SCI INSTITUTE
TECHNICAL REPORT

BioMesh3D: A Meshing Pipeline for Biomedical Computing

Michael Callahan, Martin J. Cole, Jason F. Shepherd, Jeroen G. Stinstra and Chris R. Johnson

UUSCI-2007-009

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

June 12, 2007

Abstract:

Biomedical computing applications often follow a computational pipeline: experimental data or image acquisition, mathematical modeling, geometric modeling (segmentation, mesh generation), material modeling, numerical approximation (finite element analysis, linear solvers, nonlinear optimization), visualization (of the geometric model, material model, and solutions), and validation. An important requirement of the numerical approximation and visualization methods is the need to create a discrete decomposition of the model geometry into a mesh. The meshes produced are used as input for computational simulation, as well as, the geometric basis for which many of the visualization results are displayed. Historically, the generation of these meshes has been a critical bottleneck in efforts to efficiently generate biomedical simulations which can be utilized in understanding, planning, and diagnosing biomedical conditions.

In this paper, we will outline a pipeline for more efficiently generating meshes suitable for biomedical simulations. Because of the wide array of geometries and phenomena encountered in biomedical computing, this pipeline will incorporate a flexible suite of tools that will offer some generality to mesh generation of biomedical models. We will discuss several tools that have been successfully used in past problems and how these tools have been incorporated into the suite of other tools. We will demonstrate mesh generation for a couple of example problems along with methods for verifying the quality of the meshes generated. Finally, we will discuss on-going and future efforts to bring all of these tools into a common environment to dramatically reduce the difficulty of mesh generation for biomedical simulations.

THE UNIVERSITY OF UTAH

BioMesh3D: A Meshing Pipeline for Biomedical Computing

Michael Callahan[1], Martin J. Cole[2], Jason F. Shepherd[3], Jeroen G. Stinstra[4], and Chris R. Johnson[5]

[1] Scientific Computing and Imaging Institute, Salt Lake City, UT
callahan@sci.utah.edu
[2] Scientific Computing and Imaging Institute, Salt Lake City, UT
mjc@sci.utah.edu
[3] Scientific Computing and Imaging Institute, Salt Lake City, UT
jfsheph@sci.utah.edu
[4] Scientific Computing and Imaging Institute, Salt Lake City, UT
jeroen@sci.utah.edu
[5] Scientific Computing and Imaging Institute, Salt Lake City, UT
crj@sci.utah.edu

Biomedical computing applications often follow a computational pipeline: experimental data or image acquisition, mathematical modeling, geometric modeling (segmentation, mesh generation), material modeling, numerical approximation (finite element analysis, linear solvers, nonlinear optimization), visualization (of the geometric model, material model, and solutions), and validation. An important requirement of the numerical approximation and visualization methods is the need to create a discrete decomposition of the model geometry into a 'mesh'. The meshes produced are used as input for computational simulation, as well as, the geometric basis for which many of the visualization results are displayed. Historically, the generation of these meshes has been a critical bottleneck in efforts to efficiently generate biomedical simulations which can be utilized in understanding, planning, and diagnosing biomedical conditions.

In this paper, we will outline a pipeline for more efficiently generating meshes suitable for biomedical simulations. Because of the wide array of geometries and phenomena encountered in biomedical computing, this pipeline will incorporate a flexible suite of tools that will offer some generality to mesh generation of biomedical models. We will discuss several tools that have been successfully used in past problems and how these tools have been incorporated into the suite of other tools. We will demonstrate mesh generation for a couple of example problems along with methods for verifying the quality of the meshes generated. Finally, we will discuss on-going and future efforts

to bring all of these tools into a common environment to dramatically reduce the difficulty of mesh generation for biomedical simulations.

1 Introduction

Advanced techniques in biomedical computing, imaging, and visualization are changing the face of biology and medicine in both research and clinical practice. The goals of biomedical computing, imaging and visualization are multifaceted. While some images and visualizations facilitate diagnosis, others help physicians plan surgery. Biomedical simulations can help to acquire a better understanding of human physiology. Still other biomedical computing and visualization techniques are used for medical training. Within biomedical research, new computational technologies allow us to "see" into and understand our bodies with unprecedented depth and detail. As a result of these advances, biomedical imaging, simulation, and visualization will help produce exciting new biomedical scientific discoveries and clinical treatments.

Biomedical simulations are dependent on numerical approximation methods, including finite element, finite difference, and finite volume methods, to model the varied phenomena of interest. An important requirement of the numerical approximation methods above is the need to create a discrete decomposition of the model geometry into a 'mesh'. The meshes produced are used as input for computational simulation, as well as, the geometric basis for which many of the visualization results are displayed. Historically, the generation of these meshes has been a critical bottleneck in efforts to efficiently generate biomedical simulations which can be utilized in understanding, planning, and diagnosing biomedical conditions.

In this paper, we will outline a pipeline for more efficiently generating meshes suitable for biomedical simulations. Because of the wide array of geometries and phenomena encountered in biomedical computing, this pipeline will incorporate a flexible suite of tools that will offer some generality to mesh generation of biomedical models. We will discuss several tools that have been successfully used in past problems and how these tools have been incorporated into the suite of other tools. We will demonstrate mesh generation for a couple of example problems along with methods for verifying the quality of the meshes generated. Finally, we will discuss on-going and future efforts to bring all of these tools into a common environment to dramatically reduce the difficulty of mesh generation for biomedical simulations.

2 Motivation

All of the tools discussed throughout the remainder of this paper have been developed in or integrated into the SCIRun Problem Solving Environment (PSE) [1, 15, 27]. SCIRun is an open source problem solving environment that allows

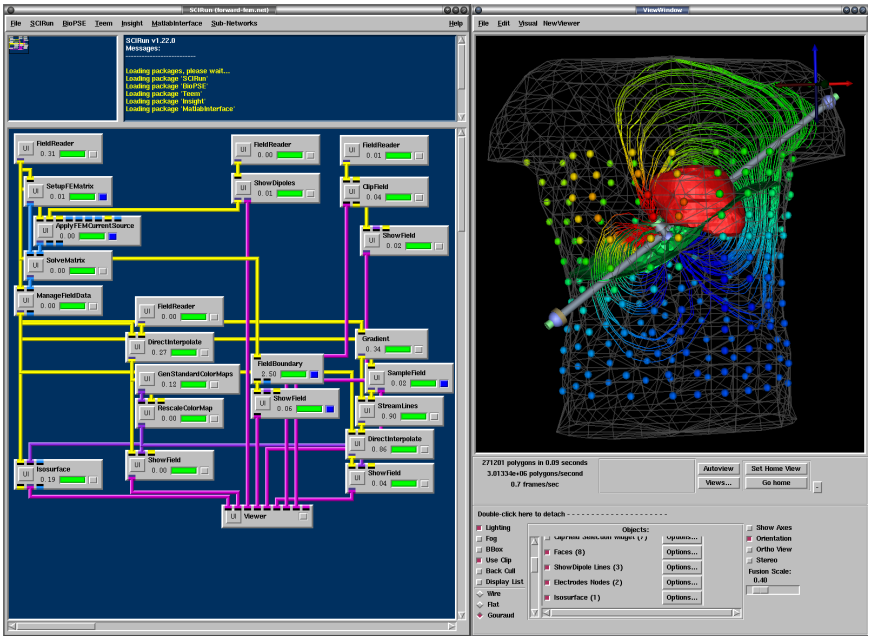

Fig. 1. The SCIRun PSE showing the module network (middle), the visualization window (right). Researchers can select UI (user interaction) buttons on many of the modules that allow control and feedback of parameters within a particular module (left).

the interactive construction, debugging, and steering of large-scale, typically parallel, scientific computations (available at www.sci.utah.edu). SCIRun provides a component model, based on dataflow programming, that allows various computational components and visualization components to be connected together. SCIRun can be envisioned as a "computational workbench," in which a scientist can design and modify simulations interactively via a component-based visual programming model. SCIRun enables scientists to modify geometric models and interactively change numerical parameters and boundary conditions, as well as to modify the level of mesh adaptation needed for an accurate numerical solution. As opposed to the typical "off-line" simulation mode - in which the scientist manually sets input parameters, computes results, visualizes the results via a separate visualization package, then starts again at the beginning - SCIRun "closes the loop" and allows interactive steering of the design, computation, and visualization phases of a simulation. An example biomedical simulation utilizing the SCIRun environment is shown in Figure 1.

There have been several over-arching goals that have governed much of the development of the meshing pipeline presented in this paper, especially in

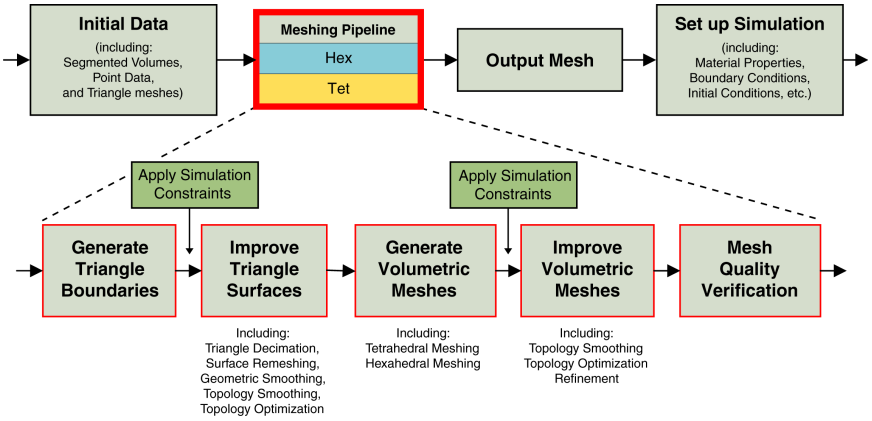

Fig. 2. The meshing pipeline in SCIRun.

regard to development of these tools in the SCIRun environment. Specifically, all of the tools in the meshing pipeline (as developed or integrated in SCIRun) must be open-source, flexible enough for broad application, and fit into a pipeline with a broad suite of other tools.

