SAND99-2852
Unlimited Release
Printed November 1999

Integration of Mesh Optimization with 3D All-Hex Mesh Generation, LDRD Subcase 3504340000, Final Report

Patrick Knupp
Scott A. Mitchell
Parallel Computing Sciences
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0847

Abstract

In an attempt to automatically produce high-quality all-hex meshes, we investigated a mesh improvement strategy: given an initial poor-quality all-hex mesh, we iteratively changed the element connectivity, adding and deleting elements and nodes, and optimized the node positions. We found a set of hex reconnection primitives. We improved the optimization algorithms so they can untangle a negative-Jacobian mesh, even considering Jacobians on the boundary, and subsequently optimize the condition number of elements in an untangled mesh. However, even after applying both the primitives and optimization we were unable to produce high-quality meshes in certain regions. Our experiences suggest that many boundary configurations of quadrilaterals admit no hexahedral mesh with positive Jacobians, although we have no proof of this.

1 Introduction.[1]

The automatic generation of high-quality, all-hex meshes for complex 3D geometries would greatly reduce the Design to Analysis to Manufacturing (D2A2M) cycle time. Two essential elements in the mesh generation process are constructing the topological connectivity of the elements and constructing the geometric position of the nodes. The connectivity problem was recently solved via the Whisker Weaving (WW) and T-HEX algorithms, which both automatically produce all-hex meshes. These algorithms do not produces meshes of *high enough quality* for certain analyses. Optimization based smoothing has recently solved the problem of positioning the nodes optimally, given a fixed mesh connectivity.

The goal of the LDRD was to improve the quality of the hex meshes produced by WW, T-HEX and other algorithms. The approach investigated was to iteratively reconnect the hex elements and optimize the nodal positions. There are algorithms in the literature for improving quadrilateral, triangular, and tetrahedral meshes based on this approach as well. A key difficulty was that, prior to this LDRD project, nobody knew how to locally reconnect a hex mesh. We developed a set of reconnection primitives; see Section 2.

Hex reconnection is much harder than reconnection for other meshes. E.g., in tetrahedral and triangle meshing, a given set of nodes has many topologically distinct meshes, but in order to change the elements in a quad and especially in a hex mesh, nodes must usually be added or removed. Also, collapsing a quad in a quad mesh is a purely local operation, but collapsing a hex in a hex mesh induces an entire column of hexes to be collapsed. For some small quadrilateral boundary meshes, no high-quality conforming hexahedral mesh is known, even though we know a well-defined hex mesh exists.[1] E.g., Schneiders's open problem, a square based pyramid with three quads per triangle and four quads on the base, has no known high-quality mesh despite years of effort. So, our intuition is that the space of hex meshes is more "sparse" than that of other types of meshes. The space of high-quality hex meshes appears to be even sparser.

The dual of a hex mesh, grouped into surfaces, provides some insight into the possible ways of filling a fixed boundary of quadrilaterals with hexahedra. A T-HEX mesh dual has a special spherical structure that we tried to exploit.

We first inspected poor quality areas and applied reconnection primitives manually, thinking that we would gain insights into how to design an automatic algorithm. However, we were unable to remove certain poor quality areas: improving the mesh locally just pushed the poor quality area somewhere else. Even after many dependent steps, negative Jacobians remained. Sometimes these negative Jacobians would be pushed near the boundary of the region. The quadrilateral surface mesh bounding the hex-meshed volume was considered to have fixed connectivity and position, so there is little freedom to reconnect the hexes near the boundary. It appears that Schneiders's open problem is not an anomaly, that many large surface meshes have features, which do not admit a high-quality hex mesh.

Despite this setback, we noticed certain poor configurations could be improved. In particular, 4-valent nodes, nodes contained in only four hexes, were common causes of poor quality. One of the primitives "rotates" a set of three hexes sharing an edge. When applied to one of the four edges of the 4-valent nodes, the connectivity was markedly improved. In a T-HEX mesh, the node at the center of every original tet is 4-valent, so this operation can be applied widely. In a WW mesh, a surprisingly large fraction of the poor quality hexes are attached to 4-valent nodes.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL8500.

We implemented an automatic algorithm that applies the rotation strategy. It also applies a pillowing strategy that subdivides high-valence nodes into two lower-valence nodes. Various reconnections are tried at poor-quality nodes. This algorithm is in progress. While we suspect that it may ultimately be able to improve certain portions of a mesh, it is unlikely that it will be able to significantly improve the minimum Jacobian in many meshes.

Another means to improve mesh quality is by node-movement strategies, also sometimes referred to as mesh optimization. A mesh quality objective function is posed that, when minimized with fixed nodal connectivity, yields a mesh with improved quality. Two objective functions figure prominently in our latest thinking, a mesh Untangle objective function and a mesh Condition Number optimizer (see [5]) for a discussion of the theory of the condition number optimizer). The mesh condition number optimizer improves element aspect ratio and skew while avoiding inverted elements. The condition number optimizer was already partially implemented at the beginning of this LDRD and works well with simplicial mesh elements. More work remains to include “boundary terms” for non-simplicial elements but this was not addressed in the LDRD. Even so, the condition number optimizer was useful for this study, particularly on T-HEX meshes, where modest mesh quality improvement was demonstrated.

The major focus in the LDRD in terms of node-movement strategies was the development of an Untangle objective function, including boundary terms, for mesh untangling. We found that boundary terms could be best included using a global objective function that includes all elements of the mesh, as opposed to a local objective function that uses only certain elements attached to each node of the mesh. The Untangle objective function, applicable to both simplicial and non-simplicial elements in two- and three-dimensions, was able to successfully untangle most tangled meshes. There is no a priori way to determine if a given mesh with fixed connectivity can be untangled. The Untangler is an effective means of doing so a posteriori. Results with the Untangler on Swept, T-HEX, Geode[4] and Whisker Weaving meshes suggests that the objective function and its implementation in CUBIT will automatically untangle any mesh for which an untangled solution exists.

Unfortunately, tangled Whisker Weaving meshes could rarely be untangled. We strongly suspect that the current WW algorithm, which uses no geometric information except on the volume boundary, usually produces mesh connectivities for which an untangled mesh does not exist. Although this is a negative result, the mesh Untangler has proven its worth by showing that this is the case. In addition, mesh untangling will prove useful in conjunction with other 3D meshing schemes in CUBIT.

