SAND2000-2273C

METHODS FOR MULTISWEEP AUTOMATION

Jason Shepherd[1], Scott A. Mitchell[1],
Patrick Knupp[1], and David White[1]
Sandia National Laboratories, Albuquerque, NM, U.S.A.
jfsheph@sandia.gov, samitch@sandia.gov,
pknupp@sandia.gov, and drwhite@sandia.gov

ABSTRACT

Sweeping has become the workhorse algorithm for creating conforming hexahedral meshes of complex models. This paper describes progress on the automatic, robust generation of MultiSwept meshes in CUBIT. MultiSweeping extends the class of volumes that may be swept to include those with multiple source and multiple target surfaces. While not yet perfect, CUBIT's MultiSweeping has recently become more reliable, and been extended to assemblies of volumes. Sweep Forging automates the process of making a volume (multi) sweepable: Sweep Verification takes the given source and target surfaces, and automatically classifies curve and vertex types so that sweep layers are well formed and progress from sources to targets.

Keywords: hexahedral mesh generation, sweeping, and automation.

1 INTRODUCTION

Sweeping has become the workhorse algorithm for creating conforming hexahedral meshes of complex models for Finite Element Analysis. Incarnations of sweeping include CooperTool,[1] BMSweep,[2] and CUBIT's Sweep[3] [11] and MultiSweep.[4] In this progress report, we address some of the outstanding limitations of CUBIT's Sweep and MultiSweep: we increase automation and extend the class of (multi) sweepable geometries. (CUBIT is a finite element mesh generation toolkit researched and developed at Sandia National Laboratories, focussed on reducing the (user) time to mesh, scalability, componentization.[5])

Many complex CAD assemblies must be manipulated in some way before they are meshable. CUBIT contains manual and feature-based geometric decomposition tools for subdividing and reshaping volumes so that they are sweepable. CUBIT also contains automatic volume scheme selection, which automatically sets surface schemes, then detects when volumes are subsequently sweepable. If a volume is not automatically sweepable, the user must either perform geometric operations, or set many parameters and force a sweep. The goal of geometric operations is to divide the volume into simpler or more blocky parts that are sweepable.

In some cases, the volume is not (many-to-one) sweepable because there are multiple source and multiple target surfaces. In this case, the volume can be meshed without further decomposition by MultiSweep; see also the CooperTool.[1]

Previous versions of CUBIT's MultiSweep [4] had robustness problems. As this paper explains, we have redesigned and re-implemented many of the fundamental MultiSweep operations, making them more reliable and general. MultiSweep remains built on the many to one CUBIT Sweeping algorithm, which has not changed except for the addition of logic to handle multiple targets. MultiSweeping is now more reliable, but still cannot mesh certain classes of geometry. As another improvement, we have taken steps to extend MultiSweeping to assemblies of volumes.

MultiSweep can dramatically reduce the time to mesh. Obviously, if the user knows this tool is available, he will initially have to perform fewer manual geometric decompositions. The time saved can be significant, as models with more volumes are more difficult to visualize, manipulate, and regroup for analysis boundary conditions.

[1] Patrick Knupp and Scott A. Mitchell were supported by the Mathematical, Information and Computational Sciences Division of the U.S. Department of Energy, Office of Energy Research. All of the authors work at Sandia National Laboratories, a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL8500.

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Also, if the user misses an initial decomposition during the geometry modification phase, adding that decomposition at the end of the meshing process can mean redoing much of the meshing process.

In some cases, a volume is not (multi) sweepable because it is not blocky enough. For example, the edges and corners of a cube may be rounded off in the CAD model. This shape detail may be unimportant for the analysis, in which case the user may add material to remake a square cube. However, sometimes changing the geometry would result in a long delay: meshing parameters may need to be tediously reset after the geometry change. Also, the user may have to return to the native CAD modeling package, or wait for a support person to return to the CAD modeling package. In addition to rounded geometry, many times a volume is not (multi) sweepable because some of the linking surfaces are not perfect rectilinear shapes, and are not automatically selected as SubMappable.[6] (For the purposes of this paper, SubMapping is a 2D surface mesher, which creates quadrilateral meshes that are a contiguous collection of structured *mxn* rectangles.)

Often the user wishes to (multi) sweep the volume anyway, perhaps because he knows that the sweep smoothing algorithm[11] will create a high quality mesh, or because he is willing to accept a lower quality mesh in that volume in return for conforming sweep directions for the entire assembly. If he wishes to sweep the volume, he must specify the source and target surfaces of the MultiSweep. In the past, he had to also specify other meshing parameters at the curve and vertex level. This paper describes Sweep Verification, a new tool that takes the given source and target surfaces, and sets curve types and linking surface schemes and vertex types so that the volume is sweepable. This tool can dramatically reduce the time to mesh: manually setting the curve and vertex types so that a volume is sweepable is a tedious and error prone process. Even a small volume might have 50 vertices, each of which, on average, might be in three surfaces. To set them all, a user must click 150 times in the graphics window of CUBIT's GUI, in addition to pulling down and selecting menu options.

Sweep Verification is a tool in the general category of Sweep Forging. The term "Sweep Forging" is a double entendre, implying either hammering the geometric and topological features of a volume into a sweepable shape, or falsifying the nature of the geometry and topology so that other algorithms think the volume is sweepable.

2 SWEEP FORGING

This section describes sweep forging, the process of taking a model that is not already sweepable, and modifying the topology or classification of topology, in order to make the volume sweepable. The geometry is not decomposed or reshaped, but is re-classified as if it were a different shape. E.g. two curves that meet at a 180 degree angle may be classified as meeting at a 90 degree angle.

