can be deduced by observing the extrusion process.

In the following discussion, we will start by describing the topology of a swept mesh, but eventually will describe swept mesh constraints in terms of the geometric surfaces and edges of a volume and the mesh schemes and intervals assigned to those surfaces and edges. This allows us to evaluate the constraints before generating mesh for any of the bounding curves or surfaces.

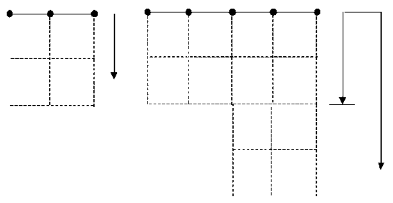

Figure 5. Extrusion of a facetted line into a mapped mesh (left); extrusion of different parts different distances into a submapped mesh (right).

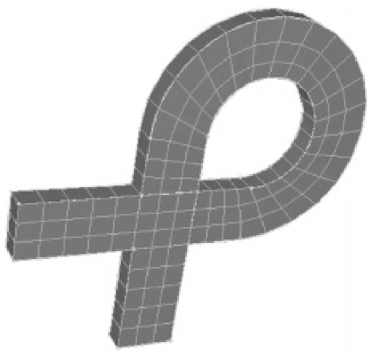

Figure 6. Volume that appears sweepable but which is not a true extrusion.

Where the terms edge and surface are used without qualification or clear context, they refer to the geometric entities rather than mesh.

By definition, extruding a logically one-dimensional segmented line into a second dimension results in another line with like topology; the line being swept is referred to as the source, while the other line is called the target. This is depicted in Figure 5, left. In three dimensions, a group of contiguous, logically two-dimensional mesh faces is swept into the third dimension, with each layer having identical mesh topology to the last and edges connecting corresponding vertices between layers; source and target surfaces are defined by the starting and ending surface(s) in the sweep. By the definition of sweeping a facetted line, as the two-dimensional facets are swept into the third dimension, the logically one-dimensional line(s) bounding them sweeps out one or more structured surface meshes (see Figure 5, right); these surfaces are referred to as the linking surfaces.

Lemma 1. A swept mesh is bounded by one or more non-intersecting chains.

Proof. Since we start with a set of mesh faces, their boundary is closed. By inspection, sweeping a closed boundary into a logically third dimension will generate a closed chain of surfaces. Since adding to or removing from the set of faces being swept does not change the fact that the faces have a closed boundary, many-to-one and many-to-many swept meshes are also bounded by chains. □

There is one situation in which chains are intersecting and a volume could be described as a swept mesh; this situation is depicted in Figure 6. However, sweeping this volume would require extruding a mesh in some portions of the volume and re-using mesh in another part of the volume. Since this is not a true extrusion, we regard this as an unsweepable volume. Indeed, none of the sweeping algorithms in References [6, 9, 10] are capable of sweeping a volume like this.

Lemma 2. The side (linking) surfaces generated by sweeping one or more contiguous sets of mesh faces into a third dimension are mapped if the boundary of the regions being added or subtracted from the sweep do not intersect the original boundary of the sweep, or submapped if the boundaries do intersect.

Proof. Consider two cases; if the boundary of the set of faces being added or removed does not intersect the original boundary, then either we are adding a set of faces which does not intersect the original set, or we are removing a subset of faces whose boundary is formed from non-boundary edges of the original set. In both cases, we are introducing a new, closed boundary, which is then swept into a linking surface which is mapped, along with the original boundary. On the other hand, if the two boundaries do intersect, then we are modifying the boundary of the original sweep; this modifies the parameter extents on the linking surface, and by definition this generates a submapped mesh.

Using the definitions of volume edge types and Sweeping, and Lemma 2, we can also observe that:

Lemma 3. In a swept volume:

(a) edges between source/target surfaces and linking surfaces are always of type End or Corner;
(b) edges between source/target surfaces are always of type Side or Reversal;
(c) edges between linking surfaces that are perpendicular to the sweep direction (i.e. edges that bound a single layer) are always of type Side or Reversal; and
(d) edges between linking surfaces that are parallel to the sweep direction (i.e. edges that bound multiple layers) can be any type.

Next, we state the following:

Lemma 4. Each set of contiguous source/target surfaces is bounded by edges having a common value of the traversed parameter with respect to the bounding linking surface(s).

Lemma 4 follows from the definitions of mapped and submapped surfaces (and their parameterizations) and Sweeping, and Lemma 2.

Lemma 5. Traversing from a set of contiguous source/target surfaces over any boundary edge of type End results in traversing a linking surface in the non-traversed parameter, no matter which End-type edge is traversed; similarly for traversing any Corner-type edge.

