Engineering with Computers (2004) 20: 222–236
DOI 10.1007/s00366-004-0290-6

ORIGINAL ARTICLE

David R. White · Sunil Saigal · Steven J. Owen

CCSweep: automatic decomposition of multi-sweep volumes

Received: 28 July 2003 / Accepted: 12 April 2004 / Published online: 31 July 2004

Abstract CCSweep is a new method to automatically decompose multi-sweepable volumes into many-to-one sweepable volumes. Multi-sweepable volumes contain both multiple source and multiple target faces. In hexahedral mesh generation, most sweeping techniques handle many-to-one sweepable volumes that contain multiple source faces, but they are limited to volumes with only a single target face. Recent proposals to solve the multi-sweep problem have several disadvantages, including: indeterminate edge sizing or interval matching constraints, over-dependence on input mesh discretization, loop Boolean restrictions on creating only loops with even numbers of nodes, and unstable loop imprinting when interior holes exist. These problems are overcome through CCSweep. CCSweep decomposes multi-sweep volumes into many-to-one sweepable sub-volumes by projecting the target faces through the volume onto corresponding source faces. The projected faces are imprinted with the source faces to determine the decomposition of the solid. Interior faces are created to decompose the volume into separate new volumes. The new volumes have only single target faces and are represented in the meshing system as real, solid geometry, enabling them to be automatically meshed using existing many-to-one hexahedral sweeping approaches. The results of successful application of CCSweep to a number of problems are shown in this paper.

Keywords Multi-sweep · Sweeping · Mesh generation · Hexahedral · Volume decomposition

Contract/grant sponsor: Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

D. R. White · S. Saigal · S. J. Owen
Department of Civil and Environmental Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213, USA

D. R. White (✉) · S. J. Owen
Sandia National Laboratories, MS 0822, PO Box 5800,
Albuquerque, NM 87185-0847, USA
E-mail: drwhite@sandia.gov
Fax: +1-505-8450833

S. Saigal
Department of Civil and Environmental Engineering,
University of South Florida,
Tampa, FL 33620-5350, USA

1 Introduction

Finite element analysis (FEA) is an increasingly important tool for scientists and engineers to simulate complex, three-dimensional (3D) physical phenomena. FEA begins by discretizing the physical domain into elementary shapes, such as hexahedra or tetrahedra. The accuracy, and for some problems, the feasibility of FEA is dependent on this discretization. For many physical problems, quadratic tetrahedral elements are both accurate and efficient, though hexahedral elements are required for other problems. Automatically filling or meshing shapes with hexahedral elements remains a difficult problem, unlike generation of tetrahedral elements where numerous automatic implementations exist. Sweeping has been introduced in the literature [1–5] as an effective tool to address hexahedral meshing. A brief description of sweeping and associated open problems is first presented.

1.1 Hexahedral meshing and sweeping

While many algorithms attempt automatic hexahedral mesh generation [6–9], user intervention is typically required before a desirable mesh can be obtained. A common approach is to decompose the shape or solid model into sub-volumes that can be individually meshed with a known hexahedral meshing primitive. Some of the common primitives employed are: rectangle or mapping, sphere, tetrahedron, and cylinder. The cylinder primitive, created by "sweeping" a quadrilateral mesh from one end to the other, has been extended to generate hexahedra for any shape where a sweeping path can be found between top and bottom faces [10]. Such a decomposition allows the process of sweeping to generate hexahedral meshes. Several types of sweeping algorithms are possible, and are illustrated in Fig. 1. *One-to-one* sweeping consists of a volume where the sweep path goes from a single top or source face to a single bottom or target face. A natural extension to one-to-one sweeping is *many-to-one* sweeping, where the algorithm generates meshes for volumes with multiple source faces that sweep to a single target face [1, 2]. The multiple source faces in a many-to-one sweep can occur adjacent to one another or separately, at different depths of the sweep path. A further extension of the sweeping technology is the *many-to-many* or *multi-sweep* [3–5] technique, where there are multiple sources and multiple target faces. Multi-sweep requires that the geometry be identified or decomposed, at a minimum, into a state where many-to-one techniques may be employed. The decomposition process is key to successful hexahedral mesh generation. This paper addresses the task of decomposing a multi-sweepable volume into many-to-one sweepable regions.

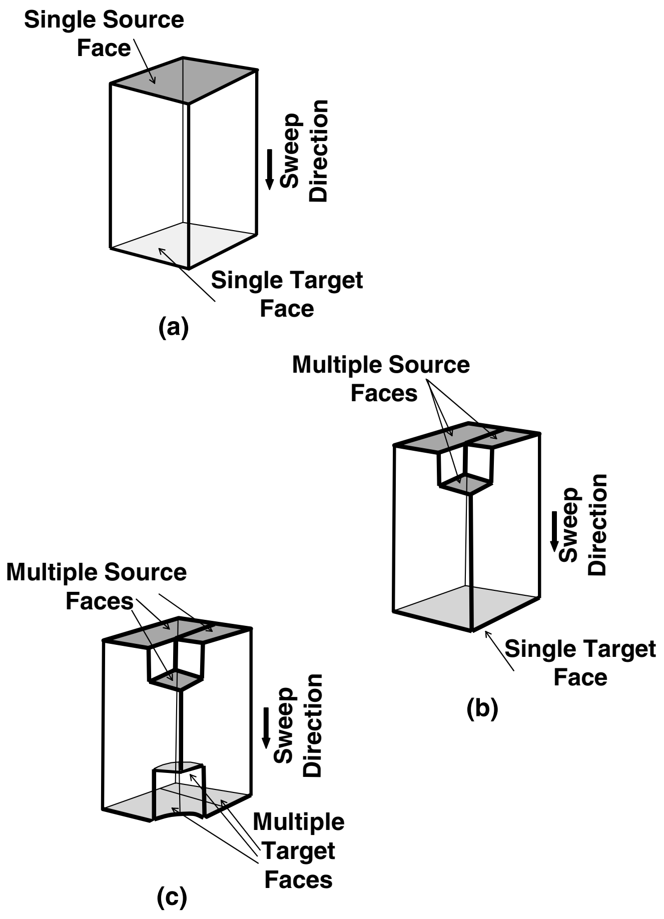


Fig. 1 **a** One-to-one. **b** Many-to-one. **c** Many-to-many or multi-sweep

1.2 Existing multi-sweep techniques

Three competing approaches introduced to solve the multi-sweep problem include those proposed by Blacker [3], Lai et al. [4], and Shepherd et al. [5]. In each of these approaches, the side walls of the sweep or the linking faces are first meshed with structured quadrilateral elements using mapping [11] or submapping [12] meshing algorithms. Next, the node loops on the target faces that consist of the ordered set of nodes formed by traversing the boundary edges of the faces, are projected upward through the volume, and are guided by the linking face meshes.

The node loops of the target faces are projected onto the appropriate source faces, and Boolean operations are performed to "imprint" the topology of the target faces onto the source faces. An imprint operation is performed by the equivalent of three separate Boolean operations; two subtractions and one intersection. Imprinting results in creating new faces that topologically match the boundaries of where the original faces overlapped. Once Boolean operations have been performed and source faces modified, the source faces can be meshed using an existing quadrilateral meshing technique [13, 14]. The hexahedral mesh is then defined by sweeping the source surface mesh to the corresponding target faces.

Several restrictions or limitations with the existing techniques described in the literature have been observed.

Indeterminate edge sizing This results when edges of the target faces are projected onto edges of the source faces. Since intersections are restricted to node matching, this means that the edge sizing on the target faces must match the edge sizing on the source faces. Prior to multi-sweeping, this information is not known since it is determined during the Boolean process itself. Figure 2a shows a multi-sweep volume that has its target faces projected onto its source faces. In Fig. 2b, the boundary nodes of a source and target face are shown projected on top of each other. In this example, because there are an unequal number of nodes, resolving the Boolean operation is impossible since the nodes cannot be made to match up exactly. Similar problems exist whenever portions of the edges are determined to match. The current techniques must rely on user intervention to match the nodes on edges prior to multi-sweeping.

Over-dependence on input mesh discretization This results from the Boolean operation relying solely on node matching. The way in which the source and target faces will be imprinted is determined by the boundary discretization. The inherent assumption made by the existing algorithms is that the size used for meshing the volume is also the best size for imprinting the source faces; this is typically not the case. While a coarse hexahedral mesh may be ultimately desired for many parts, the imprinting process often works better with a smaller size. Finer discretizations represent curved

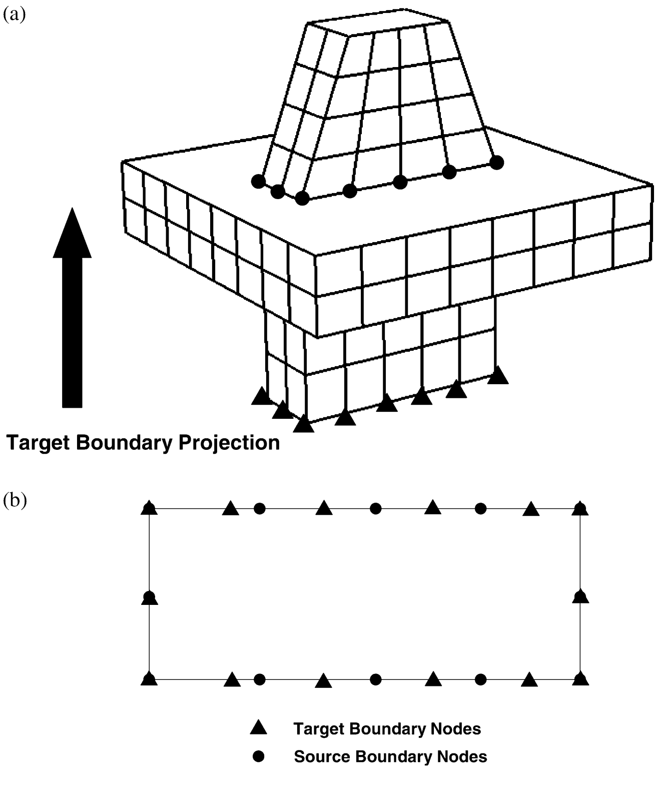


Fig. 2a, b Impossible source target node matching

boundaries more accurately and tend to produce more desirable imprints on the source faces. The current approaches couple final mesh size and imprinting discretization size, often leading to undesirable results.