3 Pipeline

Figure 2 gives a broad overview of the meshing pipeline implemented in SCIRun for preparing meshes for biomedical simulation. Because generality is desired in this pipeline, that is, the pipeline should be applicable to a broad array of possible biomedical simulations, the pipeline incorporates a suite of tools that can be used flexibly and interchangeably among the various steps within the pipeline. Each of the tools will be discussed in greater detail in the next section, and a general overview of the entire pipeline will be outlined in this section.

The choice of numerical method utilized in a biomedical simulation is often based upon the anticipated and acceptable level of error, the applicability of the method for geometic modeling a given phenomenon, and the amount of time required to prepare a model utilizing the chosen method. Two types of geometric elements are commonly used for most numerical methods: tetrahedral elements or hexahedral elements. Because of differences in the generation methods for each of these element types, we will discuss these meshing pipelines separately in this section.

Both the tetrahedral and hexahedral meshing pipelines typically start with either a stack of images, or alternatively, a three-dimensional grid of scalar values. The image stacks, or scalar grids, are often referred to as 'volumetric' data. To ensure better geometric accuracy, the volumetric data is commonly

categorized or 'segmented' into a smaller subset of discrete values that focuses the boundaries of features residing within the data. The volumetric data can also be viewed as a regular hexahedral grid, or mesh, with each node or hexahedra within this mesh containing one of the scalar values associated with the original data.

3.1 Tetrahedral Meshing Pipeline

The first step in the tetrahedral pipeline is to extract the boundary surfaces between the different segmentations. Extracting boundaries within the segmented data can be done in several ways. The first commonly used method is to run an isosurfacing algorithm, like Marching Cubes [24], to obtain a set of triangle meshes representing the boundaries between features within the volumetric data. While Marching Cube algorithms emit smoother models, in many cases these algorithms don't preserve the space partition property and require that each sub volume is fully embedded inside another sub volume, which often means that segmentations have to be altered to observe the constraints of the Marching Cubes algorithms. Another alternative is to emit a quadrilateral face between any two voxels with a different segmentation value. This results in a stairstep model of all the segmentation boundaries with some nice properties. There are no holes in the new geometry, all the boundaries are closed, and every closed piece contains a categorization.

Once the feature boundaries have been established from the volumetric data, and a triangular mesh has been created describing these boundaries, the next step in the pipeline is to optimize the mesh on the boundary to maximize the mesh quality and to embed constraints on the location of nodes (for example to embed the locations of electrodes). Because of varied constraints on the resulting boundary mesh, we have given the pipeline a suite of tools to aid in the boundary optimization. These tools include surface remeshing algorithms, mesh topology modifiers (including decimation), and geometric and mesh smoothing algorithms. These algorithms will be discussed in more detail in the next section.

Once a suitable boundary mesh has been established, the final steps in the pipeline are to create a tetrahedralization or other volumetric mesh and ensure that the mesh will be suitable for the subsequent analysis. The SCIRun pipeline currently has methods for generating the tetrahedral mesh, volumetric smoothing, and mesh refinement. The final step is to use mesh verification techniques to ensure that the meshes generated contain elements of adequate quality for computational analysis.

3.2 Hexahedral Meshing Pipeline

In addition to the tetrahedralization pipeline, SCIRun also contains support for handling hexahedral elements directly. There are two main methodologies utilized in SCIRun for hexahedral meshes: hexahedral meshes with stairstepped boundaries and hexahedral meshes with smooth boundaries. Because

the hexahedral meshes can be generated directly from the segmented model, resolution of these meshes is often greater than needed, or desired. Therefore some data reduction is often necessary. SCIRun contains algorithms for resampling this data at more coarse hexahedral representations. Additionally, a coarse lattice can also be built using resampling, with some localized refinement techniques to recover data in areas of importance back down to the level of the original data, or levels beyond the resolution of the original data which is sometimes needed for certain simulations that are constrained by other factors than the volumetric data.

The hexahedral pipeline is finished in a similar fashion to the tetrahedral pipeline with volumetric smoothing, and refinement techniques, followed by mesh verification to ensure the resulting mesh is suitable for subsequent analysis. These techniques and algorithms will also be discussed in more detail in the next sections.

4 Specific Tools in the Pipeline

In this section, we will describe in more detail the various tools which have been implemented to date in the SCIRun meshing pipeline. The format for this section will roughly follow the pipeline order described above. Where it makes sense, the tools are meant to be interchangeable between tetrahedral and hexahedral elements.

4.1 Surface Mesh Improvement Tools

Remeshing

Because the triangle meshes resulting from a Marching Cubes algorithm typically contain poor quality elements, sliver triangles, and/or dramatic size variations, it is often desireable to remesh in an effort to obtain a better set of triangles. Additionally, triangle meshes which are smoother and more uniform in size have several advantages for reduced tetrahedral element counts and higher overall element quality.

Surface remeshing is an active area of on-going meshing research [2]. While there are several tools available from the community, a tool that has demonstrated success for triangle remeshing, and was readily available to us is Afront (Advancing Front (Re)Meshing Algorithm) [28]. Afront is currently limited to isosurface boundaries (multi-material, or non-manifold, boundaries are not allowed), therefore Afront is utilized to isosurface individual materials, and provides control in offering more uniformly shaped triangles and smoother surfaces.

When creating surfaces for adjacent materials intended as input to tetrahedralization, isosurface values are chosen to prevent materials from overlapping. Overlapping surfaces lead to failure during the tetrahedralization step.

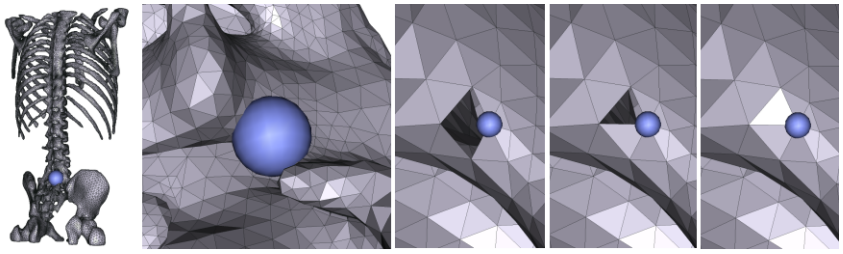

Fig. 3. A series of images showing the process for finding and filling holes in a set of triangles.

In practice some iteration is needed to find isovalues that provide non overlapping interior surfaces. Alternatively surfaces that show some kind of overlap can be intersected with each other forming small additional compartments which can be assigned later to one of the neighboring volumes.

Elemental Cleanup

In many cases, the bounding triangle meshes often contain small errors of detail which result either from algorithmic issues, improper segmentations, or unhandled exceptions. Because small errors in previous steps can lead to a mesh with a few problems, it can be desireable to have a small set of tools available for use when all other methods fail. We have developed some very basic hand-editing tools for this purpose.

In all the previous steps we hope to eliminate the need for hand editing, but a good user interface that allows for some hand-editing gives power to the process and makes some dire situations tractable. Tools used to tetrahedralize our input surfaces typically report problem areas that prevent it from completing. Finding the reported area and fixing it is the purpose of these cleanup tools. One example of this process can be seen in Figure 3. Using the probe widget (the blue sphere in Figure 3) we can locate the problem. We can then focus the view on that area and visually inspect the problem area. Command line tools are provided for deleting faces, adding triangles given specific node and face indices viewable in SCIRun. The development version of BioMesh3D has added a more user friendly selection mechanism, in which a set of faces, can be targeted for deletion, and then by selecting nodes in order faces can be added back. This allows models to more quickly be patched and made ready for the tetrahedralization.

Decimation

Triangle surface models created by isosurfacing volumes are almost always more dense than they need to be. However doing naive decimation can be

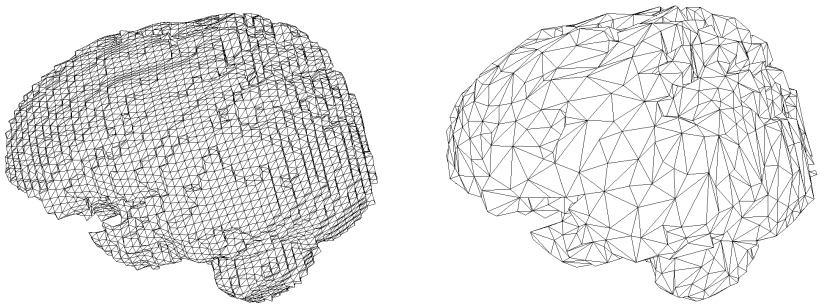

Fig. 4. Example of decimation of triangles from a mesh of a brain. The original mesh is shown at left, while the image on the right shows the mesh after decimation.

problematic for biological models where topological and geometric errors have a bigger effect on diagnostic outcome. The decimation method needs to be robust enough to preserve the topological properties of the model. Decimation should not change the topology, nor should it create degeneracies. The ideal method would also preserve the interior volume of the resulting segmentations. After an overview of the decimation literature [33] we chose a quadric based edge collapse method [12] adapted to preserve more topological constraints than is usual.