This lemma follows from Lemmas 2–4.

Lemma 6. All linking surfaces in a swept mesh can be assigned a global, consistent ij parameterization.

Proof. By consistent, we mean that anywhere linking surfaces meet at a shared edge, the type (i or j) and value of ij parameter along that edge is the same. For linking surfaces which are part of the same chain, a consistent parameterization exists because the surfaces are mapped or submapped and, by the definition of a chain, every edge assigned the traversal parameter is shared by two surfaces in the chain; thus, the surfaces in the chain can be unrolled to form a contiguous submapped or mapped mesh. The parameterization of one chain extends to chains whose surfaces share non-traversed edges with surfaces in that chain. Disjoint chains have related parameterizations by virtue of Lemma 4. □

Lemma 7. Traversing from the same set of contiguous source/target surface(s) onto a linking surface over an End-type versus a Corner-type edge will result in traversing the same ij parameter in opposite directions (e.g. $+i$ and $-i$) on the linking surface.

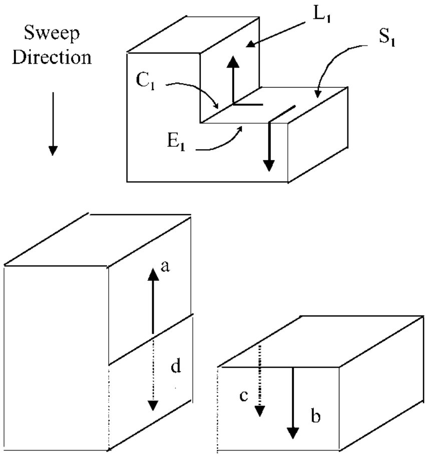

Figure 7. Traversing from a source surface over a Corner-type edge (C1) and End-type edge (E1) results in traversing the linking surface in opposite directions along the same *ij* parameter.

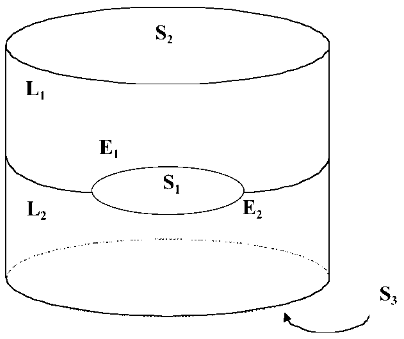

Figure 8. An example of Theorems 1(b) and (c) showing sweepability of a volume. When edges E_1 and E_2 are set to type End, the volume is not sweepable; the volume becomes sweepable if E_1 or E_2 is set to type corner.

Proof. Existence of a Corner-type edge implies the addition or removal of a region of faces from the source/target surface(s) being swept (by the definition of a Corner-type edge and of Sweeping). Consider the geometry in Figure 7 (upper), with source surface S_1, linking surface L_1, and bounding edges C_1 and E_1 of type Corner and End, respectively. If L_1 is extended into the volume, the volume is separated into two pieces which share a surface. Traversing from S_1 to L_1 over C_1 is indicated by arrow a in Figure 7 (lower), while traversing from S_1 to L_1 over E_1 is indicated by arrow b. However, by Lemma 5, this is equivalent to traversing along arrow c; because the interface is shared by both volumes, this in turn is equivalent to traversing along d. From here, it is clear that arrows a and d point in opposite directions along the same *ij* parameter of the linking surface. Using Lemmas 4 and 6, we can prove this is true even in the case where the boundary loops containing the Corner-type and End-type edges do not intersect. □

Lemmas 1–7 state several characteristics of a swept mesh; to be sweepable, a volume must at least show these characteristics. We summarize this conclusion in the following theorem:

Theorem 1. A volume is sweepable only if:

(a) the volume has one or more non-intersecting chains;
(b) for each set of contiguous source/target surfaces, traversing over all E-type or C-type edges bounding the set results in traversing in the same global *ij* parameter and direction on the linking surface(s);
(c) for each set of contiguous source/target surfaces bounded by both E-type and C-type edges, traversing over any E-type edge bounding the set results in traversing in the same global *ij* parameter and opposite direction on the linking surface as resulting from traversing over any C-type edge bounding the set.