Loops with even number of nodes Loop Boolean restrictions on creating only loops with an even number of nodes can result in poor node matches. The even-noded loop constraint comes from quadrilateral meshing theory, which proves that any loop with an even number of boundary points can be filled with quadrilateral elements [13]. This implies that, if the boundary has an odd number of points, it cannot be meshed with quadrilaterals. Restricting the node matching with these criteria can be problematic. For instance, when performing a Boolean operation on two loops that overlap, nodes are matched at the area of intersection based on two criteria: geometric proximity and the number of nodes in the loops that would result from the match. Even if a node is geometrically equivalent with another node, it may not be matched because doing so would result in an odd-numbered loop. This, in turn, may cause important intersections to be missed, resulting in problems later in the meshing process.

Unstable loop imprint when interior holes exist This occurs when attempting to determine if one loop is completely inside another loop. When the loops lie in a 2D plane, the solution is readily known [15]. During multi-sweeping, the loops on the layers of the linking mesh are not guaranteed to be planar. Approaches by Lai [2] and Shepherd [5] employ solutions to this problem by using the winding number algorithm [15]. The approach currently taken by Blacker [3] avoids this altogether by not allowing volumes with "unconnected" interior holes to be multi-swept, forcing the user to connect the interior and exterior boundaries through decomposition or to split the face with an edge. The winding number approach breaks down when the loops deviate too much from planarity. Since it is difficult to quantify "too much," and since there is no guarantee that the loops will be planar, instabilities in the algorithms appear.

This paper presents a new algorithm, CCSweep, that avoids each of the four problems encountered with existing approaches. The algorithm automatically decomposes multi-sweep volumes into many-to-one sub-volumes, where each sub-volume can be automatically meshed with any of the existing hexahedral many-to-one sweeping techniques [1, 2].

2 The CCSweep algorithm

CCSweep, named because of its similarities with the process of "*cookie cutting*," automatically decomposes a multi-sweep volume into meshable sub-volumes. The decomposition is achieved by using the target faces as cutting tools and pushing the target faces through the volume while intersecting them with appropriate source faces. As the cutting tools are pushed through the volume, internal nodes are created which are eventually triangulated to create interior surfaces. These surfaces are used to decompose the volume using virtual topology operations [16, 17]. The following outline provides a summary of the CCSweep algorithm. The process is illustrated in Figs. 3 and 4 for the simple geometry of a cylinder of radius 5.0 units and height 10.0 units, with the top and bottom faces of the cylinder split in halves. More details are provided in the following sections:

1. *Mesh linking faces*: the faces between sources and targets are meshed using a mapping or sub-mapping technique. For the multi-sweep volume in Fig. 3a, the linking face mesh is shown in Fig. 3b.
2. *Build layer information*: traverse the exterior linking mesh to determine how the nodes are connected in the sweep directions. Store on each node the node previous to it or the node in the source direction, and the node next to it, or the node in the target direction. Additionally, number each source and target face, and each node according to their respective vertical layer, starting at zero for the bottom most target faces and nodes. Figure 3b shows the layer information as it is constructed, including the vertical node connectivity for one column of nodes.
3. *Build control loops*: traverse the exterior linking mesh to determine the outer most set of nodes on each layer of the sweep. Determine the vertical

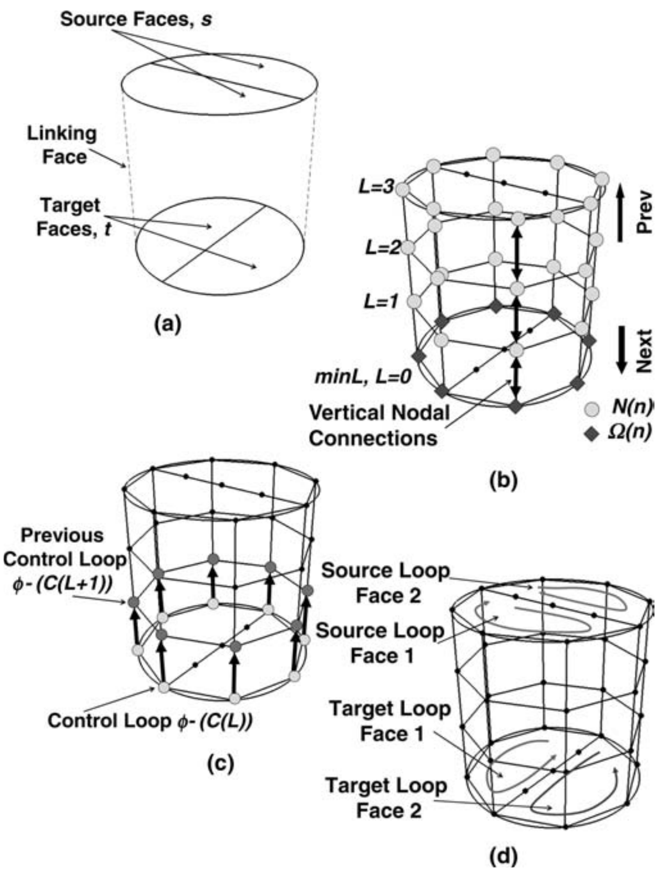


Fig. 3 a Cylinder example geometry. **b** Linking mesh and node connectivity. **c** Construction of control loops. **d** Construction of loop face data structure

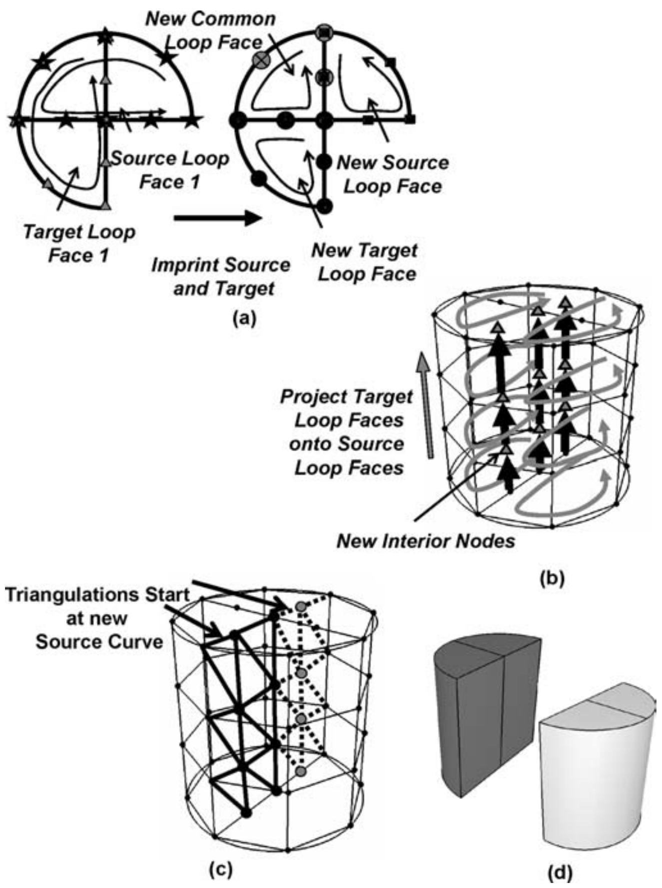


Fig. 4 a Projection of target loop faces onto source loop faces. **b** Imprinting of target face and source face. **c** Internal triangle creation. **d** Decomposition by virtual geometry operation

connectivity of these exterior node loops for later use during the node projection process. Figure 3c illustrates the control loops on the exterior of the volume and the connectivity of each loop through the linking nodes.