2.1 Definitions, Vertex and Curve Types

Vertex Type. The type of a vertex of a surface is the number of quadrilaterals in the structured mesh of that surface that contain (or will contain) that vertex; see Figure 1. For example, a mapping surface has four type 1 vertices; any other vertices are type 2. A vertex has a distinct type for each surface containing it. Reversals are rare, except for models with thin partial cuts.

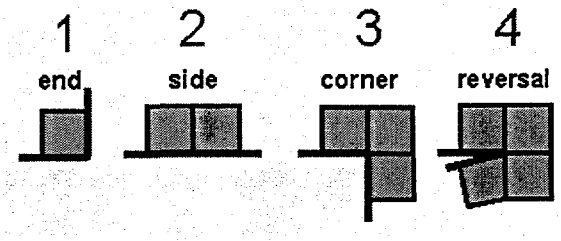

Figure 1. Vertex types.

Curve Type. The type of a curve of a volume is the number of hexahedra in the mesh of the volume that contain (or will contain) each mesh edge of the curve. Figure 2 illustrates the four curve types, which are analogous to the four vertex types.

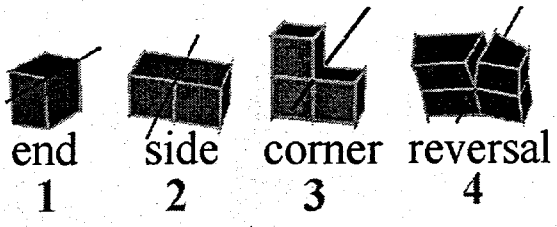

Figure 2. Curve types.

Firmness. Curve and Vertex types, and mesh schemes, have a firmness, which measures whether other tools can modify it. User-set types are hard-set, meaning that no automatic tool will change them. Automatically set types are usually soft-set, and unset types are default-set.

Additional Sweeping Terms. A swept volume has a set of source and target surfaces, which may have unstructured meshes. All other surfaces are linking surfaces, and have a structured map or SubMapped mesh. A swept mesh of a volume consists of a topological projection of the source and target meshes, along a sweep path, in conforming layers of hexes. See reference[7] for a precise definition of sweeping.

2.2 Sweep Verification

In this section we describe Sweep Verification, the process of setting surface schemes, and classifying the types of curves and vertices in order to force a volume to be sweepable. Since the vertex types on non-linking (source and target) surfaces are irrelevant for sweeping, they are ignored throughout this section.

Consider the example in Figure 3. Automatic volume scheme selection will not choose to sweep this volume, since the desired linking surfaces do not admit a good enough structured mesh. Nonetheless, suppose we want to sweep it anyway. In order to do so, we need to specify the source and target surfaces. From there, Sweep Verification will automatically assign a mapping or SubMapping scheme to the linking surfaces, set the curve types so that the source surfaces form a right angle with the linking surfaces, and set the vertex types of the linking surfaces so that the sweep layers will proceed from the source surfaces to the target surfaces.

Our Sweep Verification algorithm does not always produce the correct curve and vertex types to ensure sweepability. (As a backup, we propose casting the constraints as an integer linear program, and sacrificing some solution quality; see Section 6, Future Directions.) However, our current algorithm does automatically handle many simple examples, relieving the user from tediously becoming familiar with all of the curves and vertices in a given model and setting their type.

For example, the simple example in Figure 3 takes 2 minutes to setup and mesh in CUBIT's GUI with Sweep Verification. Without Sweep Verification, setting curve and vertex types in CUBIT's GUI increases the time to 5 minutes, assuming that the user realizes that curve and vertex types need to be set. Often, it is not obvious why a given volume will not automatically sweep. Diagnosing and fixing a similar problem in a production model last year took a sophisticated user two hours.

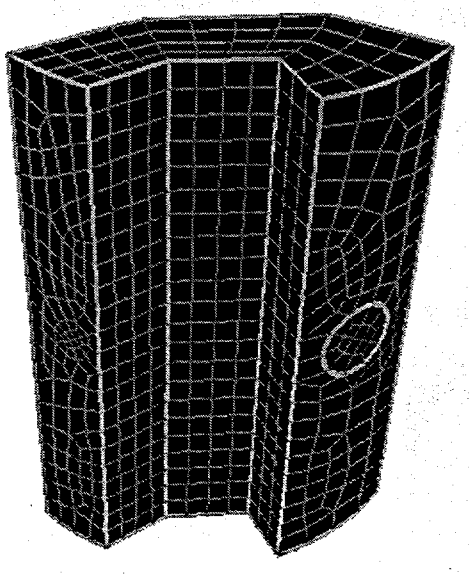

Figure 3. Sweep Verification saves a significant amount of time, even on simple examples.

When Sweep Verification does fail, the user can seed the process by manually adding a little bit of information, i.e. by manually setting a few curve or vertex types. When Sweep Verification is re-run, the user-set classifications are taken as fixed, and the algorithm has to adjust only the remaining classifications, and has a greater chance of success. Hence even when the process is not fully automatic, it is still faster for the user. Also, Sweep Verification's running time is fast, due to its serial nature.

2.2.1 Sweep Verification Algorithm

The main algorithmic steps of Sweep Verification are the following:

1. Set source and target surfaces.
2. Set curve types based on dihedral angle and adjoining surface classification
3. Set vertex types based on single surface criteria.
4. For each vertex, compute the desired sum of types over all linking surfaces based on adjoining curve types, surface classifications, and fuzzy angle criteria.
5. Adjust the vertex types so that the goal sum is met, using a pigeonhole algorithm.