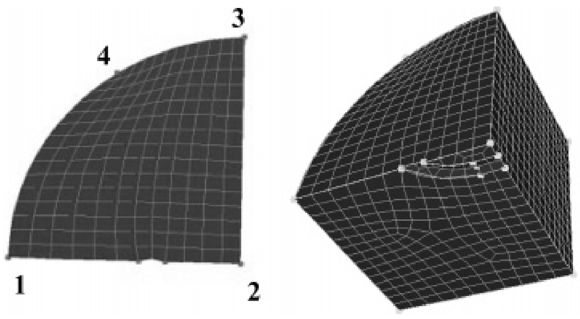

Figure 9. Vertex type adjustment for vertex 4 makes surface mappable (left); making surface mappable allows volume to be swept (right).

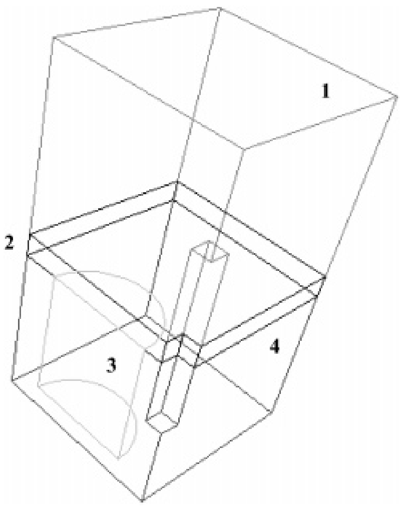

Figure 10. Four volumes demonstrating sweep order dependencies.

Proof. (a) was proven by Lemma 1; (b) and (c) were proven by Lemma 7. □

Although the conditions in Theorem 1 are necessary for sweepability, these conditions are not sufficient, even if we place no constraints on the quality of the resulting mesh. For example, consider a volume containing a through-hole, one which ties itself in a knot without its surfaces intersecting. This volume is clearly not sweepable, but still passes the constraints in Theorem 1. There is likely an additional constraint that can be placed on the bounding surface mesh which would capture examples like this; we conjecture that this constraint is similar to those discussed in Reference [15]. However, the authors have never observed an example like this in practice, nor expect to see any like this in practical applications.

Although we have proven that Theorem 1 is a necessary condition for sweepability, looking at an example will show why Theorems 1(b) and 1(c) in particular are needed. Consider the volume in Figure 8. If the edges E_1 and E_2 are designated as type End, the surfaces L_1 and L_2 each form a chain, these chains are non-intersecting, and Theorem 1(a) is satisfied. However, traversing from S_1 to L_1 and L_2 over E_1 and E_2, respectively, results in traversing the linking surfaces in opposite directions in the same *ij* parameter; this violates Theorem 1(b). On the other hand, if edge E_1 is designated type Corner, then according to Theorems 1(b) and 1(c) the volume is sweepable. In fact, the volume can be meshed by sweeping S_2 down to E_1, then adding surface S_1 and sweeping down to S_3.

When the type of edge E_1 in Figure 8 is set to Corner, the volume becomes sweepable. However, because edge E_2 is type end, quadrilateral pairs sharing edges on E_2 will be part of the same hex; the dihedral angle between these faces will be 180°. For most applications, hexes with interior angles of 180° are of insufficient quality. Therefore, even though volumes meeting the conditions in Theorem 1 are sweepable, the resulting meshes may not be of sufficient quality for analysis. Additional constraints are needed to improve mesh quality.

The combination of volume edge types and source/target surfaces uniquely determine the topology of the mesh at all geometric edges. This topology was described in the definition of the different volume edge types, and constraints on allowable volume edge types were described in Lemma 2. For the purposes of mesh quality, we must place constraints on the dihedral angles between surfaces meeting at various edge types; these constraints are similar to those used in the submapping algorithm [4].

Assuming that an ideal hex element is one whose angles are all 90°, each of the vertex types in Figure 2 (and the corresponding volume edge types) has a corresponding ideal angle; these angles are 90, 180, 270 and 360° for End, Side, Corner and Reversal types, respectively. Around these ranges are 'fuzzy' regions, approximately 45° on either side of the ideal, within which hex quality is degraded but not inadequate. Most finite element applications require that face angles be bounded between zero and 180°. These angle constraints impose limits on angles at the given edge types. The angle at End-type edges is limited to 180°; Side-type edges are limited to 360°. Corner and Reversal type edges are limited to 360°, not to prevent hex angles greater than 180° but to prevent overlapping elements. These element constraints are summarized in the following theorem.

Theorem 2. To ensure element angles less than 180° and non-overlapping elements, End-type edges must be less than 180°, and Side, Corner and Reversal edges must be less than 360°. All angles must be greater than 0°.

