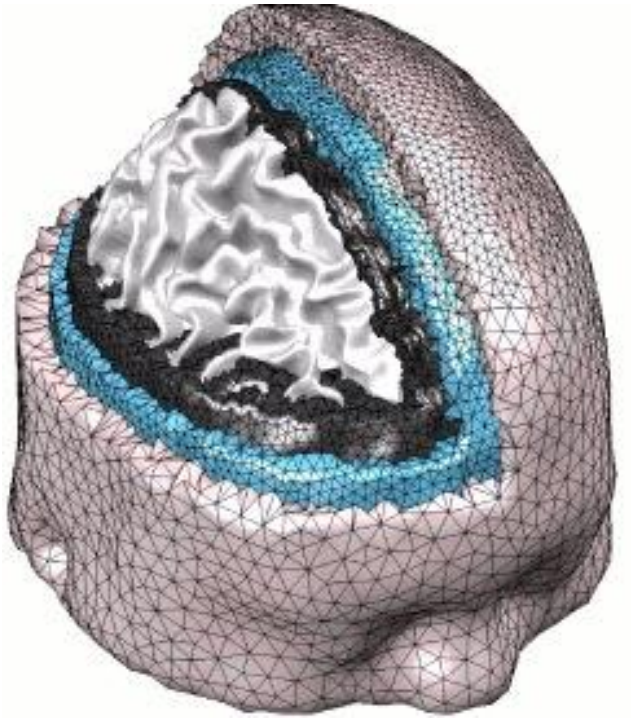
IBSim-4i 2021 Training

Hands-on: Image-based Mesh Generation with iso2mesh



Contents

Introduction

This hands-on training session is focussed on the process of mesh generation and processing from images using iso2mesh, an open-source library of Matlab/Octave-based scripts. The toolbox can generate 3D tetrahedral or surface meshes from surfaces, 3D binary and grayscale volumetric images for variety of numerical simulations (for instance, FEM, CFD, FVM). iso2Mesh is developed by Qianqian Fang, an Associate Professor at the Department of Bioengineering, Northeastern University, USA. Iso2mesh has been used and discussed in hundreds of research papers (https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=iso2mesh) and is still being developed.

If, after following this training course, you use iso2mesh in your work please cite the following papers:

- Anh Phong Tran, Shijie Yan and Qianqian Fang*, (2020) "Improving model-based fNIRS analysis using mesh-based anatomical and light-transport models," Neurophotonics, 7(1), 015008
- Qianqian Fang and David Boas, "Tetrahedral mesh generation from volumetric binary and gray-scale images," Proceedings of IEEE International Symposium on Biomedical Imaging 2009, pp. 1142-1145, 2009

Most of the tutorials in this training session are based on the examples presented in the official iso2mesh website (http://iso2mesh.sourceforge.net/) and/or in the code repository. The others have been developed by the ZCCE IBSim group at Swansea University with the use of volumetric image data sets provided by Dr Llion Evans.

The datasets used in these tutorials have been pre-processed to be ‘clean’ to facilitate the smooth running of the training sessions. Additionally, they are at lower resolutions than would normally be used to accommodate PCs/laptops with lower specifications. Pre-processing of images can be accomplished with software packages such as ImageJ (https://imagej.net/software/fiji/) but this is outside the scope of this tutorial.

The format of the tutorials is that a problem or challenge will be described that iso2mesh will be used to solve. A template script will be provided which needs modifying to complete the task. An accompanying solution script is also provided for you to check your approach. It is recommended that you do not look at these until you have completed the task or if absolutely necessary in order to progress.

Using the software

Both Octave and iso2mesh have been included in a pre-prepared Virtual Machine (VM), based on Ubuntu 18.04 environment, that is provided for use during this training course. To use this VM please first download and install Oracle's VirtualBox (https://www.virtualbox.org/wiki/Downloads). This should work on all popular operating systems (OSs) other than ARM based Macs. If you already have VirtualBox installed, it is recommended that you update to the latest version.

After installing VirtualBox you will need to download and import the VM we will use for the course.

1. Download the VM image (see link below) and note the location to where it is saved. https://ibsim.co.uk/VirtualLab/downloads/VM.html
2. To import the VM:
    a. Open VirtualBox.

Figure 1. VirtualBox icon.

    b. **'File > Import Appliance…'**

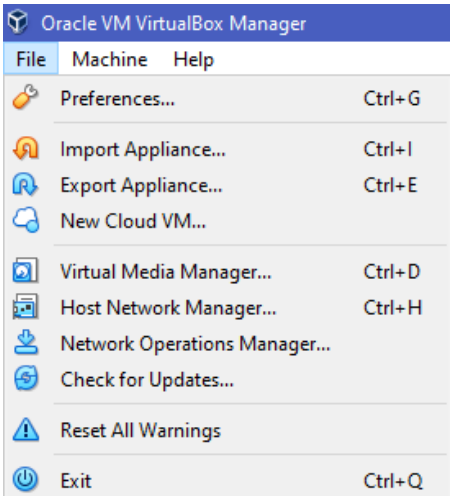


Figure 2. VirtualBox menu.

    c. In the 'File' field of the 'Import Virtual Appliance' popup, navigate to where you saved the downloaded VM image and select this file (ubuntu_VL-OS.ova), click **'Open'** then **'Next'**.
    d. On the following screen you may accept the default settings by clicking **'Import'**. You may wish to change the location of the 'Machine Base Folder' to a hard drive disc (HDD) with sufficient availability of empty storage. Importing may take a few minutes, depending on the speed of your system.

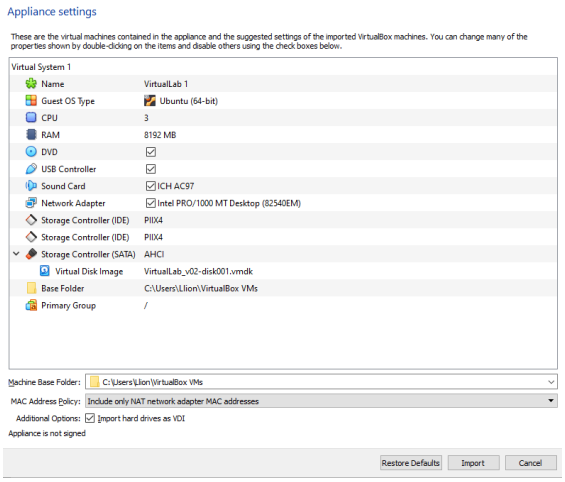


Figure 3. Appliance settings menu.

e. Once imported, you may launch the VM by double clicking the appliance (listed on the left-hand side of the VirtualBox window) or by first selecting the VM then clicking **'Start'**.

Figure 4. Button to launch the VM.

NOTE: If your machine doesn't meet the minimum requirements, you will need to amend the VM's settings from within VirtualBox before launching it.

Figure 5. Button to open the VM's settings.

The main settings you may wish to amend are:

- The amount of 'Base Memory' allocated from your system's RAM to the VM (under **'System > Motherboard'**).
- The number of 'Processors' allocated from your system's CPU to the VM (under **'System > Processor**).
- The amount of 'Video Memory' allocated from your system's CPU to the VM (under **'Display > Screen'**).

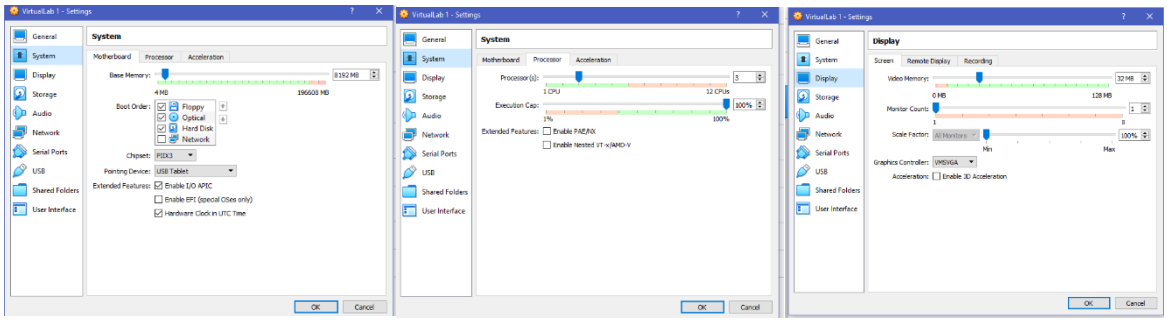


Figure 6. VirtualBox's VM settings menus.

Once the VM has launched and booted you will be presented with a full Ubuntu desktop environment. You may use this in much the same manner as your own system.

Usually, setting the display to scale to 100% is best (**'View > Virtual Screen 1 > Scale to 100%'**). However, if you encounter visualisation issues, it is recommended to change the resolution of both your host machine and the VM so that they match (e.g., 1920 x 1080).

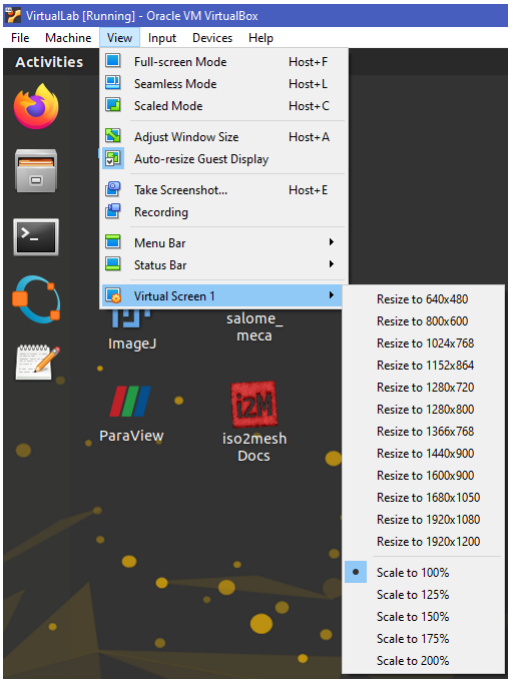


Figure 7. Options to change display settings if needed.

It is recommended that you enter full screen mode to have a near native operating system (OS) experience and in order for software to display at the correct resolution (**'View > Full-screen Mode'**).