In step 1, source and target surfaces are set either using CUBIT's automatic volume scheme selection,[7] or manually by the user. Later steps take this surface classification as fixed. Also, linking surfaces are given structured meshing schemes, i.e. map or submap, in this step. Currently, sources and targets are automatically selected successfully only when step 5 is not needed. In future work, we plan to extend automatic volume scheme selection to take advantage of Sweep Verification; see Section 6, Future Directions.

In step 2, curve types are set. A curve adjacent to two source or two target surfaces must by of type 2. A curve adjacent to one source surface and one target surface must be of type 4, which is rare. Otherwise, geometric tests are necessary. Between a non-linking and a linking surface, if the interior dihedral angle is less than 180 degrees, then the curve type is 1; otherwise it is 3. The type between two linking surfaces matters only in so far as it determines whether a curve is a reversal that should be treated as a degenerate source/target surface. If the interior angle is less than or equal to 315 degrees, then the curve is treated as type 2. (The actual number of hexes attached is arbitrary and doesn't affect Sweep Verification.) If the interior angle is greater than 315 degrees, then the curve is a reversal. The consequences of linking surface reversals are described in step 4. The curve types set in step 2 can be incorrect, since the criteria are purely local.

In step 3, vertex types are set for each linking surface independently, so that each linking surface may be meshed according to its structured meshing scheme. The algorithm described in "Choosing corners of rectangles for mapped meshing" [8] sets the vertex types for mapping surfaces.

We have a new algorithm for setting vertex types for SubMapping surfaces that works very well in practice when coupled with step 5. First, vertex types are set based on computing the interior angle (with respect to the local surface normal) between the two curves containing the vertex, and rounding to an integer multiple of 90 degrees.

Vertex types are converted to *turning types*, by subtracting 2. For an exterior loop the sum of turning types should be –4; for an interior loop the sum should be +4; and for a flat loop, such as the two loops of a cylindrical surface, the sum should be 0. A surface may have one exterior loop or two flat loops, and any number of interior loops. Once the desired turning sum for a particular loop is decided, the types are adjusted to get that sum, as illustrated by the following example. Suppose that the sum is 2 and a sum of 4 is desired. To decide which vertex types to change, we consider the *fuzziness* of the current type, defined by

$$fuzzy\_up = (vertex\_type + 1) * 90 - angle .$$

We increase the type of the two vertices with the smallest *fuzzy_up* values by 1 each. As in other steps, types that have been explicitly set by the user are taken as fixed and not adjusted.

The algorithms of step 3 are used as subroutines for step 5.

In step 4, for each vertex, we compute the desired sum of vertex types. The main observation is that the sum of vertex types, over all **linking** surfaces that contain the vertex, must be a particular value depending on adjoining curve type and surface classifications. (Recall that vertex types on non-linking surfaces are irrelevant.)

If there are no reversal curve types adjoining a vertex, we can make the following observations. For a vertex whose adjoining surfaces are all linking surfaces, the sum of vertex types must be 4. For a vertex with adjoining linking and non-linking surfaces, the sum of vertex types over all **linking** surfaces must be either 2 or 4. The sum must be 2 if there is only one adjoining linking surface or if adjoining curves between linking and non-linking surfaces are all type 1 or all type 3; see Figure 4 and Figure 5. Otherwise, there are curves between linking and non-linking surfaces of both type 1 and type 3, and the sum should be 4; see Figure 6.

We use will this curve type test in the linear program formulation mentioned in Section 6, Future Directions. However, the curve types are not always set correctly, e.g., when a linking surface is tangent to a source surface. So, in the current implementation, where we do not adjust curve types after step 2, we use a different test. We desire a vertex type sum of 4 only if the initial vertex type sum is at least 3 and the sum of interior vertex angles is at least 270 degrees.

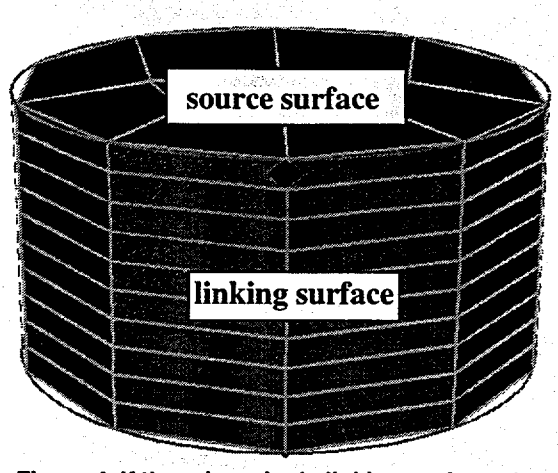

Figure 4. If there is a single linking surface, the vertex type on it must be 2.

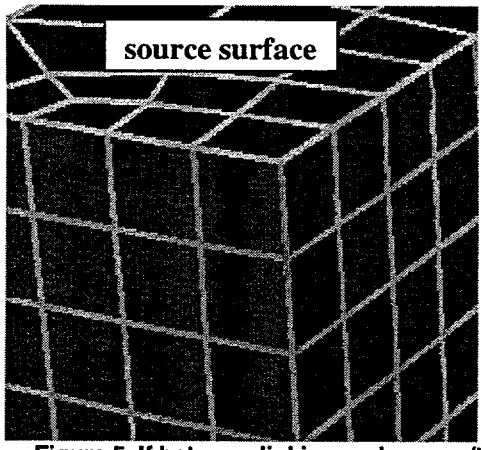

Figure 5. If between linking and source/target surfaces all curves are type 1 (or all are type 3) then the sum of types must be 2.

