Reliable Whisker Weaving via Curve Contraction

Nathan T. Folwell and Scott A. Mitchell[1]

Abstract. *Whisker Weaving is an advancing front algorithm for all-hexahedral mesh generation. It uses global information derived from grouping the mesh dual into surfaces, the* STC*, to construct the connectivity of the mesh, then positions the nodes afterwards. Currently we are able to reliably generate hexahedral meshes for complicated geometries and surface meshes. However, the surface mesh must be modified locally. Also, in large, highly-unstructured meshes, there are usually isolated regions where hex quality is poor. Reliability has been achieved by using new, provable curve-contraction algorithms to sequence the advancing front process. We have also demonstrated that sheet moving can remove certain types of invalid connectivity.*

keywords. hexahedra, mesh generation, advancing front, topology, curve contraction

1. Introduction

The finite element method (FEM) is effective for studying a wide variety of physics. Before a FEM analysis can be performed, however, a mesh of the model must be generated. Of particular interest to Sandia National Laboratories is structural mechanics simulations, often coupled with electromagnetism and radiation. For these problems, it is important that the mesh be conformal, be composed of hexahedra, and have high quality near the boundary. At Sandia, simulations are often performed on models that are different from typical industrial models; these models often have hundreds of interlocking parts, surrounded by a potting material whose geometry is the complement of the union of the other parts. The parts often have simple geometries, but the potting itself has a complex geometry that structured and semi-structured meshing algorithms have little hope of addressing. Decomposing the potting geometry into simple pieces is helpful but tedious since automatic decomposition tools are not yet mature. Obtaining a conformal mesh is difficult. Even with careful manual decomposition it is difficult to keep the sweep directions of 2.5-dimensional algorithms from colliding. Meshing complex models would be much easier if we had an algorithm for generating unstructured hexahedral meshes from a fixed bounding surface mesh. We envision that most parts would be meshed with a (semi-) structured algorithm, and the general meshing tool would address parts where sweep directions collide, and parts such as the potting that are geometrically complex and have lower quality requirements. Plastering[13] and Whisker Weaving[23] are two general meshing tools that are being developed by the CUBIT project at Sandia to meet these requirements.

Both Plastering and Whisker Weaving are based on the advancing front approach, which is motivated by the need to conform to a fixed surface mesh and for high quality near the boundary. Without both of these simultaneous requirements, alternatives abound. In [1][16][17], the medial axis is used to subdivide volumes into simpler subregions which are then meshed with a structured approach. Mesh quality is good, but the mesh doesn't necessarily conform to a fixed surface mesh. The octree or grid based approaches used in [18], [19] and [20] start with a regular mesh and then adapt the outer elements to the boundary of the volume. A new approach is to recursively insert a layer of hexes separating the volume into two smaller sub-volumes[3]. Another approach in [14] uses the dual to transform the graph of the surface mesh by placing layers of hexes hugging the boundary. The layer is removed from the problem, leaving a surface mesh with one less dual cycle. Eventually the surface mesh of a single hexahedron is obtained. The approaches [3], [14], and the current form of Whisker Weaving all concentrate on building the mesh one layer at a time. The method in [22] relies on pattern recognition, local mesh refinement and coarsening, and variational mesh smoothing techniques. Plastering advances the front based on geometric tests; for complicated geometry, however, it has trouble closing the connectivity on the interior of the mesh. Recent results on stopping with a well-shaped left-over region (*void*) followed by filling the void with tetrahedra appear promising.[12][13] [26] In contrast to Plastering, Whisker Weaving advances the front based on connectivity information inherent to a global grouping of the dual. Geometric criteria near the boundary are of secondary importance. In this interim report, we describe how Whisker

---

1ntfolwe@sandia.gov and samitch@sandia.gov, http://endo.sandia.gov/~samitch. Parallel Computing Sciences Dept., Sandia National Laboratories, Albuquerque, NM 87185. The authors were supported by the Mathematical, Information and Computational Sciences Division of the U.S. Department of Energy, Office of Energy Research, and work at Sandia National Laboratories, operated for the U.S. DOE under contract No. DE-AL04-94AL8500. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. DOE.

Weaving is able to reliably generate an all-hexahedral mesh for large and complex geometries. Currently, we must modify the surface mesh. Except for ensuring that the surface mesh has an even number of quadrilaterals, this is non-fundamental; we have an algorithm on paper for fixed surface meshes, but have not yet implemented it. This algorithm will probably produce some degenerate hexes called *knives*. Another drawback is that, despite better algorithms for removing degenerate connectivity (described here) and improving smoothing [8], the quality of the hexes are often unacceptable. For example, often isolated hexes have negative Jacobians. We plan to develop a hex mesh improvement algorithm based on a set of local connectivity-swapping operations and smoothing, as is commonly done for triangular[2], tetrahedral[6] and quadrilateral meshes. [7][21]

Given a surface mesh satisfying mild conditions, the existence of a hexahedral mesh conforming to it has been proven.[5][10][25] We developed our algorithm loosely based on the constructive steps of Mitchell's proof.[10] The proof is based on the Spatial Twist Continuum (*STC*) [15] which is the observation that the dual of a quadrilateral mesh can be grouped into an arrangement of curves, and the dual of a hexahedral mesh can be grouped into an arrangement of surfaces; see Figure 1. Conversely, given certain conditions on the arrangements, a curve arrangement dualizes into a quad mesh and a surface arrangement dualizes into a hex mesh. The problem of extending a quad mesh into a hex mesh then becomes the problem of extending the arrangement of dual curves into an arrangement of surfaces, and then fixing the arrangement to ensure that the surfaces dualize to a hex mesh.

Mitchell's proof [10] states that there exists a way to extend dual curves to dual surfaces by simultaneously manipulating the topological properties of the curves and extending the surfaces into the volume. These topological operations on the curves reduce the number of intersections among the curves. A curve shrinks to a point and disappears from the problem when it no longer intersects any other curve. For this reason, we refer to the algorithm as the *curve contraction* algorithm. This paper describes a new, provably-correct algorithm for carrying out this extension, and our robust, effective implementation. Currently we have implemented the curve-contraction algorithm for *simple* (not self-intersecting) curves and a local pre-processing algorithm that perturbs the surface quad mesh so that the dual curves are simple. Mitchell's proof also describes how to incrementally add surfaces to an arbitrary arrangement so that it dualizes to a hex mesh. This paper describes our implemented algorithm for these fix-ups that produces better quality and fewer hexes than the straightforward translation of the proof.

The remainder of the paper is organized according to the flow of the algorithm. In section 2, we recall the STC. In section 3, we describe how to perturb the surface mesh to remove dual-curve self-intersections. In section 4, we present the main result of this paper, our new algorithm for creating a surface arrangement by contracting curves. In section 5, we describe fix-ups for converting this arrangement to a reasonable mesh. Section 6 gives examples of good-quality meshes of small geometries, and bad-quality meshes of large geometries. Section 7 describes our plan for overcoming Whisker Weaving's current limitations. Conclusions follow in section 8.

2. STC Definitions

Whisker Weaving is based on the Spatial Twist Continuum, STC, the dual of a hex mesh grouped into surfaces or sheets in general position.[15] Each *sheet* is dual to a layer of hexes. The intersection of two sheets is a *chord*, the dual to a column of hexes. While the mesh is being formed, the dangling end of a chord is a *whisker* and is dual to a quadrilateral face on the meshing front. The intersection of three sheets is a *vertex*, the dual of a hex. See Figure 1. A quadrilateral surface mesh also has an STC, an arrangement of curves or *loops* in general position. A loop is defined by recursively passing from one mesh edge of a quad to the edge opposite it, until encountering the first edge or the boundary of the surface mesh. For our purposes, the surface mesh has no gaps and completely encloses the volume, so all loops will be closed. The intersection of two loops is a *point* dual to a quadrilateral.

