A Meshing Pipeline for Biomedical Computing

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

Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, Utah, USA, www.sci.utah.edu

Biomedical computing applications often require a computational pipeline that integrates data from experimental measurements or from image acquisition into a modeling and visualization environment. The latter process often involves segmentation, mesh generation, and numerical simulations. 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 both as input for computational simulation and as the geometric basis for many of the resulting visualizations. Historically, the generation of these meshes has been a significant bottleneck in efforts to efficiently create complex, three-dimensional biomedical models.

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, SCIRun, 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 SCIRun. We will demonstrate mesh generation for example problems along with methods for verifying the quality of the meshes generated. Finally, we will discuss ongoing 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 scientists 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 new biomedical scientific discoveries and clinical treatments.

Biomedical simulations depend on numerical approximation methods, such as 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 and as the geometric basis for visualization results to be displayed. Historically, the generation of these meshes has been a significant bottleneck in efforts to efficiently generate accurate biomedical models.

In this paper, we will outline a pipeline of tools for effective generation of meshes suitable for biomedical simulations. Because of the wide array of geometries and phenomena encountered in biomedical computing, this pipeline must 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 SCIRun. We will demonstrate mesh generation for example problems along with methods for verifying the quality of the meshes generated. Finally, we will discuss ongoing 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, 42, 16, 17, 30]. SCIRun is an open source, multi-platform, problem-solving environment that allows 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 modeling, image analysis, simulation, 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.

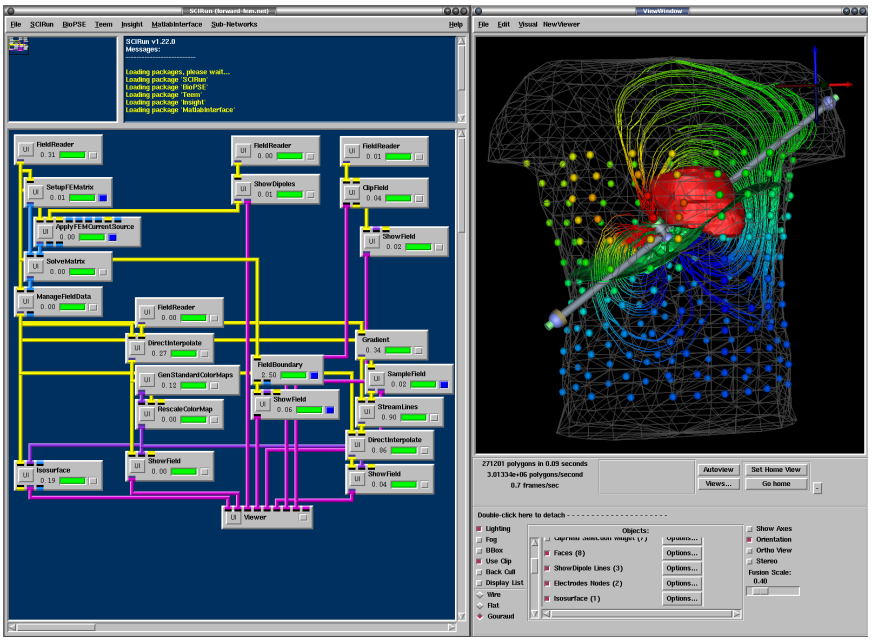

Fig. 1: The SCIRun PSE showing the module network (middle), the visualization window (right). Researchers can select UI (user interaction) buttons on many of the modules that allow control and feedback of parameters within a particular module (left).

Several overarching goals have influenced the development of the meshing pipeline presented in this paper, especially in 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 able to fit into a pipeline with a broad suite of other software tools.

3 Pipeline

Figure 2 gives a broad overview of the meshing pipeline implemented in SCIRun for preparing meshes for biomedical simulation. Because 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. The next section will discuss each tool at greater length. This section provides a general overview of the entire pipeline.

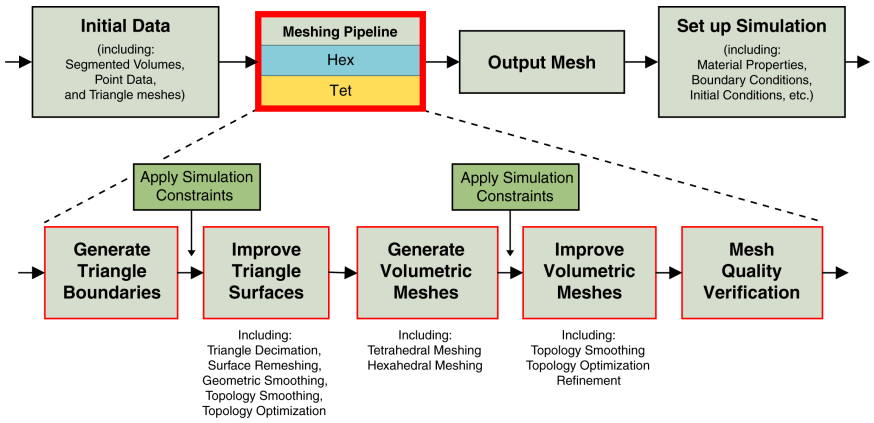

Fig. 2: The meshing pipeline in SCIRun.