2 Hex Primitives

Before this work, there were only a few ways known to change the connectivity of a hex mesh. This is in marked contrast to the situation for tet meshes.

For a tet mesh, the most common local reconnection is an edge flip: take the tets surrounding an edge, remove them, and consider the polygon formed by the edge of each tet that is opposite the removed edge. This polygon can be triangulated in a number of ways, depending on how many sides it has and its angles. The convex hull of each triangle with each node of the removed edge forms a new tetrahedron. The inverse operation is also possible. This reconnection primitive can be applied widely in a tet mesh. The reconnection is purely local: tets not containing the removed edge are unaffected.

Another common operation for tet meshes is an edge collapse: combine the two nodes of an edge into one. Some triangles turn into edges and some tets turn into triangles. Tets not containing the removed edge are unaffected, except that two originally disjoint tets may now share a triangle, etc. That is, triangles on the boundary of the primitive region have been combined.

General edge flipping does not work in a hex mesh because in many cases the hexes surrounding an edge are the only way to fill their boundary quadrilaterals. However, in some cases a primitive does exist, e.g., the rotate primitive below.

Edge collapsing does not work in a hex mesh because hexes are non-simplicial. E.g., if an edge of a quadrilateral is collapsed the quad degenerates to a triangle, not an edge. However, if two edges of a quad are collapsed, the quad degenerates to an edge. Some primitives we discovered do resemble the tet edge collapse in that elements are collapsed by collapsing some of their sub-elements, and elements on the boundary of the primitive region are combined.

2.1 Collapse two hexes

Two hexes sharing a face or more may be collapsed, removing both hexes and leaving only a collection of faces. Like the edge collapse operation in a tet mesh, the boundary of the elements is unchanged, except certain sub-elements are combined. The general principle for collapsing two hexes is to find a common face, and, for each hex, find the opposite face. Each node of the first hex's opposite face is merged with the corresponding node of the second hex's opposite face. If the hexes share more than one face, two merging nodes may already be the same. Hexes sharing one through five faces can be collapsed in this way, as well as hexes sharing a face and an edge, etc. Common cases are illustrated in Figure 1, Figure 2 and Figure 3. The inverse operation inflates a pair of hexes. Any set of faces surrounded by a ring of faces pairwise sharing an edge can be inflated: each face of the set inflates into two hexes, and each face of the surrounding ring opens into boundary faces of two inflated hexes. This large operation can be accomplished by many smaller operations. Similarly, the inverse operation collapses a large set of hexes at once, or can be built up by smaller primitives. Collapsing can lead to degenerated hexes nearby. In some cases degenerated hexes can be removed by further collapsing, or fixed by pillowing; see Section 2.3 below.

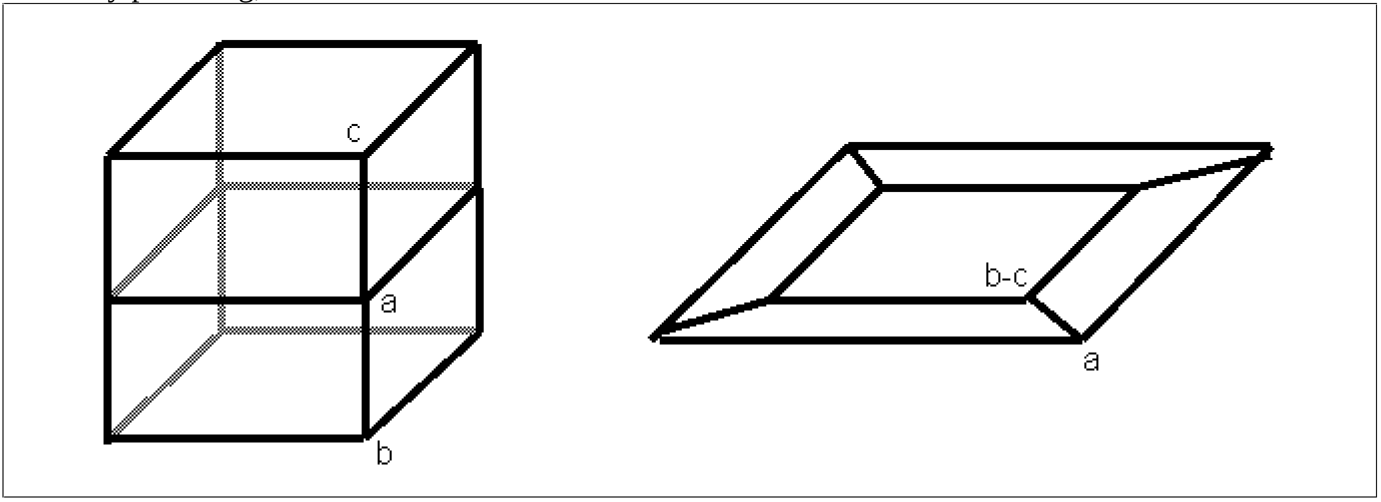

Figure 1. Collapsing two hexes sharing one face. Nodes a and b are combined.