Note that the primal and dual contain the same information, but grouping the dual into the curves and surfaces of the STC illuminates global information about how a surface mesh constrains the possible interior volume meshes. We have also found that the STC is a more flexible datastructure than the primal; the STC can generically describe certain configurations that don't dualized to a well-defined hex mesh. These configurations are useful as intermediate results, as they can be fixed later.

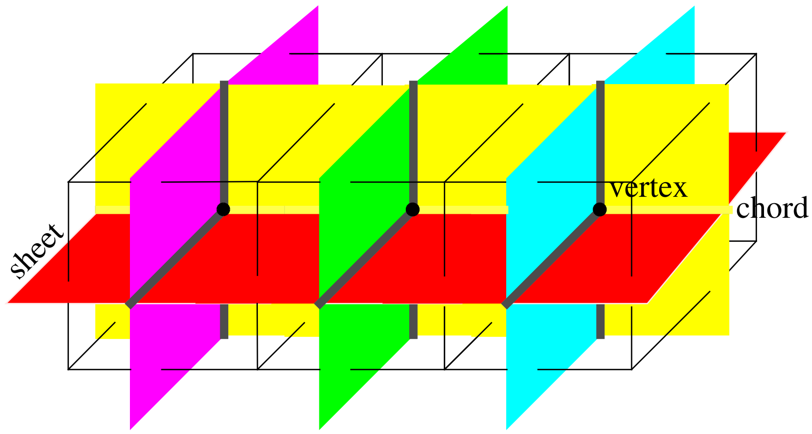

Figure 1. Three hexes and their STC.

3. Modifying the Surface Mesh to Remove Self-Intersections

We now describe how to locally modify the surface mesh so that its *loops* (dual curves) are simple. This condition is required by the current implementation of our curve contraction algorithm, section 4, but will be removed soon. The secondary goals of surface modification are to maintain good quality quadrilaterals and to change the surface mesh as little as possible.

The fact that a loop has a self-intersection is a global property that one usually wouldn't notice by looking in a small neighborhood of the self-intersection. Self-intersections arise because of unstructured surface meshes and non-rectilinear geometry, and are quite common.

The basic operation is collapsing a quadrilateral of self-intersection into two edges; see Figure 2. A *pillow* (ring of quadrilaterals) is occasionally inserted to improve the edge-valence of nodes or to meet geometric constraints; see Figure 3. Note that neither of these operations affect the topology of any other loop, so each loop may be handled in turn. The algorithm is a greedy heuristic, collapsing the most favorable face of self-intersection in the most favorable way and recursing on the resultant loop or loops. The algorithm always succeeds in removing self-intersections, but in models with thin features the surface mesh quality is sometimes poor. We suspect that minimizing the number of collapses and similar problems are NP-Complete. Another basic operation that may be useful is opening a face, but we have not explored this.

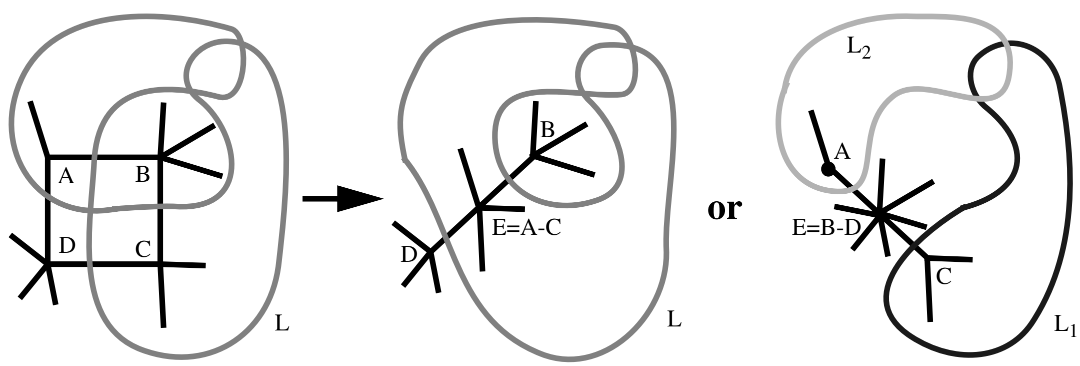

Figure 2. Two choices for collapsing: The edge-valence may differ, and the loop either remains one or splits into two.

3.1 Basic face-collapsing operation

Consider collapsing a quad into two edges by merging nodes *A* and *C* into *E* as in Figure 2. Let V(*X*) denote the *edge-valence* of $\overline{X}$, the number of edges containing *X*. The edge-valence of *B* and *D* decreases by one, and V(*E*) = V(*A*) + V(*C*) - 2. If the valence of node *X* becomes 2, then we insert a pillow around each of the nodes that are face-opposite *X*; see "Pillowing Doublets..."[11] and Figure 4 center. Similarly, regardless of edge-valence, we pillow if the interior angle between edges is nearly obtuse; see Figure 3 right. Also, if both *A* and *C* lie on curves or vertices and not a surface, then we must place a pillow around the face before collapsing it; see Figure 4.

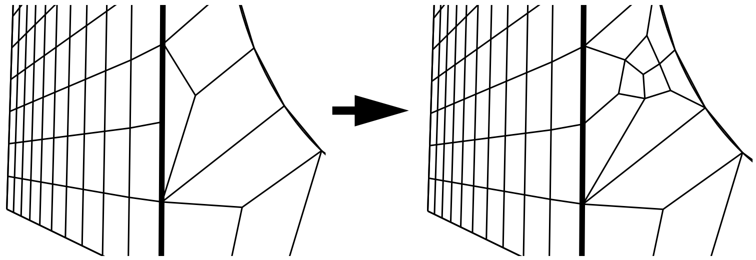

Figure 3. If an angle is nearly obtuse, we pillow all quads attached to the node opposite the large angle. The angle might be large due to poor smoothing or geometric constraints.

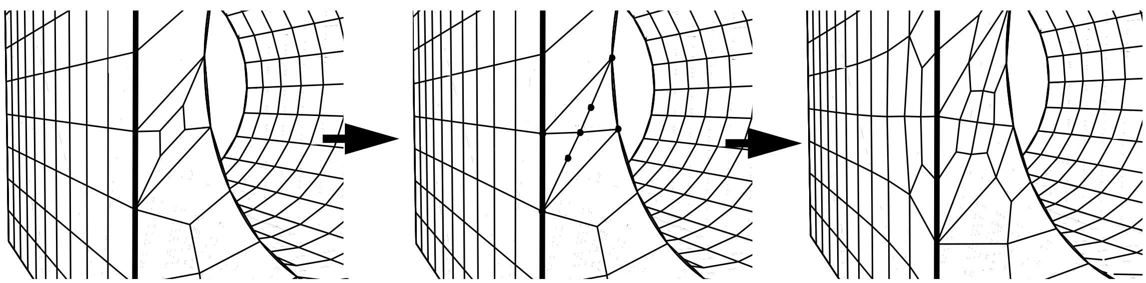

Figure 4. If both merging nodes of a collapsing face don't lie on a geometric surface, e.g. each lies on a curve, the face must be pillowed (left) before collapsing (center). Finally, the nodes opposite the resulting 2-valent nodes are pillowed (right).