The choice of the numerical method utilized in a biomedical simulation is often based upon the anticipated and acceptable level of error, the applicability of the method for geometric 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 a three-dimensional grid of scalar values. The image stacks or scalar grids are often referred to as 'volumetric' data. As electrical or mechanical properties are often categorized by tissue type, the volumetric data is commonly 'segmented' into a smaller subset of discrete values that focuses the anatomical 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 segment values. This can be accomplished in several

ways. An isosurface algorithm, such as the Marching Cubes algorithm [27], may be used to obtain a set of triangle meshes representing the boundaries between features within the volumetric data. While Marching Cube algorithms emit smoother models than other methods, in many cases these algorithms require that each sub-volume be fully embedded inside another sub-volume. Thus, the segmentations may be altered to observe the constraints of the Marching Cubes algorithms and so fail to preserve exact space partitions. A more accurate alternative involved emitting a quadrilateral face between any two voxels with a different segmentation value. This results in a stair-step model of all the segmentation boundaries. 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 surface mesh has been created describing these boundaries, the next step in the pipeline is to optimize the mesh on the boundaries 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 supply 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 to ensure that the mesh will be suitable for the subsequent analysis. The SCIRun pipeline currently has methods for generating a 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 direct generation of hexahedral meshes. There are two main types of hexahedral meshes utilized in SCIRun: hexahedral meshes with stair-stepped boundaries and hexahedral meshes with smooth boundaries. A raw model segmentation can also be thought of as a hexahedral mesh; however, the resolution of these meshes is often greater than desired. Therefore, some data reduction is often necessary. SCIRun contains algorithms for resampling this data at coarser hexahedral representations. Additionally, a coarse lattice can be built by resampling the original segmentation and then using localized refinement techniques to recover data in areas of importance. When necessary, as in simulations that are constrained by factors other than the volumetric data, such as the location of electrical sources, these refinements can even exceed the resolution of the original data.

The hexahedral meshing pipeline, which uses the segmented hexahedral model, is similar to the tetrahedral pipeline where the segmentation is used to define material boundaries. However, hexahedral meshes are generated by using a regular spaced lattice as the basis for the mesh instead of by using a template to cut the mesh elements to improve the geometric boundary for each of the materials (as done by a Marching Cubes algorithm) and then using these boundaries as starting points for the tetrahedralizer. In the hexahedral meshing pipeline, then, boundaries are obtained in one of two ways: either directly from the segmentation, thus forming a stair-stepped boundary, or through the addition of a new layer of hexahedra into the existing mesh to form a smooth boundary inside mesh. The hexahedral pipeline is finished with volumetric smoothing and refinement techniques, followed as well by mesh verification to ensure the resulting mesh is suitable for subsequent analysis. These techniques and algorithms are discussed in more detail in the next sections.

4 Specific Tools in the Pipeline

In this section, we describe in more detail the various tools that have been implemented to date in the SCIRun meshing pipeline. The format for this section will roughly follow the pipeline order described in Figure 2. 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 triangular meshes resulting from a Marching Cubes algorithm typically contain poor-quality elements, sliver triangles, and/or dramatic size variations, it is often desirable to remesh in an effort to obtain a better set of triangles. Triangle meshes that are smoother and more uniform in size are often advantageous because they field a smaller number of tetrahedral elements in the final mesh and have higher overall element quality.

Surface remeshing is an active area of ongoing meshing research [3]. While there are several tools available from the community, a tool that has demonstrated success for triangle remeshing and is readily available to us is Afront (Advancing Front (Re)Meshing Algorithm) [34, 32]. Since Afront is currently limited to isosurface boundaries (multi-material, or non-manifold, boundaries are not allowed), it is utilized to isosurface individual materials, and thus it 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. In practice, some iteration is needed to find isovalues that provide non-overlapping interior surfaces. Additionally, if the neighboring surfaces are too close, the tetrahedralization will result in a large amount of tiny elements on the boundary. Hence, in general a small void is needed between the elements when employing this technique. 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 contain small errors of detail that result either from algorithmic issues, improper segmentations, or unhandled exceptions. This can lead to small unclosed areas or slightly overlapping elements in the mesh. As such, it is often desirable to have a small set of tools available for use when all other methods fail. We have developed some basic hand-editing tools for this purpose.

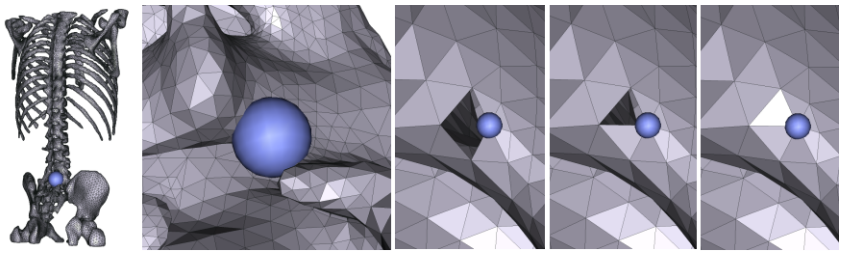

Fig. 3: A series of images showing the process for finding and filling holes in a triangular surface. This process is described in 4.1 Elemental Cleanup.

In all the previous steps we hope to eliminate the need for hand editing, but allowing a user to select and edit problem areas in the 3D scene gives power to the process and makes some dire situations tractable. Tools used to tetrahedralize our input surfaces typically report problem areas that prevent them 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 locate a hole in the mesh. We can then focus the view and visually inspect this problem area. Command line tools are provided for deleting faces and adding triangles given specific node and face indices viewable in SCIRun. The development version of our new mesh generation application, BioMesh3D, has added a more user-friendly selection mechanism. Sets of faces can be interactively added or deleted from the model by the user. 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 problematic for biological models where topological and geometric errors can have a significant 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 [38] we chose a quadric-based edge collapse method [13] adapted to preserve topological constraints of the surface intersections.

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 surface model is shown in Figure 4.

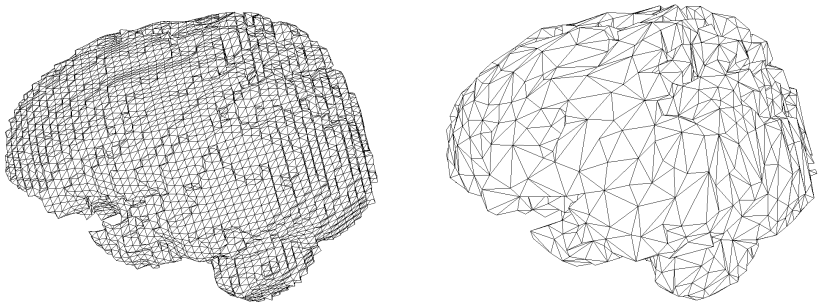

Fig. 4: Example of decimation of triangles from a mesh of a brain. The original mesh is shown at left, while the image on the right shows the mesh after decimation.