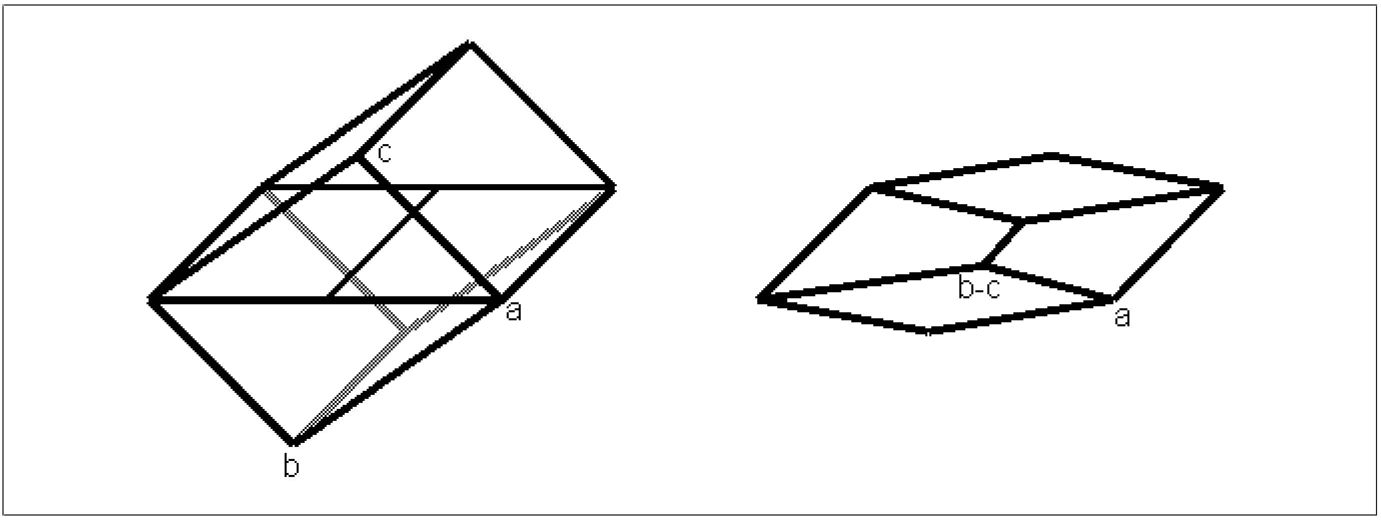

Figure 2. Collapsing two hexes sharing two faces. Nodes b and c are combined.

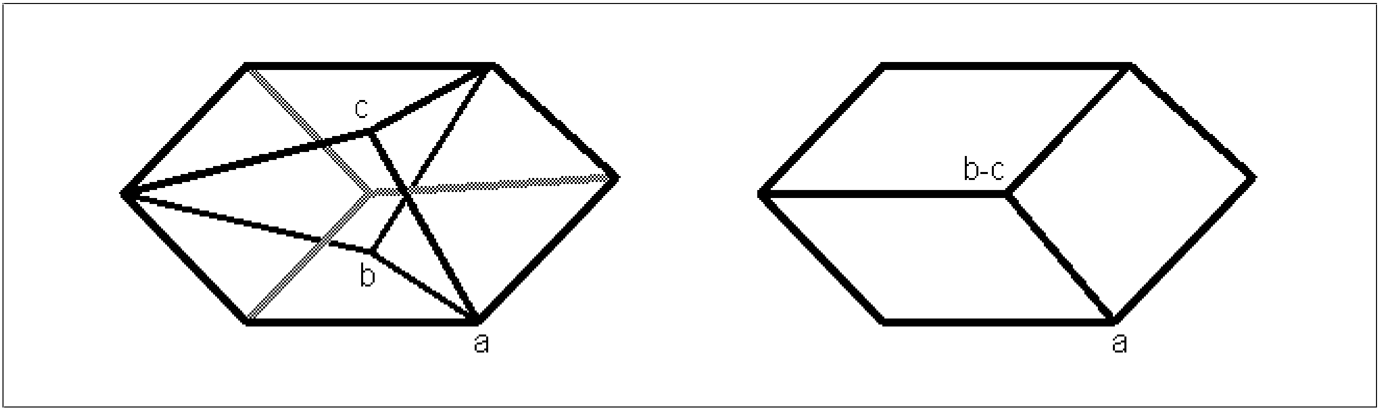

Figure 3. Collapsing two hexes sharing three faces. Nodes b and c are combined.

2.2 Rotate three hexes

The rotate three hexes primitive is applicable whenever three hexes h_1, h_2, h_3 share an edge e; see Figure 4. Consider the cycle of the six faces f_1, f_2, f_3, f_4, f_5, f_6 of h_1, h_2, h_3 not containing e, where f_1 and f_2 are in h_1, etc. We remove h_1, h_2, h_3 and replace them with hexes H_1, H_2, H_3, containing e, where H_1 contains f_2 and f_3, etc. This change is fully conforming to f_1, f_2, f_3, f_4, f_5, f_6, but not to the pattern of three faces f_7, f_8, f_9, containing just the *top* node of e. In order to restore that pattern, a single hex must inserted containing f_7, f_8, f_9 and the new top faces of H_1, H_2, H_3. If there was originally a single hex containing f_7, f_8, f_9, these two hexes are collapsed. Similarly, a hex is inserted and perhaps collapsed for the bottom node of e.

There doesn't appear to be a way to rotate an arbitrary number of hexes sharing an edge; for most internal re-arrangements around the shared edge, there isn't a set of hexes on the ends that matches the original boundary connectivity. If some hexes sharing the edge are first pillowed, so that some new edges are shared by three hexes, these hexes can be rotated, but this operations doesn't seem to be helpful.

The configuration in Figure 4 is exactly the hexes inside a single pre-subdivision tet in a T-HEX mesh. If this primitive is performed on the hexes inside two tets sharing a triangle, then we end up with six hexes instead of eight. Unfortunately, when applied to the hexes inside a tet, we were unable to find a situation in which the quality of the mesh improved. The cases in a T-HEX mesh where the quality did improve were when the primitive was applied to three hexes inside three different tets that shared an edge prior to subdivision. In WW meshes, the mesh quality improved when there was a pre-existing bottom hex that was flat (i.e. had nearly coplanar faces).

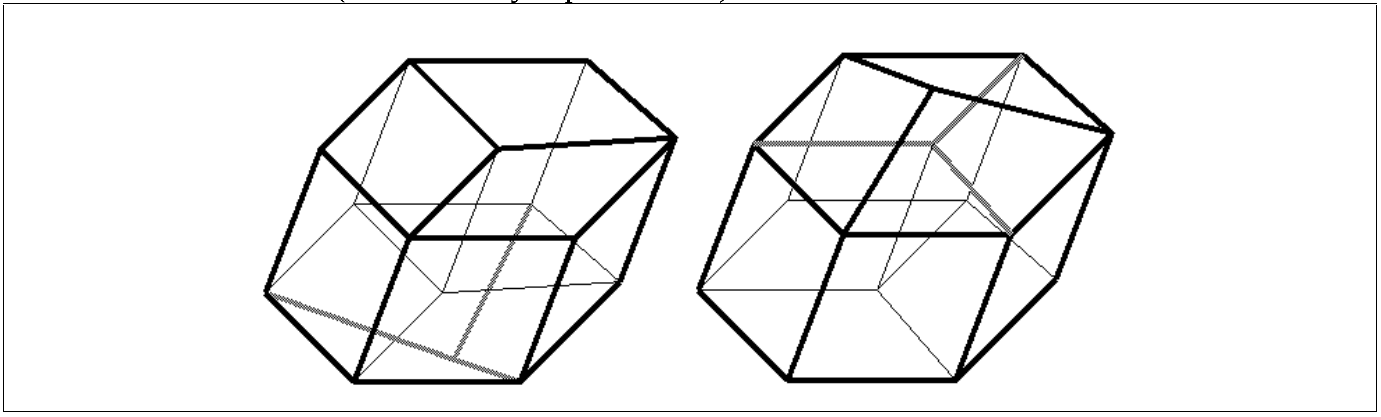

Figure 4. Rotating three hexes. Left, there is a bottom hex. Right, there is a top hex.