The SCIRun triangle decimator computes the quadric equation for the plane associated with each triangle. Then the quadrics are summed up for each node and an error metric is computed for each edge. Next the edges are collapsed in order of least error. Quadrics allow for the new metrics after a collapse to be computed quickly and for the new collapsed point to be picked optimally. In addition any collapses that would result in a topological error are discarded. An example of decimation on a mesh of a brain model is shown in Figure 4.

This algorithm works very well for our needs. However the filter for rejecting non-topological collapses could use some work to make it more robust given non-manifold surfaces as input meshes.

Geometric Smoothing

In the process of generating isosurfaces, undesired geometric features are a common occurrence. FairMesh is a geometric smoothing module in SCIRun designed to smooth a mesh based on Gaussian filtering, without shrinking the interior volume enclosed by the mesh. This module uses an algorithm developed by Taubin [34], and uses a signal processing approach to surface smoothing, reducing the problem to application of a low pass filter to the surface signal. The algorithm moves nodes towards the weighted average of its neighbors, first with a positive scale factor, then with a negative scale factor

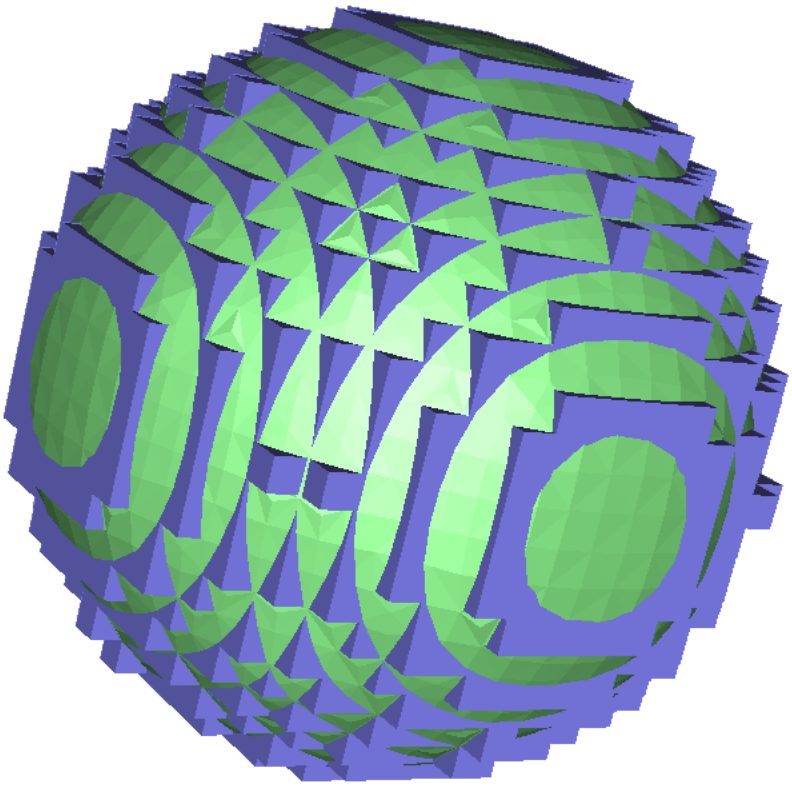

Fig. 5. The blue stair stepped mesh is the input to FairMesh, the green is the smoothed output.

that avoids the shrinking expected with Gaussian filtering. The algorithm executes efficiently especially for the simplest weighting scheme appropriate for the stair stepped meshes we obtain from Marching Cubes over medical image data. An example of this process is shown in Figure 5 where an initial stair-stepped mesh is geometrically smoothed resulting in a mesh with identical topology, but a smoother geometric boundary.

Smoothing with equal edge costs tend to equalize the lengths of the edges. While this is desirable for producing input to the tetrahedralization phase, it does not produce good results for meshes that want to apply a texture map. Desbrun [6] weights approximate the curvature normal, and tends to not move a vertex when its neighboring faces are coplanar. We provide this weighting scheme as an additional option in the FairMesh module. The curvature normal

based weights need to be recomputed at each iteration step, causing this algorithm to be computationally more expensive.

4.2 Volumetric Mesh Generation Tools

Currently in the research community, there are only a few tools for mesh generation available for use. One of the basic goals in the development described in this paper is to offer a set of tools that can be utilized and made readily available for completing mesh generation tasks. Unfortunately, volumetric mesh generation is still difficult enough that few tools with adequate track records of success and that are easily licensed for research purposes are commonly available.

Tetrahedral Mesh Generation Tools

Robust, freely available, open source tetrahedral mesh generation tools are highly desired but not commonly available to users. In this section, we will discuss our efforts to incorporate two tetrahedral mesh generation libraries into the SCIRun meshing pipeline. The major trade-offs with these tools are with respect to the quality of the resulting meshes, and the ease of licensing the product.

The current approach of generating isosurfaces as boundaries for a tetrahedral mesh generation can be problematic. In the current pipeline, the selection of isosurface values that guarantee non-overlapping surfaces gives us an artificial gap between adjacent surfaces that gets mapped to neither material. This gap gets filled with numerous small tets, of some generic material, which artificially inflates the tetrahedra count, and increases the error in finite element modeling with the artificial separating material. Additionally, this approach produces models that are visually realistic, but not computationally ideal. Therefore, we would like an isosurfacing step that produces a single surface wall separating materials in the volume. This has led us to a new pipeline approach that we describe in a later section.

CUBIT

CUBIT [5] is a full-featured software toolkit for geometric model generation and robust generation of two- and three-dimensional finite element meshes developed at Sandia National Laboratories. The main development goal for CUBIT is to dramatically reduce the time required to generate meshes, particularly large hex meshes of complicated, interlocking assemblies. The CUBIT toolkit also provides libraries to many useful meshing algorithms [4]. In particular, CUBIT utilizes the state-of-the-art GHS3D tetrahedral meshing library [25] for generation of tetrahedral meshes. For users that are able to obtain a license to CUBIT, the SCIRun meshing pipeline can be built to enable tetrahedral mesh generation using some of the tools provided by the CUBIT framework.

TetGen

TetGen [31] is an open source solution to generating tetrahedral meshes from our triangle surface input. Each surface generated from the isosurfacing step is combined into a single mesh and provided as input to TetGen. If the model has holes or self intersecting faces TetGen will fail and point out where such problems occur in the mesh. In practice we iterate over the previous steps until we have useable input.

TetGen can refine specified areas more or less densely. Separate regions can be tagged and remapped to the original material types. The entire volume will be tetrahedralized according to the input criteria. TetGen is a great tool for incorporation into a research meshing pipeline because of it's ready availability and the flexibility of the interface, i.e. the ease with which additional points and surfaces can be added as additional constraints is particularly useful for generating meshes for doing simulations. The quality of the output mesh is typically lower than meshes generated by some commercial tetrahedral packages [25], but because of fewer licensing restrictions it is a convenient alternative.

Hexahedral Mesh Generation Tools

In addition to the grid re-sampling methods for hexahedral meshes mentioned earlier, a method for generating conformal hexahedral meshes has been developed in SCIRun as outlined by Shepherd [30] and briefly described in this section.

Given an existing hexahedral mesh and a triangulated surface representing the shape of a new layer of hexahedral elements to be inserted into the mesh, the general methodology in SCIRun is the following:

1. *Locate all of the hexahedra that are intersected by one or more triangles in the triangle mesh.* A kdtree containing all of the triangles is utilized to improve the efficiency of this search. If there is a triangle in the vicinity of a given hexahedron, each edge of the hexahedron is tested for intersection with the triangles in the region. Each of the intersected hexes is marked as being intersected.
2. *Separate the hexahedra into three groups: Side1, Side2, and Intersected.* Starting with an unmarked hexahedron (i.e., a nonintersected hexahedron from the previous step), use a flood-fill algorithm to group all of the hexahedra that are connected to this hexahedron and not marked (i.e., intersected by a triangle). This group will be known as 'Side1'. All of the marked, or intersected, hexahedra are placed in a second group, known as 'Intersected', and the remaining hexahedra are placed in a third group, known as 'Side2'. An example of this process is shown in Figure 6 where a hemispherically-shaped triangle mesh is place in a hexahedral grid. The boundary of the triangle mesh is shown in black, and the 'Intersected'

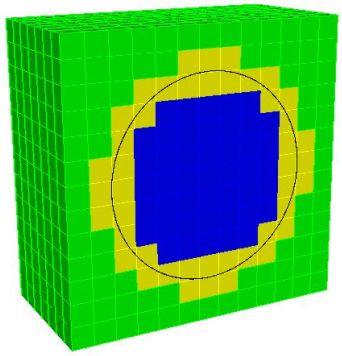

Fig. 6. A hemispherically-shaped triangle mesh (the boundary of the triangle mesh is shown in black) is placed in a hexahedral grid. The hexahedra intersected by the triangle mesh are shown in yellow, while 'Side1' is drawn in green and 'Side2' is shown in blue.

hexes are drawn in yellow. 'Side1' is drawn in green and the remaining hexahedra are placed in 'Side2' (shown in blue). The algorithm for detecting intersecting triangles and separating the hexes into these three groups is explained in further detail in [29].