Geometric Smoothing

Undesired geometric features are a common occurrence when generating isosurfaces. FairMesh is a geometric smoothing module in SCIRun designed to rely on Gaussian filtering to smooth a mesh without shrinking the volume enclosed by the mesh. This module uses an algorithm developed by Taubin [39],

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

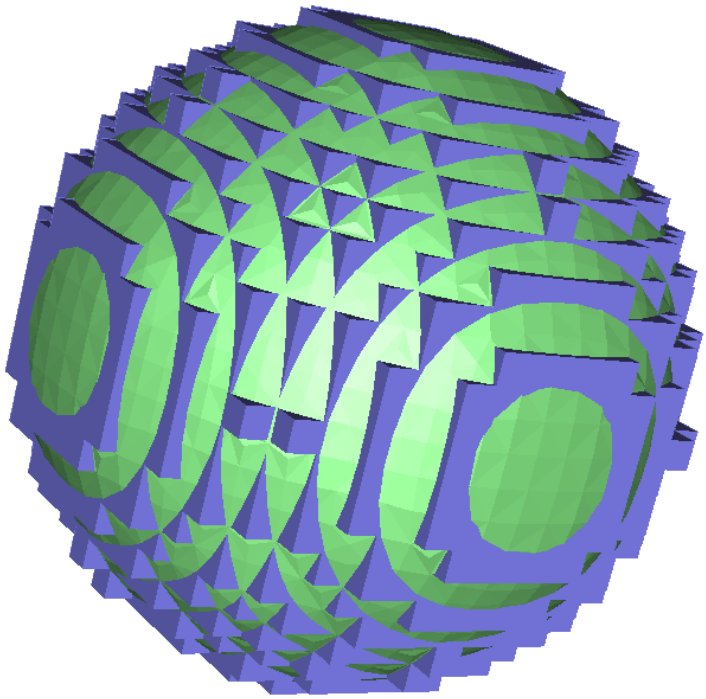

Fig. 5: The blue stair-stepped mesh is the input to FairMesh, the green is the smoothed output.

Smoothing with equal edge costs tends 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 on which one would want to apply a texture map. So-called Desbrun [7] 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.

4.2 Volumetric Mesh Generation Tools

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

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 discuss our efforts to incorporate two tetrahedral mesh generation libraries into the SCIRun meshing pipeline. The major trade-offs with these tools are the quality of the resulting meshes and the ease of licensing their use.

CUBIT

CUBIT [6], developed at Sandia National Laboratories, is a full-featured software toolkit for geometric model generation and robust generation of two- and three-dimensional finite element meshes. 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 (e.g., the Cubit Adaptive Meshing Algorithms Library (CAMAL) [5]). In particular, CUBIT utilizes the state-of-the-art GHS3D tetrahedral meshing library [28] 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. CUBIT is available for research use from Sandia National Laboratories for a small licensing fee.

TetGen

TetGen [36] is an open-source solution to generating tetrahedral meshes from a triangle surface input. TetGen is a great tool for incorporation into a research meshing pipeline because it is both free and readily available.

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 usable input.

The quality of the output mesh is typically lower than meshes generated by some commercial tetrahedral packages [28]. However, TetGen can refine specified areas more or less densely. Separate regions can be tagged and remapped to the original material types and the entire volume will be tetrahedralized according to the input criteria. The ease with which additional points and surfaces can be added as additional constraints is particularly useful for generating meshes for doing simulations.

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 [35] and is 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' 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 [33].
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.

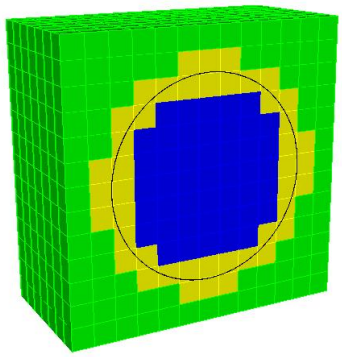

Fig. 6: A hemispherically-shaped triangle mesh (the boundary of the triangle mesh is shown in black) is placed in a hexahedral grid. The hexahedra intersected by the triangle mesh are shown in yellow, while 'Side1' is drawn in green and 'Side2' is shown in blue.

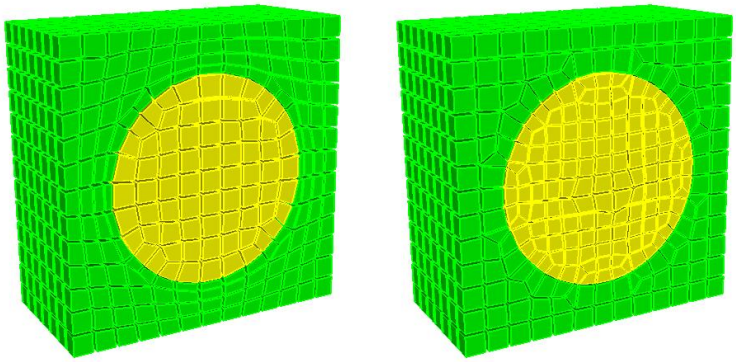

Fig. 7: Slightly different meshes result depending on which side the intersected hexes are grouped. The image on the left shows the resulting mesh if the intersected hexes are placed with Side1's hexes, while the image on the right has the intersected hexes being grouped with Side2.