2.3 Pillow

Pillowing is a very generally applicable operation that inserts an entire layer of hexes that closes on itself. A pillowing strategy that fixes two faces sharing two edges in a hex mesh is described in Pillowing Doublets[2]. Pillowing starts with a *pillow set* of hexes whose boundary is a shell of quads meeting edge to edge. Each boundary quad is split into two quads, one for a hex in the pillow set and one for the external hex sharing the boundary quad (or region boundary quad mesh). Then a new hex is created connecting each boundary quad with its split quad. Nodes are repositioned by smoothing. The inverse of pillowing removes an entire layer of hexes. If the region boundary quad mesh can be modified, then pillowing and its inverse have obvious extensions in which the layer terminates on the region boundary, rather than closing on itself. Examples of pillowing are given in Figure 5.

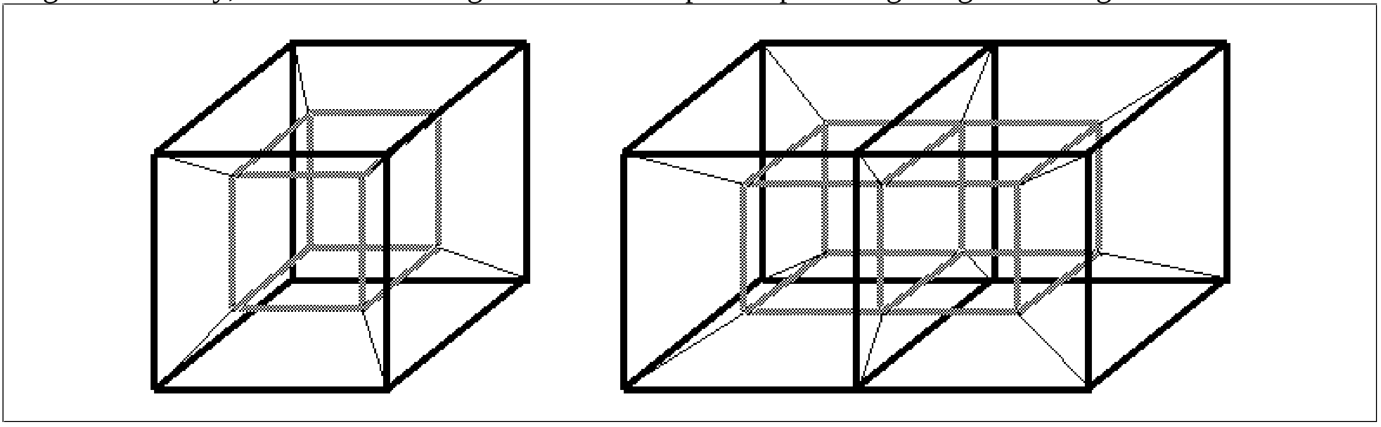

Figure 5. Left shows pillowing a single hex. Right shows pillowing two hexes sharing a face.

Pillowing can be used to decrease a node's edge-valence. Each node on the boundary splits into two, and the resulting valences can be less than the original valence of the node. Let h be the number of pillow set hexes, H the number of external hexes and f the number of boundary faces attached to a boundary node. Before pillowing, the node has $h+H$ attached hexes. After pillowing, the node has $h+f$ attached hexes and the split node has $H+f$ attached hexes. If a node has too many attached hexes, some will have a small Jacobian at the node. Pillowing can be used to split such a node and reduce the maximum hex valence. In principle, smoothing could spread the hexes out so that the minimum Jacobian is increased. Although this appears to be a useful quality improvement strategy on paper, in practice it is not effective.

2.4 Inflate hex ring

Inflating a hex ring takes a circular sequence of pairs of faces sharing an edge, and inflates each pair into a hex. A column of hexes pairwise sharing a face is created. Figure 6 illustrates the simplest form of this. The inverse operation collapses a circular column of hexes. If the region boundary quad mesh can be modified, then inflating and its inverse have obvious extensions in which the column starts and finishes on the region boundary, rather than closing on itself. Like pillowing, inflating can be used to manipulate the hex valence of nodes.

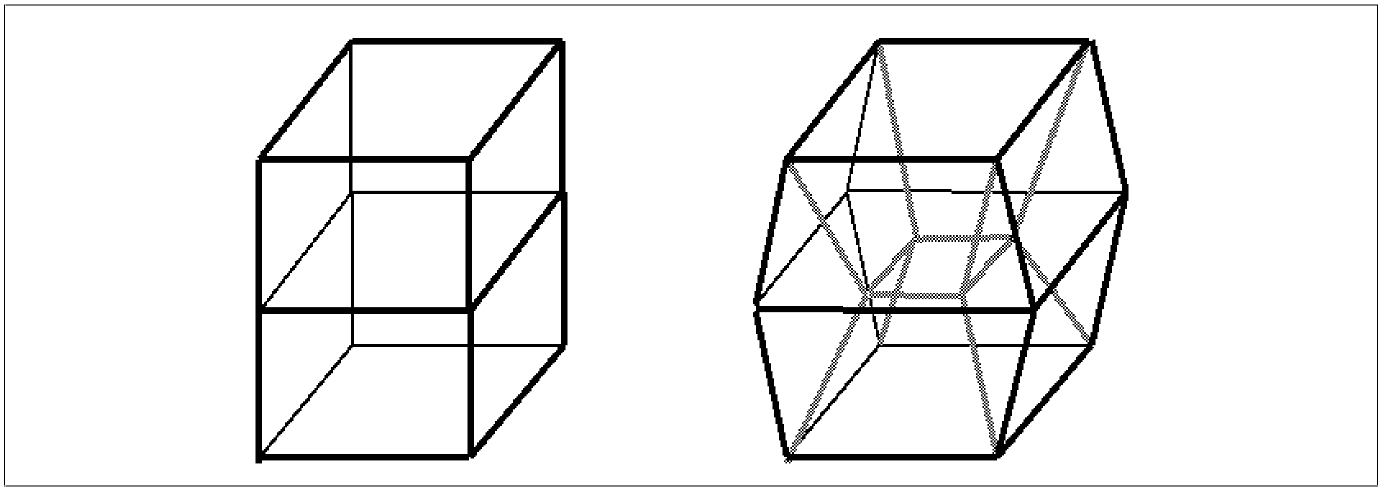

Figure 6. Inflating a ring of hexes using the pairs of faces around two hexes sharing a face.