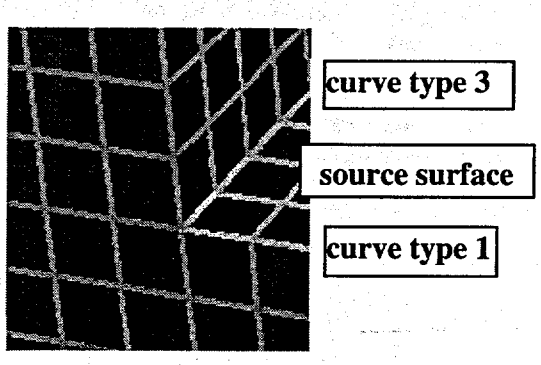

Figure 6. If between linking and source/target surfaces there are curve types of both 1 and 3, then the sum of linking surface vertex types must be 4.

If there are reversal curve types, then we enumerate some common cases, but do not enumerate all possible cases. We have reversal type curves perpendicular to the sweep direction, between two linking surfaces. Such a curve

should be treated as a degenerate source/target surface; see Figure 7. To catch this case, we check if there is a single reversal, it is between two linking surfaces, and there are no adjoining source/target surfaces. In this case the desired vertex type sum is 6.

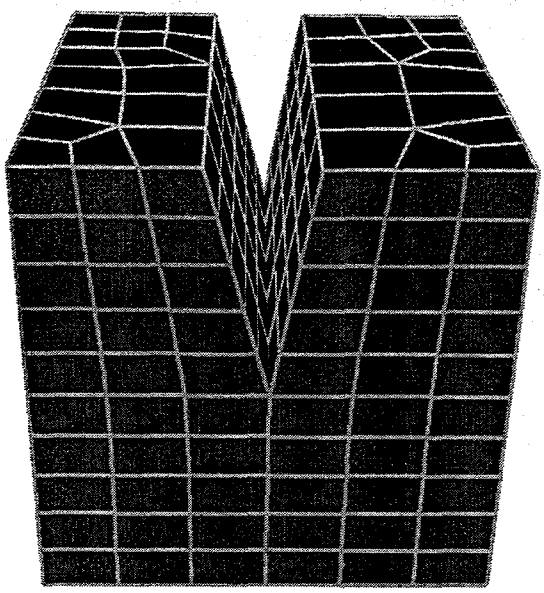

Figure 7. The curve at the bottom of the "V" is treated as a degenerate source surface during sweeping. It is a reversal curve. At the near vertex, the sum of vertex types is 6: two inside the "V" plus four on the near face.

We also check for the case of a reversal curve whose tangent is in the direction of the sweep, and which is adjacent to some source/target surfaces. This curve is not a degenerate source/target of the sweep, so the desired vertex type sum is 2 or 4 as before. Most other cases we handle implicitly by simply requiring that the sum of the vertex types must be the current sum and giving hints to the user if the types cannot be automatically adjusted; this allows the user to fix problems by explicitly setting the vertex and curve types. In all explicit and implicit cases, we require that the sum of types must be even. The evenness criterion comes from the fact that each hex containing a vertex has two faces containing that vertex parallel to the sweep direction. These faces are either on a linking surface, or are interior to the volume and paired with another hex containing the vertex.

In step 5, once the desired sum at each vertex is known, we attempt to adjust the vertex types in order to meet that desired sum and maintain the mappablity/submappability of linking surfaces. The order in which types are adjusted is important. Once a vertex type is adjusted, it is considered fixed and is not changed later in the process. The process is successful more often if the vertices where there is the least freedom are adjusted first. We have a verification loop, which proceeds until all vertex types are fixed. Within the verification loop, we have two loops.

The first loop is over surfaces of the volume. If all of the unfixed vertex types of a surface must be certain values in order to mesh the surface, those vertex types are fixed. For example, if all of the vertex types except the four ends are already fixed on a mapping surface, the remaining four are fixed.

The second loop is over vertices of the volume. If, over all surfaces of a particular vertex, the vertex types must be a certain values in order to sum up to the correct goal sum, all vertex types at that vertex are fixed. For example, suppose the goal sum is 4, and three surfaces meet at a vertex, and one of the vertex types is already fixed at type 2. Then the two other vertex types are fixed at type 1. It may be that at no vertex the types are completely fixed. In that case the algorithm makes a choice, adjusting the types at a vertex where there is the least freedom, in order to make progress. Suppose the current type sum is too small by 1, then the type of one of the vertex types for the vertex must be increased. We consider candidate vertex types in order of their fuzziness values. We change the most fuzzy vertex type, treat it as fixed, and use step 3 as a subroutine to check if the other vertex types on the corresponding surface can be changed so that a structured mesh is again possible. If a structured mesh is not possible, then we try the next most fuzzy vertex type, and so on.

This pigeonhole process works for many volumes. However, there are cases where the algorithm will fail, since we are not solving the global problem for all vertices and surfaces simultaneously. However, after telling the user which vertices the adjustment failed on, it is usually easy for him set the correct curve or vertex types by hand in the (small) problem area and re-run Sweep Verification. This still meets our goal of reducing the time to mesh by alleviating the user's need to touch all of the curves and vertices of the volume.

3 MULTISWEEPING

Having described our advances in automating the process of setting up sweeping parameters, we now focus on our advances in MultiSweep robustness. The procedure used in most sweeping methods is to identify a set of topologically equivalent source and target surfaces. The source surfaces are meshed, and then copied a single layer at a time through the volume until reaching the target surface.[2] [3] Each layer of mesh within the volume, then, is a projected copy of the mesh on the source surfaces.