3.2 Choosing which faces to collapse

Note that each face of self-intersection can be collapsed in two different ways. One way splits the loop *L* into two, L_1 and L_2, and the other way retains one loop; see Figure 2. If the loops splits into two, then points where L_1 and L_2 intersect are no longer self-intersections and need not be collapsed. Because of this we typically only have to collapse a small number of faces, which seems to grow as the cube root of the total number of self-intersections.

We favor collapsing faces such that no pillowing is required; among faces that all need or all don't need pillowing, we favor collapsing a face if the resultant nodes have edge-valence closer to 4; see [2]. Also, since collapsing a face such that the loop splits into two avoids having to collapse certain other faces, we favor a face if collapsing those other faces would have resulted in edge-valence far from 4. We conclude this section with a difficult example, Figure 5.

4. Creating Surface Arrangements by Contracting Simple Curves

Whisker Weaving is based on constructing an arrangement of *sheets* (STC surfaces) dual to layers of hexes, by iteratively performing basic weaving operations which are local. Previously, Whisker Weaving relied on a *state list* [23] similar to a first-in first-out queue to determine the order in which weaving operations should be tried. Herein we describe an alternative approach that has proven to be more reliable for larger problems. It is based on the observation that contracting a curve to a point on a shrinking sphere sweeps out a surface behind it inside the ball.[10] For Whisker Weaving, the initial curves are loops of the surface-mesh STC, and the swept out surfaces are the *sheets* of the hex-

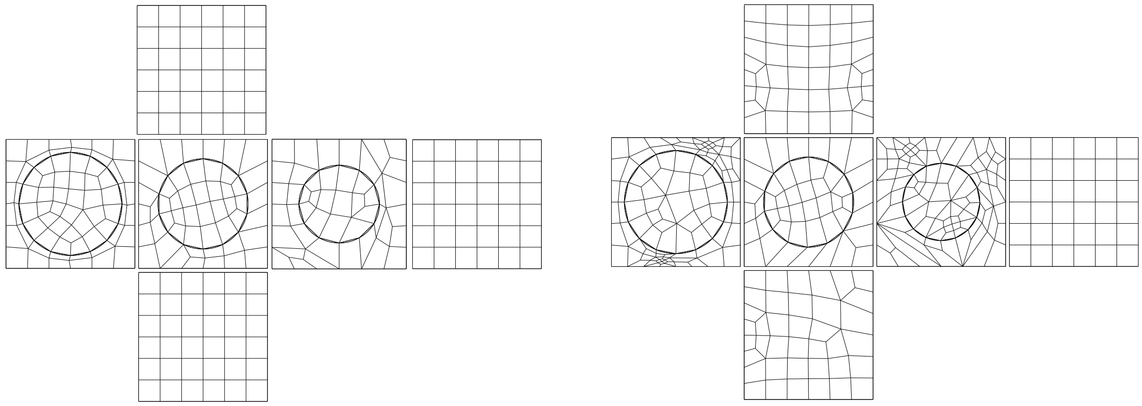

Figure 5. The unfolded surface mesh of a cube with imprinted circles. Left is the initial surface mesh, and right is after collapsing loop self-intersections and pillowing. In the initial surface mesh there are a total of six surface loops. Two of these six loops pass through **every** quad, which is typical of unstructured meshes. The thin regions around the circles make this is a difficult example for our algorithm. Of the 272 initial quads, 90 are self-intersections. Collapsing just 16 quads removed all self-intersections, and pillowing added 108 quads.

mesh STC. A shrinking curve is dual to the advancing front; as a curve is shrunk, or passed over by another shrinking curve, hex-duals are created the as front is advanced. Here we are only interested in the topological description of a sheet, namely the chords and vertices of intersection with other sheets, and not its geometric embedding.

Whisker Weaving chooses a curve to shrink and decides whether to shrink it to the left or right, then shrinks it completely. This is repeated until all curves are shrunk. Shrinking a curve is accomplished by *collapsing cells* on its boundary until no cell remains. Collapsing a cell means moving the curve so that no point of the cell is *inside* (to the right of) the curve any longer. Each cell is collapsed by a combination of three weaving operations. Both the state list and the curve contraction implementation rely on local connectivity rules not only to determine if a desired weaving operation will result in a reasonable mesh, but to automatically seam together parts of the front (the *join* operation below) and to resolve certain other degeneracies. In Figure 6, we have provided a simple example illustrating the flavor of the algorithm. A *cross* passes a curve over an intersection point. A *join* removes the empty overlap between two curves. *Cross* and *join* are defined more fully in section 4.1.

4.1 Curve Contraction Algorithm

Weaving Operations. We use three basic weaving operations in our curve shrinking algorithm; see Figure 7. In the following description we use the terminology introduced in [23] and outlined in section 2. The first operation, used for collapsing a cell of size two, is a *join*. In the STC, it corresponds to joining the whiskers (dangling ends) of two chords together to make one chord. In the primal, it corresponds to seaming two faces of neighboring hexes together into one face. The second operation, used for collapsing a cell of size three, is a *cross*. In the STC, it corresponds to extending a sheet so that it crosses the chord of intersection of two other sheets. In the primal, a cross corresponds to adding a hex containing three faces of the meshing front which pairwise share edges. The third operation, used for collapsing cells of size four or more, is a *blind chord formation*. In the STC it corresponds to extending a sheet so that it crosses a locally parallel sheet along a third sheet intersecting the previous two. The chord of intersection between these sheets does not start at a surface mesh face and is completely interior to the volume so it is called *blind*. In the primal, it simply corresponds to adding a hex that contains only two faces of the meshing front that share an edge**.**

Cell Collapsing. Given an oriented curve segment from point *a* to point *b,* we can use the basic operations to deform the segment in such a way that one of the cells neighboring the segment is collapsed.The convention we shall use is that of always collapsing to the right of the oriented segment. Recall that collapsing a cell means moving the curve so that no point of the cell is still to the right of the curve. Repeated cell collapsing will shrink a curve. Define a cell to be the segments of the points *a*, *b* and *1,...,k* as in Figure 8 left. Collapsing the cell with points *a, b, 1,..., k* is accomplished by *k -1* blind chord formations followed by a cross operation as in Figure 8 right. The running time of

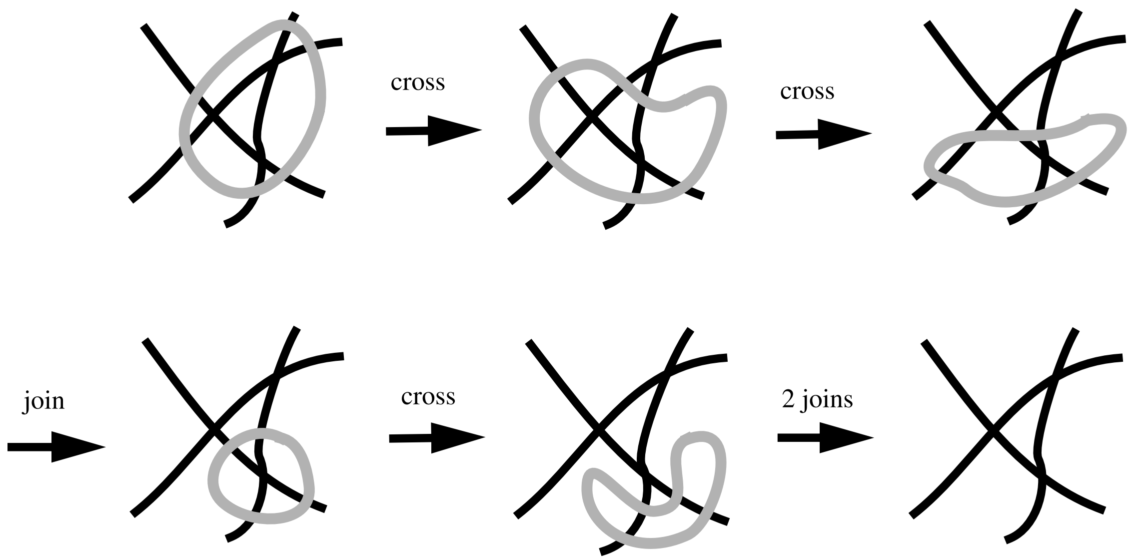

Figure 6. A simple example illustrating the curve collapsing algorithm. The grey curve is oriented clockwise.