You may be required to enter a password if the lockscreen is activated due to inactivity.

Username: ibsim
Password: ibsim

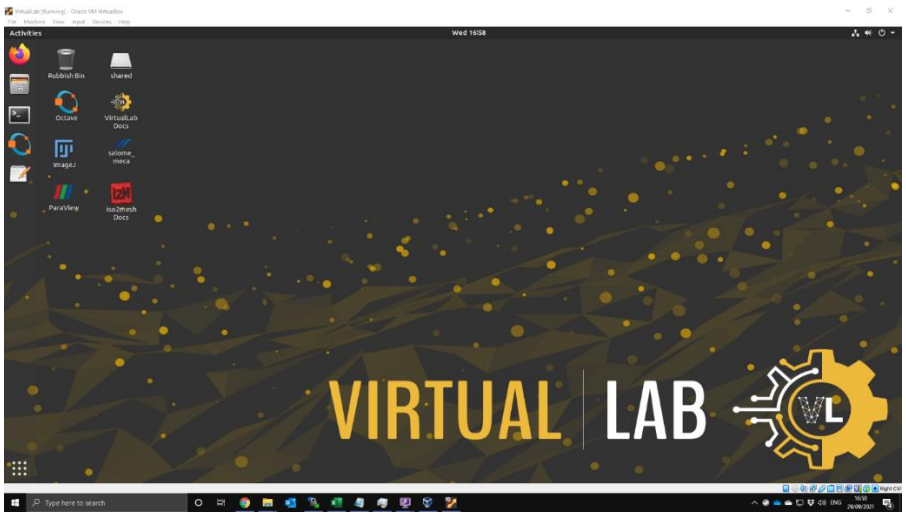


Figure 8. Ubuntu desktop environment within VirtualBox VM, can also be run in full screen to hide the host OS.

Tutorial-I: Introduction to iso2mesh

Overview:

The main purpose of ***tutorial1_surf2mesh_ex1.m*** is to demonstrate how to use the ***surf2mesh*** function to mesh an open-ended surface. In the ***tutorials*** folder, the data for a surface is available in the file ***tube_surface.mat***. This file contains the data to describe the surface of a pipe-like structure. The data consists of two variables, ***v*** and ***f***, where each variable is a three-column array (matrix).

v - the cartesian nodal coordinates of a point cloud that describes the pipe

f - elemental connectivity data, that is, which three nodes (points) make a triangle on the tessellated surface of the pipe.

The goal is to write a Matlab/Octave script which uses the iso2mesh toolbox to generate a mesh from the surface of this pipe up to a limit prescribed by a bounding box. In brief, this is achieved in two steps (with an optional third):

a) Load the surface data
b) Mesh the volume from the surface data up to a given bounding box
c) Visualise the results (optional)

Learning outcomes:

- Getting used to Octave environment and iso2mesh scripts.
- Loading image data and creating a surface mesh from it.

Steps to follow:

1. First start Octave by opening it from the 'start menu' (square containing nine dots in the bottom Left-hand corner) and searching within applications or by double clicking the icon on the desktop.

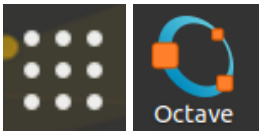


Figure 9. 'Start menu' and Octave icons.

   This will open Octave's Graphical User Interface (GUI) as shown in Figure 1 (some aspects, such as the colour scheme, may differ).

2. In Octave, the 'Current Directory' has been set to open at launch within the iso2mesh tutorials directory (folder), where you can find the tutorial files and image data sets.

   On the left-hand side (Figure 109), the 'File browser' will show the files inside this directory.

   For the first tutorial, open ***tutorial1_surf2mesh_ex1.m*** from within the 'File Browser' (Figure 109) by right clicking and select 'Open in Text Editor', this will open the file on the right-hand side in the 'Editor'. This is a script file including commands calling functions in the iso2mesh libraries.

NOTE: The percentage sign '%' is used in Matlab/Octave to denote a comment. This is used to mark a section of code that is not to be executed. You may use this to turn 'on/off' sections of the script.

NOTEL Semicolons ‘;’ are used at the end of lines to denote that the output to the console should not be verbose. That is, calculated values will be printed to the console for lines not ending in a semicolon. Whilst this may be desirable for debugging, it can severely slow the execution of a script.

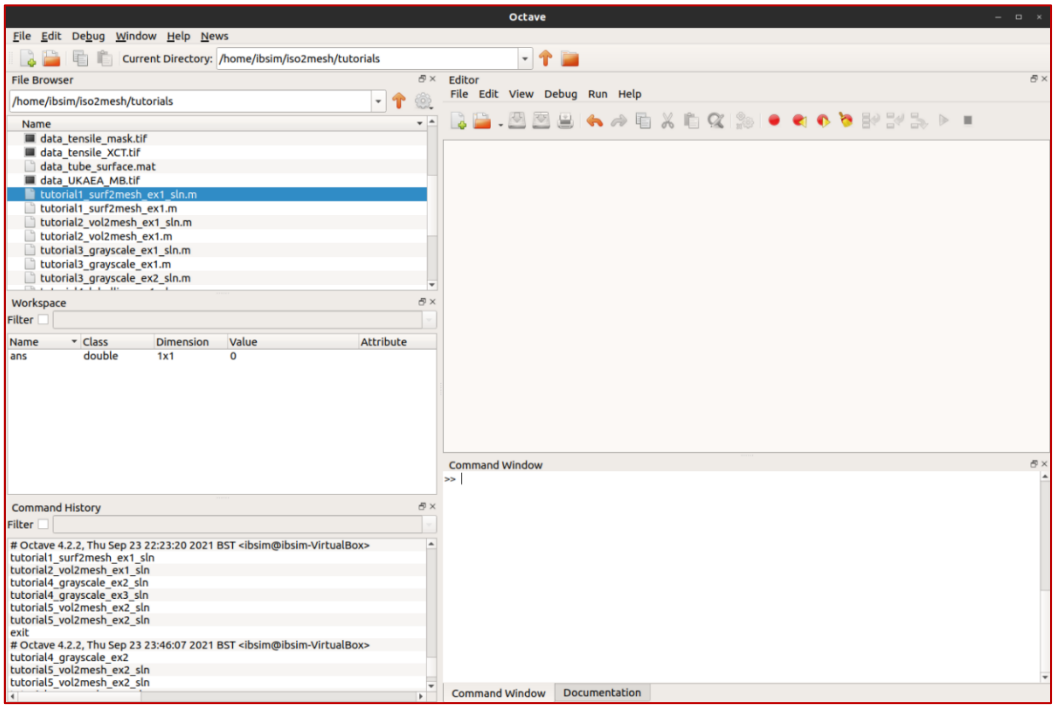


Figure 10. Octave Graphical User Interface.

3. The first thing we wish to do is load the surface data (i.e., the surface path faces, ***f***, of the open-ended surface and their corresponding nodes, ***v***). Uncomment the line “%load <filename.mat>;% in the script and change <filename.mat> to ‘data_tube_surface.mat’.

NOTE: In the scripts for these tutorials, we will use ‘<variable>’ to denote sections of code which need editing to complete the tutorial.

4. In order to visually check this data, we may use the ‘trisurf’ function which reads in the points and connectivity to construct a surface.

   Uncomment the trisurf line (you do not need to amend any code) and run the script. You may do this by pressing ‘F5’ on your keyboard or by clicking the ***save file and Run*** button.

   If successful, a window will pop open with a visualisation of the data. Take this opportunity to familiarise yourself with the viewer. It is very basic (we will look at a more accomplished alternative in a later tutorial) but serves the purpose of being able to perform a quick check.

   Once you are satisfied with the data, close the figure window and re-comment the trisurf line by placing a ‘%’ at the start.

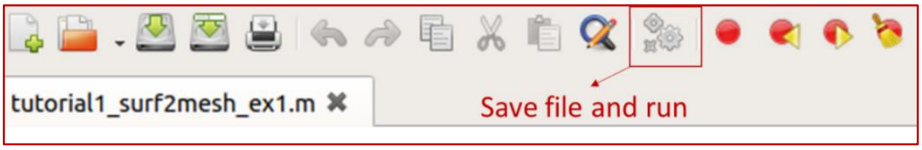


Figure 11. Button to click in order to ‘save and run’ the file in the editor.

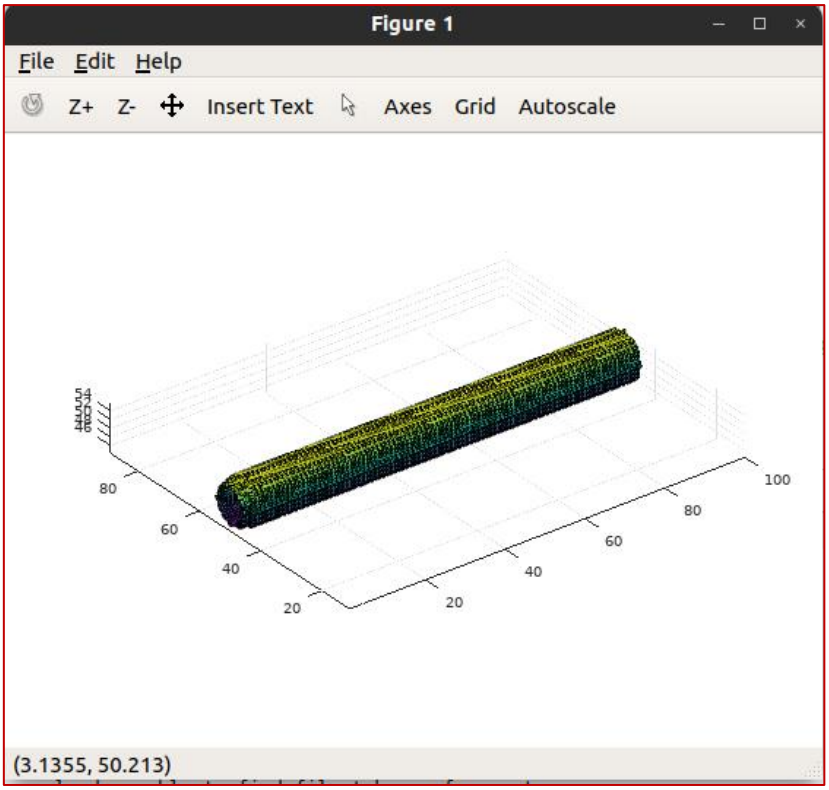


Figure 12. The open-ended surface.