2.5 Folwell Starfish

Nate Folwell, formerly dept. 9226, discovered the Folwell Starfish. This primitive applies whenever there is a chain of edges, the spine, with two symmetric sets of faces on either side. The spine is bounded by nodes a and b. Each set is bounded by the spine, and also an outer chain between nodes a and b. The outer chains are symmetric. The primitive flips the roles of the spine and outer chains. The Folwell Starfish is its own inverse.

The sub-operations of the primitive are the following: The edges of the spine are split into two chains, meeting at nodes a and b. Similarly, each face is split in two, an upper and lower face, meeting at the outer chain. The edges of the two outer chains are merged, as are symmetric pairs of upper faces, and symmetric pairs of lower faces. No hexes are created or destroyed. Geometrically, the new faces lie in a plane perpendicular to the old faces, but, for clarity, this distinction is not made in the following figures. Pairs of faces sharing two nodes are usually created or removed. If the primitive is applied to a single pair of faces sharing two edges, it separates that pair but creates a new pair of faces sharing two edges. See Figure 7, Figure 8 and Figure 9 for some other examples.

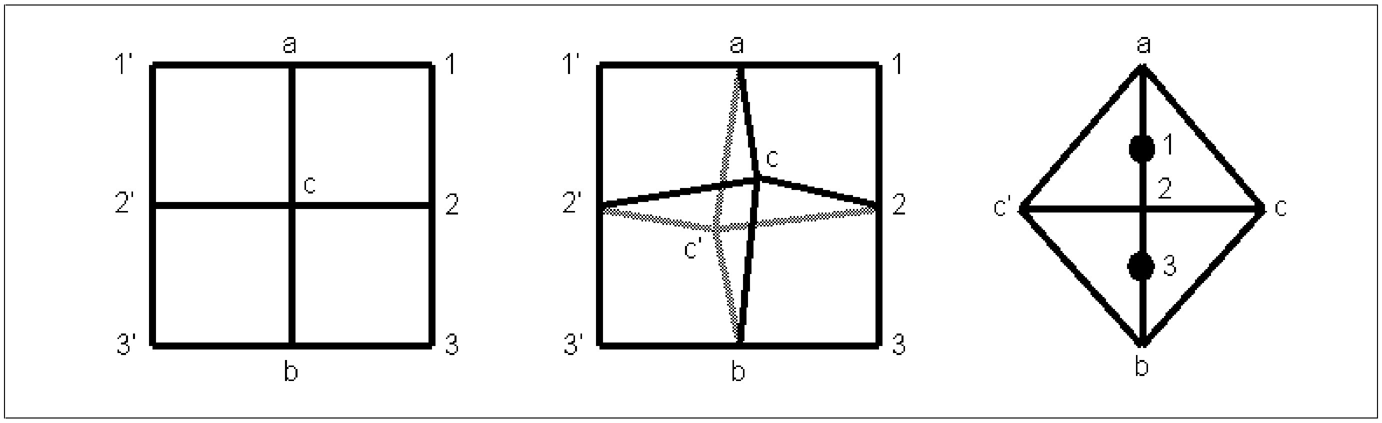

Figure 7. Folwell Starfish applied to four faces sharing a node.

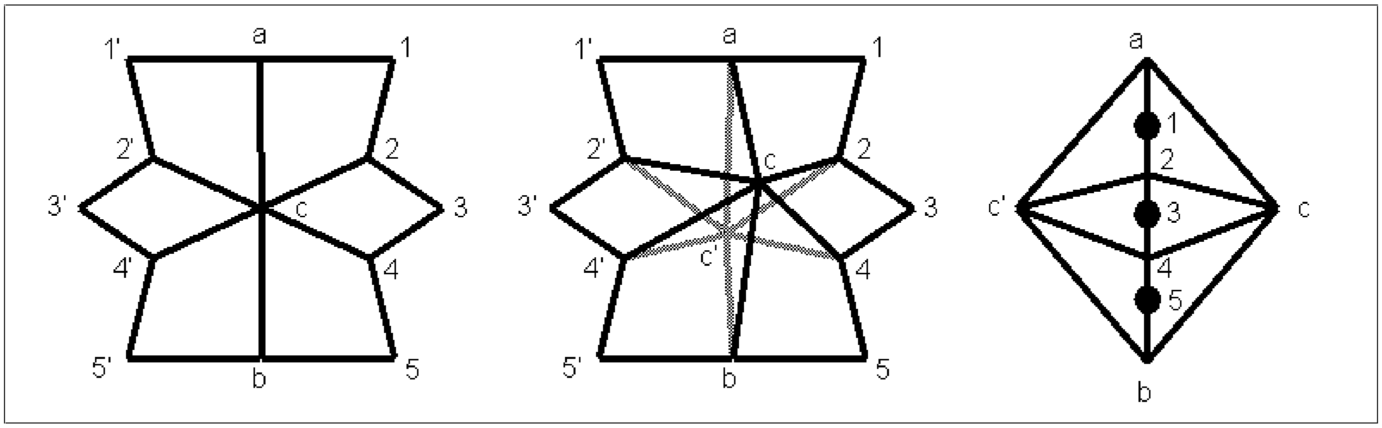

Figure 8. Folwell Starfish applied to six faces sharing a node.