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, by projecting a new node to the triangle mesh for each node on the separated boundary. 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.*

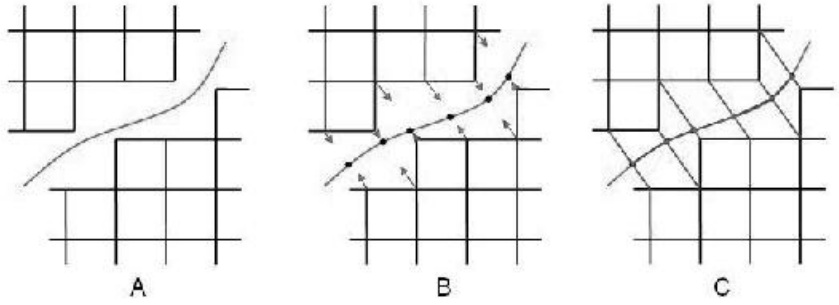

Fig. 8: Image A shows the shrunken hexahedra with the triangle mesh shown in between the hexahedra. Image B shows a newly projected node to the triangle mesh for each node on the boundary of the shrunken mesh (note that a single node on the triangle mesh corresponds to one node on each of the shrunken 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.

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 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 [29, 4] 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 [25]), 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 [14, 10], centroidal area [19], Winslow [22], angle-based [43], 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 [11], and a mesh optimization algorithm for improving the 'shape' metric, called Shape Improvement Optimization [25]. 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 currently available, the most common algorithm is known as Laplacian smoothing. The easiest way to understand 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 diminished 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 [11].

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 can therefore 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, this 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.

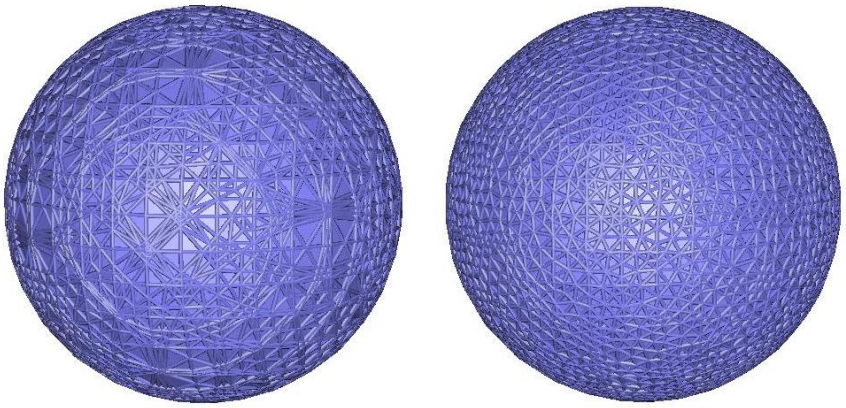

Fig. 9: Example of smart Laplacian smoothing (from the MESQUITE toolkit implemented in SCIRun) on a tetrahedral mesh. The image on the left shows the mesh before smoothing, and the images on the right is the same mesh after smoothing.

Mesh Untangling

A mesh untangling algorithm [21] 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 [24]. 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 [21, 40, 26, 12].

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 [25], condition number [21], Jacobian [23], and mean ratio [9, 24]. While these methods can be extremely effective at maximizing metric values, they can also be 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 [11]. 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 [15] and has been modified to use the four different hexahedral templates presented by that algorithm within a Marching Cubes style lookup table scheme. This method computes a refinement very quickly as 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 ideally 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 [41] 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 [31].

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 [8]), 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 display quality results utilizing the determinant of a scaled Jacobian matrix as calculated by the Verdict [41] library for each of the elements within the mesh.

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.

5.1 Pediatric Torso Mesh Generation

The overall goal for this project was to create a tetrahedral finite element mesh of a pediatric torso from a segmented volume for bioelectric field simulation and defibrillation device design. In addition, we used this project to compare different tetrahedral mesh generation tools to each other.

The input to the pediatric torso mesh pipeline was a segmented volume image data set (i.e., a NRRD file [20] 512x512x111) consisting of 11 different material categories. Each material in the segmentation was separated into a separate volume prior to isosurfacing. Using the Teem tools, each of the separate material files was resampled to 111x111x111 by filtering down in x and y to get a uniform size in each dimension.

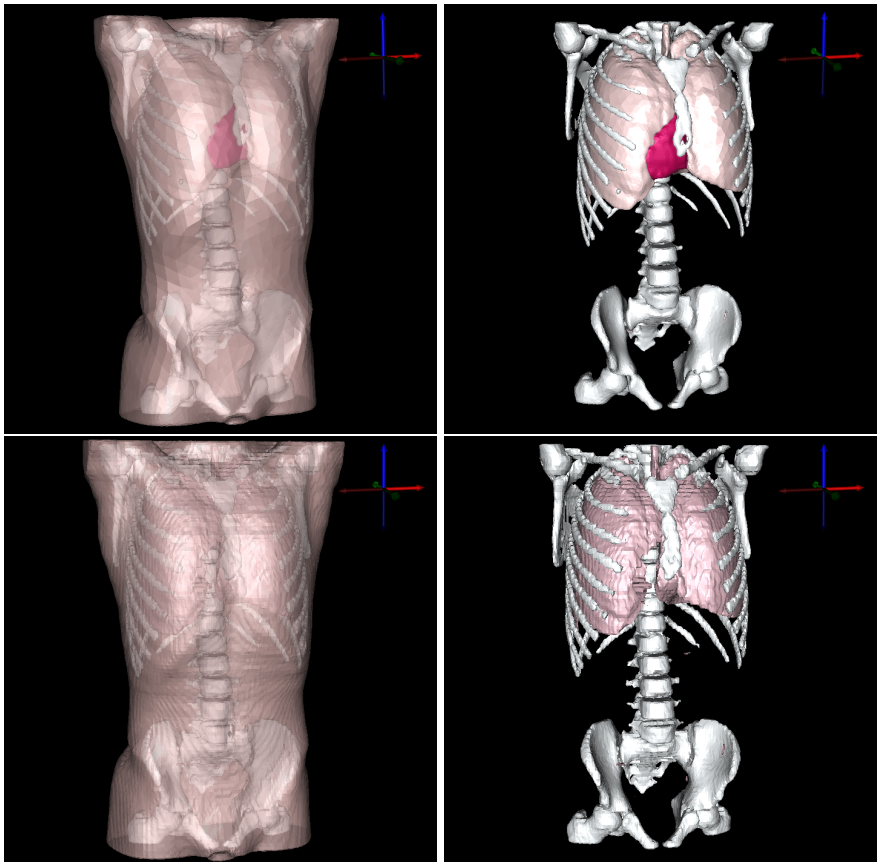

Fig. 10: Triangle surface meshes generated by Afront (top) and standard Marching Cubes (bottom).

