Meshing Pipeline Tutorial

BioMesh3D 0.1 Documentation

Center for Integrative Biomedical Computing
Scientific Computing & Imaging Institute
University of Utah

Software download:

http://software.sci.utah.edu

Center for Integrative Biomedical Computing:

http://www.sci.utah.edu/cibc

Supported by:

NIH grant P41- RR12553-07

Author(s):
Brett Burton, Jess Tate, Darrell Swenson, Jeroen Stinstra, Ayla Khan

Contents

Chapter 1

Overview

This manual gives an overview of how to operate the BioMesh3D pipeline in its current state. The BioMesh3D project aim to develop an easy to use program for generating quality meshes for the use in biological simulations. The project's primary goal is to provide a solution that does not require much interaction of the user and automatically generates a proper mesh.

Currently BioMesh3D is available is a prototype pipeline made up of a series of python scripts that are run sequentially that generates a mesh from a segmented volume. Although the plan is to generate a complete user interface for the program, the user interface is not currently available.

Chapter 2

Meshing pipeline stages

2.1 Introduction to Biomesh3D

The Biomesh3D program simplifies the meshing pipeline by breaking it up into eight basic stages described below. Each stage fulfills specific tasks and produces specific outputs for later use downstream. In order to run this pipeline a few software requirements are first essential components, namely Python and SCIRun.

Python is relatively simple to acquire, and SCIRun is an open source software package that can be downloaded from http://www.sci.utah.edu/software.html.

SCIRun software tools are required to run the BioMesh3D pipeline. SCIRun can be built on OSX and Linux platforms using the **build.sh** script that is included in the source distribution. Use **build.sh –help** for usage information. Currently no other dependencies are required to run the meshing pipeline.

Though BioMesh3D will soon have a more streamlined user interface, the current system requires a call to a python script. This python script is generated as part of the SCIRun distribution and can be found in the **bin/FEMesher** directory of SCIRun. The script that the user has to call is the **BuildMesh.py** script. This script will recursively call all the other scripts to complete the mesh.

The **BuildMesh.py** needs a description of the model it will mesh; this model is defined in a second python script that initializes the meshing parameters. This script is a user generated document that defines the path to the segmentation labelmap, the desired output directory, the names of materials to be meshed, and various other parameters discussed in the next section. Examples of these **model_config.py** can be found in the **SCIRunData** dataset in the folder **FEMesher**.

In addition to the model configuration python script, there are 5 other flags that can be used directly after the **BuildMesh.py** call.

-h provides a help menu with a brief description of each stage
-s allows the user to define what stages to run (for example: ...BuildMesh.py -s2:5 will run stages 2 thru 5)
-i executes the Biomesh3D pipeline interactively, pausing between each step and asking if the user would like to continue, while displaying the results within SCIRun.
-d displays the output of key stages of the pipeline using SCIRun without running any of the meshing pipeline. If used alone the files must already exist (that is the pipeline must have already been run) in order for this flag to work
-p path to executables used in the pipeline [only required if the script cannot find the SCIRun programs in the default location].

With the exception of *-h*, each of these flags can be used in tandem. For example *-s2:5 -i* will execute the pipeline interactively from stage 2 to stage 5. Once each stage has been completed, the output will be displayed in SCIRun and the screen will prompt the user to continue. Note: with *-d* the command would need the files in stages 2 thru 5 to already exist.

2.2 User Variables: model_config.py

The `model_config.py` file contains various variables that can be altered by the user, including:

- `model_input_file` - path to the input nrrd file
- `model_output_path` - path to the desired output directory
- `mats` - array from 0 to $n-1$ where n represents the number of materials
- `mat_names` - names the user wishes each material to be called (must be the same size as `mats`)
- `mat_radii` - radius of tightening during the tightening process (if set to 0 tightening will be skipped)
- `refinement_levels` - number of levels that the medial axis will use
- `max_sizing_field` - sets a cap on sizing field. The smaller the number the more dense the final mesh
- `tetgen_joined_vol_flags` - sets parameter flags for tetgen in generating the final mesh
- `num_particles_iters` - defines the number of iterations the particle system will execute
- `max_procs` - set the maximum number of processes run by the particle system

2.3 Initializer: BuildMesh.py

BuildMesh.py initiates the entire pipeline. It receives the command line flags *-h, -s, -i, -d, -p* or, by default, runs the entire meshing pipeline if no flags are set. Once these preliminary decisions are made, this script calls each of the following stages in order.

2.4 Stage 1: MakeSolo.py

MakeSolo.py first accepts the nrrd file defined in the `model_cofig.py` file and pads it with 0 in all dimensions. The newly padded nrrd is unoriented and a transformation matrix is extracted for use in the final stage of the pipeline (in order to realign the final data with the original data). Each material is then isolated and a separate nrrd file created for each material that is tightened (if `mat_radii` $\neq 0$ in the `model_config.py` file) and set aside for the next stage.

Output files are:

- *original-nrrd-name*`_unorient.nrrd`
- *original-nrrd-name*`_transform.tf`
- *material*`.solo.nrrd`
- *material*`.tight.nrrd`
- *material*`.lut.raw`
- *material*`.lut.nrrd`
- `medial_axis_param_file.txt`
- `make-solo-nrrd-runtime.txt`

This stage does not display anything when the *-d* flag is active.

2.5 Stage 2: ComputeMaterialBoundary.py

The material boundary is computed from the tightened nrrd files from the previous stage. These files are converted to SCIRun fld files before their respective isosurfaces are extracted.

Output files are:

- *material*`_isosurface.ts.fld`

- *material*`.tight.fld`
- `isosurface-all.ts.fld`
- `compute-material-boundary-runtime.txt`

This stage displays the isosurfaces of all materials together when the *-d* flag is active. SCIRun users can view individual materials by adjusting which field is called in the ReadField module's user interface.

2.6 Stage 3: ComputeMaterialMedialAxis.py

Both the speed and the quality of the meshing pipeline are critically dependent on the quality (accuracy AND consistency) of the medial axis computation. We have implemented a medial axis algorithm that extracts the medial axis by way of point cloud approximation. The current method produces a point cloud over the entire volume and computes which points are closest to the medial axis. These points are then used to produce another homogeneously spaced point cloud in the region of each previously computed medial axis point where it is again checked for closeness to the medial axis. This process iterates through a user-specified number of levels, each time honing in closer to the desired medial axis. Due to the number of levels and the amount of point cloud particles produced, this stage can take a long time.

Output files are:

- *material*`_ma.ptcl`
- *material*`_ma.pc.fld`
- `ma-all.pc.fld`
- `compute-material-medial-axis-runtime.txt`

This stage displays the medial axis points of each material within the isosurfaces of each material together when the *-d* flag is active. SCIRun users can view individual materials by adjusting which field is called in the ReadField module's user interface.