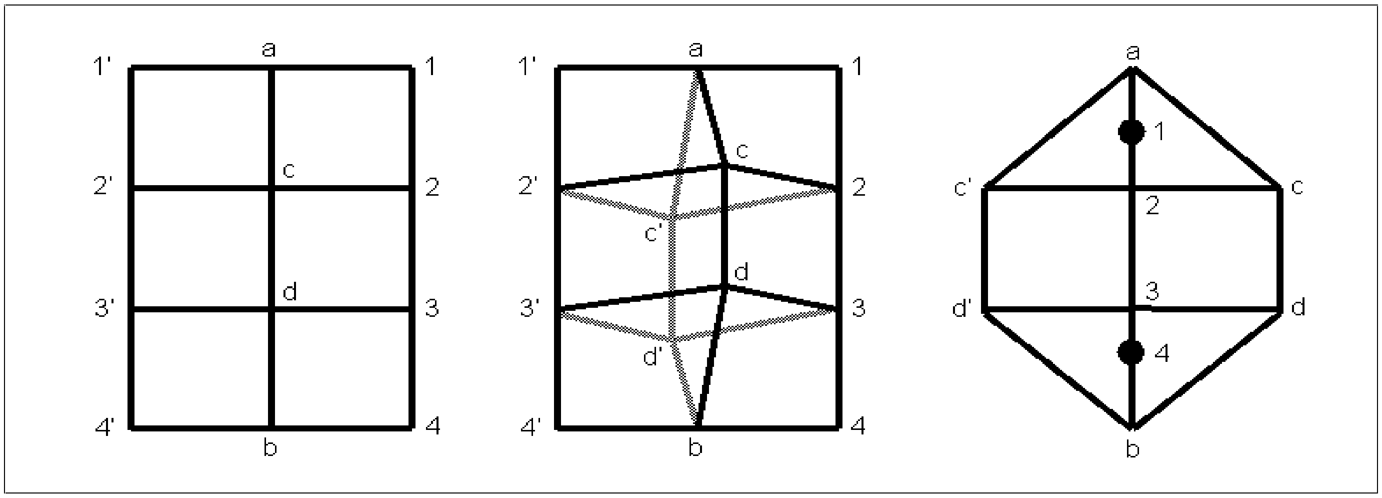

Figure 9. Folwell Starfish applied to six faces along a spine of three edges.

2.6 Dual Interpretation

Each primitive has an interpretation in the dual STC [3] of the hex mesh. Rotating three hexes locally moves a dual sheet so that it passes to the other side of the intersection of two other sheets. Inflating a hex ring moves two locally disjoint sheets so that they intersect (at the ring). Large-scale sheet moving can be created by repeatedly applying primitives. Collapsing two hexes merges two locally disjoint sheets, creating a topological handle connecting the two sheets. The Folwell Starfish merges sheets in a different way. Pillowing creates a new sheet. All known STC changes can be accomplished using the primitives in this paper, although arrangements may have degenerated hexes. In a T-HEX mesh, tet-mesh edge swaps can be simulated using the hex primitives.

3 Optimization

A mesh untangling objective function was strongly suggested by certain remarks in reference [6]. The proposed objective function referred only to simplicial elements and was not considered within the framework of global optimization. A similar objective function was independently discovered by P. Knupp and extended to non-simplicial elements with the present LDRD. By our definition, a mesh is untangled if it has no negative Jacobians at the corners of an element. Untangled meshes are highly important because they are a necessary condition for a mesh to be usable (a tangled mesh will likely cause physical assumptions such as positive energy to be violated during the course of numerical simulations). Tangled meshes rarely occur when simplicial elements are used but are frequent when advanced hex meshing algorithms are used on complicated geometries.

For each corner node of a tetrahedral mesh, we define a Jacobian matrix whose columns consist of the three edges that emanate from a node. Let α be the determinant of the Jacobian matrix. One can show that the determinant is invariant to the node at which it is computed, i.e., each tetrahedral element has a unique Jacobian determinant. Let the tetrahedral mesh be composed of M elements, with determinants α_m. The global objective function for mesh untangling that we propose is

$$f(A) = \frac{1}{2}\sum \left[|\alpha_m| - \alpha_m \right].$$

If the mesh is untangled, the determinants are all positive and the objective function attains its minimum value, zero. The sum is over all elements in the mesh. If some of the determinants are negative, then the objective function may be written as

$$f(A) = -\sum \alpha_m ,$$

where the sum includes only elements with negative determinants. For non-simplicial elements, one simply includes the determinants at all corners of the element. In this way, Jacobian determinants that reflect the “boundary terms” are included. Using the fact that the determinants are linear in their node positions, we were able to prove that the objective function is convex, thus a global minimum always exists. If the objective function at the minimum is non-zero, then an untangled mesh does not exist. Otherwise, the minimum is not usually unique since any untangled mesh will minimize the objective function. This is not a difficulty in practice because the goal is to obtain any untangled mesh. Optimization of the mesh condition number will then reduce element skew and aspect ratio.

Numerical experiments with this objective function suggested the following refinement to give users more control over the value of the minimum Jacobian determinant. Let $0 < \beta < 1$ be a user parameter and modify the untangle objective function to read

$$f(A) = \frac{1}{2}\sum \left\{ |\alpha_m - \beta| - (\alpha_m - \beta) \right\}.$$

The objective function is then minimized when the Jacobian determinant is greater than β. If β is zero, the original objective function is recovered. In this case scaled-Jacobians as small as 1.e-05 could be obtained, which potentially could cause the condition number optimizer to fail to give non-inverted elements. Condition number optimization worked well with untangled meshes for which β=0.01. If β=1, the feasible region for the objective function is generally empty, signifying that it is not possible to produce a mesh with all elements having scaled-Jacobians greater than one. In this case, the Untangling objective function acts somewhat like the "maximize-the-minimum-Jacobian" objective function we jointly proposed with L. Freitag. In general, to achieve automatic mesh untangling, β=1 is a good choice although finding the minimum for this case is slower than for β=0.

The mesh-untangling algorithm was first tried in two dimensions so that results could be easily visualized. Meshes with triangular elements are seldom tangled, so smooth scheme "randomize" was invoked to create tangled meshes. The untangle objective function readily untangled these meshes. Quadrilateral meshes generated using the paving algorithm are also rarely tangled. After tangling them with scheme randomize, the Untangler successfully untangled the paved meshes. A more challenging test problem is the Horseshoe geometry with a mapped mesh. The problem is difficult because Laplace smoothing of this mesh produces a tangled mesh in which many contiguous elements are inverted. Our original approach using a local mesh Untangler failed to untangle the Horseshoe mesh. The results suggested that the reason for the failure was that local information is insufficient to drag all the inverted elements over the zero-Jacobian line in the mesh. The global Untangler was then implemented and was able to untangle coarse Horseshoe meshes. We were unable to test the algorithm on fine meshes because it took too much CPU time; improving the efficiency of the Untangler has become a priority.