Each of the materials included in the final mesh was 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, portions of the lung and heart surfaces would share 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 set to be different for the heart and lungs and thus they 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.

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.

Totals:	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

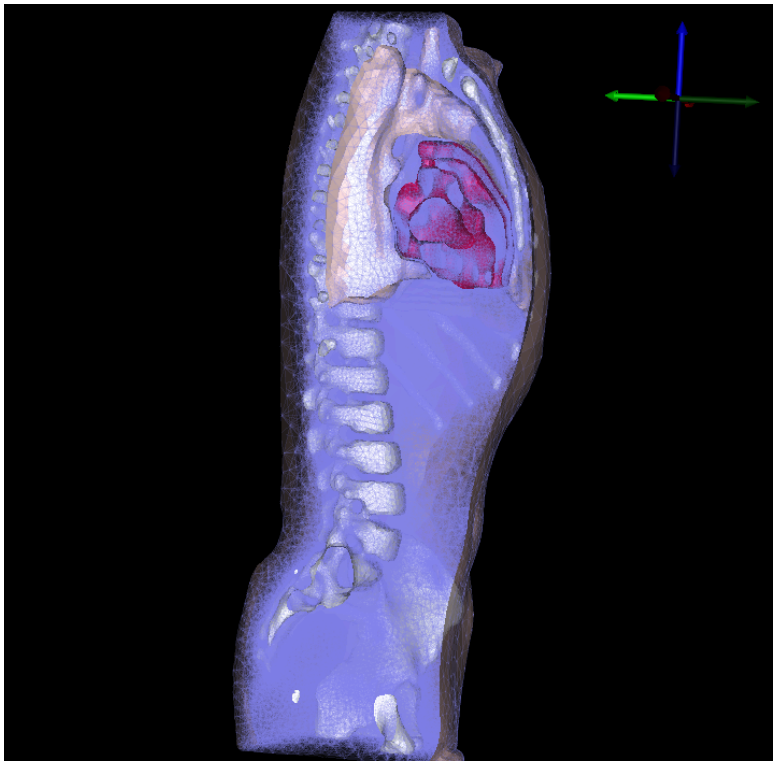

Fig. 11: Cutting plane view through torso with input surfaces and resulting tetrahedra edges shown in partially transparent blue. The tetrahedral mesh was generated by CAMAL.

The higher surface quality from the Afront generated surfaces resulted in shorter tetrahedral mesh generation timings as well as in generally 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.

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).

Totals:	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	

In Figure 12 a histogram is given of the scaled Jacobian metric per element, showing that only a few of the elements have low quality and that most elements are properly shaped. The final mesh contains 3,415,236 tetrahedra. A cut-away view of this mesh is displayed in Figure 11.

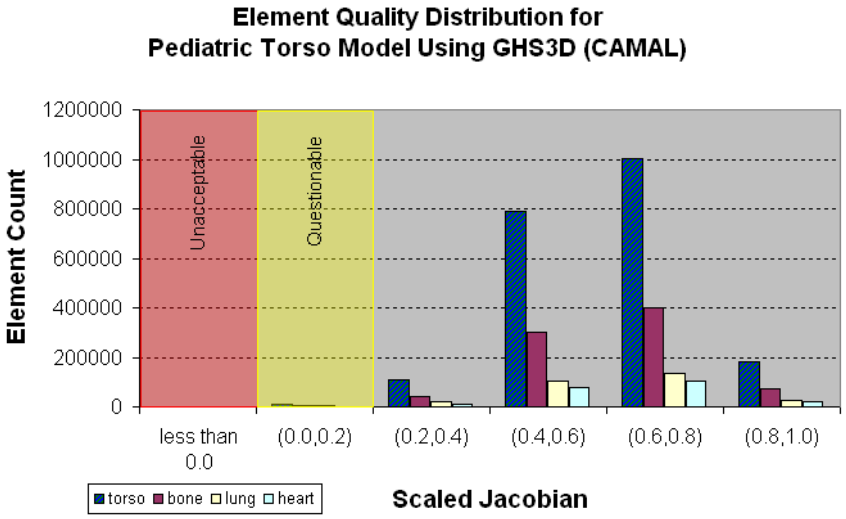

Fig. 12: Element quality of the pediatric torso mesh generated by the GHS3D (CAMAL) tetrahedral mesher as expressed in the scaled Jacobian metric.