5. Next, we call the ***surf2mesh*** function to perform volumetric meshing of the domain around the pipe surface up to a limiting bounding box.

   [node, elem, face] = surf2mesh(v, f, p0, p1, keepratio, maxvol, regions, holes, forcebox)

   These are the input arguments used on the right-hand side within the round brackets:

   i. ***v***: the cartesian nodal coordinates of a point cloud that describes the pipe
   ii. ***f***: elemental connectivity data, i.e., which three points make a triangle on the tessellated surface of the pipe
   iii. ***p0***: coordinates of one corner of the bounding box, p0=[x0 y0 z0]
   iv. ***p1***: coordinates of the other corner of the bounding box, p1=[x1 y1 z1]
   v. ***keepratio***: input, percentage of elements being kept after the simplification
   vi. ***maxvol***: input, maximum tetrahedral element volume
   vii. ***regions*****:** list of regions, specifying by an internal point for each region
   viii. ***holes***: list of holes, similar to regions
   ix. ***forcebox***: 1: add bounding box, 0: automatic

   The function returns these variables on the left-hand side within the square brackets:

   i. ***node***: nodal coordinates of the tetrahedral mesh
   ii. ***elem***: element list of the tetrahedral mesh
   iii. ***face***: mesh surface element list of the tetrahedral mesh

For this particular case, this is how we should fill the input arguments of surf2mesh:

- ***v*** and ***f*** are the imported data from the .mat file and can be passed directly to the function with the same variable names.
- We wish to use a 100 cube bounding box, therefore the x0, y0, z0 and x1, y1, z1 values should be set to 1 and 100, respectively.
- As it has been decided to keep 10% of the original elements during remeshing, ***keepratio*** should be set accordingly to '0.1'.
- Set maxvol of the tetrahedral element volumes to 25 unit^3.
- Because of the simplicity of this problem, the last three input arguments (regions, holes, forcebox) are skipped on this occasion.

NOTE: As we have not scaled the data to any unit system, processes will use a single unit to mean the width of a pixel or voxel (3D pixel).

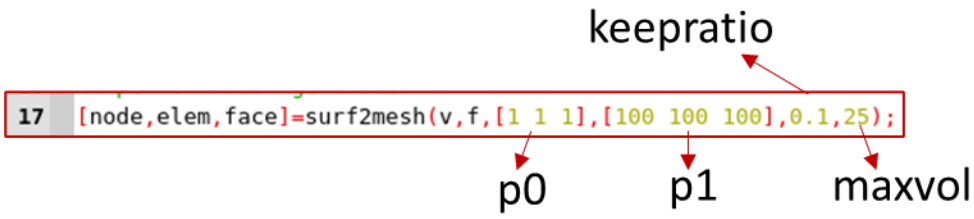


Figure 13. Line from script calling the surf2mesh function and how the argument syntax should be formatted.

6. Uncommon the '`plotmesh`' line so that the resultant mesh is visualised.

   After running the script (F5 or 'Save & Run' button) we should obtain a visualisation of the meshed domain surrounding the pipe, shown in Figure 1413.

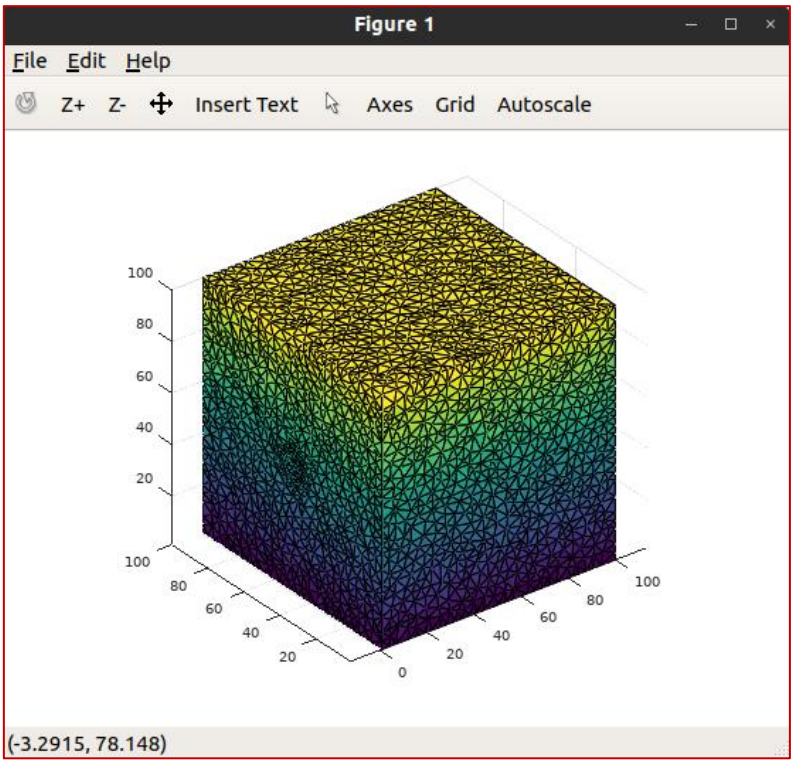


Figure 14. The surface mesh of open-ended surface.

7. You may further explore the functionality of ***surf2mesh*** by changing some of the arguments. For example, try changing the bounding box size to see what happens if you make it longer in x, e.g., set to 200, or too small for the surface in z, e.g., set to 50. Another suggestion it so make the mesh finer or coarser by changing the maxvol size, e.g., 2.5 or 250.

   NOTE: Be careful not to set the arguments to generate a mesh which is too fine, or your PC may not have sufficient resources and your script may crash. If you want to cancel a script which is being executed, click in the 'Command Window' and press 'Ctrl+c'.

Tutorial-II: Mesh generation from binarised volumetric data

Overview:

The aim of this tutorial is to mesh pre-segmented 3D image data with three different methods. The data in question is of a rat head stored in the matlab/octave matrix file format (***data_rat_heat.mat***).

Learning outcomes:

- Generating meshed bodies from binarized volumetric images such as X-ray or MRI images.
- Compare differences between ***v2m*** and ***vol2mesh*** functions.
- Comparing resultant meshes.

Steps to follow:

NOTE: Instructions already covered in previous tutorials will not be repeated in detail. Please refer back to previous sections to review these if needed.

1. If not already open, launch the IBSim VirtualBox VM, as directed in Tutorial-I.
2. First, we will visualise the rat heat image in the open source image processing and analysis platform ImageJ. We will be using the Fiji variant of ImageJ, which has a number of plugins for volumetric data pre-loaded.
    a. Open ImageJ from the VM desktop and the ImageJ menu will appear (if asked to update ImageJ, you can ignore this).

Figure 15. Image J menu.

    b. To open the import menu, click **'File > Import > Image Sequence…'**. Accept the default options. Browse to the directory containing the image sequence of tiff images in '/home/ibsim/tutorials/data_rat_head_TiffStack/' and select yes.

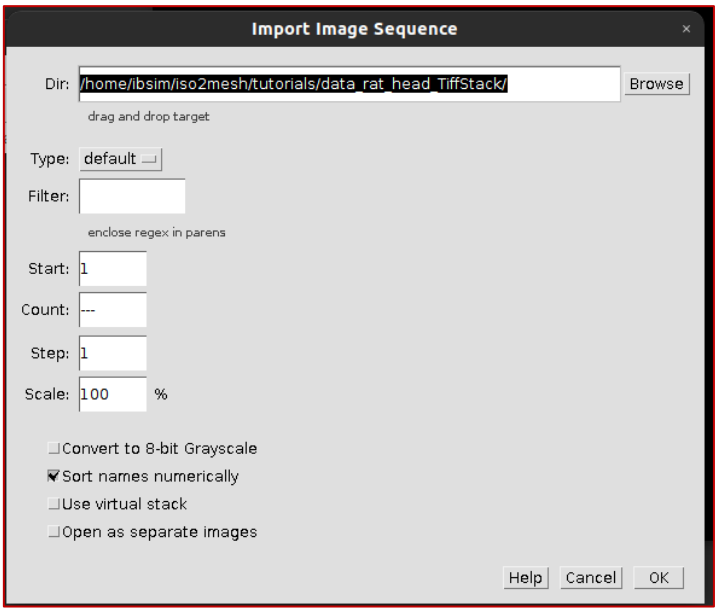


Figure 16. Image sequence import menu in ImageJ.

c. The data will open in a window which you can slide through to view 2D slices (use the magnifying glass to make the window larger).

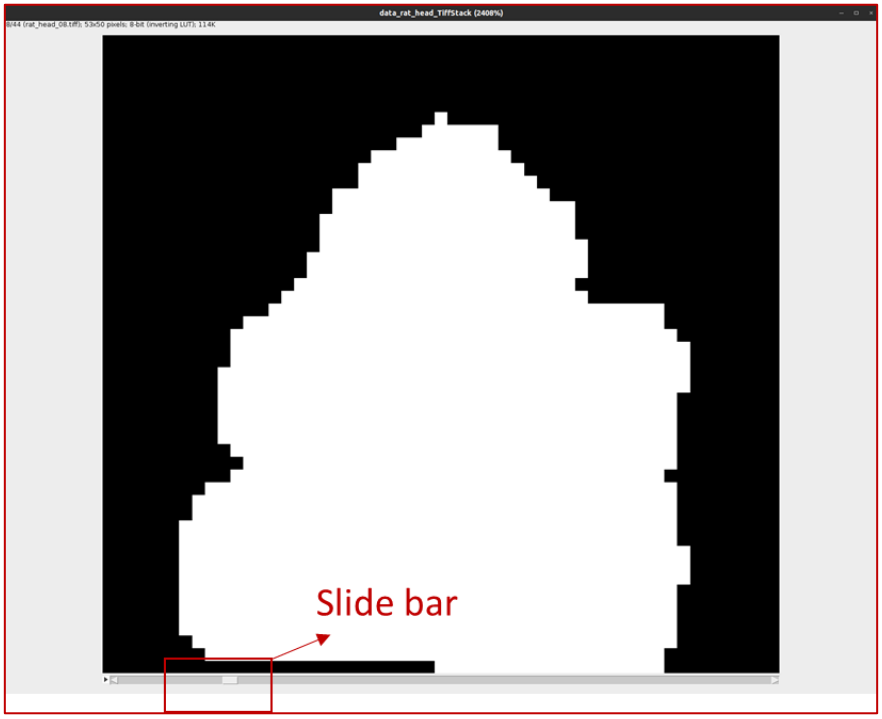


Figure 17. Image slice from binarized volumetric image of a rat head.

d. Then use *3D Viewer* in *Plugins* tab to visualise the data in 3Dto do this with the 2D view window still open go to **'Plugins -> 3D viewer'**. In the new window change the 'Resampling factor' to 1 then click OK. You will see the 3D rat head (you can rotate the view by dragging the mouse cursor).