In all these test cases, it was known that an untangled mesh existed. A more interesting test case was created using scheme Hole on a region with complex outer boundary. The resulting mesh was tangled and it was unclear if an untangled mesh existed for the given node connectivity and fixed boundary points. Visually, it appeared that an untangled mesh did not exist. This was confirmed by the Untangler which, while it improved the minimum Jacobian determinants, failed to untangle the mesh (but failed gracefully).

We then turned to the untangling of 3D meshes. The algorithm rather easily generalizes to 3D, although some issues concerning data structures within CUBIT had to be sorted out. The Untangler readily untangled Mapped, Sub-mapped and Swept meshes. The latter is significant because one current time-to-mesh bottleneck is getting untangled meshes for some Swept volumes. The method was tried on the HexTet algorithm with Geode transition element. Most examples were untangled after meshing but one case was found for which Geode resulted in a negative Jacobian element. The Untangle objective function successfully untangled the mesh.

A major disappointment was the inability of the Untangler to achieve positive Jacobians for the Whisker Weaving meshes. Weaving generates connectivity for all-hex meshes given arbitrary surface meshes, but usually fails to give a mesh with positive Jacobians. Although the Untangler typically improved the minimum scaled-Jacobians of WW meshes from –0.90 to –0.010, the resulting meshes were still tangled. Because of the success of the Untangler in untangling meshes generated by methods other than WW, we conclude that the difficulty lies not with the Untangler, but with the pathological nature of the mesh connectivities generated by the current WW algorithm. Another important use of the Untangler is thus illustrated – that of a Mesh Connectivity Quality measure – where, if a given tangled mesh cannot be untangled with the Untangler, the connectivity is useless for meshing purposes.

The ability of the Condition Number objective function to improve the quality of T-HEX meshes was investigated in this LDRD. T-HEX meshes are created by first meshing a given geometry with tetrahedral elements, then subdividing each tetrahedron with four hexahedra, giving an all-hex mesh. Unfortunately, on complex geometries, the hexahedra thus generated are often of poor quality. The basic question we asked was, could the Condition Number optimizer improve element quality enough for analysts to use the meshes?

Because condition number works particularly well on tetrahedral elements, the optimizer was applied to T-HEX meshes before they were sub-divided into hexahedra. Five complex geometries were meshed: Gear1, Hook, A-Block, DistDuct and Foam5. For each geometry, three meshes were generated: (1) T-HEX with no smoothing, (2) T-HEX with $\ell 2$ Condition Number optimization, and (3) T-HEX with $\ell\infty$ Condition Number optimization. T-HEX failed on DistDuct and Foam5 geometries because negative Jacobians were generated on the boundary due to high geometric surface curvature. Optimization was not tried on these examples because the algorithm does not presently permit movement of boundary nodes. Results for the three remaining geometries are given in the table below.

	Mesh1	Mesh1	Mesh2	Mesh2	Mesh3	Mesh3
Geometry	Max Cond.	Min J	Max Cond.	Min J	Max Cond.	Min J
Gear1	8.1	0.093	5.1	0.094	4.3	0.088
Hook	10.6	0.078	5.4	0.086	6.5	0.089
A-Block	6.7	0.111	5,5	0.143	6.2	0.146

From these results we have two conclusions: (1) Condition Number optimization can substantially reduce the worst condition number in a mesh; (2) however, since ideally one wants condition numbers below 3.0 to avoid “sliver” elements, the connectivity of T-HEX meshes is poor enough that quality T-HEX meshes of complex geometries cannot be achieved via Condition Number (or other) optimization methods. We believe that, had a quality mesh with the given connectivity existed, Condition Number would have produced it.

4 Automatic Quality Improvement Algorithm

We researched selecting primitives by hand and smoothing in order to improve quality. The most successful case was applying the rotate three hexes primitive when four hexes shared a node. We also considered the case when a node had too many hexes attached, resulting in at least some hexes having a very small solid angle at the node. In some T-HEX meshes, some of the edges attached to such a high-valent node had three attached hexes. Rotating the three hexes around that edge reduced the number of hexes attached to the node by two, and improved quality somewhat. However, an edge flip in the original T-HEX mesh may have accomplished the same thing. We were pleasantly surprised that rotating three hexes sharing an edge often improved quality in Whisker Weaving meshes as well. Rotations were often helpful in a long column of dependent hexahedral elements.

In a T-HEX mesh, for two diced tets that shared a triangle, two rotations results in six hexes instead of eight. The hexes typically have large dihedral angles, due to the edges bounding the tets having too few hexes. Pillowing all of the six resultant hexes restored good hex valences to the boundary edges, but overall the quality was poorer than in the initial mesh.

On paper, pillowing looks promising for some "waist" nodes - nodes with high edge degree, where the edges are geometrically grouped in clusters. Pillowing the hexes surrounding the clusters of edges seems like it would improve quality: the valence of the node is reduced; and, after pillowing, smoothing should have more room to spread out the cluster of edges. In real-world meshes, the existence of a good pillow set is rare. Usually, the initial set of hexes has some convex features which would make the new hexes in the pillow layer have poor quality. Adding more hexes to the set can cover the convex features, but then new convex features arise.

We found one interesting example in which the most regularly connected mesh was not the highest quality mesh; mesh quality is a matter of both connectivity and geometry. The example consists of a length-four chain of triples of hexes sharing an edge. When the middle two triples are rotated, resulting in an extra hex between the first and second, and third and fourth triples, mesh quality is better. The reason is that nearby mesh constrains the shape of the boundary of the middle two triples, so that having one internal structure is better than the other; see Figure 10 and Figure 11.