Sweeping has become reasonably robust for volumes which contain one or many source surfaces and only a single target surface. Meshing volumes with many sources and many targets is a significant extension of the current sweeping algorithms. This is due to the fact that any mesh that is placed on a source surface must eventually 'match up' with the geometric features found on the target surfaces. To overcome the potential conflicts between the mesh on the source surfaces and the topology on the target surfaces, several algorithms have been designed to resolve potential conflicts by either decomposing the volume further or imprinting the source surfaces with the

appropriate geometric features from the target surfaces.[1] [9] [10] Figure 8 shows an example of a volume which is considered multi-sweepable. Notice that the mesh on the source surfaces could not be projected correctly onto the target surfaces because of mesh-topology conflicts.

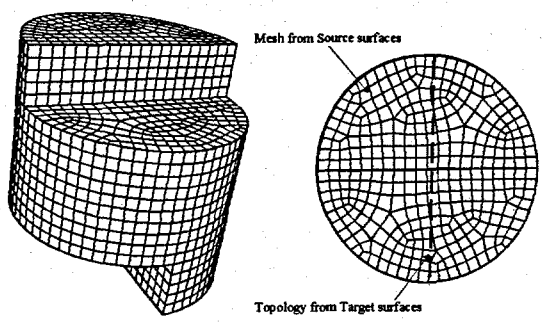

Figure 8. A MultiSweepable volume with potential mesh topology conflicts between the source and target surfaces.

3.1 The MultiSweep Algorithm

The key difficulty in extending sweeping to MultiSweeping is ensuring that the meshes of the source surfaces respect the topologies of the target surfaces. The current method used in CUBIT is based on the *loop dragging* framework designed by Lai Mingwu at Brigham Young University as part of his doctoral work.[9] We have revised the implementation of these steps since first publication, in order to make the method reliable. Each of the meshed loops of the unmeshed target surfaces are dragged up to the source surfaces, one layer at a time, using the layers specified by the linking surface meshes. Any potential mesh and topology conflicts are resolved by imprinting the source surfaces with the loops from the target surfaces.

The MultiSweep method involves four major steps.

1. Setup a mesh layering system for the volume, and mesh the linking **surfaces** and source/target **loops** (only).
2. Drag the meshed loops from the target surface layers up to the source surface layers. Combine the source surface topology and geometry with the dragged target loops to created additional source surface loops.
3. Partition the source surfaces based on the new loops, using virtual geometry.
4. Mesh the source surfaces. Mesh the volume using a combination of (many to one) sweeps: each target surface loop is swept to from its projected source surface loops.

The following subsections describe our new procedures for step 2 in detail. The other steps are straightforward or have been already described in other work. [3, 11, 12]

3.1.1 Projecting nodes and smoothing

To determine the location of source imprints, the meshed target loops are dragged (projected) one mesh layer at a time until the source surfaces are reached. The nodes of the intermediate loops are called *curtain* nodes: connecting the layers together forms a curtain partitioning the volume into the inside of the loop and the outside. The placement of curtain nodes in physical space determines the placement of the source imprints. Ideally, we desire that source imprint locations correspond (in some geometrically meaningful way) to the locations of the target imprint curves. If this is not the case, seaming operations may fail to create desirable source imprints, leading to poor mesh quality. In extreme cases, poor placement of the curtain nodes may cause seaming to fail, in which case no swept mesh can be created.

The algorithm proceeds through the sweep layers in an iterative fashion, projecting the current curtain node layer to the next layer. The user may set a *sweep imprint smoothing* flag, in which case each layer is smoothed after it is projected. As a preliminary step, the boundary of the current and next mesh layer, as defined by linking surfaces, are each assembled into a doubly-linked node list.

We first consider the case of no sweep imprint smoothing. Given the current curtain node, the two lists are used to construct a matrix that, when applied to the current node position, yields the position of the next curtain node; see Figure 9. The details of this method are described in full in reference [3]. The procedure is initialized using target imprint nodes and target bounding node loops. The matrix method is adequate when the curtain nodes can be satisfactorily placed via an affine transformation (e.g., translation, rotation, and scaling).

For more complex or warped geometries involving more general transformations between the current and next loops, we rely on sweep imprint smoothing. This smoother works similar to the one devised for the Cubit SweepTool algorithm, [11] which uses a weighted Winslow smoother for 2D unstructured meshes. The weighted Winslow smoother is, in turn, based on an unweighted Winslow smoother for 2D unstructured meshes.[12] In the sweep imprint smoothing method, the nodes surrounding the current curtain node are gathered. These *neighbor* nodes are used to determine smoothing weights, which are derived from expressions involving projections of the Winslow operator (evaluated in a Taylor Series expansion) onto the tangent plane. The weights are then used to "morph" the current node's position into the next node's position, using the positions of the next node's neighbors. Derivatives needed in the weighted Winslow smoother are again computed via a Taylor series expansion using the positions of the next node's neighbors.

Sweep imprint smoothing is a very effctive tool. When it was first introduced, the number of cases in the MultiSweep test suite that meshed successfully increased by 25%. Even with the subsequent improvements to loop imprinting, the sweep imprint smoother remains an essential part of the multi-sweep algorithm because it improves overall mesh quality.

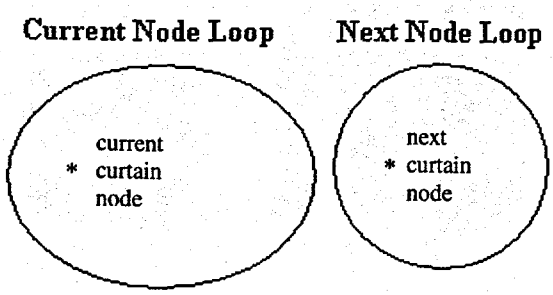

Figure 9. Left shows the geometry of the boundary of the current layer. Right shows the boundary of the next mesh layer. The geometric relationship between the current curtain node and the layer boundary is preserved in the next layer through a matrix transformation. When necessary, an additional step preserves the geometric relationship between the current curtain node and its neighbors in the next layer through sweep imprint smoothing.