3. *Collate the 'Intersected' hexahedra with either 'Side1' or 'Side2' and insert two new layers of hexahedral elements between these two groups of hexahedra.* The 'Intersected' hexahedra are subsequently added to either 'Side1' or 'Side2', and two layers of hexahedra are added around these two groups. For the example highlighted in Figure 6, depending on which side the intersected hexahedra are grouped, one of the meshes shown in Figure 7 will result.
   The new layer of hexahedral elements is inserted by (refer to Figure 8):
   a. First, determining the quadrilateral boundary between the two sides of the mesh,
   b. separating the two meshes by shrinking the elements at this interface,
   c. then, for each node on the separated boundary, project a new node to the triangle mesh. A map to each node is retained by both sides of the mesh, and once all of the projected nodes have been created on the boundary, the hexahedral connectivity for the two new layers can be developed by using the quadrilaterals on the interface boundary from both sides and the map to each of the newly projected nodes.
4. *Export the two new groups of hexahedra.*

When inserting the new elements, the shrinking process often forces some element inversion, so it is necessary to smooth the mesh to obtain the mesh quality desired. Additionally, the projection of the nodes to the triangle mesh

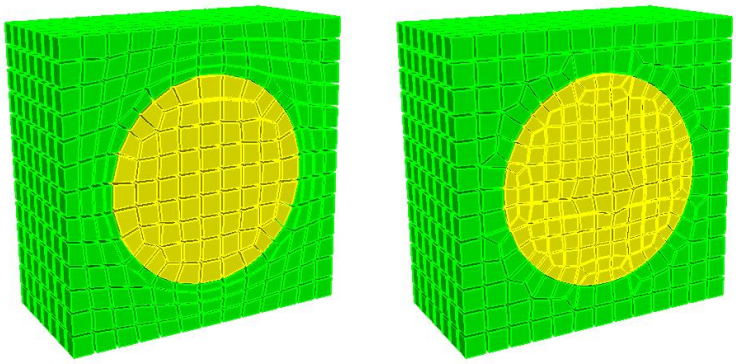

Fig. 7. Slightly different meshes result depending on which side the intersected hexes are grouped. The image on the left shows the resulting mesh if the intersected hexes are placed with Side1's hexes, while the image on the right has the intersected hexes being grouped with Side2.

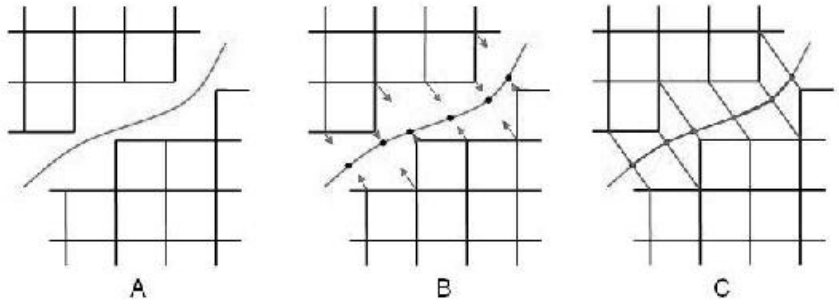

Fig. 8. Image A shows the shrunken hexahedra with the triangle mesh shown in between the hexahedra. Image B shows a newly projected node to the triangle mesh for each node on the boundary of the shrunken mesh (note that a single node on the triangle mesh corresponds to one node on each of the shrunken boundaries). Image C shows the newly created hexahedron by mapping the quadrilaterals on the boundary to the appropriate nodes (recently projected) on the triangle surface mesh.

often results in nonuniform sizing of the quadrilateral elements on the boundary. This is also remedied using a smoothing operation.

4.3 Volumetric Mesh Improvement Tools

We have utilized the TSTT Mesh Quality and Improvement Toolkit (MESQUITE) library of smoothing algorithms [26, 3] to accomplish the optimization of the meshes in the SCIRun Meshing Pipeline. Within the MESQUITE class of mesh smoothers we have access to both *untangling* smoothers (a *tangled* mesh contains elements that are inverted, or have a negative Jacobian value) and a wide range of optimization algorithms for untangled meshes. We have utilized an *inverse mean ratio* smoother (as described by Knupp [22]), that incorporates an L2-norm template with guarantees that (1) the mesh will remain untangled if the initial mesh is untangled, and (2) the average value of the inverse mean ratio will either stay the same, or be decreased.

While there are a wide variety of smoothing algorithms available, including Laplacian [13, 9], centroidal area [16], Winslow [19], angle-based [37], and many others, we will highlight three algorithms that have been implemented within SCIRun. These smoothers include Laplacian smoothing, a hybrid smoothing/optimization algorithm known as Smart Laplacian [10], and a mesh optimization algorithm for improving the 'shape' metric, called Shape Improvement Optimization [22]. In SCIRun, these smoothing/optimization algorithms are available for smoothing quadrilaterals or hexahedral meshes (as well as triangle and tetrahedral meshes). These operations can be performed on a section of the mesh or the entire mesh can be optimized. The current implementation allows for setting up constraints on the smoothing operations so nodes that need to be in a certain location for simulation purposes can be pinned to that location.

Laplacian Smoothing.

Of all the available mesh smoothing algorithms that are currently available, the most common algorithm is known as Laplacian smoothing. The easiest way to visualize Laplacian smoothing is to consider each edge attached to a node in the mesh as a spring. When the lengths of each of the edges attached to the node are identical, the spring force is balanced. When they are not equal the node is pulled towards the springs with the greatest force. Because this process iterates over each of the nodes, the spring forces may be constantly changing after each iteration, while the total force in the system should be dimished with each iteration.

The advantages of Laplacian smoothing are that the algorithm is easy to implement and is computationally efficient. However, it also has several disadvantages, including that it may generate inverted elements, element shapes are not necessarily optimized, and features may not be preserved if too many iterations are performed. Despite these disadvantages, Laplacian smoothing is a very powerful mesh optimization tool, especially when coupled with other optimization-based smoothing techniques [10].

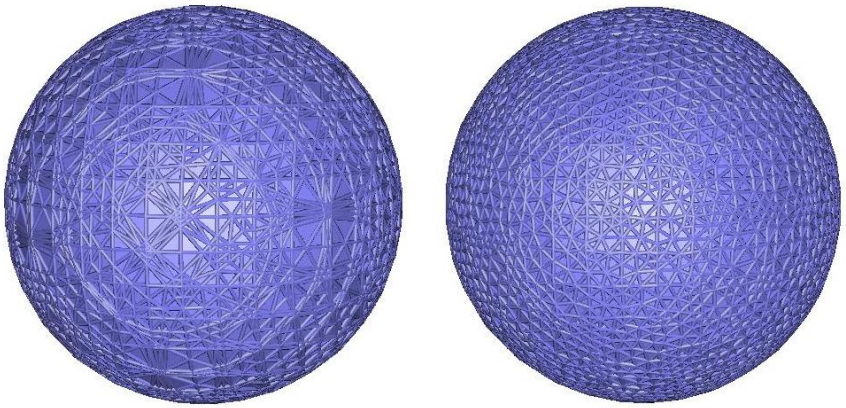

Fig. 9. Example of smart Laplacian smoothing (from the MESQUITE toolkit implemented in SCIRun) on a tetrahedral mesh. The image on the left shows the mesh before smoothing, and the images on the right is the same mesh after smoothing.

Smart-Laplacian Smoothing

Utilizing the guarantees of the L2-norm template available in MESQUITE, Laplacian smoothing can be augmented to prevent element inversion. Using this 'smart' version of Laplacian' smoothing we can assume that if the nodes on the boundary of the mesh are fixed in place while the interior nodes are free to move during optimization, then the smoothed mesh will have either the same or better quality upon completion of the optimization. We, therefore, can run the smoother until the optimization converges with the given geometry and mesh topology for the Laplacian criterion. While it may be possible to subsequently improve the quality of some of the individual elements, it would be done at the expense of the quality of the adjacent elements. Therefore, we have some confidence that the average element quality for the given mesh topology and geometry is optimal, and only modifications to the mesh topology can be utilized to gain additional average quality improvements in the reported meshes. Figure 9 visually demonstrates the difference in mesh quality after using a smart Laplacian smoother for surface meshes generated using a Marching Cubes approach.

Mesh Untangling

A mesh untangling algorithm [18] uses node movement to remove nonconvexities of elements within a mesh. The Jacobian matrix is calculated for each node with respect to an element. For each node in a tetrahedron or hexahedron, there are exactly three 'neighbor' nodes connected via an edge in the element. For a single node, the Jacobian matrix is defined as:

$$A_0 = \begin{vmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{vmatrix}$$

The minimal determinant of these matrices for each node of an element is known as the 'Jacobian' of the element [21]. By allowing nodal movement for each of these elements, an optimization problem can be formulated to maximize the following objective function (other similar functions have also been utilized):

$$f(A) = 0.5 * \sum(|\alpha_m| - \alpha_m)$$

where α_m is the determinant a single Jacobian for a mesh with m elements.

If the mesh is untangled then the summation reaches a maximum value of zero. It is common to use local optimization algorithms, such as conjugate gradient methods, to obtain a solution to the untangling optimization problem. However, because the untangling problem can be nonconvex, it is possible to reach a local maxima without obtaining an optimal solution. This is an ongoing and challenging research area [18, 35, 23, 11].