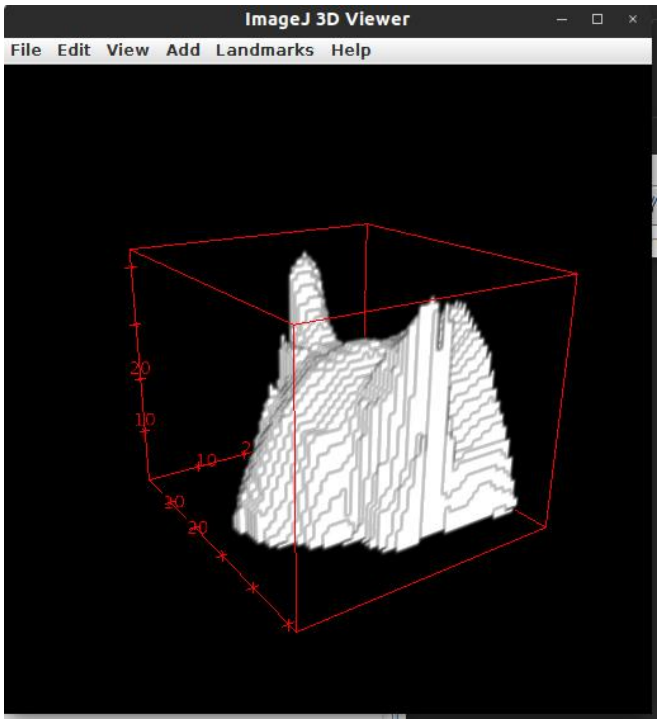


Figure 18. 3D rendering of the volumetric data.

3. Open Octave as directed in Tutorial-I.
4. In the *File Browser* (Figure 1), right click + 'open in editor' on the script tutorial2_vol2mesh_ex1.m to open the template for this tutorial.

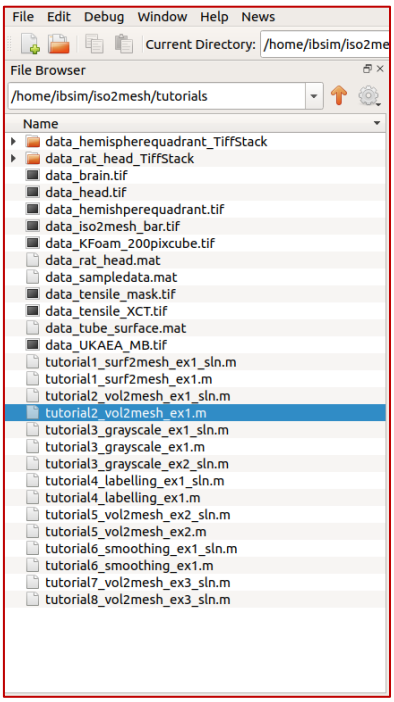


Figure 19. Open tutorial2 in *File Browser*.

5. Firstly, set the script to load the volumetric image data ***data_rat_head.mat*** by amending the <fname> with data file name.

   Once the image data has been read into the 'Workspace' (shown on the left-hand side), the ***img_vol*** variable with dimensions of 50x53x44 will appear. These are the x,y,z dimensions of the volumetric image.

   This image, which has been pre-segmented, only contains two phases: background (air) and foreground (rat head). As such, were you to inspect the ***img_vol*** matrix it would consist of zeros and ones to denote background and foreground voxels, respectively.

6. Visualise a single slice in z-direction from the segmented volume data (***data_rat_head.mat***) by amending *<zslice_number>* (for instance, 22). Do not forget to uncomment the lines starting with *img_slice* and *imagesc* by removing '%'. Click the 'save and run' button to see a single slice.

   Alternatively, the same method can be applied in the other directions, x and y, meaning in the first and second indices of the vector *img_vol*.

   After visualising the single slice, comment out the two lines back to their original form.

7. In this tutorial we will use the ***vol2mesh*** function and ***v2m*** simplified function to perform volumetric meshing of the pre-segmented rat head. The input & output arguments of ***vol2mesh*** are as shown below:

[node, elem, face] = vol2mesh(img, ix, iy, iz, opt, maxvol, dofix, method, isovalues)

This function returns 3 arrays (or matrices):

node	nodal coordinates of the tetrahedral mesh
elem	element list of the tetrahedral mesh
face	mesh surface element list of the tetrahedral mesh

Uncomment the appropriate line and amend the ***vol2mesh*** arguments to perform meshing of the image data:

img	This should be set to the variable name containing the image data (i.e., img_vol in this instance).
Ix, iy, iz	We wish to use the whole volume, therefore each of the three parameters should be set to the appropriate span (e.g. 1:xmax). The easiest way to accomplish this is to use the 'size' function to return the dimensions of the data i.e. 1:size(img_vol,1),1:size(img_vol,2),1:size(img_vol,3)
opt	This changes depending on the 'method'. In this instance it is the maximum radius of the Delaunay sphere (relating to element size). Smaller values yield finer meshes. Try setting this to '2' initially.
maxvol	Where there is less detail (e.g., away from the surface) the mesh can be coarsened to reduce the element count and thus save on computational expense. Try setting this to '2' initially.
dofix	Set this to perform mesh validation and repair. (0 = off, 1 = on)
method	(Optional) cgalsurf is the default if omitted, which is what we want here.
isovalues	(Optional) omit on this occasion.

Table 1: opt argument options

opt: argument options

if *method* is 'cgalsurf' or 'cgalpoly':
opt=a float number>1: max radius of the Delaunay sphere(element size)
opt.radbound: same as above, max radius of the Delaunay sphere
opt.distbound: maximum deviation from the specified isosurfaces
opt(1,2,...).radbound: same as above, for each levelset
if *method* is 'simplify':
opt=a float number<1: compression rate for surf. simplification
opt.keepratio=a float less than 1: same as above, same for all surf.
opt(1,2,..).keepratio: setting compression rate for each levelset
opt(1,2,..).maxsurf: 1 - only use the largest disjointed surface
0 - use all surfaces for that levelset
opt(1,2,..).side: 'upper': threshold at upper interface
'lower': threshold at lower interface
opt(1,2,..).maxnode: - the maximum number of surface node per levelset
opt(1,2,..).holes: user specified holes interior pt list
opt(1,2,..).regions: user specified regions interior pt list
opt(1,2,..).surf.{node,elem}: add additional surfaces
opt(1,2,..).{A,B}: linear transformation for each surface
opt.autoregion: if set to 1, vol2surf will try to determine the interior points for each closed surface automatically

Opt is one of the most complex arguments in ***vol2mesh***. Depending on the meshing method selected, it is used differently. You can find further information in Table 1.

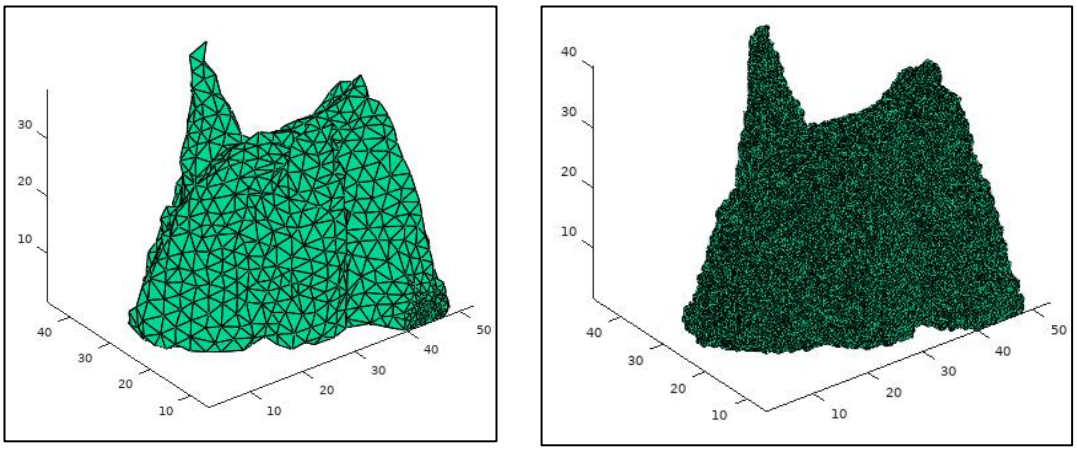
After setting the arguments for ***vol2mesh*** run the script in the usual manner.

To visualise the resultant mesh (optional), do not forget to uncomment the *plotmesh* function:

```
%% Visualise the meshed volume
plotmesh(node, face); axis equal;
```

8. Try using different values for ***opt***, which will result in different mesh densities most visible on the surface (see Figure 2019). You may also try varying ***maxvol*** which will change the mesh internally. You will not be able to observe changes with the current visualisation method, this will be shown in a later tutorial.

   Some details about the resulting mesh can be found in ***command window*** of ***OCTAVE***.



(a) (b)

Figure 20. Discretised rat head with coarse (a) and fine meshes (b).

9. Next, we wish to test out the ***'simplify'*** method with the ***vol2mesh*** function.

   To accomplish this, add the appropriate argument to the ***vol2mesh*** function call to override the default 'cgalsurf' option when no method is explicitly set (i.e., add this as an additional final argument), then re-run the script.