2.7 Stage 4: ComputeSizingField.py

The medial axis points previously generated are used to generate a sizing field nrrd file for each material based on the distance of the medial axis to the isosurface of the material. Also, a zero-crossing algorithm is used to isolate the boundary of each material generated in the MakeSolo.py script.

Output files are:

- *material*_crossing.nrrd
- *material*_lfs.nrrd
- *material*_sf_init.nrrd
- *material*_sf.nrrd
- compute-sizing-field-runtime.txt

This stage displays the sizing field and associated grid of only one material at a time when the *-d* flag is active. SCIRun users can change which material they are looking at by adjusting which field is called in the ReadField module's user interface.

2.8 Stage 5: GenerateSeeds.py

Junctions where materials meet are determined and seed points are randomly placed along these junctions. A dual junction (where two materials meet) creates a surface, the filename has a *d* in front of it followed by the two material labels. Triple junctions create lines represented by *t* and the three material labels involved, and quadruple junctions make points (*q* followed by four material labels).

Output files are:

- generate-seeds-runtime.txt
- *seeds/*

  d01_seed.pc.fld

  d01_seed.ptcl

  etc...

  t012_seed.pc.fld *if there are triple interfaces*

  t012_seed.ptcl

  etc...

  q0123_seed.pc.fld *if there are quad interfaces*

  q0123_seed.ptcl

  etc...

  seeds-all.pc.fld

This stage does not display anything when the *-d* flag is active.

2.9 Stage 6: DistributeParticles.py

Using the sizing field as an energy metric (the smaller the sizing field value, the less energy) the seeds that were randomly placed on the surface of the material adjust themselves to minimize energy. In general, the closer a seed is with its neighbor, the more energy exists between them. These particles will move away from each other in an attempt to minimize energy while still being forced to remain on the surface. By moving they interact with other particles and move again in an attempt to minimize energy. The sizing field determines the energy that each particle has as they adjust themselves into a state of lowest energy. Once the amount of iterations specified in the `model_config.py` file are met, the algorithm quits.

Output files are:

- `distribute-particles-runtime.txt`
- `ops-output-#.txt`
- *input_seeds/*

  `d01_seed.pc.fld`

  `etc...`
- *junctions/*

  `psystem_input.txt`

  `particle_params.txt`

  `m1.txt`

  `particle_params.txt_#.pts`

  `particle_params.txt_#.ptcl`

  `particle_params.txt_#.pcv.fld`

  `particle_params.txt_#.pc.fld`

  `particle-all.pcv.fld`

  `particle-all.pc.fld`

This stage displays the final position of all particles for each material together when the *-d* flag is active.

2.10 Stage 7: BuildMaterialInterfaceMesh.py

Using the particle system points, surfaces are generated that share nodes.

Output files are:

- `build-material-interface-mesh-runtime.txt`
- *junctions/*

  `particle-union.pts`

  `particle-union.node`

  `particle-union.m`

  `particle-union.ts.fld`

  `particle-union.1.node`

  `particle-union.1.face`

  `particle-union.1.ele`

  material`.m`

  material`.ts.fld`

This stage does not display anything when the *-d* flag is active.

2.11 Stage 8: BuildVolumetricMesh.py

Tetgen is used to create a volumetric mesh of each material and of the materials together as a joined field based on the surfaces produced in stage 7.

Output files are:

- `build-volumetric-mesh-runtime.txt`
- *junctions/*

  `particle-union.tets-labeled_transformed.fld`

This stage displays the tetrahedralized mesh of the entire material with an overlaid quality tet mesh that uses a scaled Jacobian measure when the *-d* flag is active.

Chapter 3

Limitations and known issues

- **Data will be node centered throughout pipeline** - Though data can be node centered or element centered, the pipeline (which accepts both) will ultimately node center all data; thus, a shift will be perpetuated in any element centered data that is used.
- **Performance** - The current system has not yet been optimized and as a result the system may take a few days to render a mesh for a large dataset. For example on our system a typical torso segmentation (512 by 512 by 400) takes, on average, one week to run to completion. Furthermore, significant hard drive space is required for these datasets (10 GB for said torso meshes). Smaller datasets with only two materials should require much less space (less than 1GB) and will be much quicker (less than a day).
- **3D Data only** - BioMesh3D does not compensate for data of other dimensions.
- **Thin segmentations produce inaccuracies** - Blood vessels and other thin, narrow (approx. 1 - 2 voxels thick) might be eliminated by the tightening algorithm. Tightening can be turned off by setting $mat\_radii = 0$ in the $model\_config.py$ file.
- **Interactive mode over forwarded X11 connections** - SCIRun has been known to crash when volume rendering over forwarded X11 connections, therefore we recommend against running BioMesh3D in interactive mode over an X11 connection. Instead run the pipeline automatically and copy the files to your local machine and run the pipeline in the -d mode, which only will do the visualizations and use all the precomputed files.
- **Large data sets require more memory than is available** - A temporary, hard coded fix has been implemented for relatively large datasets by multiplying the binning radius by 4. This will be changed to allow for a more dynamic handling of method so that large data sets will automatically reduce the size of the binning radius and allow for proper memory management.

Chapter 4

Preparing data for meshing pipeline

4.1 Segmentations with Seg3D

Segmentations of many kinds of data can be done using Seg3D. With this program domains can be identified and assigned values from images, such as various tissue types from an MRI scan. Seg3D will read many formats of slice data such as a DICOM file and other formats generated by imaging scanners. After segmenting each of the tissue types, Seg3D can then export the segmentation in nrrd format, which is used by BioMesh3D. A brief demonstration of how to use Seg3D is included in the tooth example (Chapter **??**), but for a more extensive explanation of the tools and capabilities of Seg3D, please see the Seg3D documentation.

Chapter 5

Example dataset: Mickey

To demonstrate the functionality of BioMesh3D, we will use a simple, low-resolution model. This example will walk you through how to access the example data sets and use SCIRun to create a 3D Mickey model (head and ears of Mickey Mouse).

5.1 Example Data Sets

To obtain the mickey data set, download the **SCIRunData.zip** file from *http://software.sci.utah.edu/* The SCIRunData set can be found in the SCIRun4.2 file/data repository. All the example files are stored in the *FEMesher* directory.

Select the mickey folder and download the two files (*mickey-unorient.nrrd* and *model_config.py* into the directory from which you would like to work. The *.nrrd file is our data file that is to be meshed. The *model_config.py* file is a python script full of variables that the user is able to manipulate as described above in Chapter **??**.

For BioMesh3D to work on your data set, the *model_config.py* file must point to the mickey file as well as an output directory (it does not matter whether the output directory is the same as the input directory). Open the *model_config.py* file and point the 'model_input_file' and 'model_output_path' variables at the proper paths. These paths can be relative to the *model_config.py* file. The default settings assume that the segmentation file is in the same directory as the *model_config.py* file.