Mesh Optimization

In addition to traditional mesh smoothing techniques, there has been a tremendous amount of research conducted in optimization-based smoothing. Traditional smoothing methods work heuristically and can create invalid meshes or elements containing worse quality than the original mesh. In contrast, optimization-based methods work to optimize a metric value associated with each of the mesh elements. Common metrics for optimizing include shape [22], condition number [18], Jacobian [20], and mean ratio [8, 21]. While these methods can be extremely effective at maximizing metric values, they can also be very computationally expensive. To reduce the computational expense, hybrid algorithms have been proposed which combine some of the speed advantages of the traditional methods while maintaining the quality improvement guarantees of the optimization-based techniques [10]. As mentioned earlier, a 'shape' optimization algorithm has been implemented in SCIRun using the MESQUITE toolkit.

Refinement

SCIRun contains two different methods for refining hexahedral grids. The first is based upon Harris [14] and has been modified to use the four different hexahedral templates presented in that work within a Marching Cubes style lookup table scheme. This means that it is very fast to compute a refinement as it is O(N) over the original mesh size. The most dense template is a regular 27:1 cut of each hex and thus nicely preserves the shape of the original hexes.

However the four templates presented in that work can only be used to refine a convex region of a hexahedral mesh. As a preprocess the refinement region is expanded outward until it is convex. This can result in a much larger area of refinement than desired, particularly if the refinement area is sparse. For example a wire electrode in a torso can cause the convex region to cover many more hexahedra than would otherwise be refined.

SCIRun also contains a novel hexahedral refinement scheme based upon recursively dicing up the corners of hexahedra around nodes marked to be refined. This is the one node template from the Harris method applied over and over until all the desired refinements have been made. This allows an arbitrary refinement to be made without a convex region requirement and thus offers much better results for sparse refinement areas. However the most dense template in this case is a 49:1 cut and results in hexahedra with a less regular cut than the convex scheme. If the convex region is less than two times as big as the non-convex region then the Harris method should offer better results.

4.4 Mesh Verification Tools

Significant research has gone into defining metrics for judging the quality of elements in a mesh. Element quality criteria are generally agreed upon standards for acceptance of a mesh for simulation purposes. Verdict [36] is a software library containing a comprehensive suite of mesh quality metrics for evaluating the quality of hex, tet, quad and triangle finite elements. The SCIRun Meshing Pipeline provides a module with an interface to the Verdict library for calculating and evaluating mesh quality using standardized mesh quality metrics.

5 Results

In this section, we demonstrate two example meshes generated using the tools described previously. The first example is a mesh of a pediatric torso using the tetrahedral mesh generation pipeline. The second example is a skull and cranial model using the hexahedral meshing pipeline.

Because the Jacobian matrix for a mesh element is used to map the element to a reference element in solution space in most numerical methods (particularly in finite element analysis [7]), a poor element Jacobian may result in increased error in the solution. To ensure proper element quality control within the meshes presented in this section, we will display all quality results utilizing the determinant of a scaled Jacobian matrix as calculated by the Verdict [36] library for each of the elements within the mesh.

5.1 Pediatric Torso

The goal in processing the pediatric torso was to use tools that already existed and see if we could build a finite element mesh from a segmented volume. We also desired to gauge the quality of such a mesh while also comparing how well the different tools compared to each other.

The input to the pediatric torso pipeline was a segmented volume image data set (i.e., a NRRD file [17] 512x512x111, consisting of 11 different materials categories). Each material in the segementation was separated into a separate volume prior to isosurfacing. Using the Teem tools, each of the separate material files were resampled into a 111x111x111, by filtering down in x and y to get a uniform size in each dimension.

Each of the materials included in the final mesh were isosurfaced using either Afront or a standard Marching Cubes algorithm (Figure 10 illustrates the results of both methods of isosurface generation). For this model, we chose to include the torso, bone, lung, and heart segments. The resulting surfaces could not overlap each other in any way, and had to be 'water-tight' prior to generating the volumetric elements interior to the surface. Therefore isosurface values were chosen such that the resulting surfaces would not overlap. (Ideally, the lung and heart would have portions of their surfaces that sharing a single interface surface. However, if both were isosurfaced at a common value the two surfaces along this area would overlap resulting in a failure during volumetric mesh generation. To prevent this failure, isovalues were chosen to be different and the heart and lungs were not directly interfaced.)

The sets of surfaces generated from Afront and Marching Cubes were separately configured as input for TetGen and CAMAL using the SCIRun interface. The original intention was to create four tetrahedral meshes: 1. Afront surfaces to TetGen volumes, 2. Afront surfaces to CAMAL volumes, 3. Marching Cubes surfaces to TetGen volumes, and 4. Marching Cubes surfaces to CAMAL volumes. However, using the Marching Cubes algorithm, we could not generate a valid model containing the torso, bone, lung and heart without overlap. After generating a tetrahedral mesh using the Marching Cubes surfaces of the torso, bone, and lungs and reviewing the resulting mesh generation timing and quality, it was obvious that the Marching Cubes surfaces would not provide adequate results without significant work to improve the surface meshes. Therefore, only a single model was generated using the Marching Cubes approach. These results are shown in Table 1.

The higher surface quality from the Afront generated surfaces resulted in shorter tetrahedral mesh generation timings, as well as overall higher quality tetrahedral elements. Table 2 lists several of the resulting quality metrics for the mesh generated by CAMAL using the GHS3D tetrahedral mesh generator. In figure 11 a histogram is given of the scaled Jacobian metric per element, showing that only a small portion of the elements has a low quality and that most elements are properly shaped. The final mesh contains 3,415,236 tetrahedra, and a cut-away view of this mesh is displayed in Figure 12.

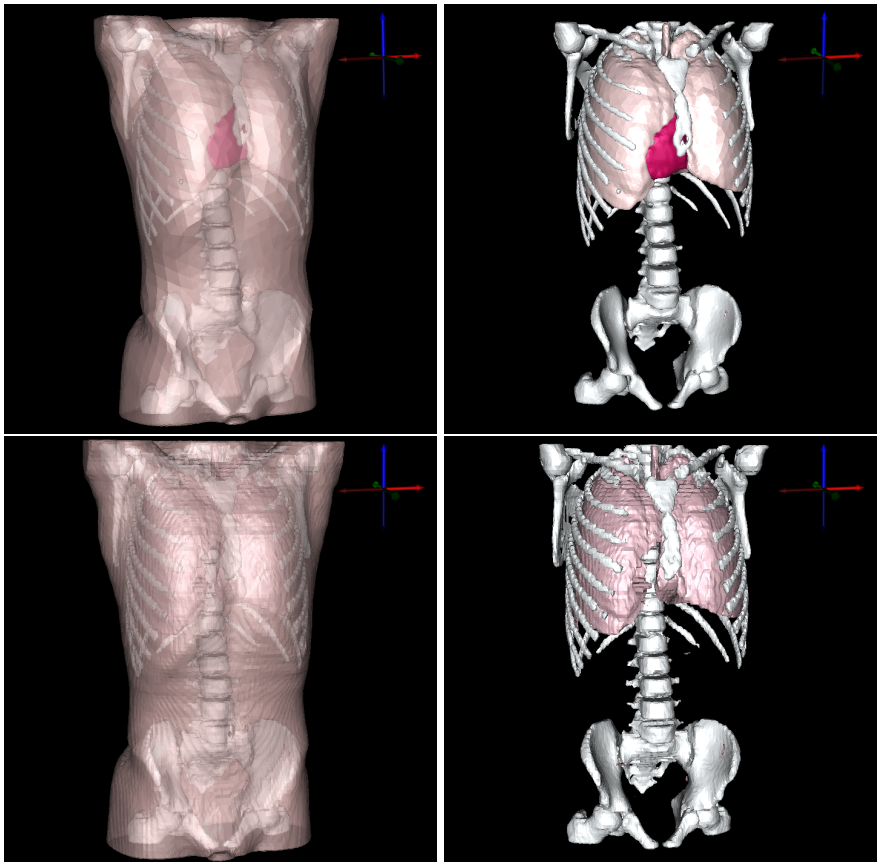

Fig. 10. Triangle surface meshes generated by afront (top) and standard Marching Cubes (bottom).

The Afront generated surfaces were also used as input to TetGen. Table 3 lists several of the resulting quality metrics for the mesh generated by TetGen

Table 1. Table indicating resulting tetrahedral quality for the pediatric torso model (torso, bone, and lung materials only using Marching Cubes surfaces and TetGen. Time to generate mesh: 1 hour 35 minutes.

Total Elements:	14,300,408 tetrahedra	29,156,724 faces	17,524,972 edges	2,668,657 nodes
Quality Metric		Low Value	Average Value	High Value
Aspect Ratio		1.00006	1.06422e+21	1.45221e+28
Volume		3.83988e-19	1.4405	266.872
Scaled Jacobian		1.40881e-15	0.37441	0.995652
Shape		1.90147e-10	0.635895	0.999953

Table 2. Table indicating resulting tetrahedral quality for the pediatric torso model (torso, bone, heart and lung materials using Afront-generated surfaces and CAMAL. Time to generate mesh: 4.42 minutes (including mesh joining).