   What differences can you observe between the resultant mesh and the previous mesh generated by the default method ('cgalsurf')? This is due to the different meshing algorithm used and the ***opt*** argument being used differently for each method. See above for more details on how ***opt*** controls the meshing. Try using different values for ***opt***, e.g., 0.1.

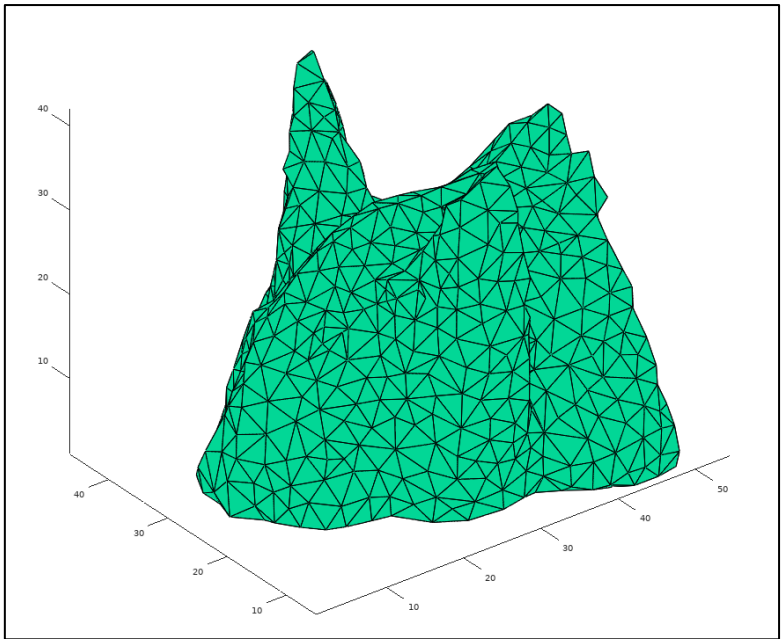


Figure 3. Discretised rat head created by the 'simplify' meshing method (opt:0.1, maxvol:2, dofix:1).

10. Finally, we will use ***v2m*** to test the 'cgalmesh' method. ***v2m*** is a simplified call providing a shortcut to ***vol2mesh*** requiring fewer arguments to be defined (and therefore offers a little less control over meshing).

    The input arguments for ***v2m*** (definitions as above):

[node, elem, face]=v2m(img, isovalues, opt, maxvol, method)

    'Comment out' the call to ***vol2mesh*** by placing a '%' character at the start of the line and the ***plotmesh*** line below. Then uncomment the ***v2m*** call and associated lines to the end of the script.

    NOTES:
    a) 'clear opt' is used to ensure that a previously set 'opt' value is not used.
    b) 'opt.radbound=2' can be used to set ***opt*** outside of the ***v2m*** call (which may be desirable, e.g., if setting multiple argument options for ***opt***), meaning that the variable name 'opt' can then be used as the argument input.

    Amend the arguments to be appropriate for use with the cgalmesh method:

img	The cgalmesh method requires data to be formatted as 8-bit unsigned integers. The data in the img_vol matrix can be converted with uint8(variable_name).
isovalues	Set this to '0.5' initially.
opt	See above note.
maxvol	Try setting this to '100' initially.
method	Set to cgalmesh.

Re-run the script and observe the differences in the mesh. It is worth noting the mesh data in the ***command window*** because, even if visibly there is little change, there can be a significant change in the number of elements due to internal changes.

Try using different values for ***opt***, ***isovalues*** and ***maxvol***.

Generally speaking, 'cgalmesh' is the most robust method if you want to produce meshes from binary or multi-region volumes. However, its limitations include:

1) only accept 8-bit unsigned integer data format.
2) cannot extract meshes from grayscale volumes (i.e., unsegmented data).

If one's goal is to process a grayscale volume, they should use the 'cgalsurf' option. The 'simplify' approach is not recommended unless the other options have failed.

Tutorial-III: Mesh generation directly from greyscale image data

Overview:

The aim of this tutorial is to generate meshes directly from grayscale data for two examples: a metallic multi-layer ring (Example 1) and a graphite foam cube (Example 2), both obtained from XCT imaging.

Learning outcomes:

- Generating meshes directly from real volumetric grayscale images, e.g., X-ray or MRI images

Steps to Follow:

Example 1:

1. First, we will visualise the data **data_UKAEA_MB.tif** in ImageJ as described in tutorial 2, but this time import the data as a TIFF Virtual Stack rather than an Image Sequence.

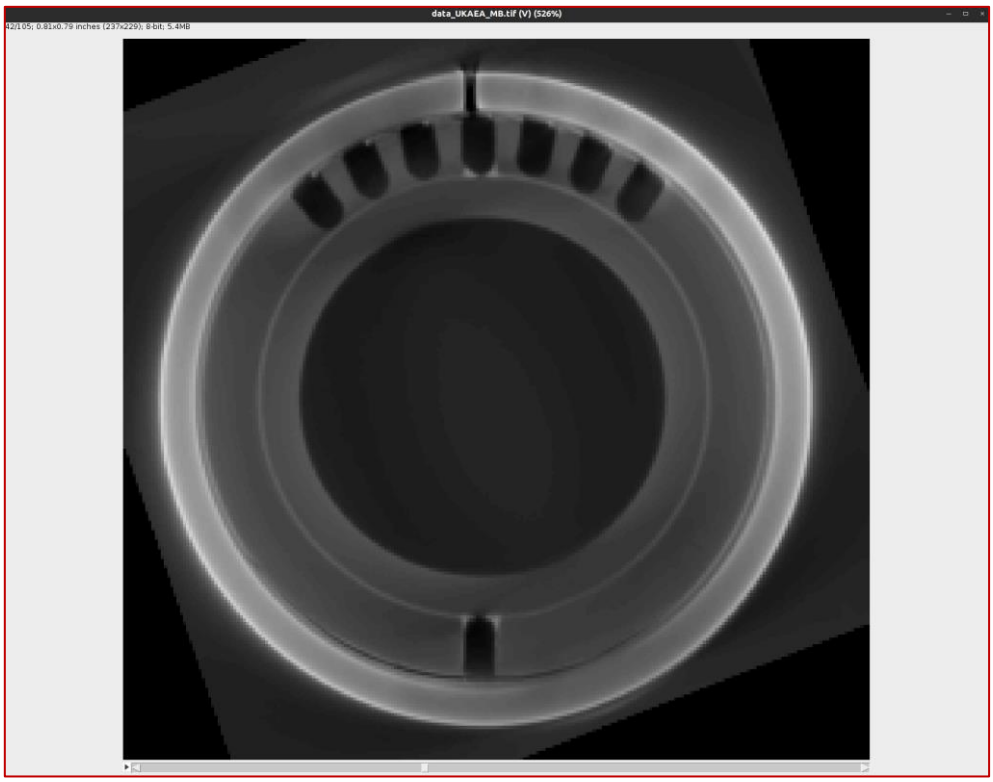


Figure 21. CT image of a metallic multi-layer pipe ring.

2. The data will open in a window which you can slide through to view 2D slices as seen in Figure 2120 (use the magnifying glass to make the window larger).
3. You can use ImageJ to get the number of slices in the image which (will be used in this tutorial).
4. Next, we need to use ImageJ to obtain the isovalue range to allow us to filter out all but the outermost ring of the multi-layer sample.

   Click **'Plugin > 3D Viewer'** which will open the viewer, then **'Edit > Adjust Threshold'.** By playing with the threshold bar, the appropriate *isovalues* can be identified for use in iso2mesh.

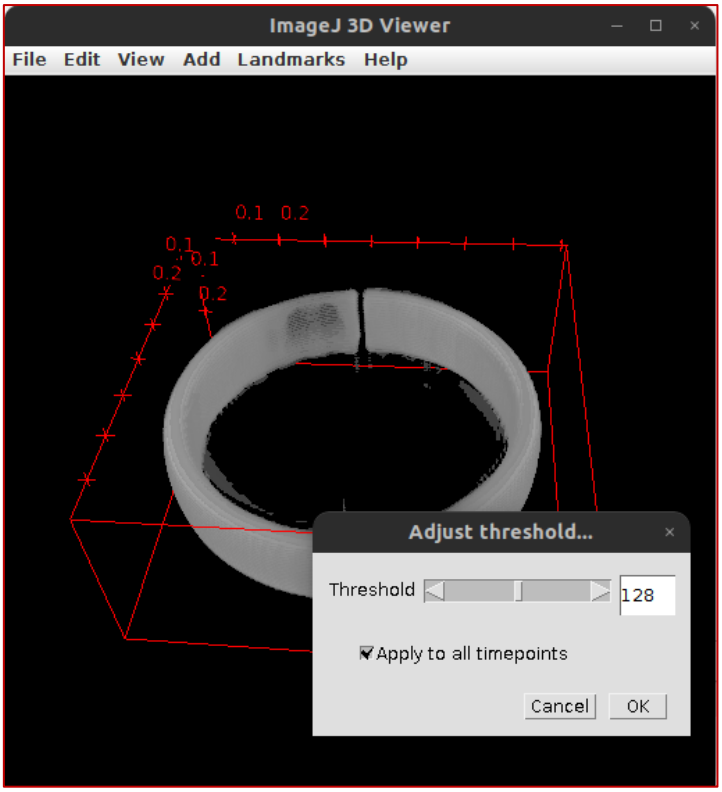


Figure 22. Visualisation of 3D segmented ring volume in ImageJ.

5. Open Octave as directed in Tutorial-I.
6. Open the template script *tutorial3_grayscale_ex1.m* in the editor.
7. First enter the number of slices (obtained in ImageJ) and load image the file *data_UKAEA_MB.tif* by replacing <number_of_slices> and <file_name>. The script runs a loop which iterates through each slice to read in and store the data in the *img_vol* matrix.
8. Then use ***v2m*** with the 'cgalsurf' method. Fill the input parameters of ***v2m*** using the value obtained from ImageJ for <isovalues>.
9. Click the save and run button. The resultant mesh is shown in Figure 2322.