4. *Build loop face data structures*: build loop face data structures to both topologically and geometrically approximate the source and target faces. Figure 3d illustrates the loop faces in the multi-sweep volume.
5. *Process each loop*: on each layer of the mesh:
   - (a) *Get loop data structures on current layer*: find the loop data structures for the source and target faces in the current layer.
   - (b) *Imprint and merge sources and targets*: imprint and merge adjacent target faces to make sure the loop data structures are conformal. Imprint and merge overlapping and adjacent source and target faces. Create new source and target loop data structures to represent the imprint. Propagate any new nodes created during the imprinting process up and down the linking face meshes. Figure 4a shows the imprinting of one of the source loop faces with one of the projected target loop faces. The figure also illustrates the result of the imprint as three connected loop faces.
   - (c) *Assign control loops*: for each loop face data structure, determine the control loop governing its transformation to the following layer.
   - (d) *Project loop faces*: for each control loop on the layer, build the affine transformation matrix and corresponding nodal residual errors. For each target loop face in each control loop, project interior nodes using the residual errors and transformation matrix. Build new loop face data structures on the new layer, copying the topology of the face on the prior layer. In Fig. 4b, the target loop faces are projected from the bottom of the volume onto the source faces. Interior nodes are created on each layer as the target loop faces progress through the volume.
6. *Build decomposition triangulations*: using the imprinted source loop faces, build internal triangulations with the interior nodes created in the main control loop. Collect the triangles into sets that represent decomposition faces. For the cylinder example, the decomposition triangulations are shown in Fig. 4c.
7. *Decompose*: use virtual geometry Boolean operators to decompose the volume into separate many-to-one sweepable volumes. In the example problem, the final decomposition of the cylinder is shown in Fig. 4d/

2.1 Starting CCSweep

Prior to beginning the CCSweep algorithm, an initial size discretization is determined and the source and target faces of the volume are identified. Currently, the size for CCSweep is chosen automatically, based on a heuristic relation that considers the volume of the part and a maximum spanning angle of 60° on curved edges. Because the size used during CCSweep is decoupled from user-desired element size, any mesh size that produces good meshes on the linking faces may be used. The goal of this sizing is to quickly obtain the minimum size that captures the features of the volume. Source and target face identification is done using the auto-sweep selection algorithm proposed by White and Tautges [18].

2.2 Mesh linking faces

The first step in the CCSweep algorithm is to discretize the linking faces of the volume. The linking faces must be meshed with a structured meshing scheme so that a system of layers between the source and target faces can be identified. Without such layering, the formation of hexahedral elements on the volume is not possible. The layering system helps to identify the transformation of the target faces onto the source faces. Like the existing multi-sweep algorithms, the linking face meshes used for CCSweep must be of good quality to enable proper transformations from the target to source faces. Mapping and submapping algorithms are used to mesh the linking faces with structured quadrilateral elements. The grid lines on the sample geometry in Fig. 2a illustrate examples of linking face meshes. Unlike the existing techniques described in the literature, the structured mesh on the linking faces is used only as a temporary mechanism to facilitate volume decomposition. The linking face meshes can be discarded once the decomposition process is complete, allowing the user to later impose any desired mesh resolution to the new CCSweep volumes.

2.3 Build layer information

Once the linking faces are meshed, the nodes on the linking faces are traversed vertically in the direction of the sweep path. This step provides two important relationships. First, it determines the layer numbering for the source and target faces, s and t. Second, it stores vertical connectivity relationships and layer numbering on the nodes, n, between s and t. The algorithm to connect linking faces is shown in Fig. 5. For the multi-sweep volume in Fig. 3a with source faces s, and target faces t, the boundary nodes $\Omega(n)$, and their respective vertical connections, $N(n)$, are shown in Fig. 3b.

In step 5 of Fig. 5, the nodes found on the boundaries of the connected source or target faces are taken and propagated away from the source or target faces until they hit another set of source or target faces. During the propagation, the layer value is increased or decreased, depending on the direction indicated from the boundary quadrilaterals. The layer value is set for each node in the quadrilateral, and the nodes connected vertically in the quadrilateral are stored for later use in the CCSweep algorithm. In step 8, as new faces are reached through the quadrilateral propagation, they are pushed onto the stack containing the faces, which are iterated over in steps 4–10.

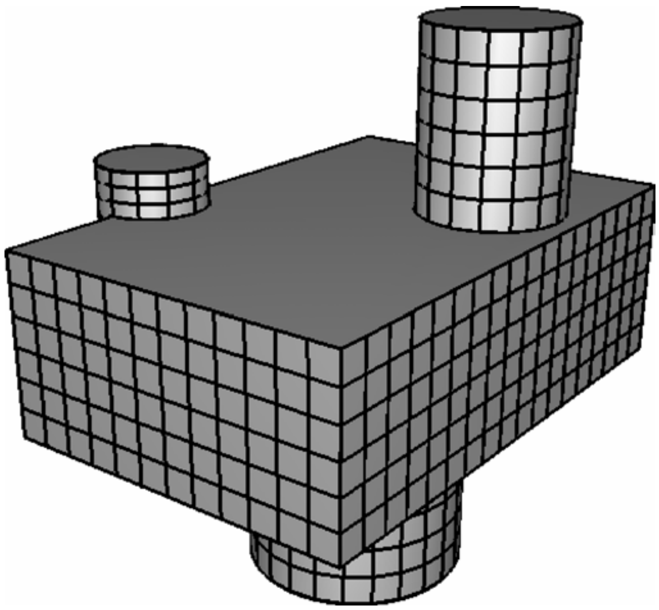


Fig. 5 Algorithm: building vertical nodal connectivity and determining layer information

Figure 3b also shows the vertical connections in the data structure for one column of linking nodes in $N(n)$. Each node stores pointers to the node above it (previous or source direction) and below it (next or target direction). The pointer to the *previous node* for each node is set to NULL if there is no node connected to it in the source direction, and similarly for the *next node* pointer if no node is connected in the target direction. When the target faces are projected to the source faces, new nodes will be generated to fill in the missing *previous* and *next* nodes.

2.4 Build control loops

Control loops are the continuous loops of nodes on the linking surfaces that bound each layer of the sweep. For multi-sweep problems, such as the one shown in Fig. 2, that have a single "barrel" to sweep along, this simply refers to all the exterior nodes on each layer. For more complicated examples, such as the one shown in Fig. 6, where parallel "barrels" exist in the sweep, there are separate groups of nodes, even on the same layer. An important function of the control loops is to define the transformation (described later) for a given section of

Fig. 6 Multi-sweep volume with parallel control loop groups

```
1.  LET F be the first target face, and assign it layer value  $L = 0$.
2.  ADD F to the stack of empty source, s and target, t faces $\lambda(s,t)$
3.  WHILE $\lambda(s,t)$ is not empty
4.     LET F be equal to first face in $\lambda(s,t)$ and remove from $\lambda(s,t)$
5.     LET $\Omega(n)$ be mesh nodes on the boundary of the faces connected to F
6.     LET $N(n)$ be the list of vertically connected nodes from $\Omega(n)$
7.     ASSIGN layer values and vertically connectivity to $N(n)$
8.     LET $\Phi(s,t)$ be the list of source and target faces
              reached at the end of $N(n)$
9.     ASSIGN layer values to $\Phi(s,t)$
10.    ADD $\Phi(s,t)$ to the stack $\lambda(s,t)$
11. CONTINUE
12. LET minL equal the mimium layer value in all the target faces t
13. ADJUST all nodes n, source faces s, and target faces t so that minL = 0
```

the volume. Each of the nodes of a control loop will contribute to the placement of interior nodes for each of the sweep layers. The control loops for the transformation of the first layer for the volume in Fig. 3a are shown in Fig. 3c.

The control loops are calculated by determining outermost boundary nodes of source and target faces $\Lambda(s)$ and $\Lambda(t)$, respectively, on each layer, L. The control loops on each layer, $C(L)$, are connected by traversing the vertical connectivity of the nodes in the linking mesh. The procedure for doing this is outlined in Fig. 7.

In step 5 of Fig. 7, the outermost boundary edges of a connected set of source or target faces are determined. Next, the nodes on these boundary edges are gathered and placed into new control loops for this layer and are added to Π $(C(L))$, which stores a list of the control loops on the current layer. In step 7, the control loops on the current layer, $\Phi(C(L))$ are used to build matching control loops on the next layer, $\Phi(C(L+1))$ through the stored vertical nodal connectivity. Once finished, the control loops for each layer are recorded in the data structure and are ready to be accessed when the target faces are projected upwards onto the source faces.

2.5 Build loop face data structures

An important aspect of the CCSweep algorithm is the data structures used to represent the source and target faces during the cutting process. Minimally, the data structures represent the source and target faces as ordered lists or loops of nodes. When the linking faces are meshed, the boundary edges of the source and target faces are also meshed. The mesh nodes on the boundaries of a source and target face are taken in order to form a loop of nodes to represent the face. Several loops are formed if there are interior holes in the face. A more complete data set is used in CCSweep to represent the faces as they are moved through the volume.