Total Elements:	3,415,236 tetrahedra	6,837,477 faces	3,997,659 edges	575,422 nodes
Quality Metric		Low Value	Average Value	High Value
Aspect Ratio		1.00002	1.32974	1553.55
Volume		3.23703e-06	6.01067	1068.21
Scaled Jacobian		0.0070007	0.618421	0.997181
Shape		0.0563958	0.829903	0.99998

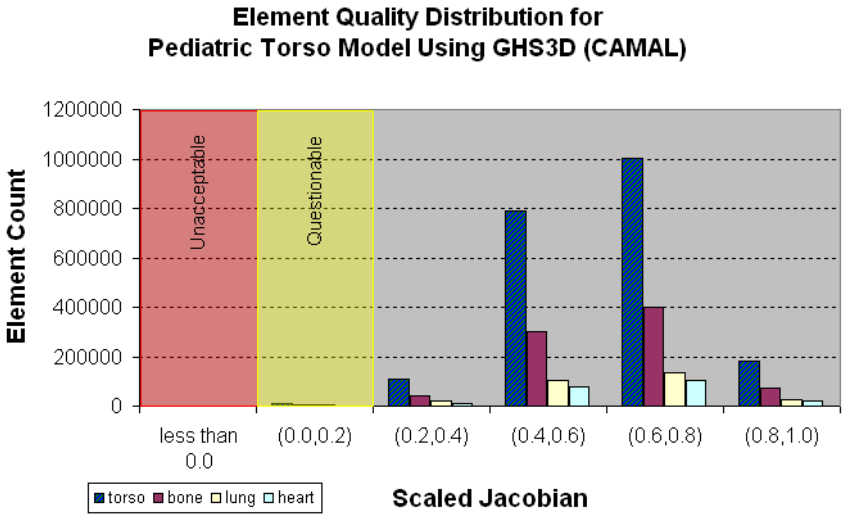

Fig. 11. Element quality of the pediatric torso mesh generated by the GHS3D (CAMAL) tetrahedral mesher as expressed in the scaled Jacobian metric.

and Figure 13 shows the histogram of element quality based on the scaled Jacobian metric. This mesh contains 4,133,993 tetrahedra. The average quality of this mesh is substantially lower than the mesh generated by CAMAL and indicates that some volumetric improvement might be useful prior to using the mesh in a subsequent simulation. A cut-away view of this mesh is displayed in Figure 14.

5.2 Skull Model

The skull model is provided courtesy of INRIA by the AIM@SHAPE Shape Repository (http://shapes.aim-at-shape.net/index.php). The difficulty in generating this model with traditional hexahedral methods is several fold. First,

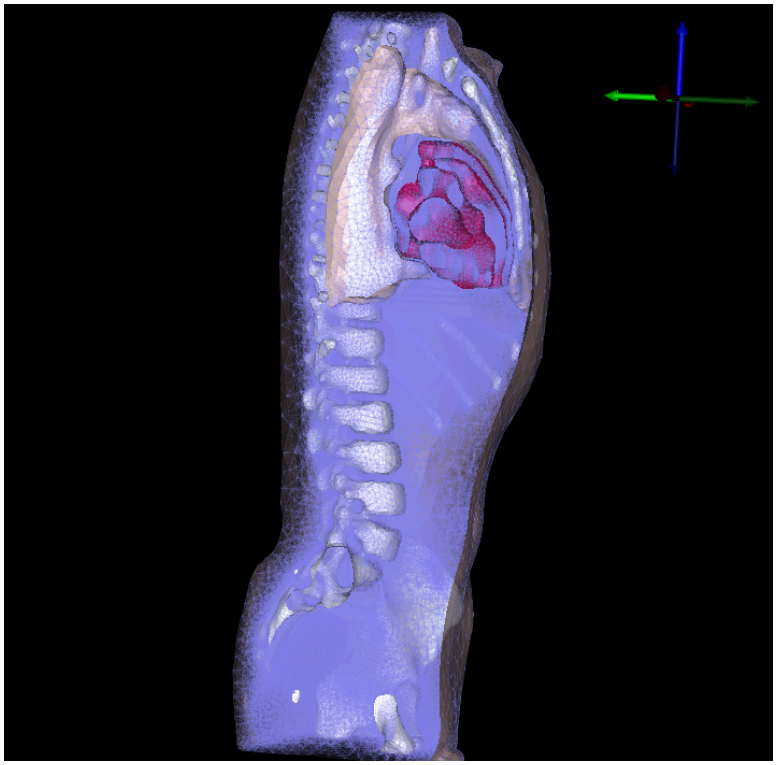

Fig. 12. Cutting plane view through torso with input surfaces and resulting tetrahedra edges shown in partially transparent blue. The tetrahedral mesh was generated by CAMAL.

Table 3. Table indicating resulting tetrahedral quality for the pediatric torso model (torso, bone, heart and lung materials using Afront-generated surfaces and TetGen. Time to generate mesh: 3.21 minutes.

Total Elements:	4,133,993 tetrahedra	8,292,237 faces	4,860,849 edges	702,606 nodes
Quality Metric		Low Value	Average Value	High Value
Aspect Ratio		1.00011	15.2247	5.00456e+07
Volume		1.74183e-07	4.96587	267.951
Scaled Jacobian		6.33094e-05	0.371769	0.990632
Shape		0.00182271	0.648091	0.999912

the original model was constructed from a triangle mesh only, and no solid model description of this model is available. Therefore traditional decomposition strategies with solid modeling operations is not readily accessible. Second, since there are no boundary curves in the model, traditional methods for determining a decomposition strategy for common hexahedral methods are not

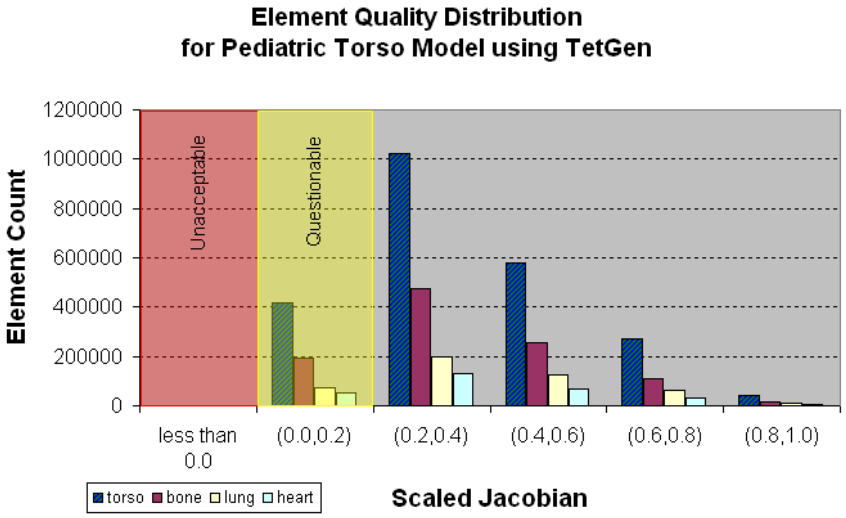

Fig. 13. Element quality of the pediatric torso mesh generated by TetGen as expressed in the scaled Jacobian metric.

present. With methods that are commonly available for performing hexahedral mesh generation, this model would be extremely difficult to produce.

The hexahedral mesh of the skull model, shown in Figure 15, was generated in SCIRun and contains 19,330 hexahedra in the skull bone and an additional 34,815 hexahedra in the mesh of the cranial cavity. The mesh is completely conformal throughout the model, but is separated into the two material blocks. A transparent view of the geometry showing the bone and cranial cavity is given in Figure 16.

This model was generated by placing a triangle mesh describing the geometry of the skull bone (minus the surface describing the cranial cavity) in a regular grid of hexahedra and inserting two layers of hexahedral elements using the triangle mesh to guide the placement of the newly formed hexahedra. The mesh exterior to the skull was discarded, and an additional set of hexahedral element layers was added using a triangle mesh describing the cranial cavity to control the placement of the new hexahedral elements. This generation process is shown in Figure 17.

The newly created hexahedral mesh was optimized using smoothing and mesh optimization routines in CUBIT [5]. First, a centroidal-area smoother was utilized on the surface of the exterior skull and the shared surface of the cranial cavity. Volumetric Laplacian smoothing was then utilized on the hexahedra in both volumes. Additional mesh untangling and condition number optimization were performed on the hexahedra in the mesh of the bone. The

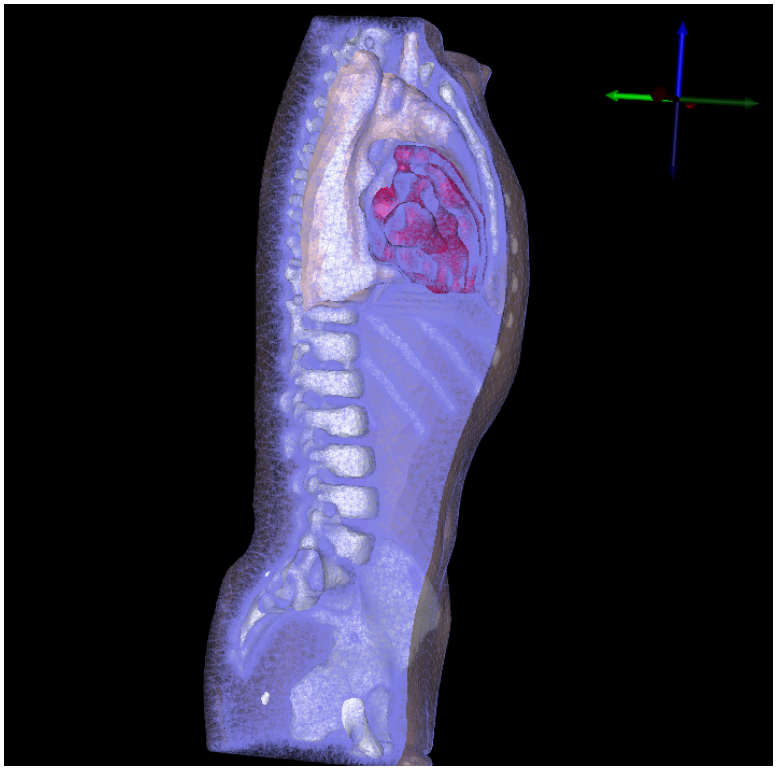

Fig. 14. Cutting plane view through torso with input surfaces and resulting tetrahedra edges shown in partially transparent blue. The tetrahedral mesh was generated by TetGen.

final mesh quality, dictated by the scaled Jacobian metric, is shown in the distribution in Figure 18.