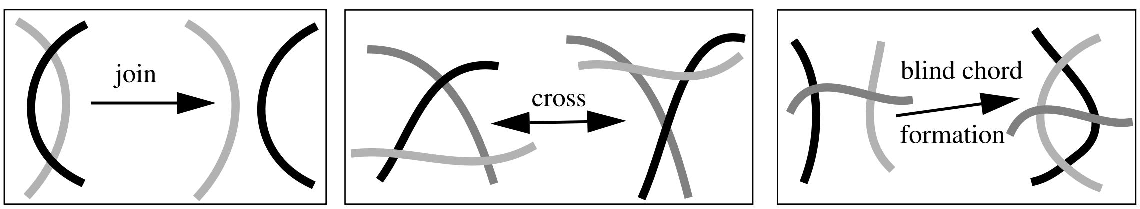

Figure 7. The three basic weaving operations used in curve contraction: join, cross and blind chord formation.

collapsing is linear in the size of the cell; each weaving operation is constant time, and collapsing takes *k* operations for a *k+2* sided cell and 1 operation for a 2 sided cell. (1 sided cells do not arise with simple curves.)

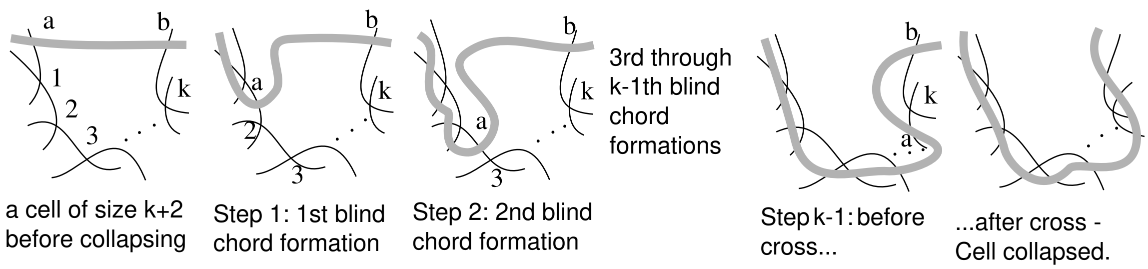

Figure 8. An example of collapsing a k+2 sided cell. Starting on the left of the figure and working to the right, we perform k-1 blind chord formations and finish with a cross operation.

Shrinking a curve. An oriented curve can be shrunk by a sequence of cell collapses. Shrinking a curve will succeed regardless of the order in which cells are collapsed. However, in practice, better meshes result if we shrink the smaller cells first. Our approach is to begin by first traversing once about the curve to find a cell *C* with the fewest number of sides *S*. We then collapse *C*. We back-up one cell, then continue traversing forward. Whenever we encounter a cell of size *S* or smaller we collapse it. If we traverse all of the way around the curve without collapsing a cell, then *S* is increased. Eventually all cells are collapsed and the curve shrinks to a point that does not intersect any other curve.

Recall Figure 6 provides a simple example. The reason that we back-up is that collapsing a cell reduces the size of the cells to either side; see Figure 9.

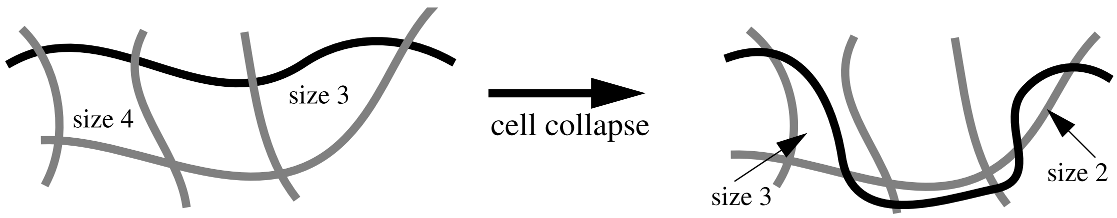

Figure 9. Collapsing a cell reduces the size of the neighboring cells by one.

To avoid forming self-intersections, we will not collapse a cell if the shrinking curve forms two or more of its sides; see Figure 10 for a simple example.

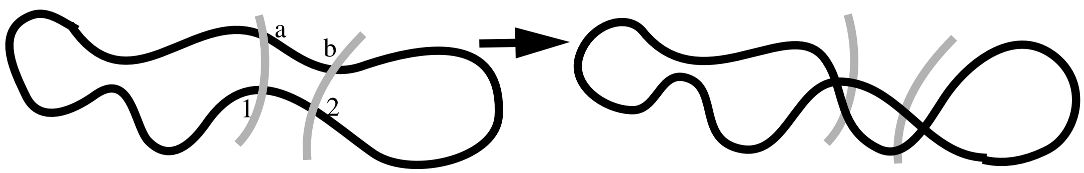

Figure 10. An example of a cell where the shrinking curve forms two sides. Collapsing the cell with points a,b,1,2 would cause a self-intersection as shown on the right, and does not lead to progress. Hence, we do not shrink such cells.

The curve shrinking algorithm provably eliminates a curve from the front. (The local rules may prevent our implementation from achieving this, however. Section 4.2 describes how we overcome this.) Collapsing a cell of size k yields one of two forms of progress. First, if k is 3 or greater, then collapsing the cell reduces the number of intersection points inside the curve by k-2. Second, if k is 2, then collapsing the cell performs a join which reduces the number of points on the curve by 2.

As implemented, the running time of shrinking a single curve is $O(n\ S_{\max})$ where n is the number of intersection points on or to the right of the curve, and $S_{\max}$ is the maximum cell size; for each cell size, we may have to traverse once about the curve, which is always $O(n)$. Also, each point to the right of the curve is moved to the left of the curve by a weaving operation in constant time. In practice $S_{\max}$ is small, usually less than 20. A single curve could consist of all the points on the front, so n could be large, but by amortizing over all curves this is not a problem. If we shrunk cells as they were encountered regardless of size, then the algorithm would take time $O(n)$. Either way, the number of hexes created is equal to the number of points to the right of the curve, up to modification by the local rules.

Choosing which curve to collapse. Repeatedly applying the curve shrinking algorithm to each curve on the front creates a complete STC. On paper, the algorithm will succeed regardless of the order in which curves are shrunk. In practice, the best order is by increasing *weight* (defined below). The weights of all curves is computed, then the curves are shrunk in that order. The running time of the curve shrinking algorithm as implemented is $O(NS_{\max} + cn)$, where c is the number of curves, n is the number of points on the front, and $N = O(cn)$ is the number of hexes generated.