Finally, the available sweep algorithm may determine additional constraints on the topology of surfaces in S or L. The 'standard' sweep algorithm allows the extrusion of one surface (the 'source' surface) onto a single 'target' surface of identical topology; this is often referred to as a one-to-one sweep [9, 16]. There also exist other varieties of sweeping which have fewer constraints on source and target topology, e.g. 'many-to-one' or $n \rightarrow 1$ sweeps [17], and 'many-to-many' or $n \rightarrow m$ sweeps [10, 18]. Many-to-many sweeps may require modifications to the toplogy of source or target surfaces, depending on implementation [18]. We summarize these notes in the following Theorems:

Theorem 3. Volumes in which source/target surfaces share an edge of type Reversal require multiple source and target sweeping; volumes in which source/target and linking surfaces share edges of type Corner require multiple source/single target or multiple source/target sweeping; volumes in which linking surfaces share edges perpendicular to the sweep direction of type Reversal require multiple source/single target sweeping; and volumes containing multiple source/target surfaces require multiple source and possibly multiple target sweeping.

Some algorithms may also impose topological restrictions on linking surfaces; for example, some implementations require the linking surfaces to all be mappable (having only four sides), while others allow more general submappable surfaces with imprints which form interior holes. These additional constraints can be evaluated after the determination of whether a volume is sweepable. These considerations lead to a fourth type of sweepability criterion.

Theorem 4. Sweepable volumes contain constraints on the topology of linking surfaces; these constraints depend on the Sweeping implementation being used.

Fuzzy regions around the ideal angles for the four vertex and edge types allow the use of submapping and mapping schemes on surfaces, and sweeping on volumes, where they might not typically be used. For example, the surface mesh shown in Figure 9 was obtained by setting vertex 4 to type End, even though by angle measurement it was closer to type Side. Although this degraded mesh quality locally, it allowed the volume to be swept, and the resulting mesh had an acceptable overall mesh quality.

For application to mesh generation, the definition of a sweepable volume is less constrained geometrically, since it is not strictly required that the cross section of the extrusion have a constant area or even that its normal maintain a constant direction. This makes the detection of mesh-sweepable volumes considerably more efficient, since there is no need to evaluate global geometric constraints. At the same time, an extruded volume in the mesh generation application is more constrained in terms of topology, as represented by Theorems 1 and 3 above. There are also local geometric constraints, represented by Theorem 2. However, topological and local geometric constraints are easier and faster to evaluate than the global geometric constraints used in the past.

4. AUTO SWEEP DETECTION ALGORITHM

Theorem 1 in the previous section identified the conditions necessary for a volume to be sweepable. In this section, we describe the algorithm used to verify that the conditions in Theorem 1 are met by a given volume. The auto sweep detection algorithm consists of the following four steps:

(1) *Classify surface mesh schemes locally using surface vertex types.*
Automatic surface mesh scheme assignment is accomplished by first computing surface vertex types for all vertices on a surface, then summing the angles at the vertices (assigning the numbers 1, 0, -1 and -2 to end, side, corner, and reversal vertex types, respectively). If this sum is equal to four, the surface is submappable [4]. Note that the criteria used to assign vertex types for automatic surface mesh scheme assignment are somewhat more conservative than those used to find the 'corners' in the submap algorithm when the user assigns that scheme explicitly. More conservative vertex type criteria lead to fewer surfaces that are assigned a submap scheme; the surfaces not assigned submap are usually assigned the paving scheme [19].

(2) *Group mappable and submappable surfaces into chains.*
Once the corners have been chosen, mapped and submapped surfaces have an implied, logical 2D parameter space defined. This parameter space is implied even before meshing occurs, since the corners define the structure of the mesh if not the actual mesh. This parameterization can be used to traverse from a given edge across the surface to the 'opposite' edge (see Figure 3). This traversal can continue across the surface sharing the second edge, and so on until either it meets up with the original edge again or it encounters a surface that is not assigned the map or submap mesh scheme. If a complete loop of surfaces results, this group of surfaces define a chain.

If a submapped surface is encountered during a traversal, the loop may branch into more than one section; this branching is analogous to a parallel circuit. For example, Figure 3 shows one such chain, part of which branches into two parallel sections (parallel in a topological sense, not a geometric sense).