6 Future Plans

The goal of BioMesh3D is to generate a stand alone application that is easy accessible to a large variety of biomedical computing users. The tools presented in this paper are currently in different stages of development and will be released as open source software along with the SCIRun problem solving environment available from www.sci.utah.edu.

In order to come up with an intuitive and easy to use interface, we are currently integrating these meshing tools into our SCIRun framework. The goal is to integrate the meshing tools inside a full modeling pipeline. Here the modeling pipeline refers to the process starting from extracting segmentations

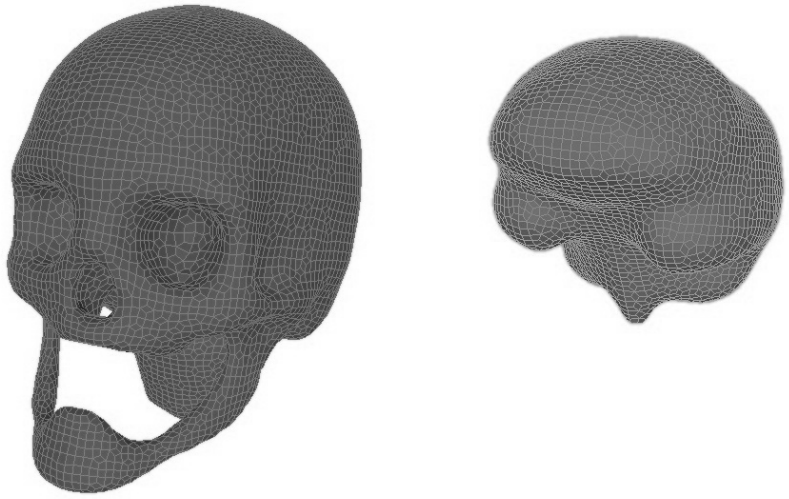

Fig. 15. Hexahedral mesh of the skull model. Bone (left) and cranial cavity (right) meshes are shown separately.

from medical images and ending in the analysis of biomedical parameters that are useful for clinicians. As the SCIRun framework contains tools for building finite element models and tools for doing simulations, embedding our pipeline in this framework will ensure that we can test the effectiveness of the different meshing strategies for different biomedical applications.

In order to evaluate the effectiveness of the algorithms, we are currently setting up modeling pipelines for a variety of biolectromagnetic applications. The first targeted application is the evaluation of defibrillation thresholds in children with an Implantable Cardiac Defibrillator (ICD). In this project we are computing the effects on the electric field inside the heart resulting from anatomical differences in the children. Other examples we are working on include localization of electrical sources in the brain, and simulating the propagation of action potentials in the heart. Each of these application adds a different focus to the meshing. For instance, in the case of simulating the propagation of action potentials, the regularity of the mesh is important, whereas preliminary results for the defibrillation project show that local refinement to obtain a high mesh density around the electrodes is important and can improve performance [32]. Because of the broad range of potential requirements within biomedical applications, a flexible pipeline containing a wide array of tools that can be applied differently depending on the biomedical application is necessary.

The application design incorporates the following paradigms: (1) the input for the application will consist of segmented data and additional geometrical

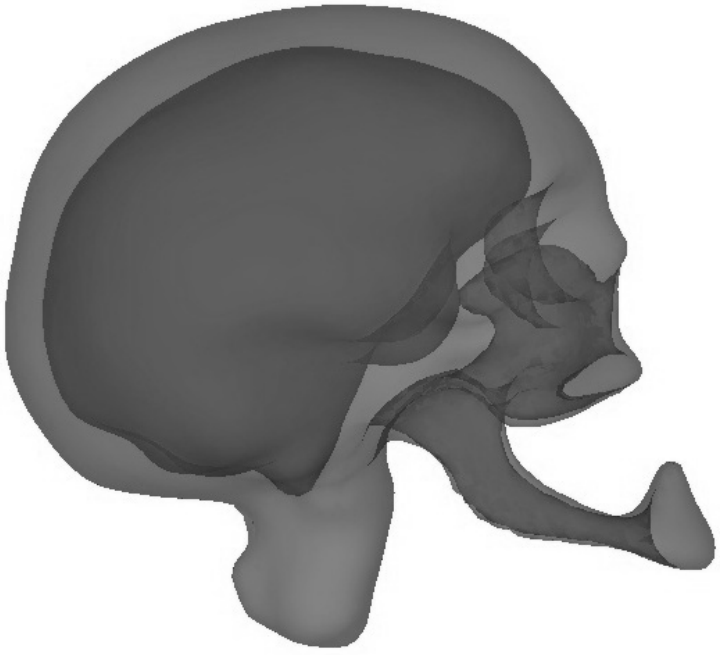

Fig. 16. Transparent view of the combined geometry generated from the facets of the hexahedral mesh of the skull model.

models and points for the boundary conditions, (2) the output will be a mesh or a series of meshes with data assigned to the mesh for subsequent simu-

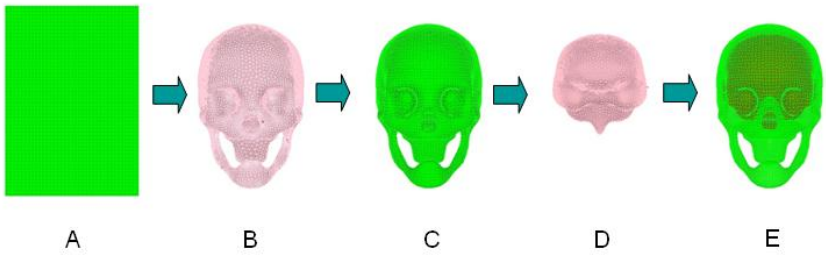

Fig. 17. Pictorial flow chart demonstrating the mesh generation process for creating the hexahedral mesh of the skull. Triangle meshes (pink) are utilized to guide placement of layers of hexahedral elements into existing hexahedral meshes to achieve new meshes that are conformal with the original solid geometry.

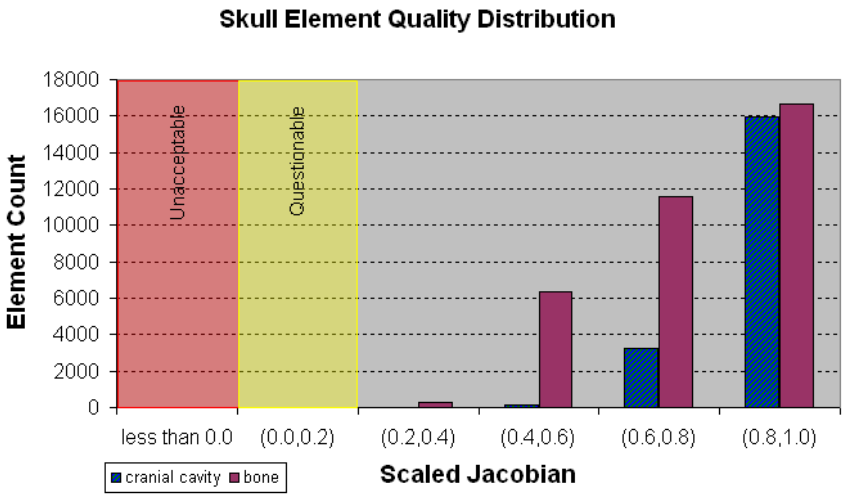

Fig. 18. Element quality of the skull model generated by the proposed hexahedral pipeline as expressed in the scaled Jacobian metric.

lations, (3) the application has to be visual (i.e., the user should be able to browse through the mesh each step of the process), (4) the meshing pipeline should be reasonably simplistic (i.e., the program should have only a few steps that user needs to follow in order to build a mesh), and (5) the program will both render meshes with tetrahedral as well as hexahedral elements, and optionally triangulated surfaces can be exported for methods like boundary elements. Figure 19 gives an example framework for the integrated BioMesh3D application.