Open a terminal from which you can run command line code. Type the command described in the first section of Chapter **??** (if you are already in the SCIRun directory, there is no need to include the path to your SCIRun directory). The files will generate themselves as the program runs. If you would like to view the stages as they progress use the *-i* flag. If you wish to only run certain stages is the *-s* flag; however, if this is a fresh run, you will have to start from stage 1 in order to get any results.

The following results were obtained by using the following command: From the SCIRun directory

bin/BuildMesh.py -i
<path-to-mickey>/Mickey/model_config.py

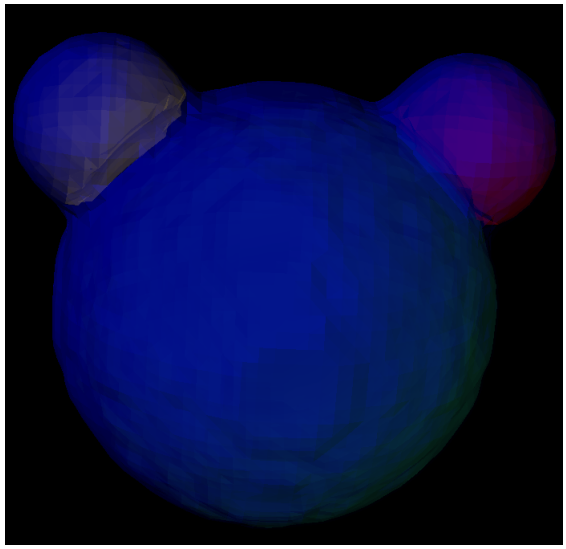

Figure 5.1. Stage 2: Extract Material Surfaces - Isosurfaces of the three materials that make up mickey (head, left-ear, and right-ear)

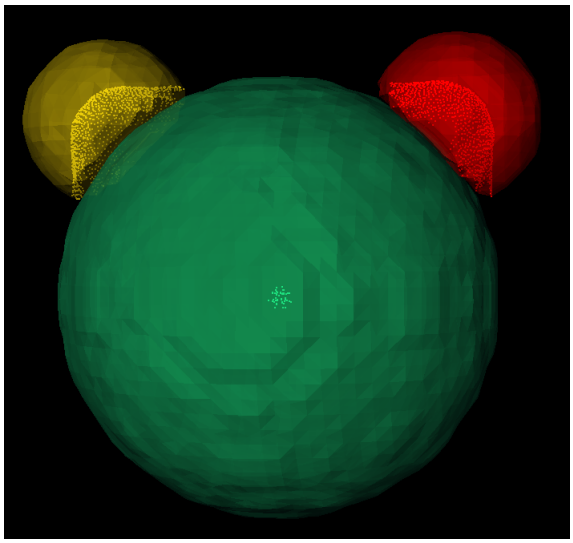

Figure 5.2. Stage 3: Compute Medial Axis - Medial axis points of the mickey.

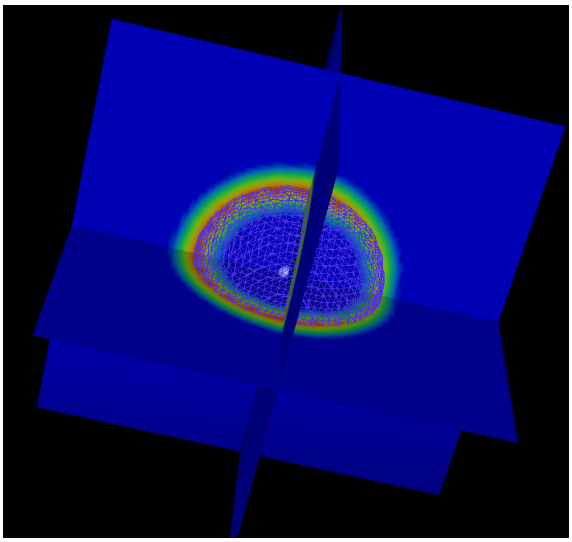

Figure 5.3. Stage 4: Compute Sizing Field - Computed from the medial axis, the sizing field is a distance map from the medial axis points to the surface of the material. Sizing field files are only shown one material at a time (this being the head).

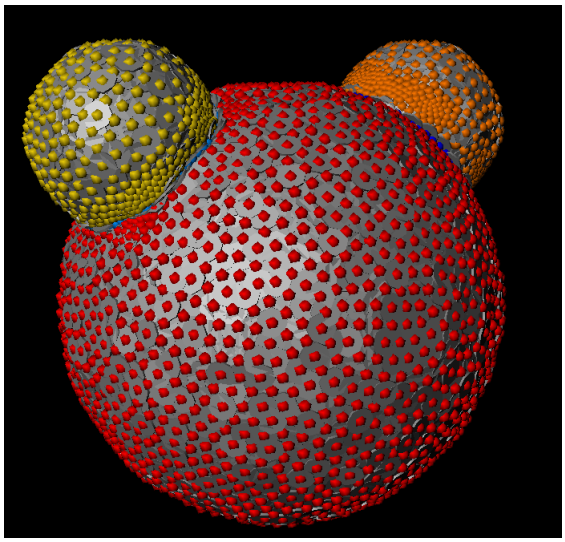

Figure 5.4. Stage 6: Run Particle System - A high resolution particle system (that is when max_sizing_field and SIZING SCALE VAR values are low) will produce tightly packed particles such as these.

Stage 7: Generate Surface Mesh (no visualization)

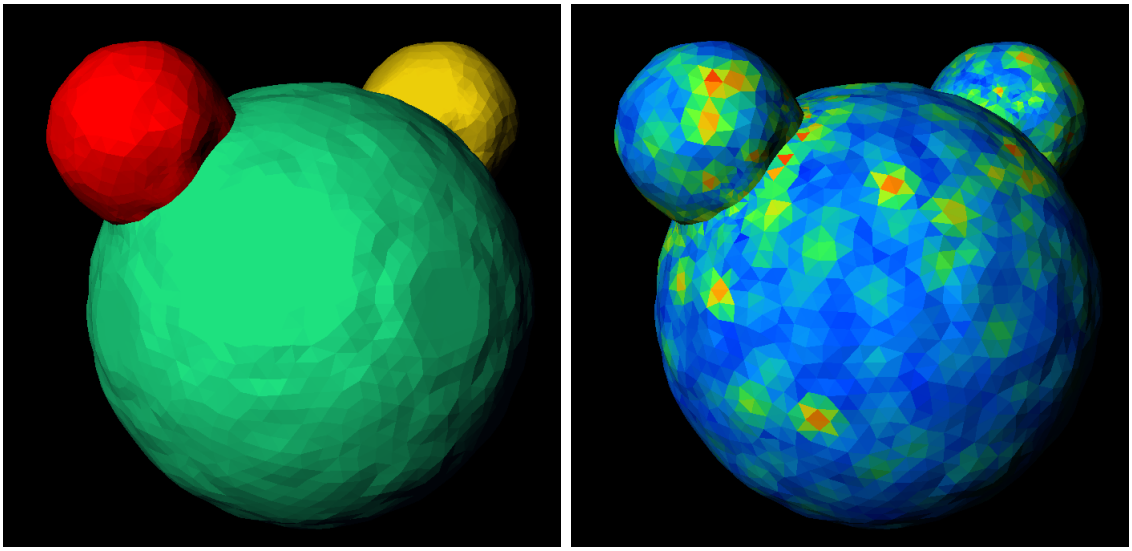

Figure 5.5. Stage 8: Generate Volume Mesh - Final mesh (left) and associated quality field (right). Red tetrahedra are higher quality than blue.