Each oriented curve has two weights, corresponding to shrinking it to the *left* (from the equator towards the north pole) or *right* (from the equator towards the south pole). Let n_{rp} denote the number of points to the right of the curve, n_{lp} the number of points to the left of the curve and n_{pc} the number of points on the curve. The left-weight is defined as $w_l = n_{lp} + n_{pc}$ and the right-weight as $w_r = n_{rp} + n_{pc}$. We currently know no better way to compute these weights than the straightforward method which takes $O(cn)$ steps, where c is the number of curves and n is the number of points on the front. Note that for a rectangular parallelepiped with a regular surface mesh, shrinking in order of our weights produces a regular hexahedral mesh. In practice, our shrinking order produces a small mesh compared to other shrinking orders.

4.2 Blending Curve Shrinking with Local Rules

The curve shrinking algorithm forms a weave without directly considering the geometry or connectivity of the hexes formed. Within Whisker Weaving there is a set of local rules[24] that prevent hexes from being formed that violate certain geometric and connectivity constraints, and also automatically perform basic weaving operations to fix certain configurations. For example, before forming a hex next to the boundary of the model, one local rule requires that there must be a certain angle between the corresponding two or three faces. We have blended these local rules into the decision structure of the curve shrinking algorithm to increase mesh quality.

In the blended algorithm, curve shrinking chooses a weaving operation, and passes it to the local rules. If the operation is consistent with the local rules, it is performed. Otherwise, it is kicked back to the curve shrinking algorithm; the current cell being collapsed is returned to its initial state and curve shrinking moves on to another cell. Also, local rules automatically remove degenerate hexes such as two hexes sharing two edges, and join chords; this can cause radical changes in the curve being shrunk. This turns the provable algorithm into a reliable heuristic that produces a better arrangement of surfaces than the pure algorithm.

There are two sets of local rules. The first set is geometry rules, which only applies to hexes being formed on the geometric boundary of the model. The second set is connectivity rules, which applies universally, and prevents two hexes from sharing three faces and the like. We first attempt to weave with both sets of rules on. If we get stuck, we turn the geometry rules off, remove the partially completed weave, and start weaving over. If we still get stuck, we turn the connectivity rules off and restart. It is usually necessary to turn the geometry rules off in order to complete the weave, but rarely necessary to turn the connectivity rules off.

A structured mesh has eight hexes attached to each node. In a Whisker Weaving mesh, one cause of poor mesh quality is *high valence nodes* - nodes with high numbers of hexes attached. We treat high valence nodes much the same way as we treat local degeneracies, by a type of local rule that attempts to avoid forming them. While choosing which cell to collapse, we check the size of the cell and the size of the polyhedra behind the cell. This corresponds to taking a node on the front and checking the number of attached faces on the front and the number of attached hexes behind the front. If the sum of these two values is above a certain threshold, we collapse that cell next. This buries the node behind the front, preventing the number of hexes attached from increasing.

5. Converting a Surface Arrangement to a Well-Defined Hex Mesh.

5.1 Removing Degeneracies by Pillowing

Mitchell's [10] existence proof enumerates the possible degeneracies that would keep an arrangement of surfaces from dualizing to a well-defined hexahedral mesh. There are three types of degeneracies possible. The first is *through-cells*: the arrangement does not resolve distinct portions of the surface mesh. For example, normally a 3-cell contains at most one 2-cell of the surface mesh. If it contains one, then the dual node of that 3-cell is actually the surface mesh node of that 2-cell. If it contains more than one, it is dual to all of those nodes. One interpretation is that in order to realize the mesh, we must first collapse the surface mesh nodes; see Figure 11 for a two-dimensional example.

The second degeneracy is *non-simplicial meets*: the intersection of two cells is non-simplicial. That is, the two cells share two or more maximal sub-cells. E.g. two 3-cells sharing two distinct 2-cells is dual to two nodes being connected by two distinct edges.

The third degeneracy is *non-distinct sup-cells*: the cells containing a cell are not distinct. A vertex should be in eight distinct 3-cells, twelve distinct 2-cells, and six distinct 1-cells (STC-edges). Similarly for STC-edges and 2-cells. E.g. if a vertex lies in only seven 3-cells, then the dual hex has only 7 distinct nodes.

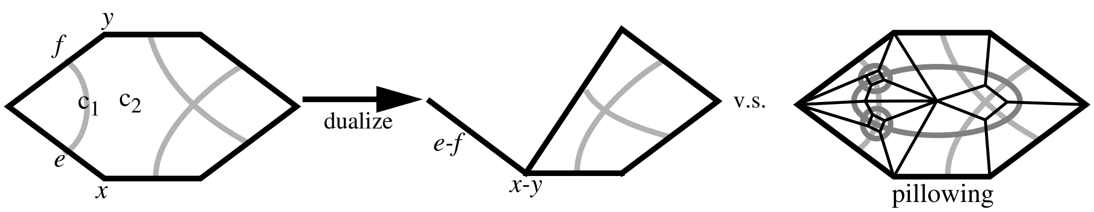

Figure 11. Left, an STC with through-cells dualizes to a mesh that conforms to a collapsed version of the bounding mesh: the 2-cell c_2 is dual to both nodes x and y, and 1-cell c_1 is dual to both edges e and f. Right, adding pillow-sheets removes this, but adds many elements.

Mitchell [10] also gives a provable construction for removing these degeneracies. However, the straight-forward implementation of that would lead to many more hexes than is necessary, and poorer quality. Our algorithm is as follows. First, we remove through-cells. We remove through-2-cells and any adjoining through-3-cells and through-1-cells by *sheet moving*; see section 5.2. Rarely we have a through-3-cell that is not part of any through-2-cell. These are removed by inserting a *pillow-sheet*, a new surface that surrounds all non-surface vertices of the through-3-cell, as in Figure 11 right.

Second, geometric checks are done on the STC-vertices connected to the surface mesh. If any vertex is dual to a hex with two or more surface-mesh quads that make a large dihedral angle, we insert a pillow-sheet that puts a boundary layer around the surface mesh. This pillowing is almost always needed if the geometry rules were turned off during weaving. We have experimented with adding a pillow-sheet surrounding just a neighborhood of the vertex, but in practice these sheets are nearly as large as the complete boundary layer and have poorer quality.

Third, we remove non-simplicial meets by pillowing the non-simplicial intersection. For each vertex, we traverse the attached 1-cells, checking that each pair has only the vertex in common. If a pair has two common vertices, we add each vertex to a list of vertices to pillow. Each vertex must be in separate pillows, since a pillow around both vertices will not remove the degeneracy. Similarly, we traverse the attached 2-cells. If any 2-cell pair has more than an edge in common, then we collect the vertices of intersection for later pillowing. We collect maximal components that are edge-connected by edges shared by both 2-cells; each components must be in a separate pillow. If any attached 3-cells have more than a 2-cell in common, we gather their intersection for later pillowing. After all these checks for the vertex, we pillow maximal sets of vertices that need not be kept separate, then proceed to the next vertex. Occasionally this fails, and we fall back on the provable algorithm of pillowing non-simplicial meets one-by-one as they are encountered.

Fourth, we remove non-distinct sup-cells. This is rarely necessary, usually just in cases where we had to pillow non-simplicial meets one by one. For each vertex, we gather its 3-cells. We pillow vertices appearing more than once in a single 3-cell.

5.2 Sheet moving for through-cells

A through-2-cell can be removed by inserting a pillow sheet surrounding all of its non-surface vertices. However, when there is a *boundary-through-3-cell*, moving the 2-cell so that it is cut by other sheets removes the degeneracy and produces fewer hexes and a better quality mesh; see Figure 12.