7 Conclusion

In this paper we have outlined a pipeline for generating computational meshes suitable for biomedical simulations. Because of the wide array of geometries and phenomena encountered in biomedical computing, this pipeline incorporates a flexible suite of tools that offer some generality to mesh generation of biomedical models. Many of these tools have been successfully used in past problems, including surface remeshing strategies, geometric smoothing, mesh smoothing and optimization, volumetric mesh generation, refinement and decimation techniques, and mesh verification using standardized mesh quality metrics. These tools have been incorporated into suite of other tools in the SCIRun Problem Solving Environment. Using this environment, we have demonstrated mesh generation for a couple of example problems of in-

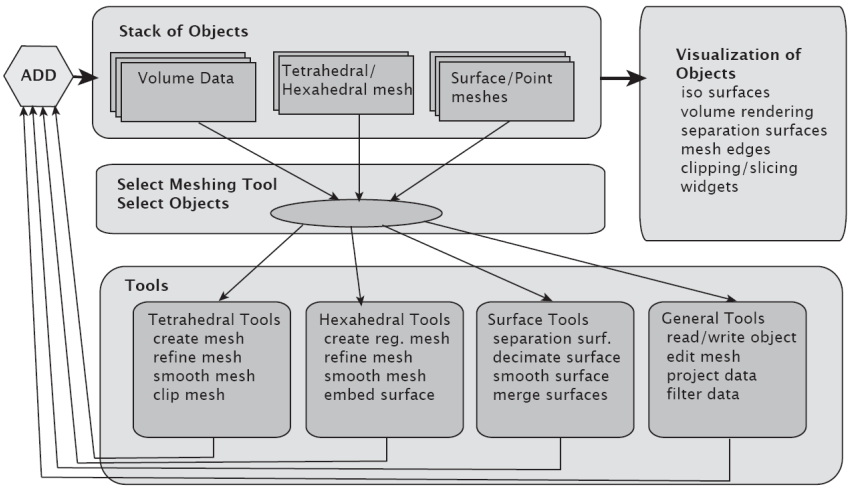

Fig. 19. Dataflow diagram of a stand alone meshing application that will combine all the different algorithms for creating a mesh. BioMesh3D will combine hexahedral and tetrahedral meshing into one application.

terest to the biomedical community including a tetrahedral mesh of a pediatric torso and a hexahedral mesh of a skull and cranial cavity. Using standardized mesh quality metrics, we showed that these meshes would be suitable for use in biomedical simulation. We plan to incorporate the tools presented in this paper into an integrated meshing tool for use by the biomedical community. It is hoped that this tool can be utilized to dramatically reduce the difficulty of mesh generation for biomedical simulations.

In this paper we compared meshing techniques for a few examples by means of the scaled Jacobian metric. However there may be other factors determining what is suitable for an application (i.e., speed of the meshing procedure, control of element sizing, simulation boundary conditions, etc.). To determine what is appropriate to use for certain application, one should consider the full modeling pipeline. For instance, if the intent is to use a single mesh for a number of of time-intensive simulations, it may be best to optimize the mesh with fewer elements with higher quality, whereas a mesh is to be used for a single simulation, a speedier meshing method that results in a lot of elements may be more appropriate for the problem. Hence, having the meshing tools incorporated with segmentation, simulation and visualization tools within a larger problem solving framework like SCIRun, will have the added benefit of being able to test these different tools within a common environment. In this way, the performance of the meshing algorithms can also be compared by looking at the results of a visualization or simulation, providing an alternative metric for assessing performance of the meshing strategy.

References

1. 2007. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI), Download from: http://software.sci.utah.edu/scirun.html.
2. P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. Recent advances in remeshing of surfaces. *Research Report, AIM@SHAPE Network of Excellence*, 2005.
3. M. Brewer, L. Freitag-Diachin, P. Knupp, T. Leurent, and D. J. Melander. The MESQUITE mesh quality improvement toolkit. In *Proceedings, 12th International Meshing Roundtable*, pages 239–250. Sandia National Laboratories, September 2003.
4. The CUBIT Adaptive Meshing Algorithm Library, Sandia National Laboratories, http://cubit.sandia.gov/camal.html, 2007.
5. The CUBIT Geometry and Mesh Generation Toolkit, Sandia National Laboratories, http://cubit.sandia.gov/, 2007.
6. M. Desbrun, M. Meyer, P. Schroder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Siggraph'99 Conference Proceedings*, pages 317–324, August 1999.
7. G. Dhatt and G. Touzot. *The Finite Element Method Displayed*. John Wiley and Sons, 1984.
8. L. F. Diachin, P. Knupp, T. Munson, and S. Shontz. A comparison of inexact Newton and coordinate descent mesh optimization techniques. In *Proceedings, 13th International Meshing Roundtable*, pages 243–254. Sandia National Laboratories, September 2004.
9. D. A. Field. Laplacian smoothing and Delaunay triangulation. *Communications in Applied Numerical Methods*, 4:709–712, 1988.
10. L. Freitag. On combining Laplacian and optimization-based mesh smoothing techniques. *AMD Trends in Unstructured Mesh Generation, ASME*, 220:37–43, 1997.
11. L. A. Freitag and P. Plassmann. Local optimization-based simplicial mesh untangling and improvement. *International Journal for Numerical Methods in Engineering*, 49(1):109–125, September 10-20, 2000.
12. M. Garland. *Quadric-Based Polygonal Surface Simplification*. Published Doctoral Dissertation, Carnegie Mellon University, May 1999.
13. P. Hansbo. Generalized Laplacian smoothing of unstructured grids. *Communications in Numerical Methods in Engineering*, 11:455–464, 1995.
14. N. Harris, S. E. Benzley, and S. J. Owen. Conformal refinement of all-hexahedral meshes based on multiple twist plane insertion. In *Proceedings, 13th International Meshing Roundtable*, pages 157–168. Sandia National Laboratories, September 2004.
15. C. Johnson, R. MacLeod, S. Parker, and D. Weinstein. Biomedical computing and visualization software environments. *In Communications of the ACM*, 47(11):64–71, 2004.
16. T. R. Jones, F. Durand, and M. Desbrun. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics*, 22(3):943–949, 2003.
17. G. Kindlmann. Teem, http://teem.sourceforge.net/, December 2005.
18. P. Knupp and S. A. Mitchell. Integration of mesh optimization with 3D all-hex mesh generation, LDRD subcase 3504340000, final report. SAND 99-2852, October 1999.

19. P. M. Knupp. Winslow smoothing on two-dimensional unstructured meshes. In *Proceedings, 7th International Meshing Roundtable*, pages 449–457. Sandia National Laboratories, 1998.
20. P. M. Knupp. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities: Part II - a framework for volume mesh optimization and the condition number of the Jacobian matrix. *International Journal for Numerical Methods in Engineering*, 48:1165–1185, 2000.
21. P. M. Knupp. Algebraic mesh quality metrics. *SIAM J. Sci. Comput.*, 23(1):193–218, 2001.
22. P. M. Knupp. Hexahedral and tetrahedral mesh shape optimization. *International Journal for Numerical Methods in Engineering*, 58(1):319–332, 2003.
23. P. M. Knupp. Hexahedral mesh untangling and algebraic mesh quality metrics. In *Proceedings, 9th International Meshing Roundtable*, pages 173–183. Sandia National Laboratories, October 2000.
24. W. E. Lorenson and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (Proceedings of SIGGRAPH '87)*, 21(4):163–169, 1987.
25. M. Loriot. TetMesh-GHS3D v3.1 the fast, reliable, high quality tetrahedral mesh generator and optimiser, http://www.simulog.fr/mesh/tetmesh3p1d-wp.pdf, 2006.
26. MESQUITE: The Mesh Quality Improvement Toolkit, Terascale Simulation Tools and Technology Center (TSTT), http://www.tstt-scidac.org/research/mesquite.html, 2005.
27. S. Parker, D. Weinstein, and C. Johnson. The SCIRun computational steering software system. In E. Arge, A. Bruaset, and H. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 1–40. Birkhauser Press, Boston, 1997.
28. C. Scheidegger and J. Schreiner. Afront, http://sourceforge.net/projects/afront/, January 2007.
29. J. Schreiner and C. Scheidegger. Algorithm for separating hexahedra given a triangle mesh. *SCI Institute Technical Report*, UUSCI-2007, 2007.
30. J. F. Shepherd. *Topologic and Geometric Constraint-Based Hexahedral Mesh Generation*. Published Doctoral Dissertation, University of Utah, May 2007.
31. H. Si. TetGen: A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, Research Group: Numerical Mathematics and Scientific Computing, Weierstrass Institute for Applied Analysis and Stochastics, Mohrenstr. 39, 10117 Berlin, Germany, http://tetgen.berliso.de, 2007.
32. J. G. Stinstra, M. Jolley, M. Callahan, D. Weinstein, M. Cole, D. H. Brooks, J. Triedman, and R. S. MacLeod. Evaluation of different meshing algorithms in the computation of defibrillation thresholds in children. *to appear in Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, August 2007.
33. J. O. Talton. A short survey of mesh simplification algorithms. manuscript, University of Illinois at Urbana-Champaign, Octoboer 2004.
34. G. Taubin. A signal processing approach to fair surface design. In *Siggraph'95 Conference Proceedings*, pages 351–358, August 1995.
35. P. Vachal, R. V. Garimella, and M. J. Shashkov. Mesh untangling. LAU-UR-02-7271, T-7 Summer Report 2002.
36. The Verdict Mesh Verification Library, Sandia National Laboratories, http://cubit.sandia.gov/verdict.html, 2007.

37. T. Zhou and K. Shimada. An angle-based approach to two-dimensional mesh smoothing. In *Proceedings, 9th International Meshing Roundtable*, pages 373–384. Sandia National Laboratories, 2000.