Chapter 6

Example dataset: Tooth

This Example will walk you through the basics of generating a mesh from raw data. There are two basic steps: creating the segmentation and running the BioMesh3D script. The data file is available in the SCIRunData/FEMesher/tooth/volume-tooth/ directory. Similarly, in the SCIRunData/FEMesher/tooth/ directory, there is a **tooth.nrrd** file that is similar to the one that will be created in the tutorial in the following section. Therefore, the first section of the tooth tutorial, wherein Seg3D is used to make a segmentation can be skipped. However, Seg3D is a useful tool that is compatible with BioMesh3D and SCIRun, so it is encouraged to use this tutorial or the tutorial provided for Seg3D to become familiar with it if the user intends to make meshes from raw scan data.

6.1 Creating a segmentation using Seg3D

Open Seg3D and create a new project. Under the 'File' menu, choose 'Import Layer From Single File ' and open *tooth.nhdr* that was just downloaded or from the path mentioned above. Import the file as a 'Data Volume.' A grayscale volume will appear. Become familiar with the basic controls and navigation in Seg3D. Be sure to 'save session' often as you segment to ensure that you do not lose any data. For further instruction in the operation of Seg3D, please consult the Seg3D documentation.

With the tooth volume loaded, perform a median filter on the data. This filter will make it easier to perform segmentation tasks later. This is done by clicking 'Median' option under the 'Data Filters' menu. Leave the radius at 1 and press the Run Filter button. Another layer will appear containing a blurrier but smoother tooth volume. Notice that the three main intensity regions of the tooth are more uniform (Figure **??**). we will label each of these three regions to be identified by BioMesh3D and other programs.

We will begin with the darkest region of the tooth. Select 'Neighborhood Connected' under the 'Data Filters' menu. This filter will use seed points to label bordering pixels of similar values. To set the seeds, simply click on the dark region of the tooth (Figure **??**). Set enough seed points that will encompass enough value range

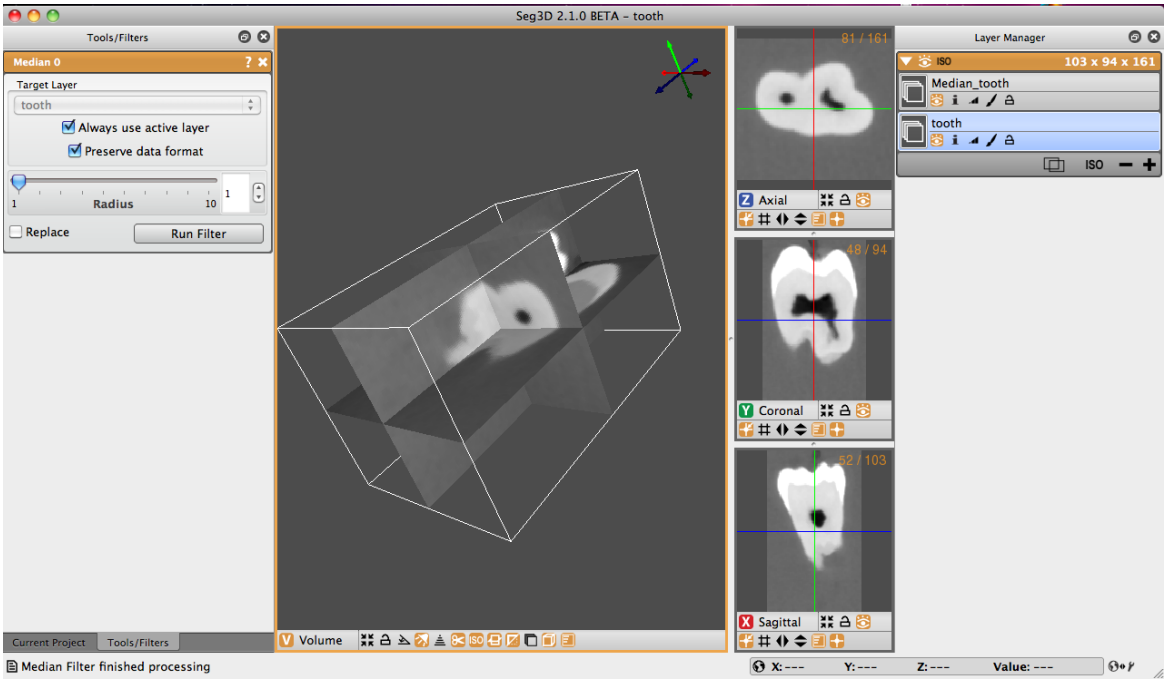

Figure 6.1. Tooth volume after Median Filter.