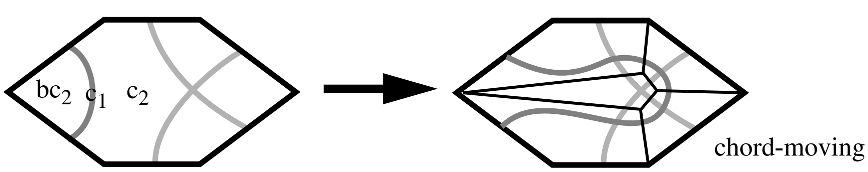

Figure 12. A two-dimensional example of sheet-moving (chord-moving) to remove through-cells: bc_2 is a boundary-through-2-cell. We move c_1 so that it goes around the vertices of c_2, where c_2 is the non-boundary through-2-cell adjacent to c_1.

Consider the 3-cells on either side of the through-2-cell. Typically, one of these cells is a *boundary-through-3-cell*: all of its vertices are either on the surface mesh or on the through-2-cell. If not, then we must resort to pillowing. Otherwise, we move the sheet containing the through-2-cell so that it surrounds the vertices of the other, non-boundary through-3-cell. In practice, there is usually a series of through-2-cells on a sheet, and we move all of these simultaneously. Occasionally a through-2-cell will pass over another. We can detect which is farther from the surface mesh, and move that one first.

6. Examples

The following small examples are good-quality, totally automatic Whisker Weaving meshes of real-world parts. While small, these parts exhibit true 3D character; to mesh these parts with a sweeping-type algorithm would require decomposing the geometry by hand, which is quite difficult in some cases.

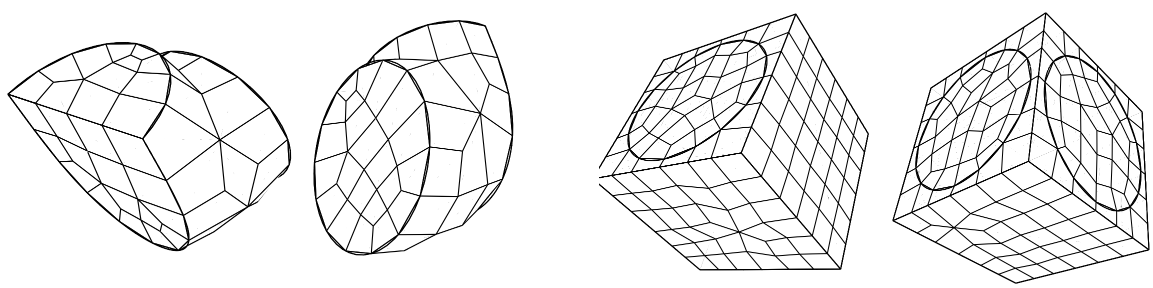

Figure 13. Left: Two views of the bent-pipe example. It has 130 hexes and the scaled jacobian ranges from 0.54 to 0.93. In the original model, the flat faces of the pipe are next to sweepable parts. Right: Two views of a complicated nugget in a large model, courtesy of Clay Fulcher. It has 499 elements. The scaled jacobian ranges from 0.12 to 0.99.

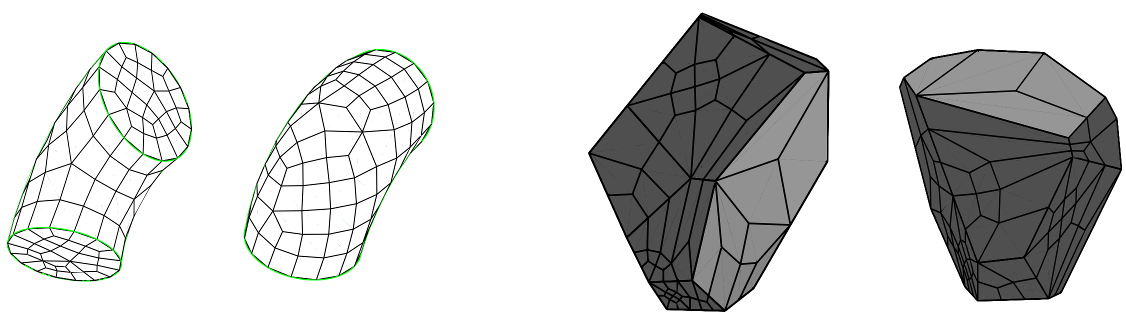

Figure 14. Left: Two views of the macaroni. It has 320 elements and the scaled jacobian runs from 0.26 to 0.95. Right: Two views of a single component of a metal grain. It has 173 elements. The scaled jacobian runs from 0.18 to 0.8.

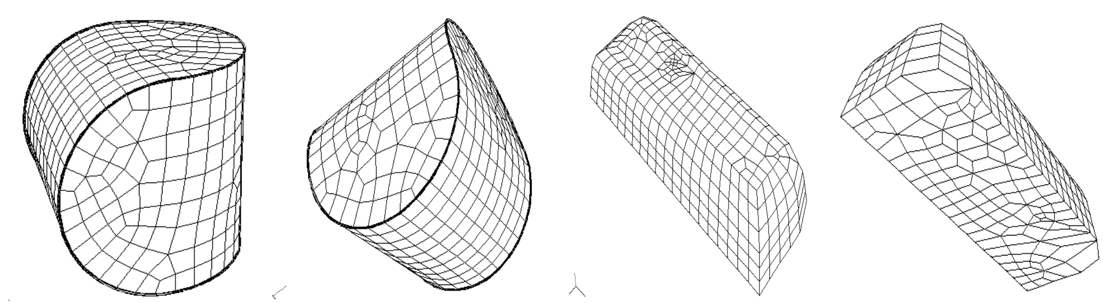

Figure 15. Left: two views of a non-sweepable geometry. It has 1320 elements and the scaled jacobian runs from 0.18 to 0.98. Right: Two views of the bathtub courtesy of Ford. It has 1041 elements and the scaled jacobian runs from 0.001 to 0.99.

The following are Whisker Weaving meshes of large, complex surface meshes and geometries. Typically 2% to 5% of the hexes have negative jacobians at nodes, making them unusable for most FEMs. Despite this, we feel these examples demonstrate proof-of-concept