Table I. Pseudo-code for traversing source/target surfaces.

```
• find a non-submapped surface r; insert r in source/target list S; call
  prop_to_next_src_tgt (r, S)
• Procedure prop_to_next_src_tgt (Surface r, Src/Tgt List q):
  • put r in source/target list q
  • for each edge e in surface r:
      • find other surface sharing that edge, s
      • if s is already in S, T or L, continue to next edge
      • else if edge e is an end or corner, call link_to_next_src_tgt (r,e,q)
      • else if e is side or reversal type:
        • if e is a side type, choose p = q
        • else if e is a reversal type, choose p = opposite (q)
        • call prop_to_next_src_tgt (r, p)
• Procedure link_to_next_src_tgt (Surface r, Edge e, Src/Tgt List q):
  • if r is neither mapped nor submapped, or has no valid chains, return NOT_
    SWEEPABLE
      • put r in linking surface list L
      • for each edge f of surface r:
        • if edge f has ij parameter different from that of e, continue at next
          edge
        • get other surface s sharing edge f
        • if s is already in S, T or L, continue at next edge
        • else if edge f has type side, call link_to_next_src_tgt (s, f, q)
        • else:
          • set type of next source/target surface s by calling p=next_type
            (e, f, q)
          • call prop_to_next_src_tgt(s, p)
• Procedure next_type(Edge e, Edge f, Src/Tgt List p)
  • If e and f are different type (End-Corner or Corner-End), return p
  • Else if e and f are same type (End-End or Corner-Corner), return opposite (p),
    where opposite(SRC_LIST) = TGT_LIST, etc.
```

The result of looping the surfaces is a number of chains. If the number of chains is zero, there are no complete loops of mappable/submappable surfaces, therefore the volume is not sweepable. If on the other hand, all the bounding surfaces of a volume are mappable or submappable, then the volume itself is submappable. Finally, there are many cases where a submappable surface is part of two chains, but the volume is not submappable. In these cases, one or more source/target surfaces will be mappable or submappable.

(3) *Compute volume edge types.*
The computation of volume edge types is not done until looping the surfaces bounding the volume has been done; the cost of this step is saved in the cases where the volume is either submappable or when no chains are found. Note that computing volume edge types is an operation local to the edge and the two surfaces connected to it.

(4) *Traverse surfaces, grouping them in S and L.*
In addition to determining whether a volume is sweepable, it must be determined which are the source and target surfaces for the sweep. This is done by traversing the surfaces; pseudo-code for this traversal process is shown in Table I.

There are several things to note in the pseudo-code shown in Table I. First, the traversal of source/target and linking surfaces is recursive, and ends when either all the surfaces have been put into one of the lists (S, T or L), or a surface is encountered which should be a linking surface (i.e. submappable and part of a valid chain) but which is not.

The result of this traversal will be a determination of whether or not the volume is sweepable, and if so, what the source and target surfaces are. This latter information is useful for evaluating criteria in Theorems 1(b), (c) and 2–4. The assignment of source and target lists is arbitrary; that is, these lists can be interchanged. This can be useful when the target surface(s) are meshed or when there are multiple targets and a single source (switching the lists in this case simplifies the sweep).

5. AUTOMATIC SWEEP GROUPING

After assigning volume sweep schemes, these volumes can be meshed individually. However, in many applications, the volumes are part of a larger assembly, where the volumes share surfaces and their corresponding mesh. In this case, there are additional constraints on the order of meshing volumes that must be met to avoid further complications in sweeping the volumes.

For example, consider the four volumes in Figure 10. If these volumes are meshed in an arbitrary order, for example volumes 1, 2, 4 then 3, the last volume will already have mesh on all its source and target surfaces specified. Changing the mesh on these surfaces is not possible without changing the adjoining volume meshes, which can be difficult. Therefore, meshing volume 3 will be difficult because the source and target meshes will have to coincide. By Theorem 1(a), the number of faces on the source and target surfaces must be the same; if they are not, the volume will not be sweepable.

It is clear that the volumes in Figure 10 can be meshed in an order which avoids any of the above difficulties; for this example, the order is volume 3, 4, 2 then 1. In fact, it is not difficult to build this order automatically by traversing the connected volumes over their source and target surface lists. The algorithm for this traversal, referred to as ‘automatic sweep grouping’, is shown in Table II.

Automatic sweep grouping is useful primarily for resolving dependencies in sweep order when sweeping assemblies of volumes. However, it is also useful for separating a large collection of volumes into independent groups, each of which can be meshed independently of the others.[‖] This facilitates the meshing of large assemblies by teams of users, and has significantly reduced the wall clock time for hex meshing these assemblies.

[‖]These sweep groups are not completely independent, because they must satisfy interval constraints as described in Reference [21]. However, these constraints can be satisfied before meshing begins; the mesh on linking and other interface surfaces is then deterministic, so it does not matter which volume owning one of these surfaces generates its mesh.