In addition, CCSweep requires that the nodes from the target faces be first projected and then their loop topology inscribed onto the source faces. We propose a

Fig. 7 Algorithm: build control loops

```
1.  FOR each sweep layer L
2.     LET $\Phi(C(L))$ be the known list of control loops on L
3.     LET $\Lambda(s)$ and $\Lambda(t)$ be lists of the source and target faces on L
4.     IF $\Lambda(s)$ OR $\Lambda(t)$ is not empty, THEN
5.        LET $\Pi(C(L))$ be the list of control loops bounding
              the connected face groups in $\Lambda(s)$ and $\Lambda(t)$
6.        ADD $\Pi(C(L))$ to $\Phi(C(L))$
7:     LET $\Phi(C(L+1))$ be equal to the loops vertically connected to $\Phi(C(L))$
```

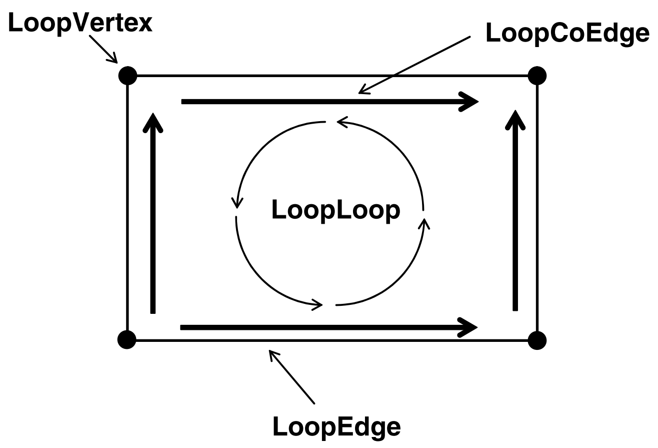
simple topology representation that will facilitate and help keep track of this projection and imprinting procedure. During the imprinting process, additional information is needed to maintain the orientation of the loops and provide an efficient method to access data. A bidirectional graph of new topology entities is chosen to describe the topology of the source and target faces. For example, given the square face in Fig. 8a, the face is represented by a LoopFace, the edges by LoopEdges, the vertices by LoopVertices, and the orientation of the face is represented by both the LoopLoop and LoopCoEdge data structures. The connectivity of the data structure for this example is shown in Fig. 8b. The mesh nodes are stored on the LoopEdges and LoopVertex data structures. This configuration allows adjacent source or target faces to share common LoopEdge data structures and, therefore, the mesh nodes. A LoopCoEdge serves to orient the LoopEdge with respect to the current face, while a LoopEdge may be shared by more than one face.

Imprinting is a combination of subtraction and intersection Boolean operations to split up overlapping faces into new faces that either overlap or do not overlap. Boolean operations assume that the faces are oriented in the same direction. In a volume, source and target faces are naturally oriented opposite each other. To correct this, when the loop face data structure is created, the orientation of the source faces is reversed to match that of the loop faces created from the target faces. This reversal only affects the loop face data structure, and has no bearing on the real source or target faces.

2.6 Project loop faces

In the CCSweep algorithm, projection of the loop faces does not occur until after gathering and imprinting the source and target loop faces. Imprinting, however, will not occur until at least the second layer of the process,

(a)



(b)

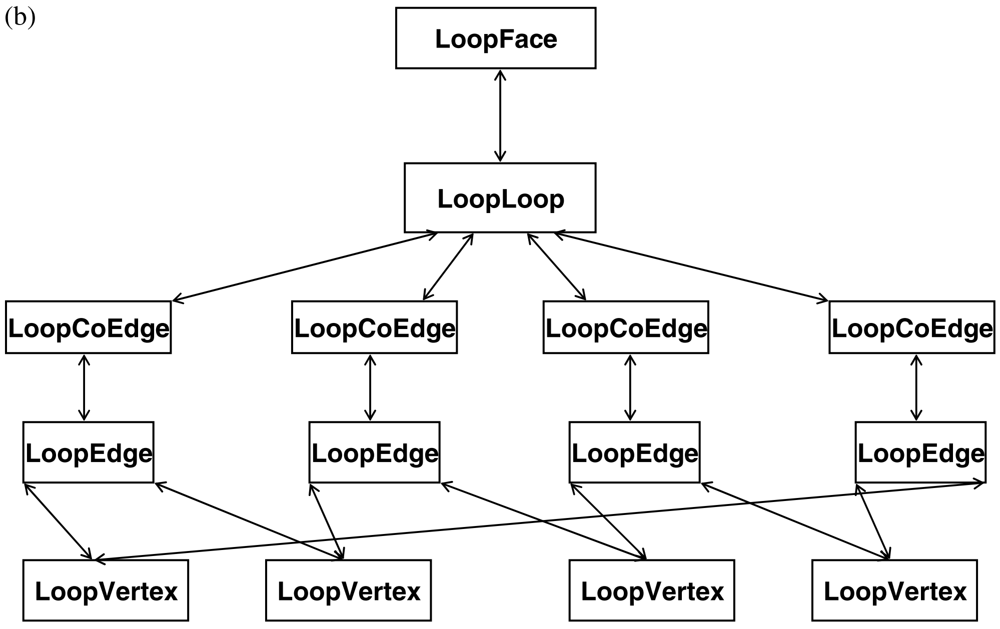


Fig. 8a–c LoopFace data structure

since source and target faces cannot both exist on the first layer. To move the target loop faces to the previous layers, nodes must be projected from the nodes that have NULL *previous node* pointers. This process is called the projection process. CCSweep uses the affine transformation described by Knupp [1] with additional refinement of the transformation provided by the weighted residual errors of several of the closest nodes, as described in the following.

Given the affine transformation matrix, $[T]$, between control loops on two adjacent layers, and nodal positions n_j on the current control loops, and the nodal positions $\breve{n}_j$ in the control loop on the previous layer, the residual error R_j is given as:

$$R_j = \breve{n}_j - [T]n_j \tag{1}$$

For example, Fig. 9a shows a set of four nodal positions, n_1, n_2, n_3, and n_4, in the current control loop and the corresponding nodal positions that they are vertically connected to; $\breve{n}_1, \breve{n}_2, \breve{n}_3$, and $\breve{n}_4$. Figure 9b illustrates the projection of the nodes on the current control loop to the previous layer through the transformation matrix $[T]$. The residual vectors, R_1, R_2, R_3, and R_4, show the error in the transformation matrix.

Projecting nodal positions, s_i, onto the new layer yields nodal positions $[T]s_i$ with an unknown residual error, R_i. R_i can be approximated by:

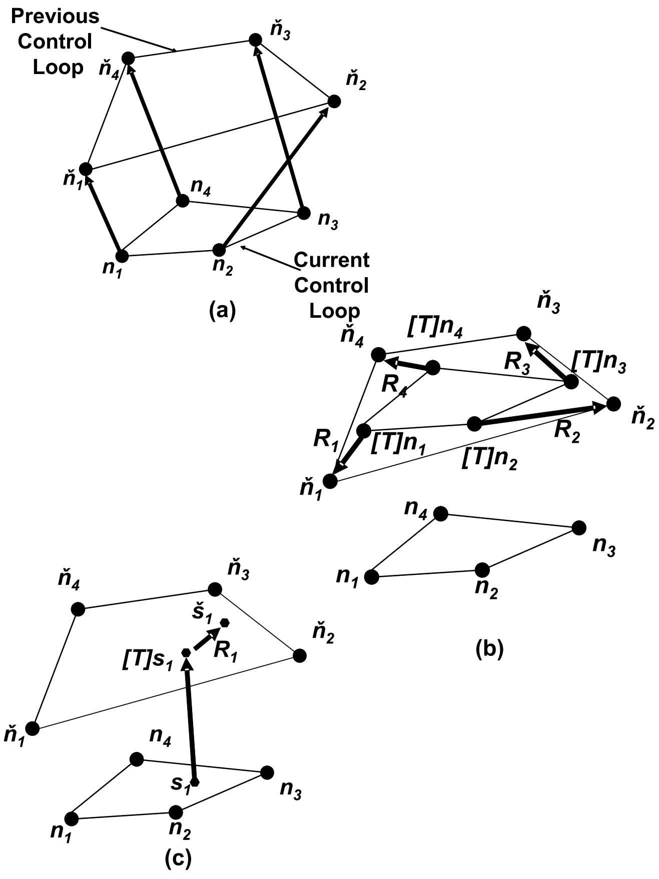


Fig. 9 Weighted residual correction

$$R_i = \sum_{k=1}^{j} w_k R_k \tag{2}$$

where w_k is an inverse distance weighted interpolant, described by:

$$w_k = \frac{d_k^{-2}}{\sum_{n=1}^{j} d_n^{-2}} \tag{3}$$

and

$$d_n = |[T]s_i - [T]n_n| \tag{4}$$

where the operator $|\cdot|$ indicates the magnitude of the computed vector, and $\hat{s}_i$, the corrected projection, is computed as:

$$\hat{s}_i = [T]s_i + R_i \tag{5}$$

For the previous example in Fig. 9a, the projection of new point, s_i, to $\hat{s}_i$ is illustrated in Fig. 9c.

The current implementation uses the ten closest nodes in $\breve{n}_j$ to the position $[T]s_i$ to compute the approximate residual error. This is sufficient for accuracy while limiting computational expense. An R-Tree [19] data structure combined with kth nearest neighbor searches [20] is employed to determine efficiently the ten closest nodes. Nodal projections with a combined affine transformation and weighted residual improvement are an important part of the success of the CCSweep algorithm. Experience has shown that, if the projections are not successful, the imprinting process will fail, resulting in termination of the algorithm. The method described in this section has been shown by experience to work well, however, there is no guarantee of success nor is there a known algorithm with such properties.

2.7 Imprint and merge source and target faces

Prior to imprinting source and target faces, adjacent target faces are first imprinted and the coincident edges are consolidated or merged in order to allow traversal of the loop faces through common LoopEdges and LoopVertices. When LoopFaces are projected upwards, new loop topology is created on the next layer. Once all the LoopFaces have been projected, coincident LoopEdges are merged. Coincident LoopEdges are those that are defined by the same set of nodes, and with the same start and end LoopVertices. When two LoopEdges are merged, the LoopCoEdges that reference these edges are updated to point to the merged LoopEdge, and to reflect the possible new orientation of the merged LoopEdge.