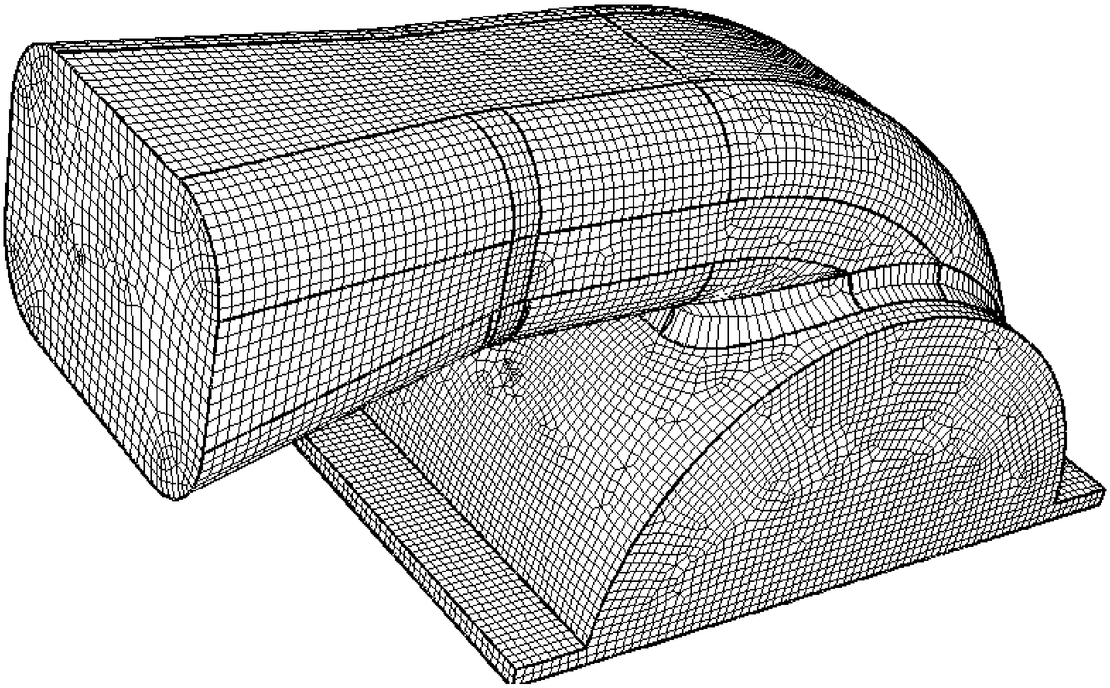

Figure 16. A view of a blower geometry complements of Rick Garcia. It has 105,999 elements and 575 negative jacobians.

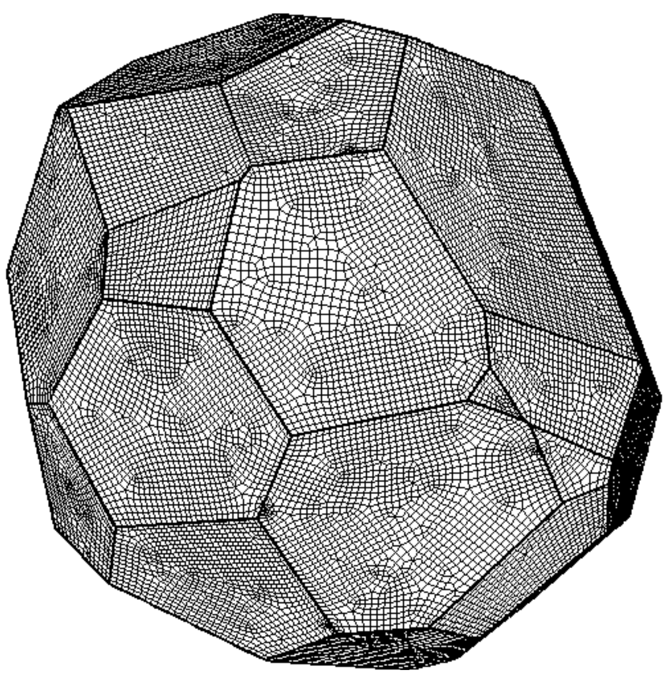

Figure 17. A single view of another component of a metal grain. It has 240,724 elements and 33 negative jacobians.

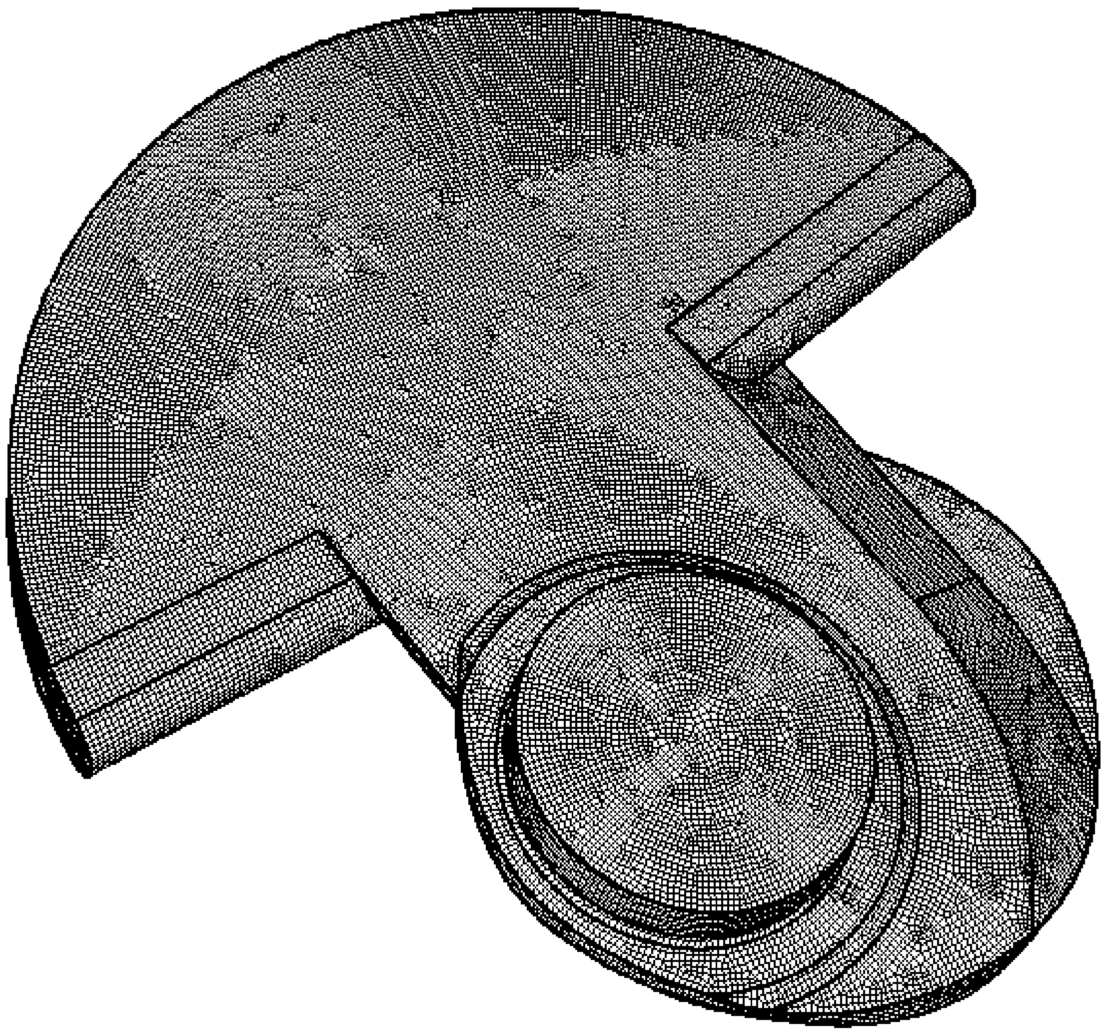

Figure 18. A view of a throw arm for a crank shaft, courtesy of Ford. It has 622,209 hexes and 15,652 negative jacobians.

7. Extensions: Contracting Self-Intersecting Curves, Non-ball Geometry.

We have algorithms for handling the self-intersection case using basic weaving operations. These are currently only worked out on paper, but appear to require not much more effort to implement than the simple curve shrinking algorithm. Progress is measured by reducing the number of self-intersections. After all self-intersections are removed, the simple curve contraction algorithm takes over. Note that the problem of simplifying self-intersecting polygonal curves has been considered in a computational setting.[9][27] However, these approaches concentrate on the geometry of the shrinking. In our case, we are interested in the topological events of the shrinking, e.g. keeping the number of events to a minimum, and, except for the initial surface-mesh loop, we have no geometry for the sheets.

We also have various algorithms on paper for reducing non-ball geometries to a small collection of meshable ball-type geometries. These algorithms have various degrees of efficiency, sophistication and provable properties. Some are geometric and some are purely topological. The common theme is to cut the handles of the geometry until only balls remain.