Table II. Pseudo-code for building sweep dependencies.

```
• Collect all sweepable volumes and unmark them
• Collect all sweepable “end” volumes (volumes with source/target surface not shared
  by another sweepable volume)
• For each end volume v:
  • If v is marked, continue to next volume
  • Mark v
  • Call prop_from_source (v, source_volume_list)
  • Call prop_from_target (v, target_volume_list)
  • Sweep_group = source_volume_list + target_volume_list
• If any unmarked sweeepable volumes, resolve cyclic sweep groups

Procedure prop_from_source (volume v, volume list source_vol_list)

• Get source face list for v
• For each source face f:
  • a = adjacent volume from v across f
  • If a is NULL, continue to next face
  • Else if a is not assigned sweep scheme, continue
  • Else if a is marked, continue
  • Else
       • Mark a
       • Call prop_from_source (a, source_vol_list)
       • Insert a in source_vol_list
       • If f is a source face for a:
           • Call prop_from_target (a, source_vol_list)

Procedure prop_from_target (volume v, volume list target_vol_list)

• Get target list for v
• For each target face f:
  • a = adjacent volume from v across f
  • If a is NULL, continue
  • Else if a is not assigned sweep scheme, continue
  • Else if a is marked, continue
  • Else
       • Mark a
       • Call prop_from_source (a, target_vol_list)
       • Insert a in target_vol_list
       • Call prop_from_target (a, target_vol_list)
```

6. DISCUSSION AND EXAMPLES

There are many uses for an automatic sweep detection algorithm. First and foremost, the algorithm is useful for automating the specification of source and target surfaces for volumes that the user knows are sweepable. For example, the geometry shown in Figure 11 consists of 33 sweepable volumes; auto sweep detect determines that these volumes are sweepable, and sets the source and target surfaces for each automatically. Note that in this example, the block contains both blind holes and through-holes. More examples of sweepable volumes detected by this algorithm are shown in Figures 12–14.

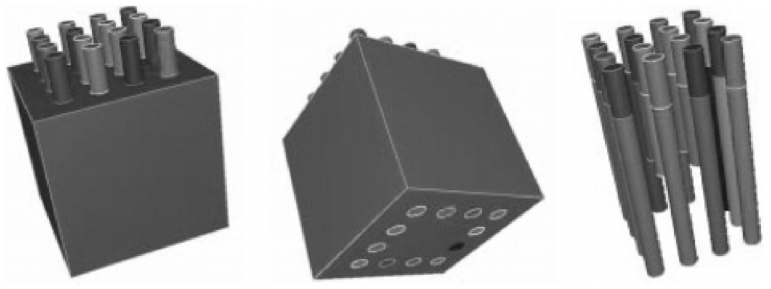

Figure 11. Collection of volumes whose schemes, source and target surfaces were detected automatically. Note inner core of rods which produces blind holes in block.

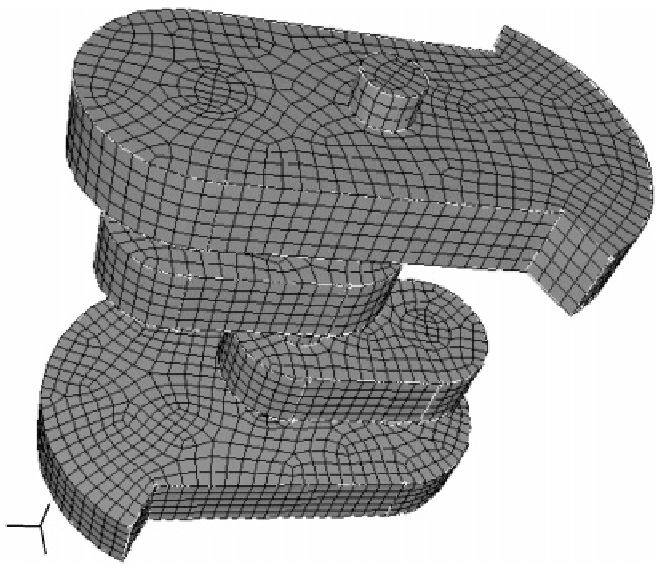

Figure 12. Volume whose sweep scheme and source and target surfaces were determined automatically.

A less obvious application of auto sweep detect is to filter a large group of volumes into sweepable and non-sweepable ones; this allows the user to quickly determine which volumes are not meshable and need to be decomposed further (in the case where a 3D auto hex algorithm is not available). For example, on a large meshing application at Sandia, this algorithm identified 220 of 245 volumes as being sweepable, leaving only 10% of the volumes for further inspection and decomposition [20]. Thus, this algorithm can be used to quickly reduce the number of volumes needing further interactive decomposition.