Often during projection, new LoopFaces are created adjacent to existing LoopFaces that were formed from target faces reached on the new layer. In these situations, it is possible for adjacent target LoopFaces to have non-matching LoopEdges due to LoopVertices coming from the current layer or the layers below. For example, in

Fig. 10a, a multi-sweep geometry is shown with three target LoopFaces. In Fig. 10b, LoopFace 1 is projected two layers through the volume until it reaches the layer where LoopFace 2 is. While LoopFaces 1 and 2 will share common mesh nodes at a boundary, their loop data structure is not conformal. In these cases, the LoopEdges must be split to match adjacent edges. Since adjacent target faces must share common nodes, this is easily done by matching up nodes and splitting LoopEdges where an interior node also belongs to a LoopVertex. After splitting unmatched LoopEdges, coincident LoopEdges are merged. In Fig. 10c, LoopEdge (LE) 2 on LoopFace 1 is split with LoopVertex (LV) 6 from LoopFace 2 into two new LoopEdges, LE 10 and LE 11, respectively. LE 9 is then merged together with LE 10 which also causes LV 3 and LV 9 to be merged. The result is that LoopFaces 1 and 2 now share a common LoopEdge in their respective topology graphs.

While there are shared nodes between adjacent or overlapping target faces, source and target faces are not guaranteed to share nodes, LoopEdges, or LoopVertices. Intersections of the faces must, therefore, be computed in order to properly determine the imprint. The imprinting process may introduce new boundary nodes because of the discrete intersection process it uses during the imprinting process. After imprinting, any new nodes created are propagated vertically up or down the volume, depending on the situation. An overview of the imprinting process is given in Fig. 11. In the algorithm, the list of target loop faces, $\Lambda(t)$, on the current layer, is imprinted with the source loop faces, $\Lambda(s)$, on the same layer. The results of the imprint are stored in three lists, $\Theta(o)$, $\Theta(s)$, and $\Theta(t)$, respectively. $\Theta(o)$ contains the list of faces that came from sections of the two faces that overlapped each other. $\Theta(s)$ and $\Theta(t)$ contain the lists of faces that were exterior to any overlap and were on the original source or target face, respectively. An example of the three results is shown in Fig. 4b, when half-circle-shaped source and target faces are imprinted. The result of the imprint is that $\Theta(o)$, $\Theta(s)$, and $\Theta(t)$ each contain one new loop face. After the imprinting, the loop faces in $\Theta(t)$ and $\Theta(s)$ are added to $\Lambda(t)$ and $\Lambda(s)$, respectively, for further imprinting until all the faces on the layer are either common or do not intersect.

The actual imprinting of the LoopFaces occurs in step 4 of Fig. 11. The imprinting process proceeds by collecting the nodes on the boundaries of the two LoopFaces into lists. Line segments from one LoopFace defined by the connectivity of the boundary node lists are then intersected with the line segments representing the other LoopFace. The places of intersection are stored in the data structure, and the loop face data structure is updated to reflect intersecting edges and coincident edges and vertices. Both the faces are traversed, making counter-clock-wise turns at vertices, to create new LoopLoops and LoopFaces. An outline for the imprinting and merging operation used in step 4 of Fig. 11 proceeds as follows:

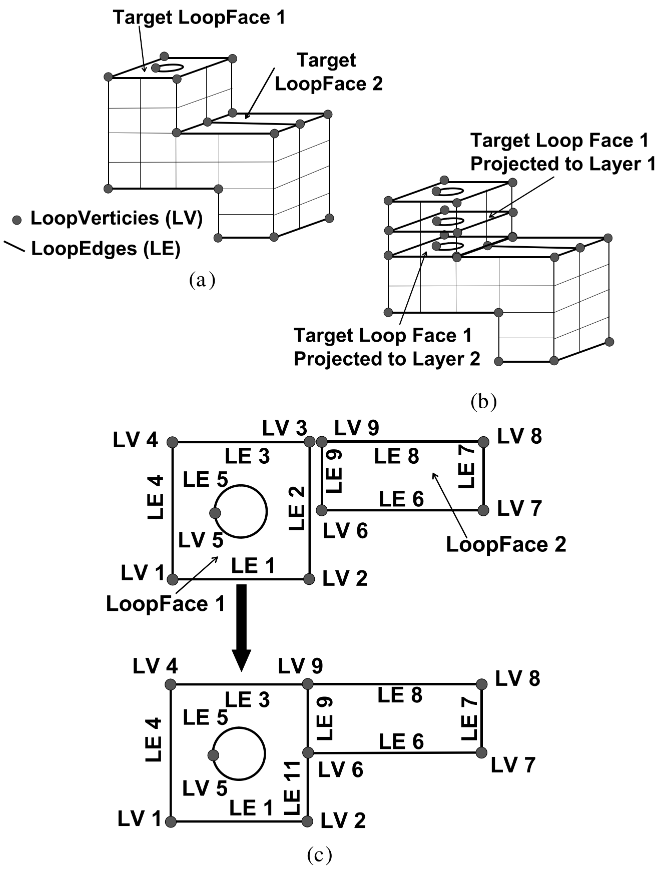


Fig. 10 Imprinting target LoopFaces

1. *Get boundary nodes from source and target LoopFaces*: form loops of sequential nodes representing the boundaries of the source LoopFace and target LoopFace.
2. *Intersect boundaries*: calculate intersections of the line segments formed by the sequential node loops. Insert new nodes when the loops intersect on a line segment. Merge boundary nodes when the intersection occurs exactly on a node. Store the intersection locations on the nodes.
3. *Split LoopEdges*: based on the calculated intersections, split the LoopEdges that own nodes where intersections occurred. Create LoopVertices at the locations of intersection.
4. *Merge coincident LoopVertices*: merge LoopVertices on the source and target LoopFaces that are now coincident as a result from the LoopEdge splitting. Merged vertices will now be referenced by both the source LoopFace and the target LoopFace.
5. *Merge coincident LoopEdges*: merge LoopEdges on the source and target LoopFaces that shared common start and end vertices. During the merging of two LoopEdges, the connected LoopCoEdges are also modified to point to the remaining LoopEdge and their sense data is updated to reflect a possible new LoopEdge orientation. After merging two coin-

Fig. 11 Algorithm: imprint and merge source and target faces

```
1.  WHILE $\lambda(t)$, the stack of target loop faces being imprinted, is not EMPTY:
2.      LET $T$ be the first item in $\lambda(t)$ and remove from $\lambda(t)$
3.      FOR EACH source loop face $S$ in the list of source faces, $\lambda(s)$
4.          LET $\Theta(o)$, $\Theta(s)$, and $\Theta(t)$ be the result $S\Delta T$, where $\Delta$ is the imprint
                operator, $\Theta(o)$ are the loop faces that originate from overlaps in
                $S$ and $T$, $\Theta(s)$ are the loop faces that only originate from $S$, and
                $\Theta(t)$ are the loop faces that only originate from $T$
5.          IF $\Theta(s)$ is not EMPTY, THEN
6.            REMOVE $S$ from $\lambda(s)$
7.            ADD $\Theta(s)$ to $\lambda(s)$
8.          IF $\Theta(t)$ is not EMPTY, THEN
7.            ADD $\Theta(t)$ to $\lambda(t)$
8.            BREAK FOR EACH
9.          ELSE THEN ADD $T$ to $\Pi(t)$, where $\Pi(t)$ is the list of target faces that
10.                          continue to be projected after imprinting
11. CONTINUE
```

cident LoopEdges, both source and target LoopFaces reference the same LoopEdge.

6. *Create new LoopLoops*: the topology graph of each LoopFace is traversed to create new LoopLoop data structures. The new loops are created by making "left" turns at the vertices in the graph.
7. *Create new LoopFaces*: from the new LoopLoops, new LoopFaces are created. As the faces are created, the faces are classified into one of three categories. The first category is the faces that were interior to the intersection, or where the faces overlapped. The second category is the faces that were exterior to the intersection and came from the source face. The final category of faces are the ones that were exterior to the intersection and came from the target face.

In step 2, *intersect boundaries*, the intersection is produced using the procedure outlined for the boundary imprinting algorithm described by White and Saigal [21]. For this application, LoopFaces represent the faces rather than CAD faces. Virtual topology operations are not used, but similar topology operations for LoopFaces are performed in steps 3–7 of the imprinting and merging outline. Unlike boundary imprinting, in steps 3–7, the boundary edges and vertices are merged in addition to being made coincident. This merging allows the CCSweep to traverse the topology graph to get from one face to an adjacent one. Topology traversals are crucial to the method to build the new LoopLoop data structures after the intersection process.

In step 6 of the imprint and merge outline, the original LoopFaces are traversed to determine any new loops of LoopCoEdges resulting from the intersection process. The traversal starts by traversing each LoopLoop in a LoopFace in the direction of the first LoopCoEdge in that LoopLoop. The LoopEdges are traversed by making "left turns" at each LoopVertex junction. Making left turns is done by starting with a current LoopEdge, getting the set of LoopEdges connected to it in the traversal direction, and choosing the LoopEdge from this set that makes the smallest interior angle between it and the current LoopEdge. The smallest interior angle is relative to the orientation of the LoopFace. When intersections between the LoopFaces occur and when the left turns are made, LoopEdges of the other LoopFaces are also traversed because of the connected topology graph discussed earlier. As the edges are visited, they are marked to avoid visiting the same edge twice while traversing the same face.