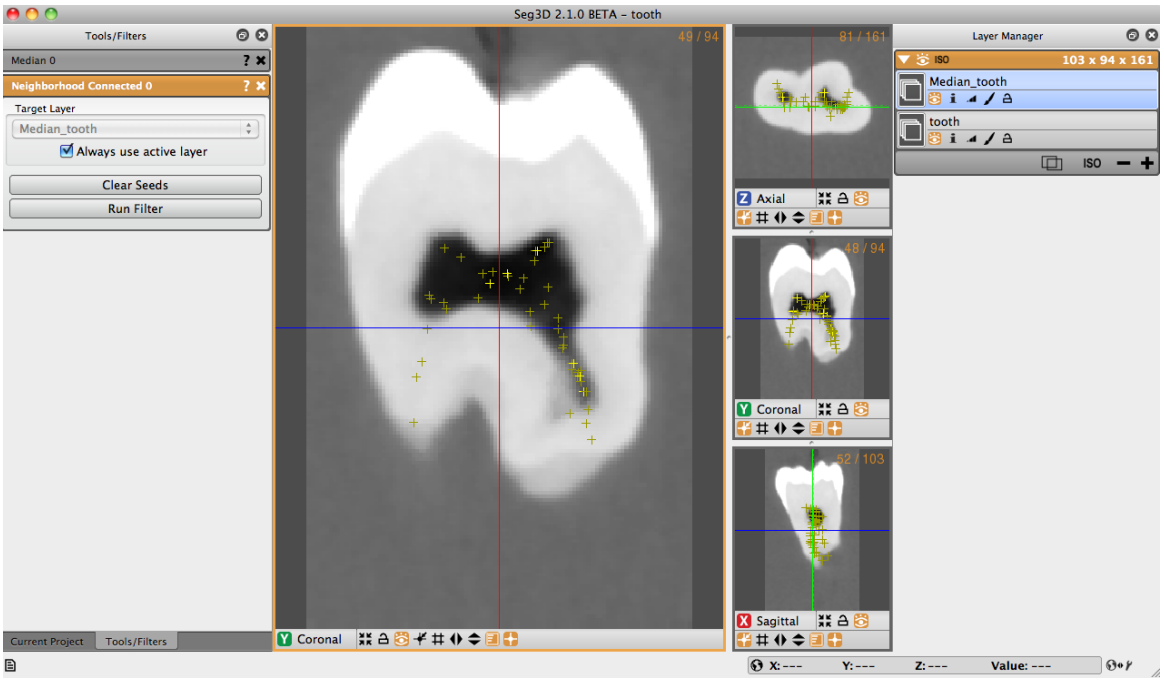

Figure 6.2. Choosing the seed points.

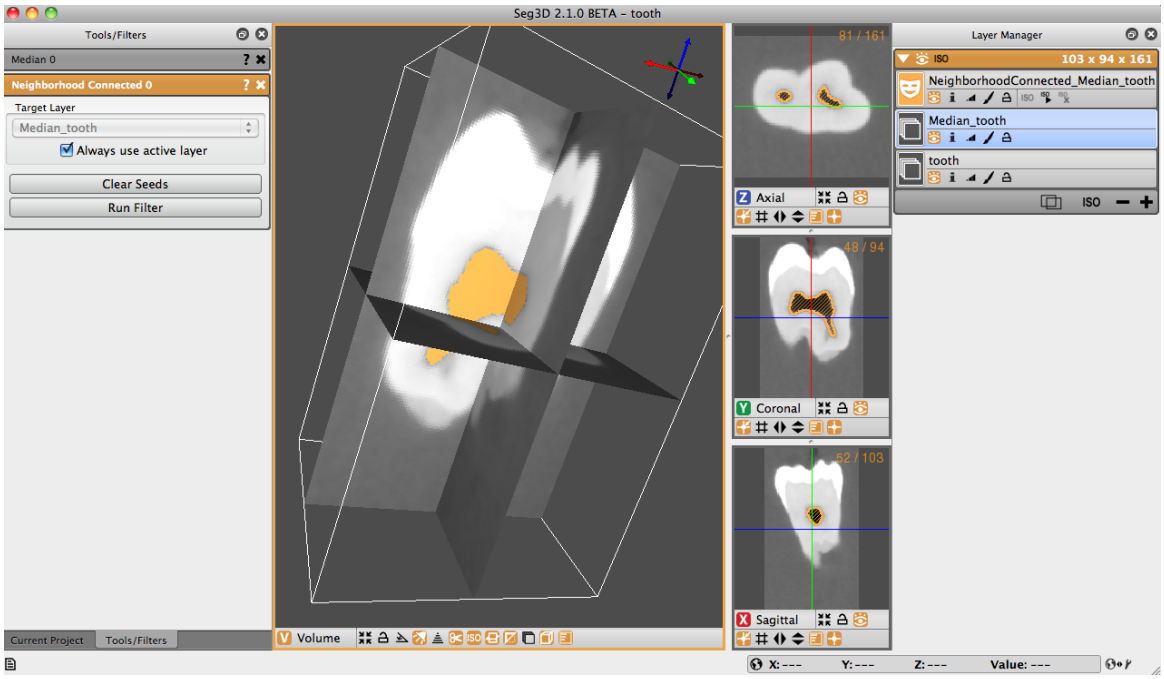

Figure 6.3. Darkest region labeled.

to get the area wanted. Scroll through multiple slices using the arrow keys while the cursor hoovers over the view you want to change, and then set seeds on more than one slice.

This will probably require many seed points and some experimenting. Keep running the filter and adding more seed points as needed (the seed points will be saved until they are cleared) until the entire darkest area is covered by the label (Figure **??**). Delete all the extra labels that were generated while you practiced. Now you have one of the regions finished. Rename this label to 'DarkRegion'.

For the second region, select 'Threshold' under the 'Tools' menu. This tool will label all the pixels in the volume that have a value between the set thresholds. The thresholds can be set by seed points, but there are also slider bars to select values to label. Choose threshold values that will label the entire tooth volume except the darkest regions (Figure **??**). Make sure that the area selected by the threshold filter overlaps the label that you made for the darkest regions slightly. Press the 'Run Filter' button. Rename the layer to 'Threshold'.

To get rid of the overlap, choose the 'BooleanRemove' tool from the 'Mask Filters' menu. Set the Target Layer to 'Threshold' or the layer you just made. Set Mask Layer to 'DarkRegion' or the layer you made from the darkest region. Run this filter by clicking 'Run Filter'.

(Figure **??**). A new label should appear that does not overlap called 'REMOVE_Threshold' if you have renamed the labels as described above.

Open the 'Threshold' tool once again. Now set the threshold to select the highest