We are also focused on overcoming the quality problem. We wish to post-process the mesh by optimizing the position of nodes in tandem with improving connectivity regularity, as is commonly done in triangular, tetrahedral, and quadrilateral meshing. This requires research in mesh positioning, namely weighted and boundary-term smoothing. This also requires research on hex mesh connectivity swapping. Currently, poor quality concentrates around nodes with too many or too few hexes attached, where we want to swap hexes between nodes. Few swapping operations are known for hexahedral meshes, unlike for tetrahedral meshes. We plan to develop swap operations using the global information provided by the STC. The key difficulties are finding swapping operations that are local and do not affect the surface mesh, and finding a complete set of operations. We have had some preliminary success by moving dual surfaces for a special case which occurs near the bounding surface mesh; see section 5.2.

8. Conclusions

We have presented algorithms for contracting curves. These guide Whisker Weaving's basic operations and provide provable hexahedral mesh generation on paper. We have implemented a local pre-processing algorithm that perturbs the surface quad mesh so that the dual curves are simple. We have implemented the simple-curve version of curve-contraction, blended it with local geometry and connectivity rules, and implemented heuristics to improve the STC surface arrangement so that it dualizes to a hexahedral mesh. The result is that, given the freedom to perturb the surface mesh, we can reliably generate a hexahedral mesh conforming to the surface mesh. The hex meshes often have unacceptable quality in isolated regions, and our current research is focussed on overcoming this problem.

We are also currently extending our implementation to weave from a fixed surface mesh whose dual curves may be non-simple. The extension appears straight-forward. Unfortunately, surface-mesh quads where dual loops self-intersect are closely related to the formation of degenerate hexes called *knives* inside the volume. Knifes appear to be acceptable FEM elements, [4] but it would be better to avoid them. On paper knives are avoidable if the surface mesh has an even number of quads, but it is unclear if there are practical ways to avoid or remove all knives.

References

[1] Armstrong, C.G., D.J. Robinson, R.M. McKeag, T.S. Li, S.J. Bridgett, R.J. Donaghy and C.A. McGleenan, Medials for Meshing and More, *Proceedings, 4th International Meshing Roundtable*, Sandia National Laboratories, pp.277-288, October 1995.

[2] S. Canann, S. Muthukrishnan and R. Phillips, Topological Refinement Procedures for Triangular Finite Element Meshes, *Engineering with Computers*, Springer-Verlag, Vol 12, pp.243-255, December 1996.

[3] N. Calvo and S. Idelsohn, All-Hexahedral Element Meshing by Generating the Dual Mesh, *Computational Mechanics: New Trends and Applications,* CIMNE, Barcelona, Spain 1998.

[4] B. Clark and S. Benzley, Development and Evaluation of a Degenerate Seven-Node Hexahedron Finite Element, *Proceedings, 5th International Meshing Roundtable,* Pittsburgh, Pennsylvania, pp. 321-331, October 1996.

[5] D. Eppstein, Linear Complexity Hexahedral Mesh Generation, *Proceedings, 12th Annual ACM Symposium on Computational Geometry*, Philadelphia, pp. 58-67, May 1996.

[6] L. Freitag and C. Ollivier-Gooch, Tetrahedral Mesh Improvement Using Swapping and Smoothing, *International Journal for Numerical Methods in Engineering*, Wiley, Vol 40, pp. 3979-4002, 1997.

[7] P. Kinney, CleanUp: Improving Quadrilateral Finite Element Meshes, *Proceedings, 6th International Meshing Roundtable*, Park City, Utah, pp 449-461, October 1997.

[8] P. Knupp, Optimizing the 3-D Jacobian, work in progress.

[9] K. Melhorn and C. Yap, Constructive Hopf's Theorem: or How to Untangle Closed Planar Curves, *Springer Lecture Notes in Computer Science*, Vol 317, pp. 410-423,

[10] S. Mitchell, A Characterization of the Quadrilateral Meshes of a Surface Which Admit a Compatible Hexahedral Mesh of the Enclosed Volume, *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, Springer, pp. 465-476, 1996.

[11] S. Mitchell and T. Tautges, Pillowing Doublets: Refining a Mesh to Ensure that Faces Share at Most One Edge, *Proceeding, 4th International Meshing Roundtable*, Sandia National Laboratories, pp. 231-240, October 1995.

[12] S. Mitchell, The All-hex Geode Transition Template for Conforming a Diced Tetrahedral Mesh to any Diced Quad Mesh, submitted to *7th International Meshing Roundtable*.

[13] R. Meyers and P.Tuchinski, The "Hex-Tet" Hex Dominant Meshing Algorithm as Implemented in CUBIT, submitted to *7th International Meshing Roundtable*.

[14] M. Muller-Hannemann, Hexahedral Mesh Generation by Successive Dual Cycle Elimination, submitted to *7th International Meshing Roundtable*.

[15] P. Murdoch and S. Benzley, The Spatial Twist Continuum, *Proceedings, 4th International Meshing Roundtable,* Sandia National Laboratories, pp. 243-251, October 1995.

[16] M. Price an C. Armstrong, Hexahedral Mesh Generation by Medial Surface Subdivision: Part 1. Solids with Convex Edges, *International Journal for Numerical Methods in Engineering,* Vol. 38, 3335-3359, 1995.

[17] M. Price an C. Armstrong, Hexahedral Mesh Generation by Medial Surface Subdivision: Part 1. Solids with Convex Edges, *International Journal for Numerical Methods in Engineering,* Vol. 40, 111-136, 1997.

[18] R. Schneiders, R. Schindler and F. Weiler, Octree-based Generation of Hexahedral Element Meshes. *Proceedings 5th International Meshing Roundtable*, Pittsburgh, 1996.

[19] R. Schneiders, A Grid-based Algorithm for the Generation of Hexahedral Elemenet Meshes. *Engineering with Computers* vol 12, 1996, pp. 168-177.

[20] M. Shepard and M. Georges, Automatics Three-dimensional Mesh Generation by the Finite Octree Technique, *International Journal for Numerical Methods in Engineering*, Vol 32, pp. 709-749, 1991.

[21] M. Staten and S. Canann, Post Refinement Element Shape Improvement for Quadrilateral Meshes, AMD-Vol.220 *Trends in Unstructured Mesh Generation*, ASME, pp.9-16, July 1997.

[22] R. Taghavi, Automatic, Parallel and Fault Tolerant Mesh Generation from CAD, *Engineering with Computers* vol 12:178-185, 1996.

[23] T. Tautges, T. Blacker and S. Mitchell, The Whisker Weaving Algorithm: A Connectivity-based Method for Constructing All-hexahedral Finite Element Meshes, *International Journal for Numerical Methods in Engineering,* Vol. 39, pp. 3327-3349, 1996.

[24] T. Tautges, S. Mitchell, Whisker Weaving: Invalid Connectivity Resolutions and Primal Construction Algorithm, *Proceeding, 4th International Meshing Roundtable*, Sandia National Laboratories, pp. 115-127, October 1995.

[25] W. Thurston, Hexahedral decomposition of polyhedra, Posting to sci.math., 25 Oct., 1993.

[26] P. Tuchinsky, The Hex-Tet Hex Dominant Automesher: An Interim Progress Report, *Proceedings, 6th International Meshing Roundtable*, Park City, Utah, pp 183-13, October 1997.

[27] G. Vegter, Kink-Free Deformations of Polygons, *Proceedings of the 5th Annual Symposium on Computational Geometry*, pp 61-68, June 1989.