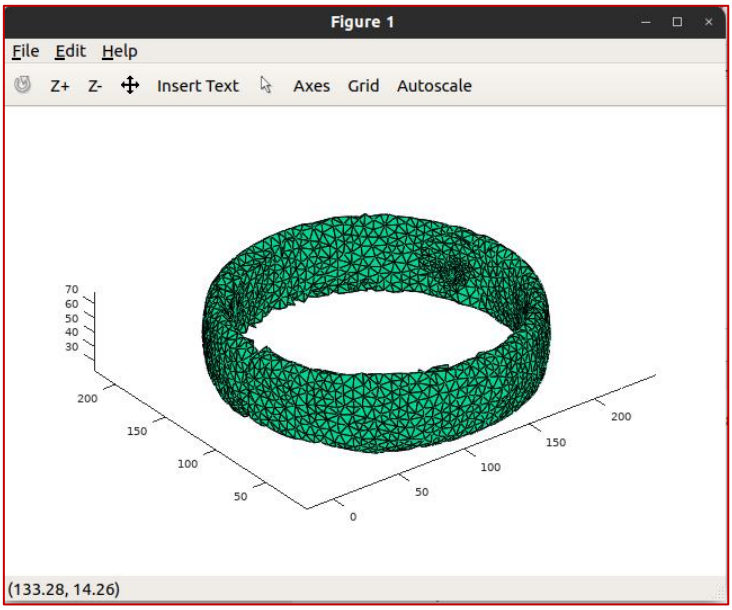


Figure 23. Mesh of ring generated by v2m (isovalues:128, opt:5, maxvol:100, method:cgalsurf)

Example 2:

In this example, real grayscale CT data of a block of KFoam obtained from CT imaging is meshed.

1. First, we visualise our CT volume data **data_KFoam_200pixcube.tif** in ImageJ as described in the previous example by importing the data as TIFF Virtual Stack.

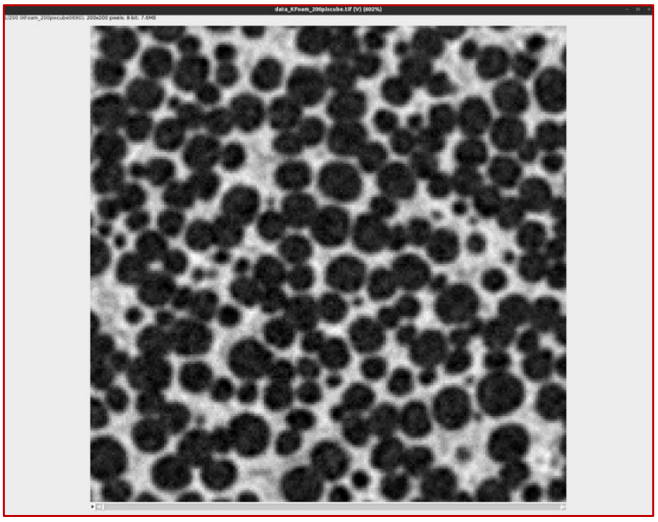


Figure 24. CT slice of KFoam.

2. Obtain the number of slices and then the *isovalues* by playing with *Adjust Threshold* to obtain the best view in the *3D Viewer* as described in the previous example.
3. Open Octave.
4. As we will follow the same procedure as in the previous example, we can use the same template script, i.e., *tutorial3_grayscale_ex1.m*.
5. Enter the number of slices and load the image file *data_KFoam_200pixcube.tif*.
6. This time we will use ***v2m*** to test the 'cgalsurf' method. Fill the input parameters of ***v2m***.
7. Click the save and run button. The resultant mesh is shown in Figure 5.

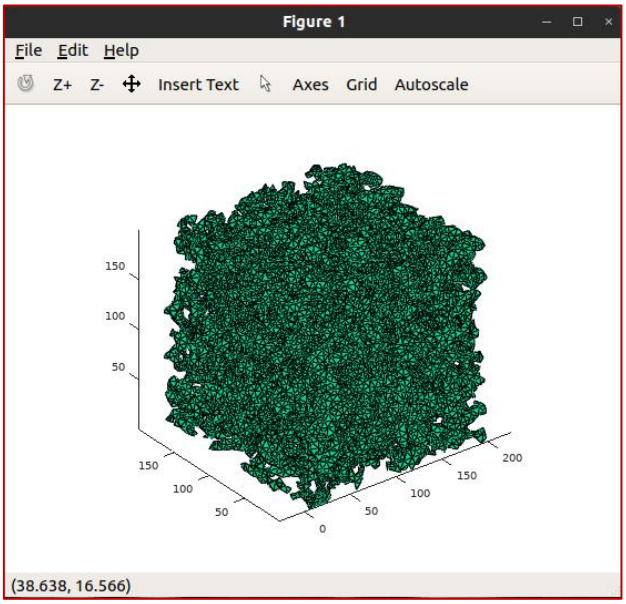


Figure 25. Mesh of Kfoam generated by v2m.

Tutorial-IV: Mesh generation from multi-phase pre-segmented volumetric data

Overview:

The aim of this tutorial is to mesh a pre-segmented volumetric image data composed of variety of regions (or phases) with unique meshing parameters, i.e., local meshing.

Learning outcomes:

- Visualising a multi-region body in ImageJ.
- Generating meshes with element sizing parameters unique to specific regions of volumetric data such as X-ray or MRI images.

Steps to follow:

1. First, we will visualise the data **data_hemispherequadrant.tif** in ImageJ.
    a. Open ImageJ from the VM desktop and import the data by clicking: **'File > Import > TIFF Virtual Stack…'**. Accept the default options.
    b. As shown in Figure 2625, you can see different regions of 1/8th sphere in the window. To see all distinct phases, you may need to adjust the image's contrast level:
    'Image > Adjust > Brightness/Contrast…' (or 'Shift+Ctrl+c').
    c. You can use ImageJ to get the number of slices in the image which (will be used in this tutorial).

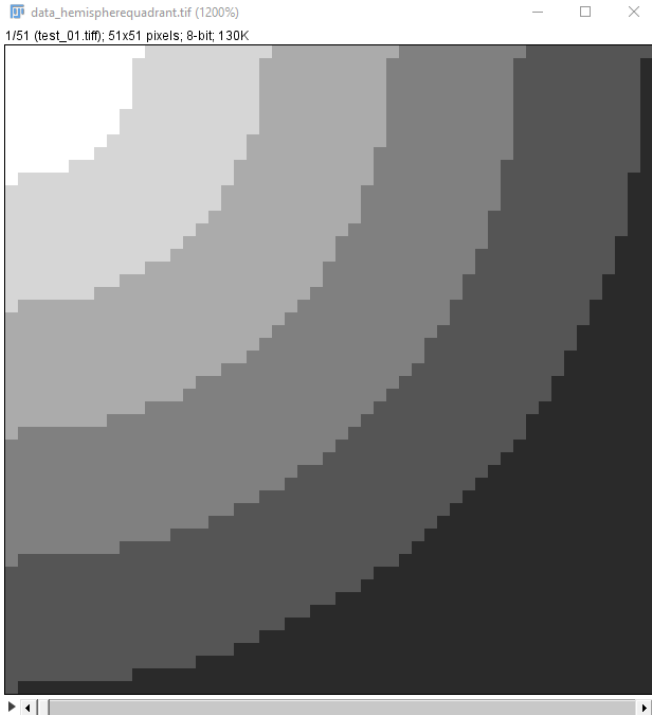


Figure 26. Image slice from grayscale volumetric image of a ball.

NOTE: Each layer is distinguished by a greyscale value ranging from 1-5 (outer to inner) with 0 representing the background. You can verify this by hovering your mouse cursor over the image, the x,y,z and greyscale value will be displayed along the bottom of the ImageJ GUI menu.

2. Within Octave, open the template script *tutorial4_labelling_ex1.m* in the editor.

3. First enter the number of slices (obtained in ImageJ) and load image the file **data_hemispherequadrant.tif**.

4. We will use ***v2m*** to mesh the volume:

[node, elem, face]=v2m(img, isovalues, opt, maxvol, method)

Set the parameters for v2m:

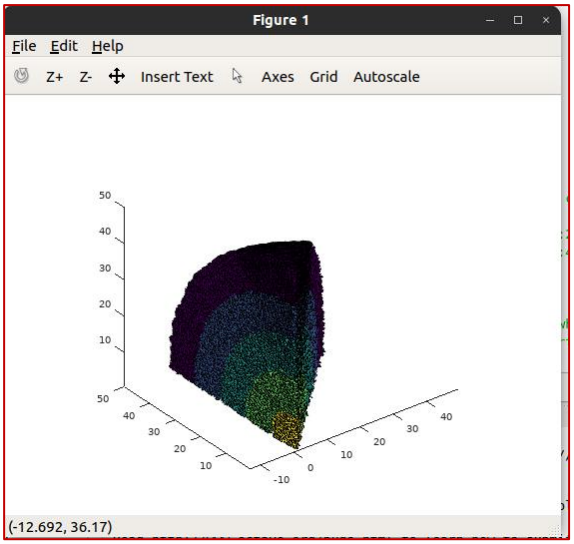
img	The cgalmesh method requires data to be formatted as 8-bit unsigned integers. The data in the img_vol matrix should already be in this format, but you can ensure this with uint8(variable_name).
isovalues	In this example with pre-segmented data, leave the isovalues argument empty by entering '[]' which creates a null array.
opt	In this example we are investigating the 'maxvol' parameter, therefore set opt to be 5 for all examples in this tutorial.
maxvol	See notes below for details.
method	Set to 'cgalmesh'.

5. In this tutorial, the aim is to mesh the regions (or labels) of the ball using different mesh sizes by controlling the *maxvol* argument for each region individually.
    a. For the first example we will set maxvol to be equal for all regions. We can do this for all labels by using '1' in *maxvol*. Click the save and run' button. You should have meshes with similar size elements in each label, as seen in Figure 2726 (a-b)
    b. Now we will test another *maxvol*. Comment out the first *maxvol* line and uncomment the following line, i.e., where maxvol is '**1**=2:**2**=1:**3**=2:**4**=1'. The bold numbers represent the id of each region and the following numbers are the local *maxvol* values.