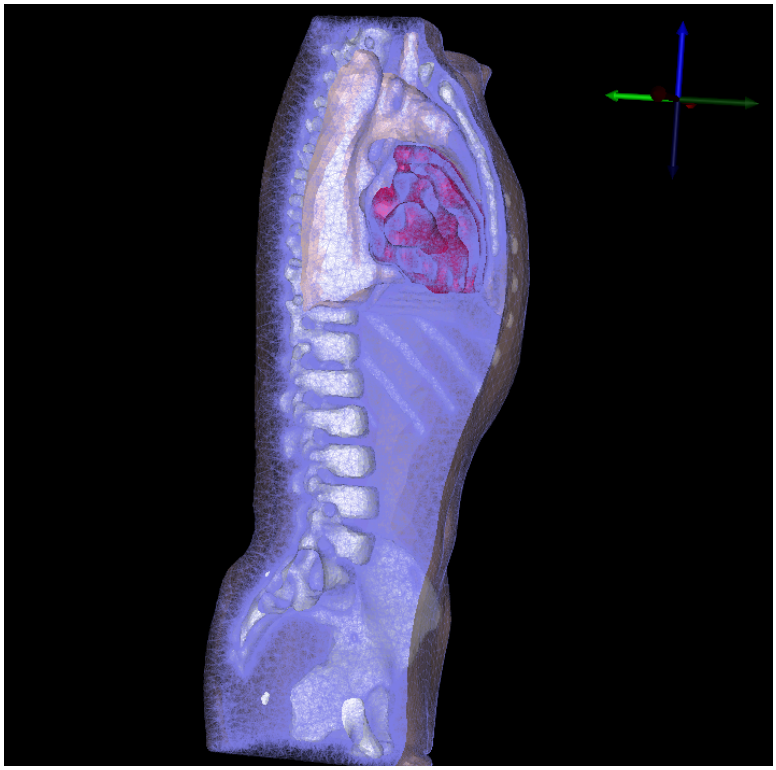

Fig. 13: Cutting plane view through torso with input surfaces and resulting tetrahedra edges shown in partially transparent blue. The tetrahedral mesh was generated by TetGen.

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 and Figure 14 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 lower than the mesh generated by CAMAL, which 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 13.

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.

Totals:	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	

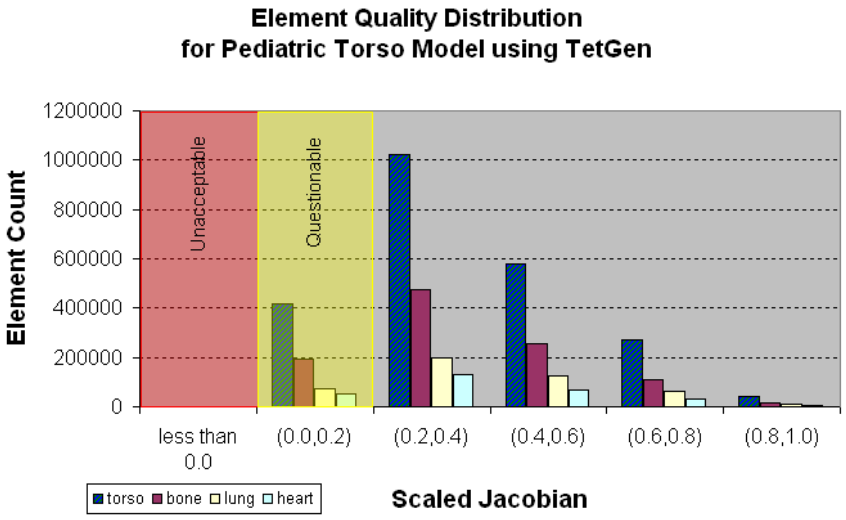

Fig. 14: Element quality of the pediatric torso mesh generated by TetGen as expressed in the scaled Jacobian metric.

5.2 Skull Model Mesh Generation

The skull model was provided courtesy of INRIA by the AIM@SHAPE Shape Repository (http://shapes.aim-at-shape.net/index.php). There are a number of difficulties in generating this model with traditional hexahedral methods. First, 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 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.

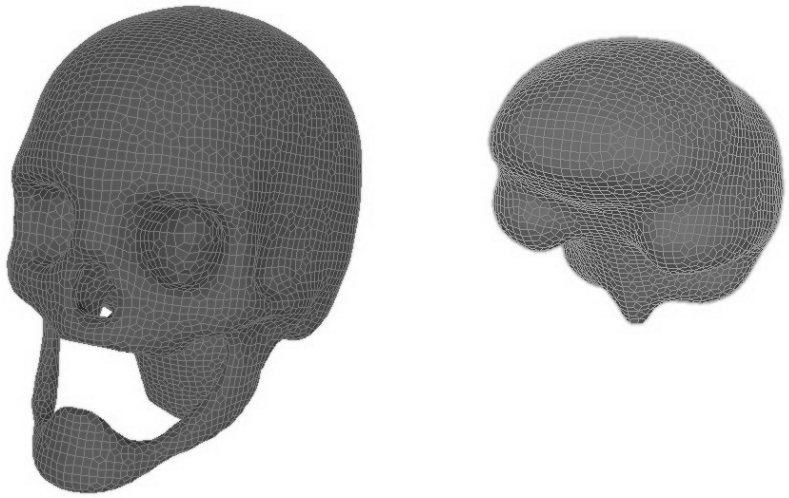

Fig. 15: Hexahedral mesh of the skull model. Bone (left) and cranial cavity (right) meshes are shown separately.

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

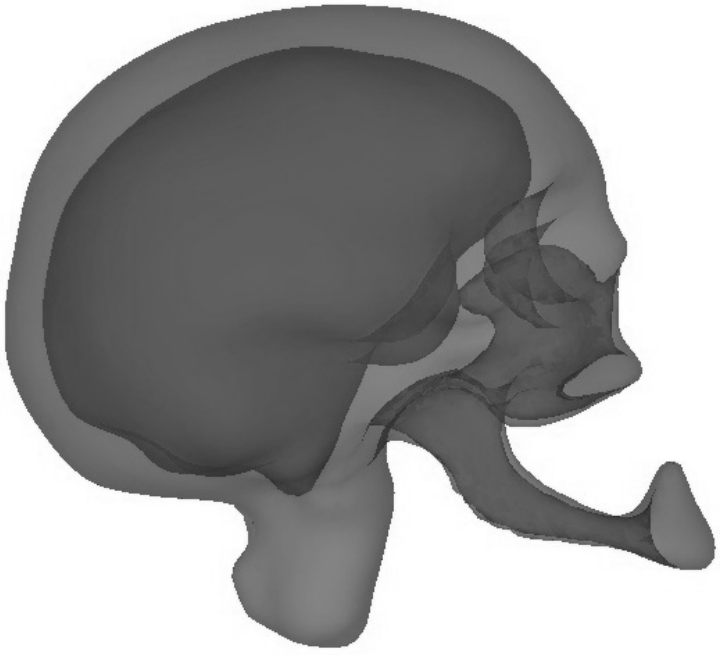

Fig. 16: Transparent view of the combined geometry generated from the facets of the hexahedral mesh of the skull model.