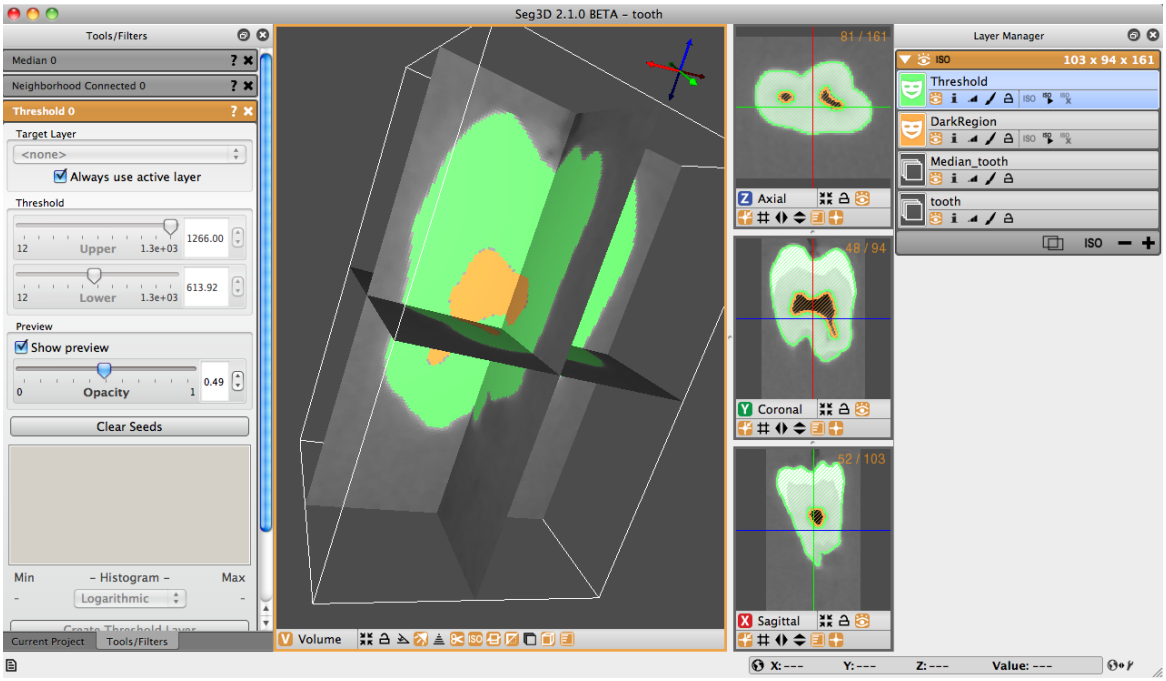

Figure 6.4. Selecting the tooth using the threshold tool.

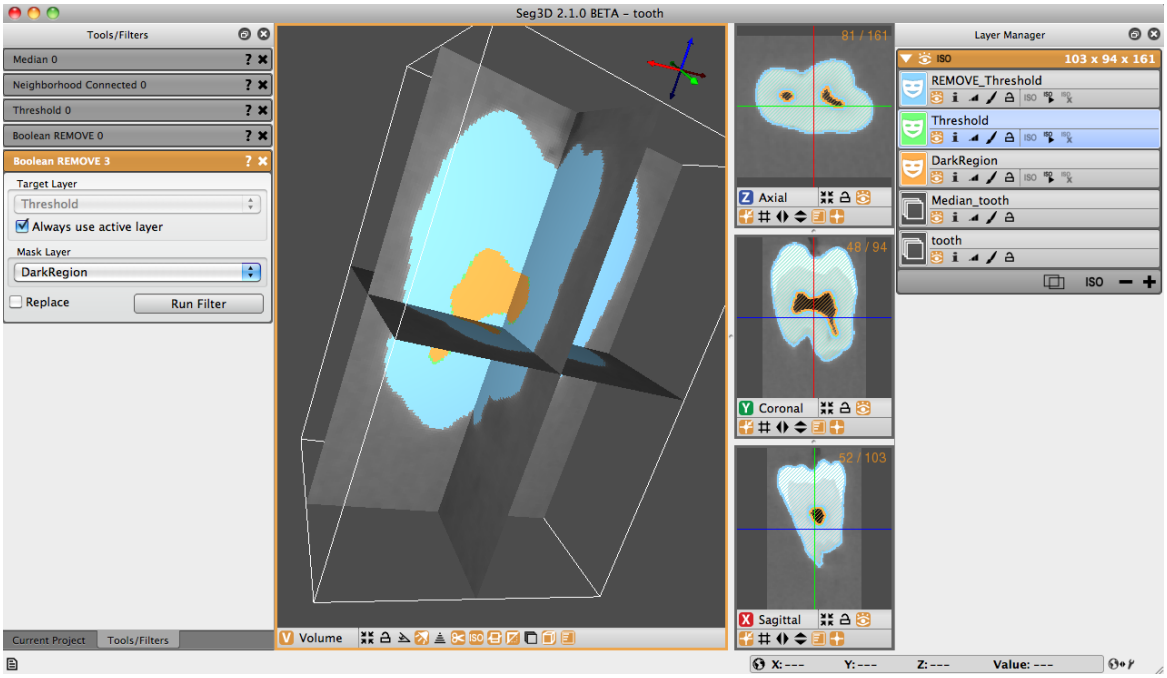

Figure 6.5. Removing the overlapping regions using 'Boolean Remove'.

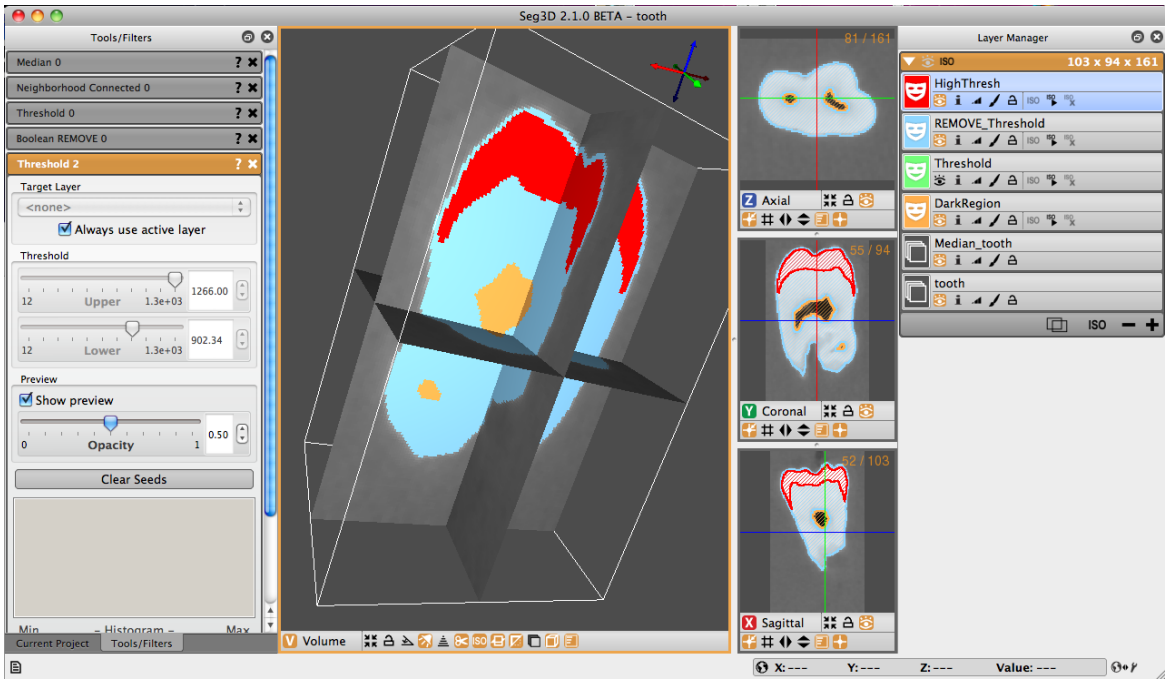

Figure 6.6. Using the paint brush tool to modify the labels to map areas not selected by the filter.

intensity region in the tooth. Make sure that it is as big a threshold as possible without getting stray pixels. The label for the whitest region is meant to extend to the top of the tooth so that there is none of the previously made layers above it. If this is not the case, then open the 'Paint Brush' from the 'Tools' menu and set the bigger label as 'Mask Constraint 1'. It will probably be called 'REMOVE_ Threshold' under the 'Edit' menu. Now simply paint the remaining pixels to extend the white region (Figure **??**). The mask will help make painting easier, but this still has to be done slice by slice, so the painting can be skipped. It just makes the mesh look nicer. Rename the layer to 'HighThresh'.