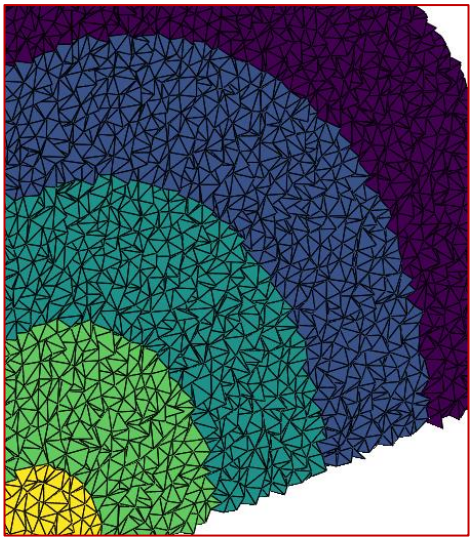
       Run the script again which should yield the resultant mesh (Figure 2726 (c-d)) with alternating fine and coarse layers. Notice how we did not specify a value for region 5. Any regions not specified will use the coarsest level set amongst the other regions. For example, '**2**=2:**3**=4:**4**=1:**5**=0.5' is the same as '**1**=4:**2**=2:**3**=4:**4**=1:**5**=0.5'.
    c. You do not need to specify the region number when declaring maxvol. Therefore, our third example where maxvol is '2:1:2:1' is, in fact, the same as our second example. Comment / uncomment the appropriate lines to test this, Figure 2726 (e-f).

       However, by not specifying the region number, the first entry will always be assigned to the first region. That is, by removing the entry in this example to '1:2:1', this would be read as '1=1:2=2:3=1', then regions 4 and 5 would be set to the largest value which is 2.
    d. The fourth example combines both approaches where maxvol is set to '3=2:1:0.5'. Here, the third region is set explicitly, and the following values are assigned to the following regions in order, regions 1 and 2 are set at the coarsest level, i.e., this is read as '2:2:2:1:0.5'. It is also possible to combine the approaches, e.g., '2=3:4=1:0.5' becomes '3:3:3:1:0.5'. Be careful not to explicitly set regions out of order, e.g., '3=3:2=1:0.5' may lead to unexpected behaviour.

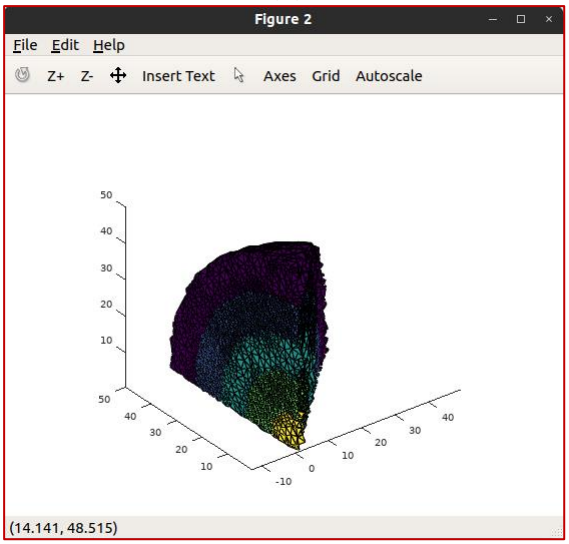
NOTE: It is worth noting that the current plot function is set to display a cross-section of the mesh in order to observe the internal mesh transitions between the labels. If you wish to view the entire meshed labels, comment out the active plot function by adding '%' in front of it and uncomment the previous plot function *plotmesh(no,el)* by deleting '%'. Click the save and run button one more time.



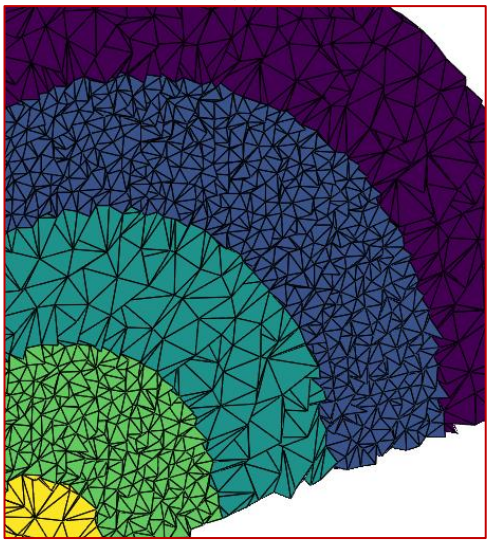
(a)



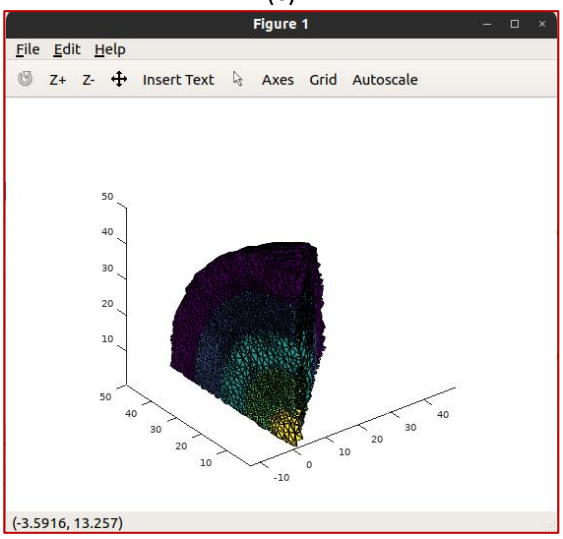
(b)



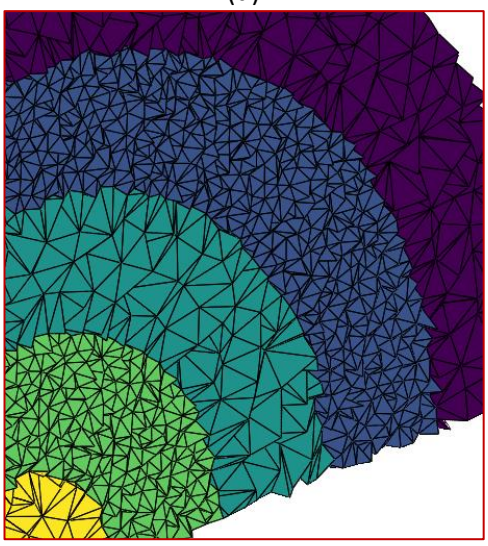
(c)



(d)



(e)



(f)

Figure 27. Fine and coarse-meshed labels (a-c-e) and their zoomed cross-sectional views (b-d-f).

Tutorial-V: Mesh generation from 2D image

Overview:

The aim of this tutorial is to extrude a volume from a 2D image and mesh it.

Learning outcomes:

- Creating meshed bodies from single 2D images.
- Investigating the resultant meshes in ParaView (an open source pre- and post-processor).

Steps to follow:

1. In order to visualise the 2D image **data_iso2mesh_bar.tif**, double-click the file in the tutorial directory, i.e., '/home/ibsim/iso2mesh/tutorials' (see Figure 2827).

Figure 28. Single 2D image to be extruded for 3D mesh generation

2. Within Octave open the template script *tutorial5_vol2mesh_ex2.m* in the editor.
3. First load the image file **data_iso2mesh_bar** by replacing < file_name > & <extension_type>.
4. Next step is to extrude the volume from the 2D image. For this example, we wish to create 30 slices, but you can choose any slice numbers. Replace <number_of_slices> in the script with 30. We, now, have a 3D image where each slice is identical.
5. In this tutorial, the ***vol2mesh*** function is used to mesh the 3D image. The necessary input arguments have been already set up in the script. We only need to click the 'save and run' button to mesh the volume and visualise the resultant mesh, see Figure 2928.

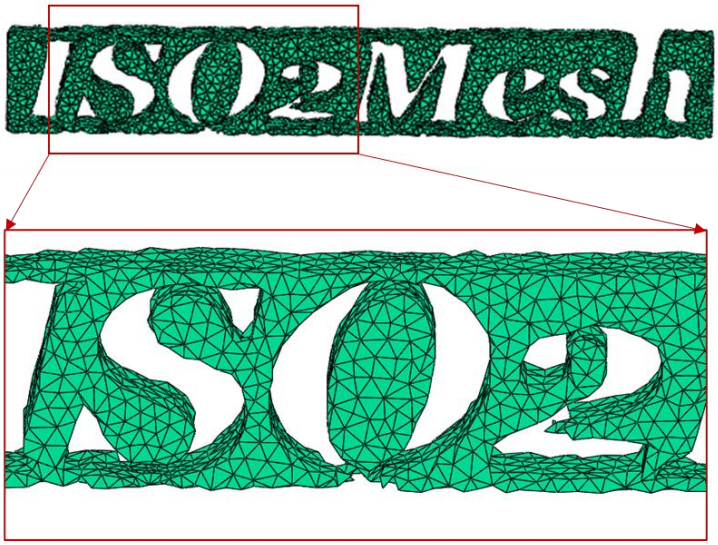


Figure 29. Mesh structure generated by iso2mesh from the extruded 2D image.

6. Note that the mesh has some disconnected regions (or floating islands), for example the centre of the letter 'O' as shown in the zoomed in view of Figure 2928. Features such as floating islands can cause issues for the FEA solver. To ensure no islands there are different approaches (e.g., connecting the island to the main body or removing it completely). In this instance, we wish to connect it with the rest of the mesh. To do this we will add additional slices as a 'background' to the text.

   Uncomment the line of script shown below (which adds 30 white slices), 'cut & paste' it to the line after the ' fullimg=repmat…' line then re-run the script.

   fullimg(:,:,31:60)=repmat(ones(size(img)),[1,1,30]);

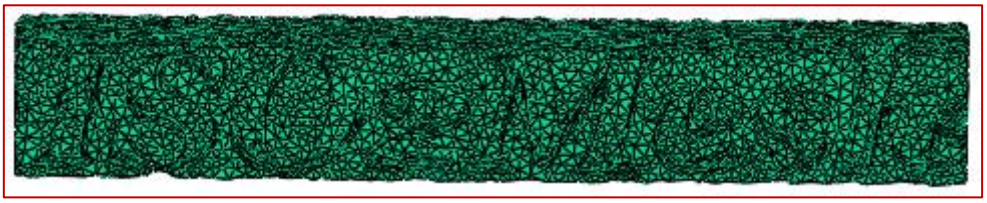


Figure 30. Mesh of the extruded 2D image after adding 30 'background' slices.