In step 7 of the *imprint and merge* outline, the new LoopFaces are created. The process is simple for loops where there are only single exterior loops. But, when multiple interior and exterior loops are present, it becomes necessary to determine the exterior loop in which the interior loops belong. In existing multi-sweep approaches, the winding number algorithm [15] is used to determine if an interior loop is internal or external to a specific exterior loop. This algorithm breaks down quickly when the loops are not planar. CCSweep overcomes this difficulty by introducing a triangulation of the exterior control loops. This is accomplished by first projecting the nodes on the loops to a best-fit plane [22]. The boundary nodes on the plane are then triangulated using a Boyer–Watson Delaunay method [23, 24], followed by a boundary edge recovery procedure [25]. The boundary nodes are next projected back to their original coordinates, bringing the connected triangles with them. A point on the interior of a loop can now be tested by projecting it onto the triangulation to determine if the point lies inside or outside the exterior loop [26]. The introduction of the proposed discrete method of determining the sided-ness of a node, with respect to a non-planar loop, greatly increases the robustness of the multi-sweep process and overcomes previous difficulties introduced by the winding number procedure. With the discovery of these external to internal relationships, the LoopFaces resulting from the imprinting process are constructed.

After the imprinting process on a layer is complete, any new nodes that are created during this process must be copied vertically to later enable creation of the triangulated decomposition faces. During the imprinting process, new nodes can be introduced in three ways: the source LoopFace, the target LoopFace, and/or both the source and target LoopFaces. If a node on the boundary of the target LoopFace intersects the boundary of the source LoopFace, but is not within geometric tolerance of another node, the node becomes "new" to the boundary of the source LoopFace during the intersection process, and, similarly, for a node on the source LoopFace. If the two faces intersect on their boundaries and no nodes from either boundary are within tolerance of the intersection, then a new node is created at the point of intersection and becomes "new" to both boundaries.

New nodes on the boundaries of the LoopFaces from the target faces are then propagated back upwards through the volume until the boundary of the original target face is reached. As the nodes are propagated upwards, the *next* and *previous* data structures are updated so that, after completion, the new nodes are similar to other nodes projected from the target faces. New nodes on the boundaries of source LoopFaces are propagated downwards toward other source faces, if any exist. Nodes that are new to both boundaries are propagated in both directions. The example shown in Fig. 2b demonstrates that, during the imprinting process of the LoopFaces shown in Fig. 2a, new nodes may need to be added to both the source and target LoopFaces to enable them to exactly match. Figure 12 illustrates the vertical node propagation of the new nodes.

Allowing new nodes to be added to the source and target boundaries during the imprint operation is one of the fundamental advantages of CCSweep over the existing approaches. This enables boundaries on the source and target faces to be matched through the intersection process, without having to previously ensure that the edge interval sizes on the source and target faces match. The matching edges can only be determined by projecting the target faces onto the source faces and then intersecting them. Through CCSweep, the edge interval sizes do not need to be matched prior to the algorithm.

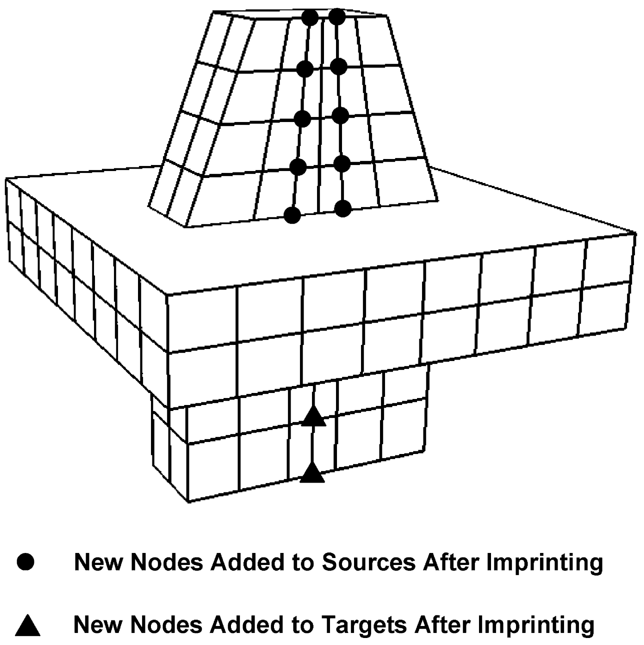

● New Nodes Added to Sources After Imprinting
▲ New Nodes Added to Targets After Imprinting

Fig. 12 Propagation of new nodes added to source and target LoopFaces after imprinting

2.8 Build decomposition triangulations and volume decomposition

The decomposition faces are constructed after projecting and imprinting the target and source faces. As the target LoopFaces are projected through the volume, nodes on the interior of the volume are created. These nodes are gathered into structured columns. The columns begin at LoopEdges on source faces that were projected from the target faces and terminate at the original target face boundaries. The triangulations are formed by following adjacent nodes vertically through the volume, creating two triangles between each layer. The vertical nodes and the triangulations that follow from a single LoopEdge are considered as a decomposition set. The triangles in the decomposition set are used to represent a cutting face during the volume decomposition process. The decomposition sets are then passed to the virtual geometry engine [16] to create facet-based surfaces and to decompose the volume.

As the decomposition sets are inserted into the volume, they may or may not completely decompose the volume. For instance, as in Fig. 3a, there are two decomposition sets shown in Fig. 4c. When the first set is inserted into the volume, the virtual geometry engine simply adds the surface as an interior non-manifold surface. When the second set is inserted, a complete dissection of the volume occurs and the volume is split into two volumes, as shown in Fig. 4d. As soon as one set successfully decomposes the volume into two separate volumes, the remaining sets are geometrically sorted into the proper new volume. The geometric sorting is done by finding the closest distance to the two volumes. The volume that is closest is matched up with the decomposition set, and the process continues until all the remaining sets are transferred to either of the two new volumes.

After all of the decomposition sets have been inserted, the original volume can be decomposed, resulting in multiple volumes. Each of these new volumes can now be meshed automatically with any sweeping approach that handles many-to-one volumes. Additionally, the linking meshes and interior nodes are removed. This is done because the source and target faces may no longer have boundaries with even numbers of nodes. By removing the mesh, and reassigning intervals to ensure an even number of nodes on the boundaries, the linking surfaces can be remeshed at the mesh density desired by the user.

3 Results

CCSweep was implemented in C++ within the CUBIT [27] framework to take advantage of the required support algorithms of auto-scheme [18], submapping [12], mapping [11], paving [13], many-to-one sweeping [1], and virtual topology operations [16]. Several examples are given to demonstrate various aspects of the CCSweep algorithm. These examples cannot be automatically completed using the published algorithms [3–5].

3.1 Tapered protrusion example

The geometry of the tapered protrusion example is shown in Fig. 13a. This example was also shown previously in Fig. 2 to demonstrate problems with other multi-sweep approaches while only using node matching in the imprinting process. CCSweep successfully decomposes the model into two separate parts. The algorithm generates four decomposition sets to accomplish the decomposition. The result of decomposition performed by CCSweep is shown in Fig. 13b. CCSweep is able to correctly match the interior loops from the source and target faces without manually setting the intervals along the matching edges. After automatic decomposition by CCSweep, the part can be meshed. The resulting mesh, after choosing an appropriate mesh size and again applying auto-scheme selection to determine the sweeping directions of the two new volumes, is shown in Fig. 13c. This example demonstrates that CCSweep is able to match edges through the volume without first matching the interval sizes on the edges.

3.2 Piston section example

The piston section shown in Fig. 14a is a portion of a CAD model used to model an engine piston. This example demonstrates CCSweep on a volume that does not contain strictly planar sweeping layers. Additionally, because of the cylindrical protrusion, the proposed discrete inside/outside testing is utilized to determine the matching of internal and external loops during the imprinting process. CCSweep decomposes the original volume by creating four decomposition sets to cut the volume into three many-to-one meshable volumes, shown in Fig. 14b. Following the decomposition, the volume is meshed with a mesh size equal to twice the size of the smallest feature in the model. The resulting mesh is shown in Fig. 14c.