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

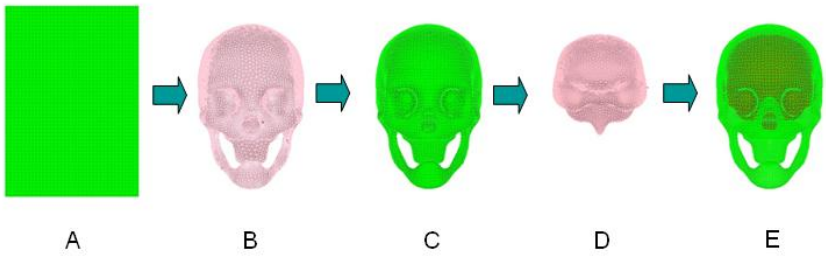

Fig. 17: Pictorial flow chart demonstrating the mesh generation process for creating the hexahedral mesh of the skull. Triangle meshes (pink) are utilized to guide placement of layers of hexahedral elements into existing hexahedral meshes to achieve new meshes that are conformal with the original solid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

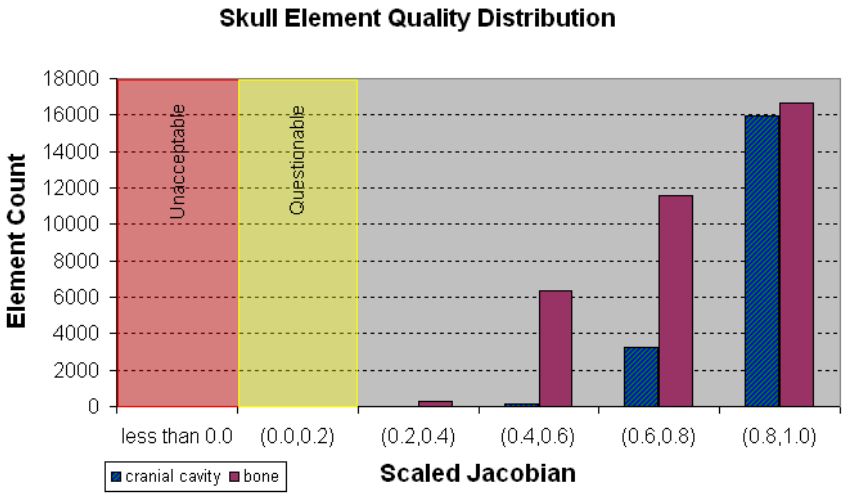

Fig. 18: Element quality of the skull model generated by the proposed hexahedral pipeline as expressed in the scaled Jacobian metric.

6 Future Plans

Given our experience with tetrahedral and hexahedral mesh generation tools and the SCIRun problem solving environment, we are currently developing an open-source, stand-alone mesh generation application, BioMesh3D, that will be useful for a large variety of biomedical computing applications. BioMesh3D and the mesh generation 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 from images using a tool such as Seg3D [2], creating a volume mesh, performing simulations, and visualizing the results, resulting in the analysis of biomedical systems that are useful for researchers and 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 setting up modeling pipelines for a variety of biomedical simulation applications. The first targeted application is the evaluation of defibrillation thresholds in children with Implantable Cardiac Defibrillators (ICD) [18]. In this project we are computing the strengths of electric fields within the heart that are caused by shock that the ICD delivers. The latter measure is a measure for the efficacy of defibrillating a heart. Due to anatomical differences between patients and differences in location of the ICD electrodes, the size and orientation of these fields vary; hence, reconstructing these fields interactively will help in finding optimal locations for implantation. Other examples include localization of electrical sources in the brain and simulating the electrical activity propagation in the heart. Each of these applications adds a different focus to the meshing. For instance, in the case of simulating the propagation of electrical activity within the heart, 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 [37]. 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 of BioMesh3D incorporates the following considerations: (1) the input for the application will consist of segmented data and additional geometrical 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 simulations, (3) the application has to provide visual interac-

tion (i.e., the user should be able to browse through the mesh each step of the process), (4) the meshing pipeline should be reasonably simple (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 permit optionally triangulated surfaces to be exported for methods like boundary elements. Figure 19 gives an example framework for the integrated BioMesh3D application.

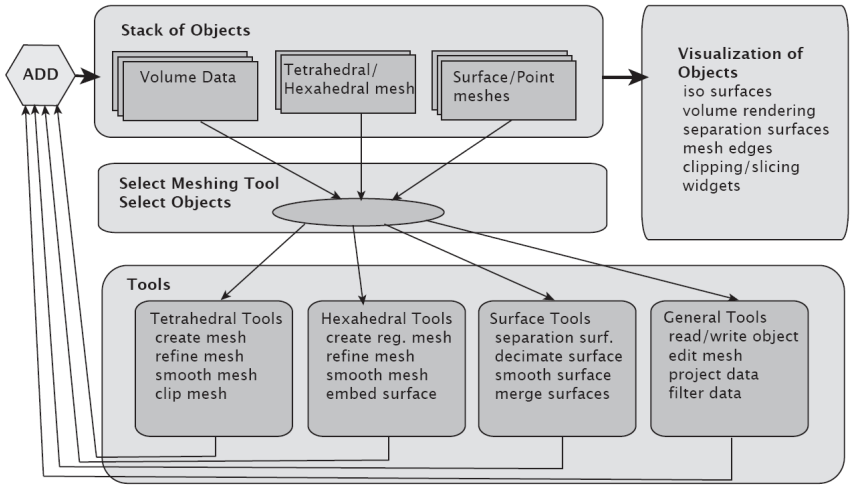

Fig. 19: Dataflow diagram of a stand alone meshing application that will combine all the different algorithms for creating a mesh. BioMesh3D will combine hexahedral and tetrahedral meshing into one application.