3.1.2 Seaming

As the nodes from the target loops reach the source surface levels, some loops from below will inevitably 'line up' with loops on the source surfaces. Whenever the loops line up, a seaming operation is used to simplify the loops and combine nodes between loops where possible.

Two types of seaming operations can be performed, 'proximity' seaming and 'smart' seaming. Proximity seaming operations occur based on geometric proximity to other nodes. From a node in one loop, the closest node in a boundary loop is found and the proximity to the first node is determined. If the two nodes were within a quarter of a mesh edge length between each other, the nodes were combined into a single node. This process was continued for each node in the first list, combining the nodes in the two loops, where possible.

'Smart seaming' has been added to supplement the proximity checks. Smart seaming starts at a common node between two loops and looks at the next nodes of both loops. If the angle between the two next nodes and the common node is less than forty-five degrees, is within reasonable geometric proximity, and the nodes can be combined without affecting quality or producing inverted elements, then the nodes are combined. Smart seaming has greater flexibility in combining nodes than the proximity checks alone, and reduces the number of small or sliver surfaces that may be generated during partitioning.

In addition to smart seaming, numerous checks are made to ensure that any nodes being seamed will not reduce the quality of the resulting mesh. In particular, the nodes are checked to ensure that any merging completed between the nodes will not produce poor quality or inverted elements, and that nodes on the geometric curves of the volume are not seamed together. Figure 10 shows an example of a seaming procedure.

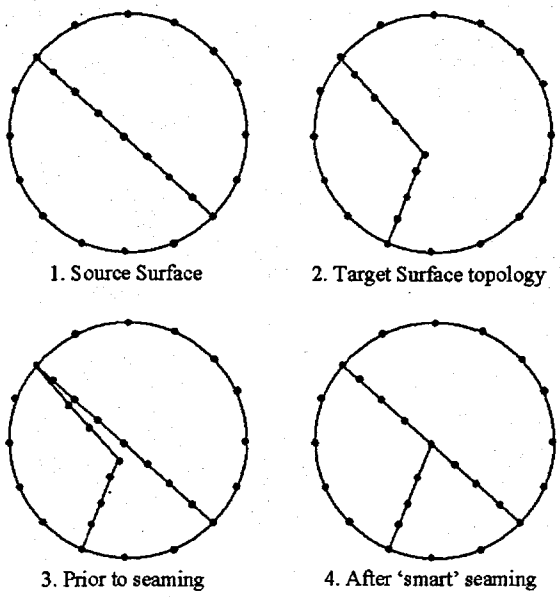

Figure 10. Loop seaming prior to imprinting on a source surface.

3.1.3 Loop Boolean Operations

As the loops are brought up from the target surfaces to the source surfaces, the loops need to be broken up into new loops and correctly assigned to the source surfaces. Breaking the loops into smaller loops based on the interaction with the other loops belonging to the source surfaces is accomplished by using loop Boolean operations: union, intersection, and subtraction.

When creating the new loops on the source surfaces, the most important constraint to remember is that to produce an all-quad mesh, the number of nodes in a loop must be even. Therefore when combining nodes, care must be taken to ensure that the new, simpler loops contain an even number of nodes. It is also important to realize that this procedure for combining nodes can affect the ability to successfully produce an even numbered loop in subsequent Boolean operations with other loops. Figure 11 shows an example of combining nodes to form odd and even loops.

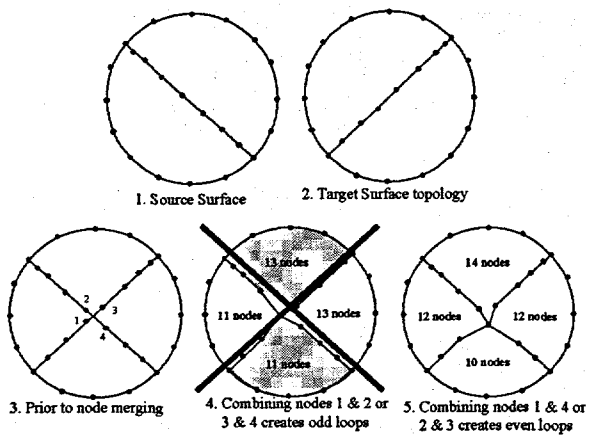

Figure 11. Combining nodes to form odd and even loops.

The current procedure for generating the simplest set of loops on a source surface is to first generate a set of loops that lie completely on the source surface, i.e. a set of loops that do not overlap any loop of the source surface and do not lie outside the source surface. This is accomplished by taking the target loops from below and using the subtraction Boolean with the source surface boundary loops, taking care that resulting loops have an even number of nodes. Once the overlap has been resolved, the remaining loops on the source surface then go through a series of subtraction and intersection Booleans to form the simplest loops on the source surface. Each of the simple loops corresponds to either a new source surface, or a hole within a pre-existing surface.

3.2 Multi-volume MultiSweeping

Another way in which we have extended the usefulness of MultiSweeping is extending it to meshing assemblies, or groups, of volumes. This presents some new challenges. A single volume in the group may be meshable using one-to-one, or many-to-one, sweeping techniques; however, the collection taken together may require MultiSweeping before the collection will be meshable. A simple example of this with three cylinders is shown in the series of Figure 12, Figure 13, Figure 14, and Figure 15. Individually, all of the cylinders are sweepable with many-to-one, or one-to-one sweeping techniques. However, meshing the parts singly would probably result in a non-contiguous mesh at the inter-volume surfaces.

Figure 12. Three stacked cylinders that are individually many-to-one or one-to-one sweepable. The collection taken together is only meshable with MultiSweeping.