3.3 Bulkhead example

The bulkhead CAD geometry is shown in Fig. 15a. This example demonstrates CCSweep on parts with small features and interior through-holes. As identified in Fig. 15a, the small features are ledges which create small edges with lengths as small as 0.008 units, where, for comparison, one of the side edges is of length 2.30 units. In previous multi-sweep approaches, even if the interior

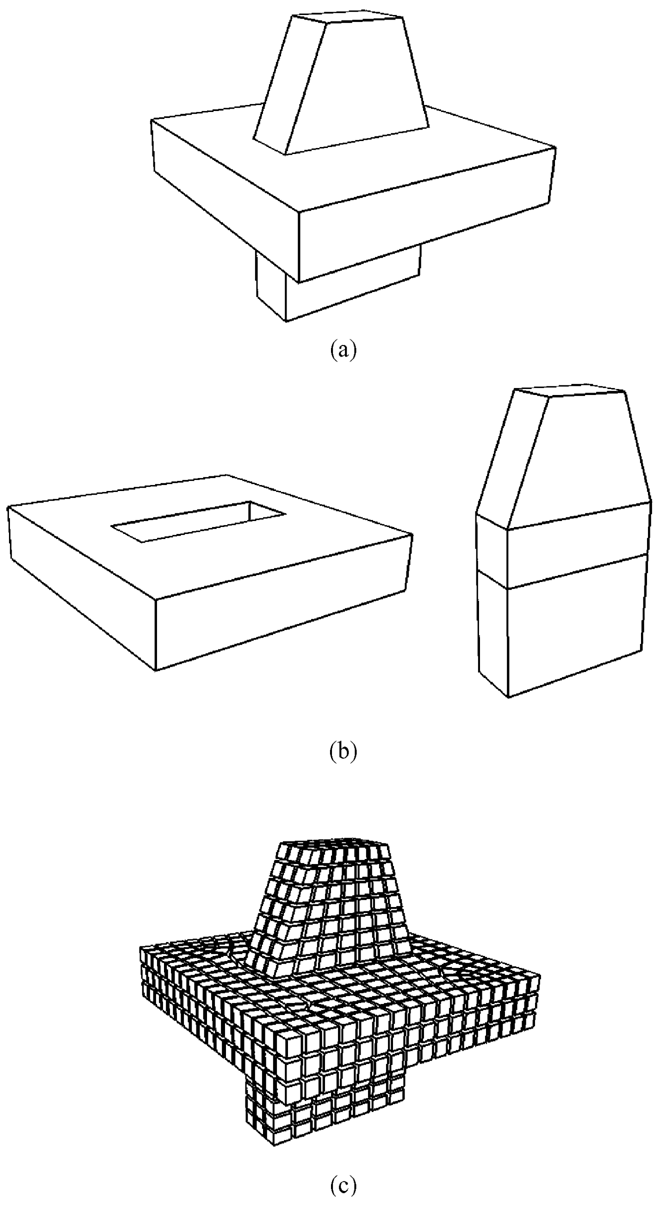


Fig. 13 **a** Tapered protrusion example. **b** Decomposition. **c** Mesh

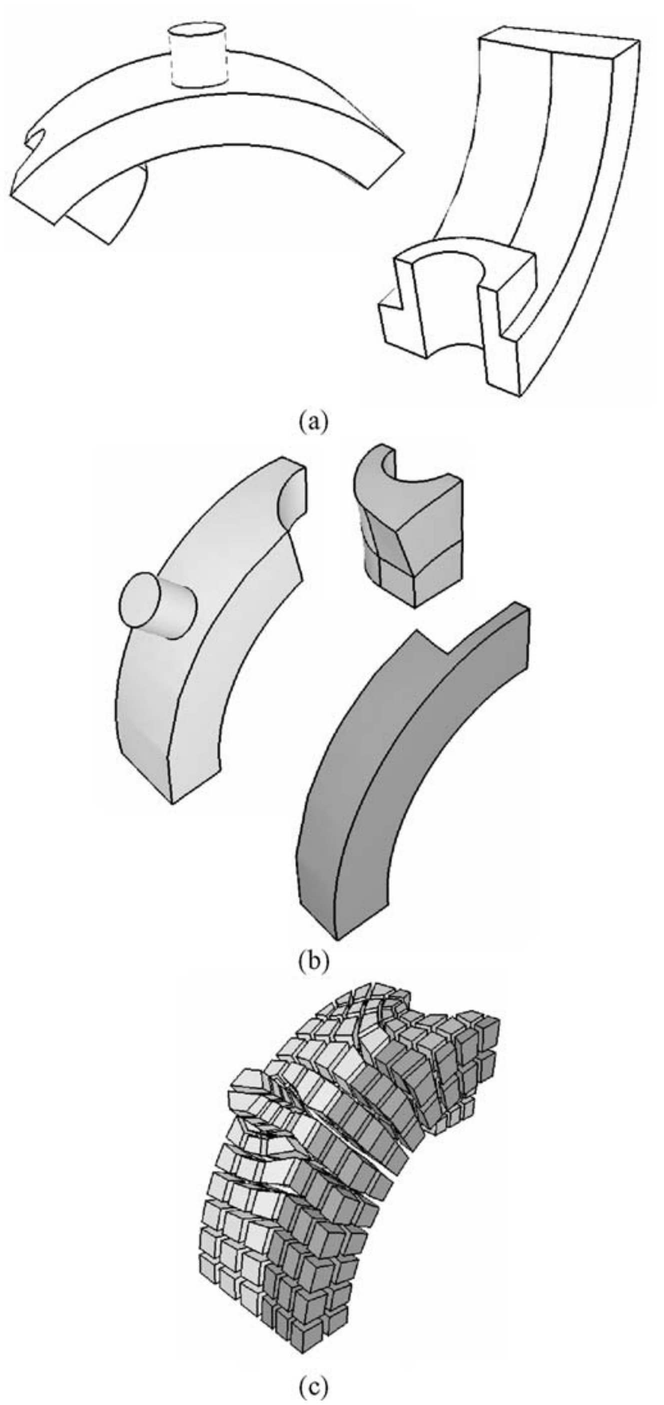


Fig. 14 **a** Different views of piston section geometry. **b** Decomposition. **c** Mesh

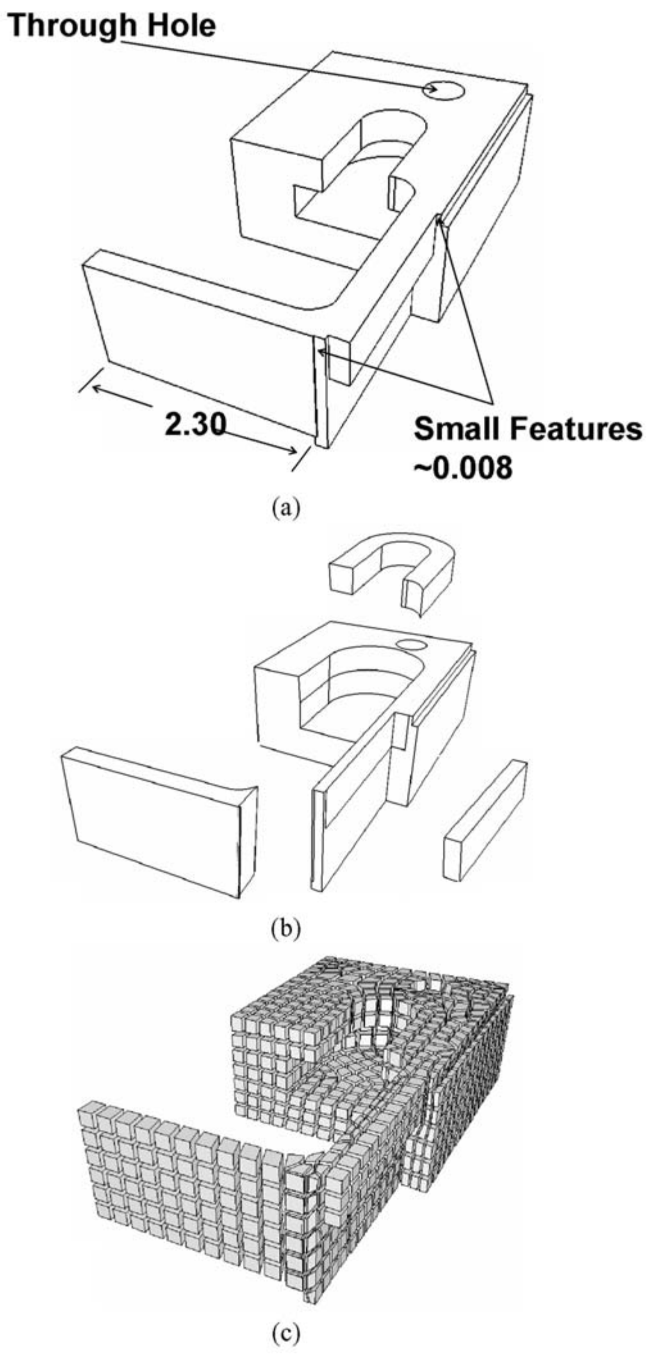


Fig. 15 **a** Bulkhead model with small features. **b** Decomposition. **c** Mesh

loop classifications are successful, a smaller mesh size would be required to successfully generate a mesh because of the small features. CCSweep must also use a small discretization size but, after CCSweep decomposes the bulkhead into the four separate volumes shown in Fig. 15b, a much coarser element size can be used to mesh the resulting volumes. CCSweep creates 6 cutting faces, or decomposition sets, to decompose the model. The resulting mesh of the model, with an element size of 0.15 units, is shown in Fig. 15c.