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 interest 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 BioMesh3D, an integrated, open-source 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 used the scaled Jacobian metric to compare meshing techniques for a few examples. 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 time-intensive simulations, it may be best to optimize the mesh with fewer elements with higher quality, whereas if 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. Hence, incorporating the meshing tools with segmentation, simulation, and visualization tools within a larger problem solving framework like SCIRun will provide the added benefit of enabling the user 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. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI). Download available from: http://software.sci.utah.edu/scirun.html.
2. Seg3D: Volumetric Image Segmentation and Visualization. Scientific Computing and Imaging Institute. Download available from: http://www.sci.utah.edu/cibc/software/seg3d.html.
3. P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. Recent advances in remeshing of surfaces. *Research Report, AIM@SHAPE Network of Excellence*, 2005.
4. 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.
5. The CUBIT Adaptive Meshing Algorithm Library, Sandia National Laboratories, http://cubit.sandia.gov/camal.html, 2007.
6. The CUBIT Geometry and Mesh Generation Toolkit, Sandia National Laboratories, http://cubit.sandia.gov/, 2007.
7. 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.

8. G. Dhatt and G. Touzot. *The Finite Element Method Displayed*. John Wiley and Sons, 1984.
9. 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.
10. D. A. Field. Laplacian smoothing and Delaunay triangulation. *Communications in Applied Numerical Methods*, 4:709–712, 1988.
11. L. Freitag. On combining Laplacian and optimization-based mesh smoothing techniques. *AMD Trends in Unstructured Mesh Generation, ASME*, 220:37–43, 1997.
12. 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.
13. M. Garland. *Quadric-Based Polygonal Surface Simplification*. Published Doctoral Dissertation, Carnegie Mellon University, May 1999.
14. P. Hansbo. Generalized Laplacian smoothing of unstructured grids. *Communications in Numerical Methods in Engineering*, 11:455–464, 1995.
15. 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.
16. 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.
17. C. Johnson, S. Parker, D. Weinstein, and S. Heffernan. Component-based problem solving environments for large-scale scientific computing. *Concurrency and Computation: Practice and Experience*, 14:1337–1349, 2002.
18. M. Jolley, J. Stinstra, D. M. Weinstein, S. Pieper, R. S. J. Estpar, G. L. Kindlmann, R. S. MacLeod, D. H. Brooks, and J. K. Triedman. Open-source environment for interactive finite element modeling of optimal icd electrode placement. In F. B. Sachse and G. Seemann, editors, *FIMH*, volume 4466 of *Lecture Notes in Computer Science*, pages 373–382. Springer, 2007.
19. T. R. Jones, F. Durand, and M. Desbrun. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics*, 22(3):943–949, 2003.
20. G. Kindlmann. Teem, http://teem.sourceforge.net/, December 2005.
21. 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.
22. P. M. Knupp. Winslow smoothing on two-dimensional unstructured meshes. In *Proceedings, 7th International Meshing Roundtable*, pages 449–457. Sandia National Laboratories, 1998.
23. 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.
24. P. M. Knupp. Algebraic mesh quality metrics. *SIAM J. Sci. Comput.*, 23(1):193–218, 2001.
25. P. M. Knupp. Hexahedral and tetrahedral mesh shape optimization. *International Journal for Numerical Methods in Engineering*, 58(1):319–332, 2003.

26. 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.
27. 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.
28. 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.
29. MESQUITE: The Mesh Quality Improvement Toolkit, Terascale Simulation Tools and Technology Center (TSTT), http://www.tstt-scidac.org/research/mesquite.html, 2005.
30. 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.
31. P. P. Pebay, D. Thompson, J. F. Shepherd, atrick M. Knupp, C. Lisle, V. A. Magnotta, and N. M. Grosland. New applications of the verdict library for standardized mesh verification: Pre, post and end-to-end processing. In *Proceedings, 16th International Meshing Roundtable*, pages 535–552. Sandia National Laboratories, October 2007.
32. C. Scheidegger and J. Schreiner. Afront, http://sourceforge.net/projects/afront/, January 2007.
33. J. Schreiner and C. Scheidegger. Algorithm for separating hexahedra given a triangle mesh. *SCI Institute Technical Report*, UUSCI-2007, 2007.
34. J. Schreiner, C. Scheidegger, and C. Silva. High quality extraction of isosurfaces from regular and irregular grids. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2006)*, 12(5):1205–1212, September-October 2006.
35. J. F. Shepherd. *Topologic and Geometric Constraint-Based Hexahedral Mesh Generation*. Published Doctoral Dissertation, University of Utah, May 2007.
36. 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.
37. 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.
38. J. O. Talton. A short survey of mesh simplification algorithms. manuscript, University of Illinois at Urbana-Champaign, Octoboer 2004.
39. G. Taubin. A signal processing approach to fair surface design. In *Siggraph'95 Conference Proceedings*, pages 351–358, August 1995.
40. P. Vachal, R. V. Garimella, and M. J. Shashkov. Mesh untangling. LAU-UR-02-7271, T-7 Summer Report 2002.
41. The Verdict Mesh Verification Library, Sandia National Laboratories, http://cubit.sandia.gov/verdict.html, 2007.

42. D. Weinstein, S. Parker, J. Simpson, K. Zimmerman, and G. Jones. Visualization in the SCIRun problem-solving environment. In C. Hansen and C. Johnson, editors, *The Visualization Handbook*, pages 615–632. Elsevier, 2005.
43. 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.