Figure 13. Three stacked cylinders after the target surfaces have been imprinted with the source surface topology.

Figure 14. Three stacked cylinders after the source surfaces have been imprinted with the target surface topology.

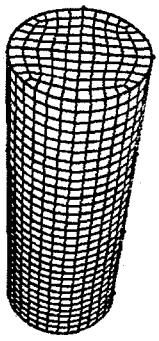

Figure 15. Completed mesh of the three cylinders.

In order to get a contiguous mesh, we must drag imprints across multiple volumes in a particular order. We are able to do this automatically using a slight modification to the sweep grouping techniques discussed by White and Tautges [7] and the MultiSweep loop imprinting algorithms described in Section 3.1.

The first step is to align the source and target lists on the volumes, such that each target surface corresponds to a source surface on the next volume. Once the volumes have been ordered from the uppermost source surface down to the lowest target surface, the MultiSweep algorithm is applied to the volumes in order, imprinting the source surface topologies onto the corresponding target surfaces. When all volumes have been imprinted from source to target, the order is reversed and the topology from the target surfaces is imprinted onto the source surfaces. The imprinting of source surfaces to target surfaces and then back from target surfaces to source surfaces guarantees contiguity between the meshes. Each volume is then meshed, in the previously determined order, using sweeping.

4 EXAMPLES

Figure 16 and Figure 17 are examples where Sweep Verification is necessary.

The relatively poor quality in Figure 16 was nonetheless sufficient for the desired analysis. In the future, for Sweep Forging we propose to explore methods for automatically improving mesh quality and connectivity, similar to Pillowing[13] and Grafting[14].

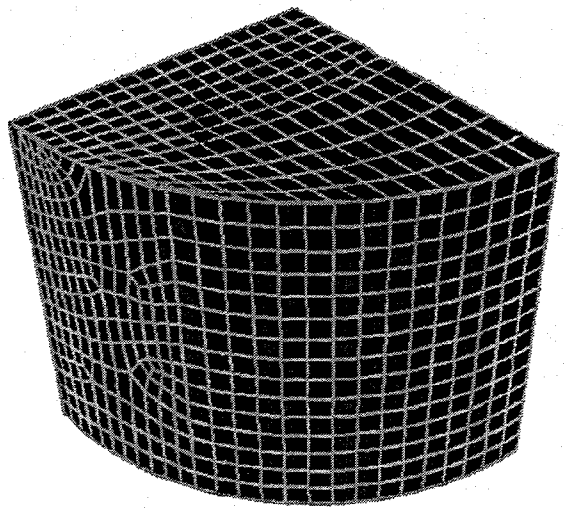

Figure 16. Sweep Verification forced a surface tangency to be treated as a 90 degree angle.

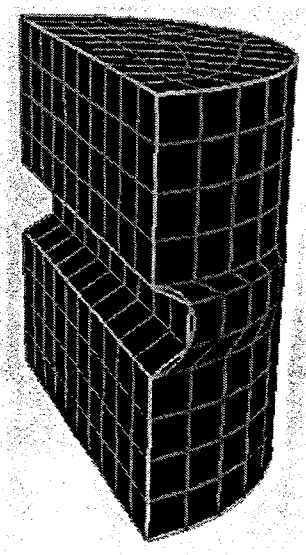

Figure 17. Sweep Verification forced the corners of the curved linking surface to be correct for sweeping the entire volume.

The following examples in Figure 18, Figure 19, Figure 20, and Figure 21 did not reliably MultiSweep before implementing the improved dragging, imprinting, and seaming operations described in this paper.

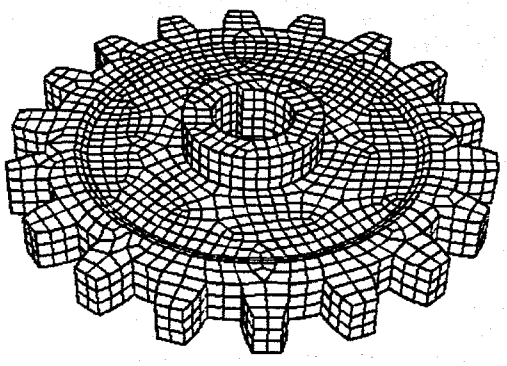

Figure 18. A MultiSwept gear (courtesy of Caterpillar).

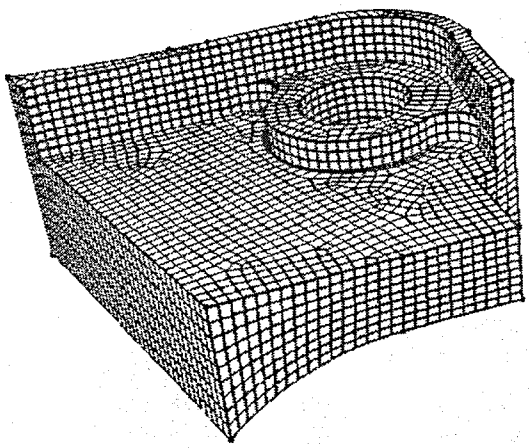

Figure 19. A MultiSwept volume requiring sweep smoothing (courtesy of Caterpillar).

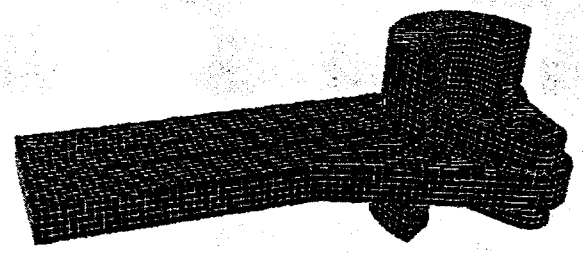

Figure 20. To fully automate the meshing of this volume, interval assignment needs to be run after imprinting (courtesy of Caterpillar).