Once the label for the whitest region is fully selected, use the 'BooleanRemove' tool again. Select the Target Layer as 'REMOVE_ Threshold' and the Mask Layer as 'High Threshold'. Also select the 'Replace' box before clicking the 'Run Filter' button. Rename REMOVE_ Threshold to MiddleThresh. There should be three labels for each of the obvious regions in the tooth (Figure **??**). They should not over lap or contain any gaps between them. If there are, the mesh can still be run through BioMesh3D, but it won't be as pretty. Once the three regions of the tooth are correctly labeled 'Export Segmentation' under the 'File' menu. Uncheck the group box and then select the three of the correct labels, HighThresh, MiddleThresh, and DarkRegion. Then click Continue. Give the output .nrrd a name like 'tooth' and verify that the labels are numbered as you would like and then click Done. This is the file that will be used in subsequent steps. Once the segmentation is exported from

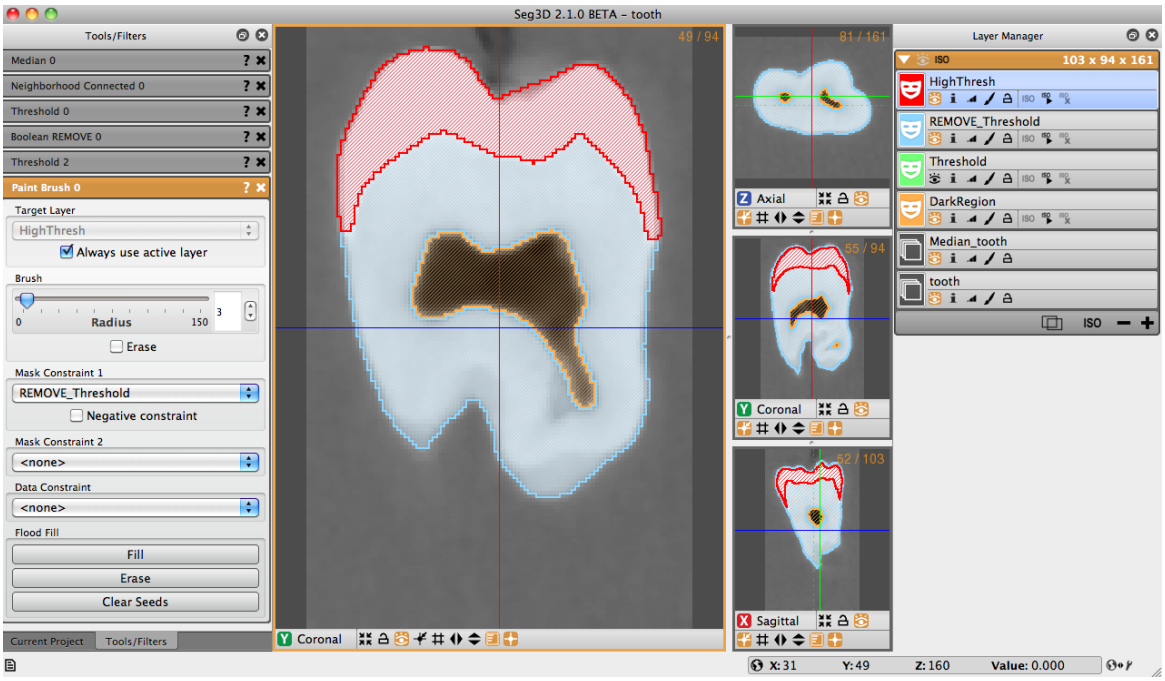

Figure 6.7. Final three regions segmented, ready for export.

Seg3D, it can run through BioMesh3D.

6.2 Running BioMesh3D

The actual running of BioMesh3D does not require much work on your part, just start it and let the computer crunch through it. The *model_config.py* setup file needs to be edited to fit your machine and project. There are examples of this configuration file in the directory **bin/FEMesher**.

It is suggested that a *model_config.py* file be made for each dataset run in BioMesh3D.

There are several variables in the *model_config.py* that need to be set.

The variables are as follows:

- model_input_file: The path for the mesh that will be input into the pipeline, in our case the segmentation of the tooth (.../tooth.nrrd).
- model_output_path: The path for the output files that will be generated by BioMesh3D (.../tooth).
- mats: The mask or data values of the materials in the input fields. In the case of our tooth segmentation, there are four materials: the three different regions

that we segmented, and the air around it. These are all assigned a value by Seg3D beginning with 0. With the tooth file that we are using, we will assign this variable (0,1,2,3).

- mat_names: The names of the materials labeled in the segmentation. Each material labeled in the mats variable needs to have unique name. For simplicity, we will assign the mat_names variable ('air', 'mat1', 'mat2', 'mat3').

- mat_radii: This variable is a parameter for the tightening that is performed in the second stage to make the meshes smoother. To turn off tightening, such as for very thin (one or two voxels thick) meshes, set this parameter to 0. For the tooth file that we are using, we can leave the default value 0.8.

- refinement_levels: This will modify the generation and refinement of the medial axis points. A higher value will allow for more refinement, and thus medial axis points will be closer together. This variable can be set higher if the segmentation is very small or narrow.

- max_sizing_field: Modifies the sizing field algorithm. A lower number generates more final elements on the tetrahedral mesh.

- SIZING_SCALE_VAR: A lower number generates more final elements on the tetrahedral mesh.

- num_particle_iters: The number of iterations used in the particle system in stage 6 of the pipeline. The particle system generates equally spaced points to generate tetrahedra. For the tooth.nrrd file, set the variable to 500.

- tetgen_joined_vol_flags: This variable is similar to the previous, except it applies to the tetrahedral models generated for all the materials combined into a single file. The flags available in this option are the same as the previous variable. It can be left as "zpAAqa10".

- max_procs: Caps the number of processes used by the particle system (stage 6).

Once the *model_config.py* file is set to your liking, you're ready to run the script to execute the pipeline. The command to run the script has three parts: the path to the BuildMesh.py script and the path to the *model_config.py* file.