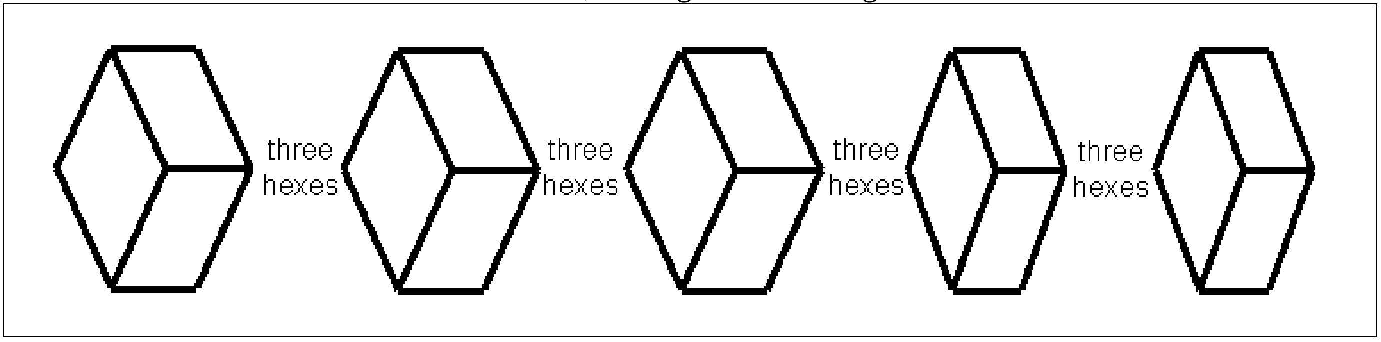

Figure 10. Face cross-sections between a chain of four triples of hexes with regular connectivity.

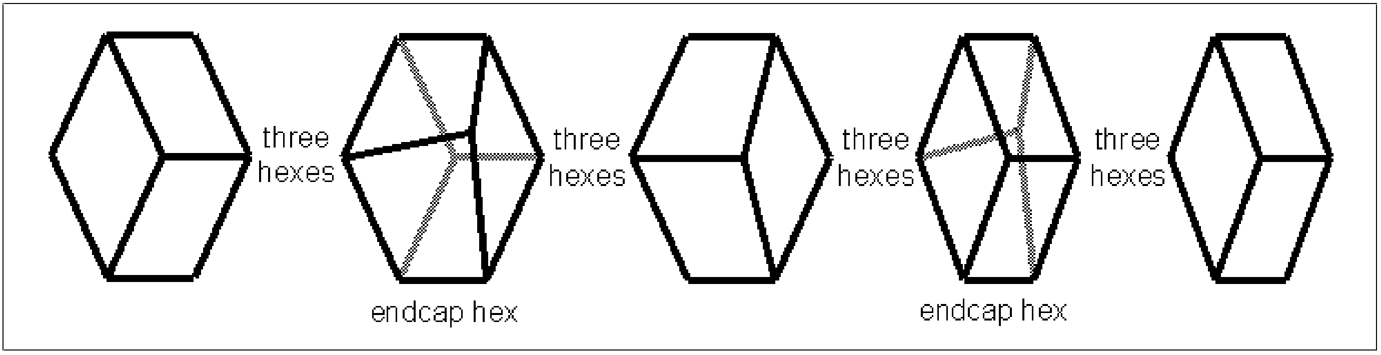

Figure 11. Face cross-sections between a chain of four triples of hexes with irregular connectivity: the middle two triples have been rotated. This internal mesh happens to have better quality due to the geometry of the boundary surfaces.

We wrote an automatic algorithm that applies the rotation primitives to degree-three edges next to flat hexes, and the pillowing strategy near nodes with a small solid angle. The control structure loops through all hexes of the mesh several times, evaluating the quality and shape of the hex. If a poor quality hex of the right type is found, we attempt to improve the quality locally. The current quality is evaluated in a neighborhood around the hex, and the connectivity of the neighborhood is saved. Next, we apply a primitive to change the local connectivity, and the mesh is smoothed. The new quality in a neighborhood near where the old hex was is evaluated, and the neighborhood is saved. If more primitives could have been applied to the original mesh, the original connectivity of the neighborhood is restored, and those primitives are tried as well. When all options have been tried, the connectivity with the best quality is retained. This process destroys some hexes, so care is taken to ensure that deleted hexes are never referenced.

Saving and restoring the neighborhood is currently accomplished using CUBIT's general mesh export and import capability, exporting and importing the entire mesh of the 3d region to a file. We also partially implemented a more efficient mesh rollback mechanism that stores just the local neighborhood affected by the primitives in a stack in memory. The mesh rollback mechanism saves information about the boundary and original connectivity inside a neighborhood, so that the original connectivity can be stitched back into place after the primitives change the connectivity inside the neighborhood. This general mechanism should be useful in future mesh-connectivity improvement work, but is not fully implemented at present.

5 Conclusions

There is a small but rich set of hex primitives that locally change the connectivity of a hex mesh. Global optimization can position the nodes of a hex mesh optimally, with respect to untangling the mesh and optimizing the condition number of the Jacobian matrix of a hex. We can locally reconnect and optimize a hex mesh and fix some quality problems. However, in many real-world examples we are unable to remove certain knots of poor quality. We suspect that this is due to fundamental constraints imposed by the connectivity and geometric embedding of the quadrilateral mesh of the region boundary. Constraints imposed by the self-intersections of dual curves have been studied,[3] but overall the conditions under which a quad boundary mesh admits a high-quality hex mesh are not well understood.

References

[1] S. A. Mitchell. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume, Proc. 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS ’96), Lecture Notes in Computer Science 1046, pages 465-476, Springer, 1996.

[2] S. A. Mitchell and T. J. Tautges. Pillowing Doublets: refining a mesh to ensure that faces share at most one edge, Proc. 4th International Meshing Roundtable, pages 231-240 (1995). Sand. Report 95-2301, Sandia National Laboratories, PO Box 5800, MS0847, Albuquerque, NM, 87185.

[3] N. T. Folwell and S. A. Mitchell. Reliable Whisker Weaving via curve contraction, Proc. 7th International Meshing Roundtable ’98, pages 365-378 (1998).

[4] S. A. Mitchell. The all-hex Geode-template for conforming a diced tetrahedral mesh to any diced hexahedral mesh, Proc. 7th International Meshing Roundtable ’98, pages 295-305 (1998).

[5] P.M. Knupp. Matrix Norms & the Condition Number, Proc. 8th International Meshing Roundtable ’99, pages 13-22 (1999).

[6] T. Coupez, Grandes transformations et remaillage automatique. PhD thesis, Ecole Nationale Superieure des Mines de Paris, November 1991.

Distribution:

MS-0188	Donna L. Chavez, LDRD Office, E04001
MS-0612	Review and Approval Desk, 4912 For DOE/OSTI
MS-0847	David R. White, 9226
MS-0847	Robert W. Leland, 9226
MS-0899	Technical Library, 4916
MS-9018	Central Technical Files, 8940 – 2
MS-0161	Patent and Licensing Office, 11500