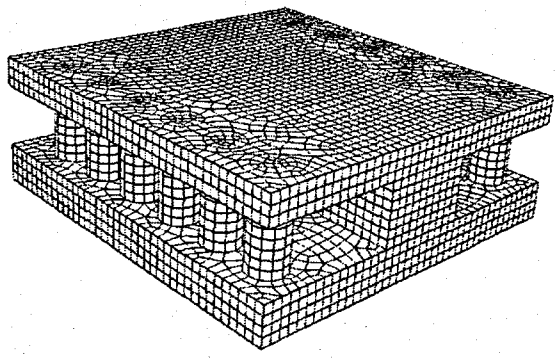

Figure 21. In this MultiSwept example, loops are picked up and lost during the sweep.

5 CONCLUSIONS

In summary, in this progress report we have shown that the time to create hexahedral meshes can be reduced by extending sweeping to reliable multiple source, multiple target sweeping of assemblies. Also, the process of setting up a MultiSweep can be sped up through automatically classifying surface, curve and vertex parameters with Sweep Verification.

6 FUTURE DIRECTIONS

Currently, CUBIT's automatic volume scheme selection algorithm only works if the vertex types chosen on local surface criteria make the linking surfaces form valid chains, i.e., if the vertex types chosen by step 3 of Sweep Verification are correct and do not need to be adjusted by step 5. In future work, we intend to modify Sweep Verification to return a measure of how much the curve and vertex types needed to be adjusted, so that automatic volume scheme selection can use that measure to select sources and targets. This is analogous to how corner picking is used by automatic surface scheme selection in CUBIT; see references [7][8].

Also, Sweep Verification may be recast as an integer linear program, provided certain desires and constraints are relaxed. This would serve as a useful backup algorithm when the existing algorithm does not find a feasible set of vertex types.

MultiSweeping is also in the process of being extended. One extension that will make the algorithm much more flexible is to have an interval assignment phase after imprinting. (The loop imprinting should still be done using a nodal representation; the mesh density affects the result of Booleans, and this is a desirable feature.) There are two main motivations. The first motivation is that often times loops are totally coincident in the seaming process, but are not directly connected by linking surfaces, such as the top and bottom loops in Figure 20. In this case, the intervals are not forced to be the same before the imprinting phase, but after imprinting we know the loops correspond, and should have equal intervals. The second motivation is that sometimes intervals

should be adjusted slightly in order to make features match up better. For example, sometimes nodes are seamed which are far apart, in order to ensure that resulting loops are even. Sometimes loop Boolean operations fail or result in excess mesh skew, again because loops must remain even.

REFERENCES

[1] T. Blacker, "The Cooper Tool," Proc. 5th International Meshing Roundtable, pp. 13-29, 1996.

[2] M. L. Staten, S. A. Canann, and S. J. Owen, "BMsweep: Locating Interior Nodes During Sweeping," Proceedings 7th International Meshing Roundtable, Sandia National Laboratories, pp. 7-18, 1998.

[3] P. Knupp, "Next-Generation Sweep Tool: A Method for Generation All-Hex Meshes on Two-and-One-Half Dimensional Geometries," Proc.7th International Meshing Roundtable, Sandia National Laboratories, pp. 505-513, 1998.

[4] L. Mingwu, S. Benzley and D. R. White, "Automated Hexahedral Mesh Generation by Generalized Multiple Source to Multiple Target Sweeping," Proc. 5th U.S. National Congress on Computational Mechanics, pp. 94, 1999.

[5] CUBIT homepage, http://endo.sandia.gov/cubit.

[6] D. White, Automatic, "Quadrilateral and Hexahedral Meshing of Pseudo-Cartesian Geometries Using Virtual Subdivision," Published Masters Thesis at Brigham Young University, 1996.

[7] D. R. White and T. J. Tautges, "Automatic Scheme Selection for Toolkit Hex Meshing," Proc. 5th U.S. National Congress on Computational Mechanics, pp. 90, 1999. To appear in IJNME, Sept. 2000.

[8] S. A. Mitchell, "Choosing corners of rectangles for mapped meshing," Proc. Thirteenth Annual Symposium on Computational Geometry, pp. 87-93, 1997.

[9] L. Mingwu, "Automatic Hexahedral Mesh Generation by Generalized Multiple Source to Multiple Target Sweeping," Published Doctoral Dissertation at Brigham Young University, 1998.

[10] S.-S. Liu, and R. Gadh, "Automatic Hexahedral Mesh Generation by Recursive Convex and Swept Volume Decomposition," Proceedings 6th International Meshing Roundtable, Sandia National Laboratories, pp. 217-231, 1997.

[11] P. Knupp, "Applications of Mesh Smoothing: Copy, Morph, and Sweep on Unstructured Quadrilateral Meshes," Intl. J. for Num. Meth. Engr., Volume 45, Issue 1, pp. 37-45, 1999.

[12] P. Knupp, "Winslow Smoothing on Two-Dimensional Unstructured Meshes", Engr. With Computers, 15: 263-268, 1999.

[13] S. A. Mitchell, T. J. Tautges. "Pillowing doublets: refining a mesh to ensure that faces share at most one edge," Proc. 4th International Meshing Roundtable, pp. 231-240, 1995.

[14] S. R. Jankovich, S. E. Benzley, J. F. Shepherd, and S. A. Mitchell, "The Graft Tool: an all-hexahedral transition algorithm for creating a multi-directional swept volume mesh," Proc. 8th International Meshing Roundtable, pp. 387-392, 1999.