Open a terminal window, and type the following as one command:

[...]**bin/BuildMesh.py** [*flags*] [...] **/model_config.py**

To see the other flags available for BioMesh3D, please refer to Section **??**.

The script may take a while to finish, depending on the system you are using and the size of the dataset. The tooth file will probably take between twenty minutes and an hour. If it takes longer, then it is likely that the script crashed and needs to be

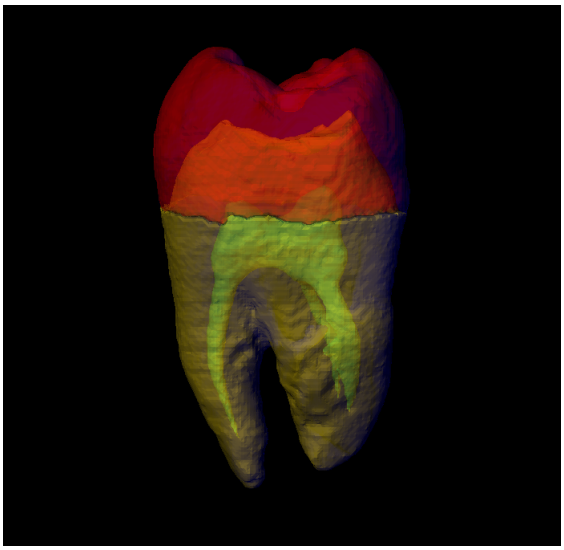

Figure 6.8. The output of Stage 2. This is the isosurface of each of the material type.

restarted. Look for error messages as the script runs and check the parameters in the *model_config.py* file accordingly.

Once BioMesh3D confirms that it is complete, look in the junctions directory in the defined output directory. There will be a file named particle-union.tets-labeled_transformed.fld, which is a tetrahedral mesh labeled with the values given for each material in the *model_config.py*.

6.3 BioMesh3D Output

The following images are the output of each of the display stages (using the the -d flag). To see these images, it is recommended that the entire script run to completion, then re-execute the script with the *-id* flags. This will show the displays without re-executing the script.

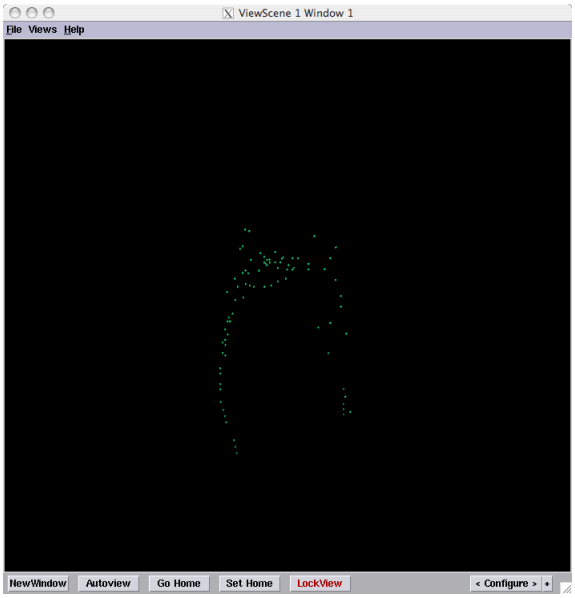

Figure 6.9. The output of Stage 3. These points are the medial axis points. The medial axis points are generated for each material, but only the first material is shown by default.

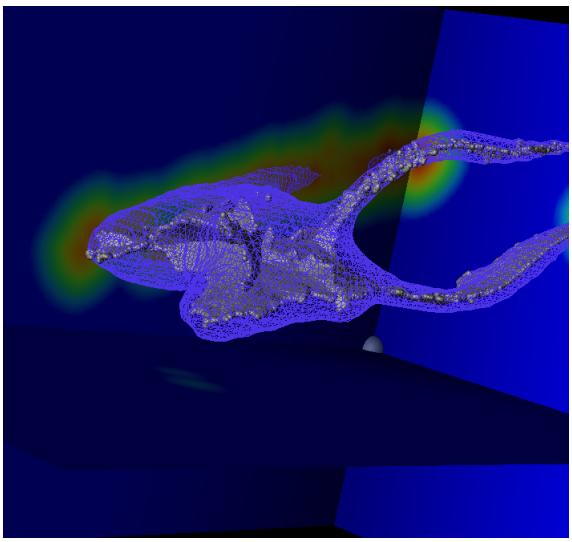

Figure 6.10. The output of Stage 4. This shows the sizing field and the medial axis points meshed together.

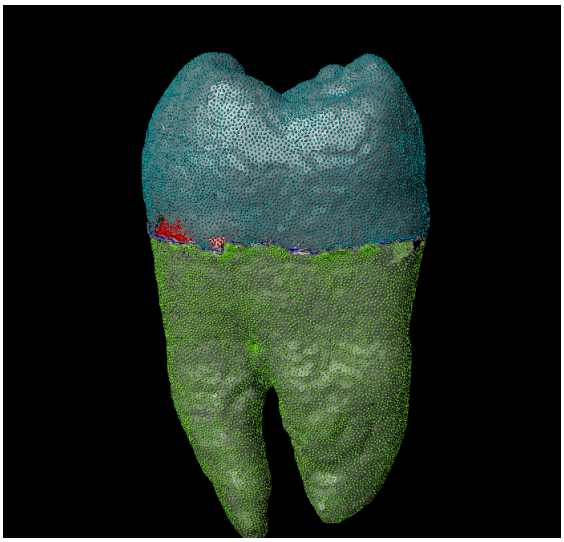

Figure 6.11. The output of Stage 6. The particles shown are the points used to generate the tetrahedral meshes.

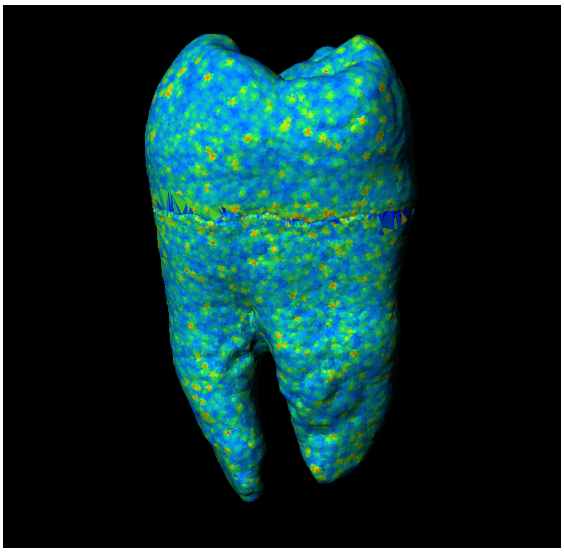

Figure 6.12. The final mesh generated by BioMesh3D. The color map represents the mesh quality of the tetrahedra. Red is high and blue is low.