3.4 Brake example

The final example is a CAD model of a part used in a braking system. The model is shown in Fig. 16a. This example demonstrates CCSweep on a model that has

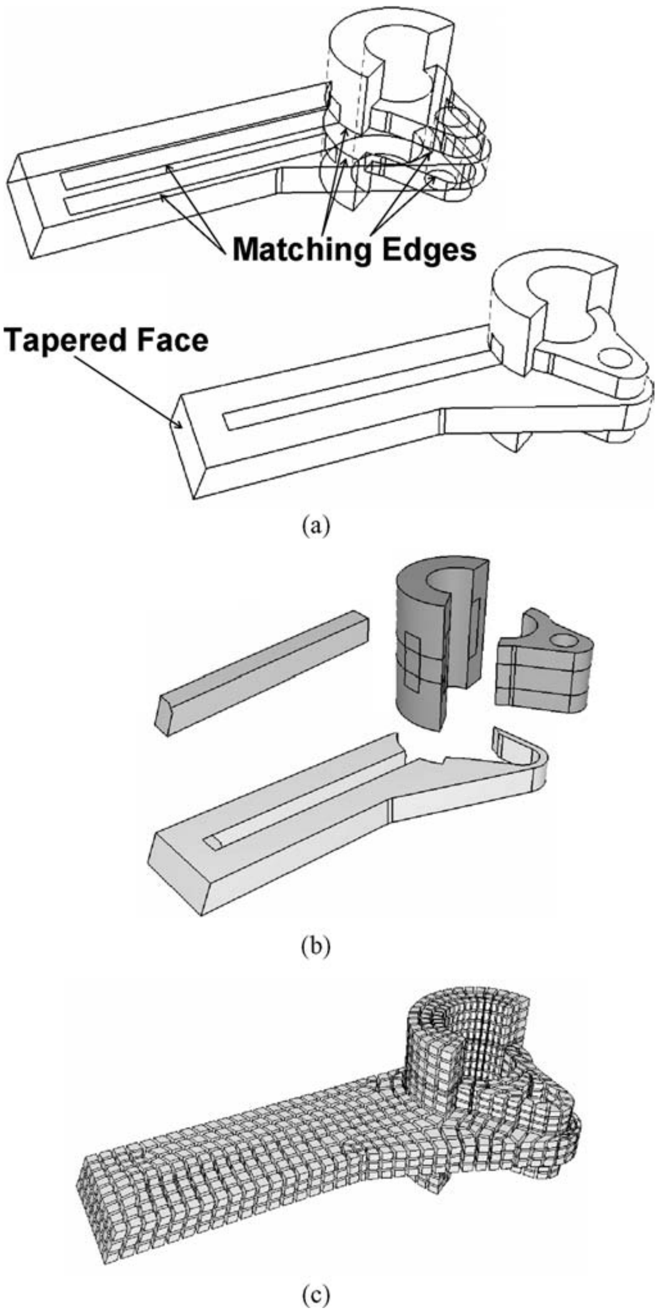


Fig. 16 a Wire frame and hidden line views of brake geometry. **b** Decomposition. **c** Mesh

non-planar projections combined with edges that match across the volume. In other approaches, several of the element sizes on edges in the model would have to be manually matched. Additionally, because of the tapered face on the left of Fig. 16a, the projections will not properly align the edges. For other approaches, this would further complicate the node matching process. CCSweep is able to automatically decompose the model into four separate volumes using sixteen decomposition sets, as illustrated in Fig. 16b. Each of the volumes can then be automatically meshed using many-to-one sweeping. The resulting mesh is shown in Fig. 16c.

4 Conclusions

CCSweep is a new algorithm for automatically decomposing multi-sweep volumes into volumes that can be readily meshed with traditional hexahedral sweeping techniques. CCSweep achieves this by discretizing the linking faces of the volumes with structured quadrilateral elements. The structured discretization provides a layering system to traverse the volume vertically. The target faces are approximated using the loop face data structure consisting of the ordered boundary nodes. The approximated target faces are then pushed through the volume. This is accomplished by following the exterior layering and projecting new interior nodes. The nodal projections are calculated through affine transformations corrected locally by approximate residual errors. Once projected onto source faces, the approximated source and target faces are imprinted and merged, enabling source faces to topologically match target faces. Then, the interior nodes created in the projection process and the results from the imprinting are used to create interior triangulations representing cutting surfaces to decompose the volume. These triangulations are passed to a virtual topology operator to decompose the volumes along these triangulations. The volumes are decomposed into separate volumes which are individually meshable with existing many-to-one hexahedral sweeping algorithms. CCSweep successfully enables automatic mesh generation of multi-sweep geometries.

CCSweep overcomes the main problems of existing approaches to meshing multi-sweep volumes. Specifically, CCSweep uses discrete boundary intersections to perform imprint operations of projected target faces with source faces, enabling the algorithm to handle mismatched edge sizes. The discretization size CCSweep uses to imprint source and target faces is decoupled from the mesh size, which enables more accurate and stable imprinting. This decoupling is facilitated by the decomposition of the multi-sweep volume, which allows users to mesh and remesh new volumes at various mesh densities, without having to repeat projecting and imprinting operations. CCSweep contains a stable method to handle inside–outside testing for handling source or target faces that have interior holes. Multiple examples demonstrate that the CCSweep algorithm is both effective and robust.

The implementation of the CCSweep algorithm demonstrates its ability to automatically decompose multi-sweep volumes into many-to-one volumes. The resulting volumes are easily meshed automatically. CCSweep decouples the discretization size it uses to decompose the volumes and the size used to achieve the final mesh. This decoupling enables greater flexibility for the user when generating large assemblies of meshes and performing convergence studies. Importantly, the decoupling allows greater flexibility in resolving the intersections of the source and target faces, improving the robustness of the overall algorithm.

References

1. Knupp P (1998) Next-generation sweep tool: a method for generating all-hex meshes on two-and-one-half dimensional geometries. In: Proceedings of the 7th international meshing roundtable, Dearborn, Michigan, October 1998, pp 505–513
2. Lai M, Benzley S, Sjaardema G, Tautges T (1996) A multiple source and target sweeping method for generating all-hexahedral finite element meshes. In: Proceedings of the 5th international meshing roundtable, Pittsburgh, Pennsylvania, October 1996, pp 217–228
3. Blacker T (1996) The Cooper tool. In: Proceedings of the 5th international meshing roundtable, Pittsburgh, Pennsylvania, October 1996, pp 13–30
4. Lai M, Benzley S, White D (2000) Automated hexahedral mesh generation by generalized multiple source to multiple target sweeping. Int J Num Methods Eng 49:261–275
5. Shepherd J, Mitchell S, Knupp P, White D (2000) Methods for multisweep automation. In: Proceedings of the 9th international meshing roundtable, New Orleans, Los Angeles, October 2000, pp 77–87
6. Schneiders R, Schindler R, Weiler F (1996) Octree-based Generation of hexahedral element meshes. In: Proceedings of the 5th international roundtable, Pittsburgh, Pennsylvania, October 1996, pp 205–216
7. Tautges T, Blacker T, Mitchell S (1996) The Whisker weaving algorithm: a connectivity-based method for constructing all-hexahedral finite element meshes. Int J Num Methods Eng 39:3327–3349
8. Blacker T, Meyers R (1993) Seams and wedges in plastering: a 3D hexahedral mesh generation algorithm. Eng Comput 2: 83–93
9. Ymakawa S, Shimada K (2001) Hexhoop: modular templates for converting a hex-dominant mesh to an all-hex mesh. In: Proceedings of the 10th international meshing roundtable, Newport Beach, California, October 2001, pp 235–246
10. Staten M, Canann S, Owen S (1998) BMSWEEP: locating interior nodes during sweeping. In: Proceedings of the 7th international meshing roundtable, Dearborn, Michigan, October 1998, pp 7–18
11. Cook W, Oaks W (1983) Mapping methods for generating three-dimensional meshing. Comput Mech Eng 1:67–72
12. Whiteley M, White D, Benzley S, Blacker T (1996) Two and three-quarter dimensional meshing facilitators. Eng Comput 12:155–167
13. Blacker T (1991) Paving: a new approach to automated quadrilateral mesh generation. Int J Num Methods Eng 32:811–847
14. Owen S, Staten M, Canann S, Saigal S (1999) Q-morph: an indirect approach to advancing front quad meshing. Int J Num Methods Eng 9:1317–1340
15. O'Rourke J (1998) Computational geometry in C. Cambridge University Press, Cambridge
16. Kraftcheck J (2000) Virtual geometry: a mechanism for modification of cad model topology for improved meshability. PhD thesis, University of Wisconsin, Madison
17. Sheffer A, Blacker T, Clements J, Bercovier M (1997) Virtual topology operators for meshing. In: Proceedings of the 6th international meshing roundtable, Park City, Utah, October 1997, pp 49–66
18. White D, Tautges T (2000) Automatic scheme selection for Toolkit hex meshing. Int J Methods Eng 49:127–144
19. Guttman A (1984) R-Trees: a dynamic index structure for spatial searching. In: Proceedings of the ACM international conference on management of data (SIGMOD'84), Boston, Massachusetts, June 1984, pp 47–57
20. Roussopoulos N, Kelley S, Frederic V (1995) Nearest neighbor queries. In: Proceedings of the ACM international conference on management of data (SIGMOD'95), San Jose, California, May 1995, pp 71–79
21. White D, Saigal S (2002) Improved imprint and merge for conformal meshing. In: Proceedings of the 11th international meshing roundtable, Ithaca, New York, September 2002, pp 285–296
22. Pratt V (1987) Direct least-squares fitting of algebraic surfaces. Comput Graphics 21:145–152
23. Bowyer A (1981) Computing Dirichlet tessellations. Comput J 24(2):162–166
24. Watson D (1981) Computing the Delaunay tesselation with application to Voronoi polytopes. Comput J 24:167–172
25. George P, Hecht F, Saltel E (1991) Automatic mesh generator with specified boundary. Comput Methods Appl Mech Eng 92:269–288
26. Owen S, White D (2001) Mesh-based geometry: a systematic approach to constructing geometry from a finite element mesh. In: Proceedings of the 10th international meshing roundtable, Newport Beach, California, October 2001, pp 83–96
27. Shepherd J (11/2003) CUBIT home page. Available at http://cubit.sandia.gov.