The identification of chains of surfaces emerging in the sweeping process has also been used to identify independent sweep paths, which in turn is used to determine mesh interval constraints which guarantee that the resulting volume can be meshed [21]. This is useful when sweeping volumes containing through-holes, for example.

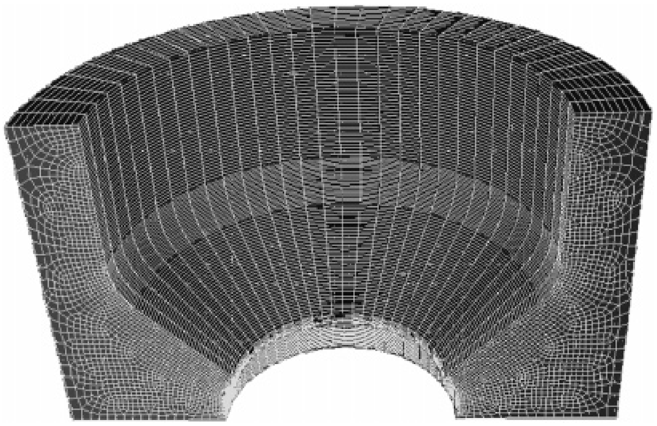

Figure 13. Volume whose sweep scheme and source/target surfaces were determined automatically.

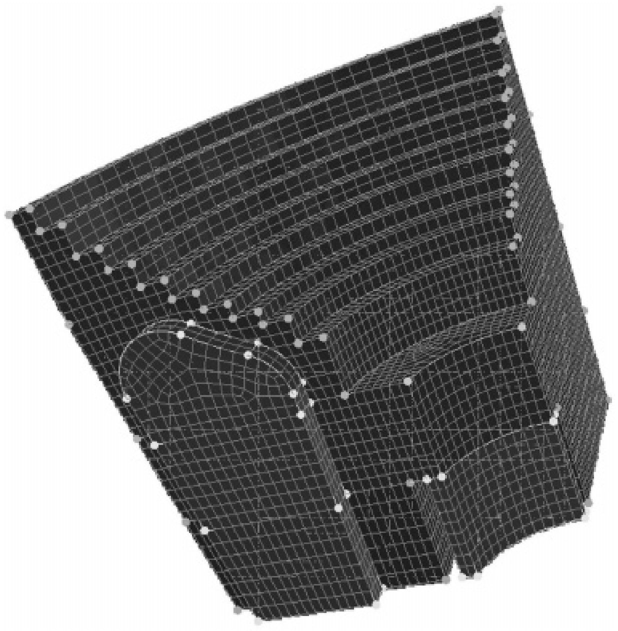

Figure 14. Group of volumes whose sweep schemes and source and target surfaces were determined automatically.

This algorithm is also useful for guiding an automatic geometry decomposition tool that is being developed [22]. Auto sweep detect can provide a stopping criterion for the decomposition tool, terminating decomposition when all the pieces are sweepable. Chains of surfaces bounding volumes which are not sweepable may also be useful for guiding further decomposition.

7. CONCLUSIONS

In this paper we have listed conditions under which a volume is sweepable, and have proven that these conditions are necessary for sweepability. These conditions are listed in three theorems, and are based on global topology and local geometry. We have also shown that these conditions are not sufficient for some cases, namely volumes whose genus is not zero.

An algorithm for determining whether or not a volume is sweepable, and if so, identifying the source and target surfaces, has also been described. This algorithm relies on local geometry in the form of vertex and edge types, as well as on surface topology. Because only local geometric information is used, the algorithm is quite efficient and reliable.

Finally, it has been shown that there are also sweep order dependencies which must be met to avoid difficulty when meshing assemblies of volumes; in some cases, not satisfying these dependencies can result in some of the volumes not being sweepable. An algorithm which computes these dependencies has also been given; this algorithm relies only on topology information for swept volumes, and is therefore quite efficient.

The auto sweep detect algorithm has been used to reduce interactive user time required to generate all-hexahedral meshes by filtering out non-sweepable volumes needing further decomposition. Parts of the algorithm have also been used to identify independent sweep paths, for use in volume-based interval assignment. The automatic sweep grouping algorithm has also been useful for resolving sweep dependencies, and therefore reducing interactive time meshing large assemblies. It has also been useful for splitting large meshing jobs into independent pieces which can be meshed concurrently by multiple users.

The auto sweep detect algorithm may also prove useful for guiding automatic geometry decomposition, both by providing a stopping criterion for such a tool, and for guiding further decomposition using completed chains on unsweepable volumes.

ACKNOWLEDGEMENTS

We would like to acknowledge the insightful questions posed by Dr. Graham Carey of UT-Austin during the presentation of this paper. We are also grateful for the helpful comments from Dr. Scott A. Mitchell of Sandia National Labs in verifying the correctness and clarity of the theorems derived in this paper. Finally, we acknowledge the staff and sponsors of the CUBIT mesh generation project at Sandia National Labs, without whom this work would not have been possible.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin company, for the United States Department of Energy under contract DE-AC04-94AL85000.

REFERENCES

1. Folwell NT, Mitchell SA. Reliable whisker weaving via curve contraction. *Proceedings of the 7th International Meshing Roundtable*, SAND98-2250, Sandia National Laboratories, Albuquerque, NM, October 1998.
2. Scheffer A, Etzion M, Rappoport A, Bercovier M. Hexahedral mesh generation using the embedded voronoi skeletons. *Proceedings of the 7th International Meshing Roundtable*, SAND98-2250, Sandia National Laboratories, Albuquerque, NM, October 1998.
3. Muller-Hannemann M. Hexahedral mesh generation by successive dual cycle elimination. *Proceedings of the 7th International Meshing Roundtable*, SAND98-2250, Sandia National Laboratories, Albuquerque, NM, October 1998.
4. White DR. Automated hexahedral mesh generation by virtual decomposition. *Proceedings of the 5th International Meshing Roundtable*, SAND96-2301, Sandia National Laboratories, Albuquerque, NM, September 1996.
5. Stephenson MB, Blacker TD. Using cojoint meshing primitives to generate quadrilateral and hexahedral elements in irregular regions. *Proceedings of the ASME Computations in Engineering Conference*, 1989.

6. Knupp PM. Applications of mesh smoothing: copy, morph, and sweep on unstructured quadrilateral meshes. *International Journal for Numerical Methods in Engineering* 1999; **45**:37–45.
7. FEMAP, Enterprise Software Products, Inc., http://www.femap.com.
8. ANSYS, ANSYS, Inc., http://www.ansys.com.
9. Staten ML, Canann SA, Owen SJ. BMsweep: locating interior nodes during sweeping. *Proceedings of the 7th International Meshing Roundtable*, SAND98-2250, Sandia National Laboratories, Albuquerque, NM, October 1998.
10. Blacker T. The Cooper Tool. *Proceedings of the 5th International Meshing Roundtable*, SAND96-2301, Sandia National Laboratories, Albuquerque, NM, September 1996.
11. Subrahmanyam S, Wozny M. An overview of automatic feature recognition techniques for computer-aided process planning. *Computers in Industry* 1995; **26**:1–21.
12. Razdan A, Henderson MR, Chavez PF, Erickson PA. Feature based object decomposition for finite element meshing. *The Visual Computer* 1989; **5**:291–303.
13. Blacker TD *et al*., CUBIT mesh generation environment, vol. 1: user's manual. SAND94-1100, Sandia National Laboratories, Albuquerque, NM, May 1994.
14. Mortenson ME. *Geometric Modeling*. Wiley: New York, 1985.
15. Mitchell SA. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume. *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, Springer: Berlin, 1996; 465–476.
16. Gilkey AP, Sjaardema GD. GEN3D: A GENESIS database 2D to 3D transformation program. SAND89-0485, Sandia National Laboratories, Albuquerque, NM, February 1994.
17. Mingwu L, Benzley SE, Sjaardema G, Tautges T. A multiple source and target sweeping method for generating all-hexahedral finite element meshes. *Proceedings of the 5th International Meshing Roundtable*, SAND96-2301, Sandia National Laboratories, Albuquerque, NM, September 1996.
18. Mingwu L, Benzley SE, White DR. Automated hexahedral mesh generation by generalized multiple source to multiple target sweeping. *International Journal for Numerical Methods in Engineering* 2000; **49**:261–275.
19. Cass RJ, Benzley SE, Meyers RJ, Blacker TD. Generalized 3-D paving: an automated quadrilateral surface mesh generation algorithm. *International Journal for Numerical Methods in Engineering* 1996; **39**:1475–1489.
20. Knupp P, Melander DJ, Mitchell SA, Tautges TJ, White DR. Personal experience modeling the MC4380 Neutron Generator Tube, Sandia National Laboratories, 7/98.
21. Shepherd J, Benzley S, Mitchell SA. Interval assignment for volumes with holes. *International Journal for Numerical Methods in Engineering* 2000; **49**:277–288.
22. Lu Y, Gadh R, Tautges TJ. Feature-based hexmeshing methodology: feature recognition and volume decomposition. *Computer Aided Design* 2000, to appear.