7. In the next step, the mesh will be exported in the Gmsh format (.msh) to be used in other software (e.g., FEA simulation). Uncomment the four consecutive lines starting with '*ext_out=…'.*

   Ext_out is the file format in which the mesh data will be exported. Re-run the script and, when the meshing and export operations are complete, you will see the file **data_iso2mesh_bar.msh** in the directory **'/home/ibsim/iso2mesh/tutorials/'**.
8. In order to investigate the mesh, we will use ParaView. Double-click the icon on the desktop which will open the software's GUI.

Figure 31. ParaView Icon.

9. Import the mesh file **data_iso2mesh_bar.msh** by ***'File > Open File'*** (browse to the directory shown above, select the file and click 'OK'). After the file has been loaded into the 'Pipeline Browser' you must click the '*Apply'* button under the 'Properties' tab (bottom left).
10. Amend the ***'Representation'*** style on the left-hand side from ***'Surface'*** to ***'Surface With Edges'*** in ***'Properties > Representation'*** to view the mesh elements, as shown in Figure 3231. Once finished, change the representation back to ***'Surface'***.

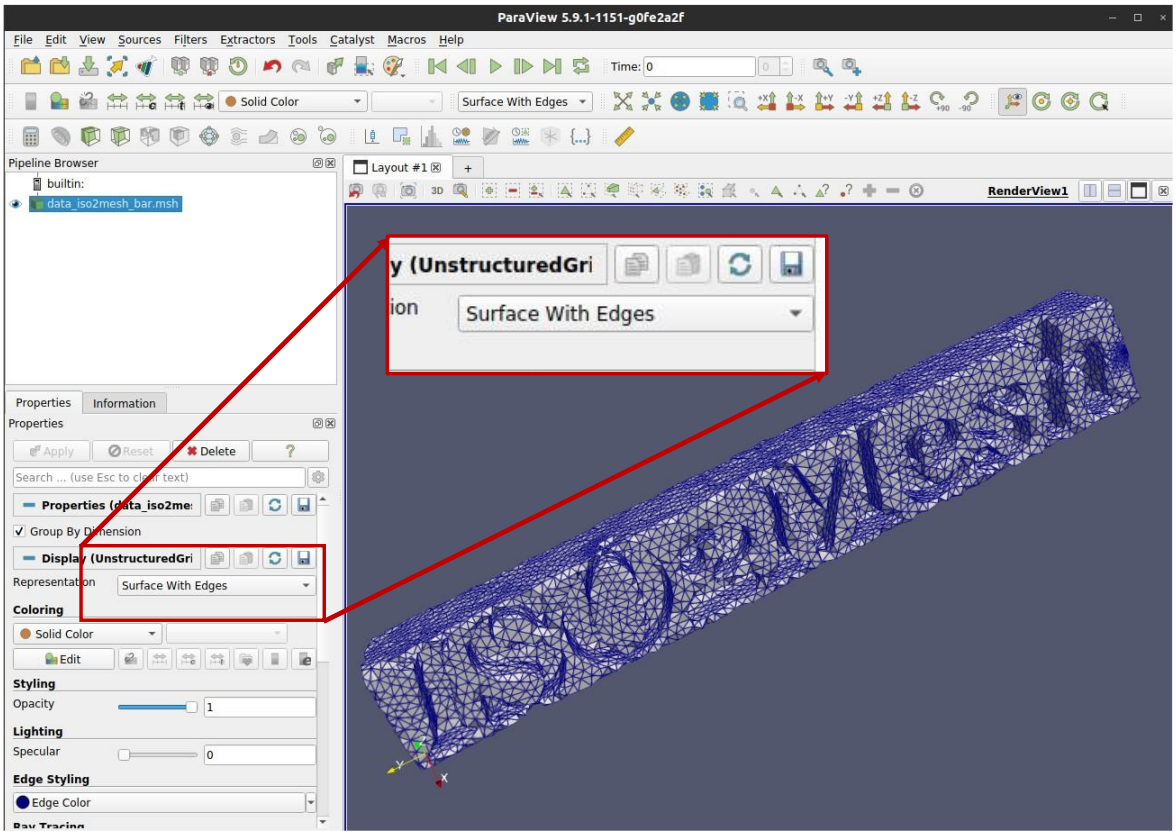


Figure 32. Amended 'Representation' from 'Surface' to 'Surface with Edges' in ParaView.

11. ParaView can be used to measure the mesh quality. To do this we must add a 'filter' by clicking ***'Filters>Alphabetical>Mesh Quality'.*** The filter allows us to measure (and visualise) the quality of triangles, quads, tets and hexes. The mesh only contains tetrahedral elements, therefore, of the four options available, that is the only category of interest in this tutorial. The dropdown menu allows us to select between various ways of measuring quality, such as the aspect ratio, which is the ratio between the shortest and longest edge length in the element (an ideal element has an aspect ratio of one). From the drop-down menu, which gives a list of mesh quality metrics to choose from, for "Tet Quality Measure" select **Aspect Ratio** for the measuring method then click ***'Apply'***.

    To facilitate viewing the location of low-quality elements we will add a threshold filter. Whilst the mesh quality filter is selected in the 'Pipeline Browser' click on ***'Filters > Alphabetical > Threshold'*** which adds another element to the pipeline. Use the minimum and maximum sliders in the properties menu to select threshold values, then click ***'Apply'***. When the min and max are set to their extremities, all the elements will be visualised, as you increase the min slider only the lowest quality elements will remain (if 'aspect ratio' was selected as your measurement metric).

To visualise where those lowest quality elements are located within the mesh toggle the visibility of **data_iso2mesh_bar.msh** by clicking the 'eye' symbol to the left of the element in the 'Pipeline Browser'. Then, with that element selected, reduce its opacity with the opacity slider under ***'Properties > Styling > Opacity'***. You should end up with a visualisation similar to that shown in Figure 3332. You may need to rescale the colourmap if you switch the mesh quality metric by clicking the ***'Rescale to Data Range'*** button directly above the opacity slider.

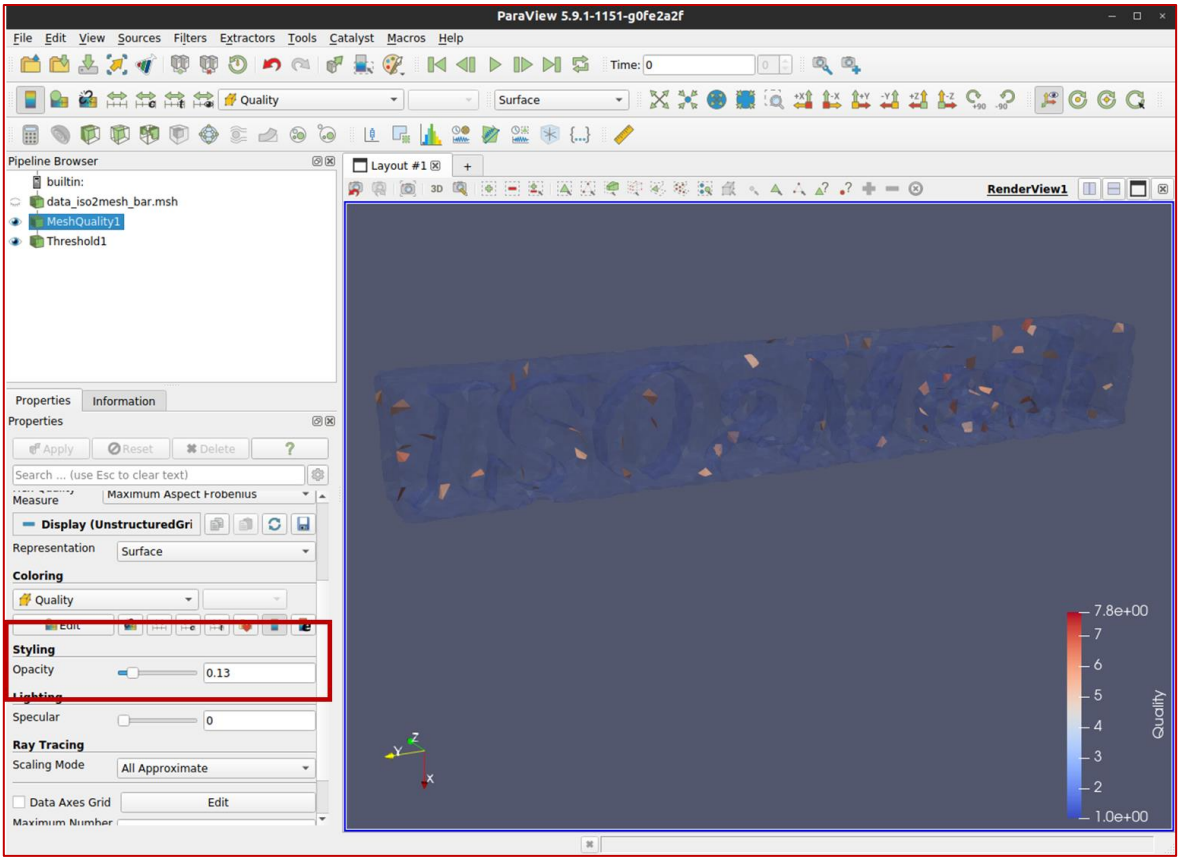


Figure 33. Visualisation of elements below a threshold mesh quality within the volume.

Tutorial-VI: Surface smoothing

Overview:

The aim of this tutorial is return to the 'rat head' dataset (**data_rat_head.mat**), which was investigated in Tutorial-II, to investigate various iterative surface smoothing techniques applied to the mesh of a binarized volumetric dataset.

Learning outcomes:

- Investigating the effects of the chosen smoothing techniques.
- Investigating the effect of different numbers of iterations during the smoothing process.

Steps to follow:

1. Within Octave open the template script *tutorial6_smoothing_ex1.m* in the editor.
2. In the script, after loading **data_rat_head.mat**, set the number of iterations by replacing <number_of_iterations> with, for instance, 12.
3. The meshing of the volume data is performed in the next step using the ***v2s*** function, which is a simplified version of the ***vol2surf*** function, with the 'cgalmesh' meshing method. You do not need to amend anything, but you may wish to come back to this section at the end.
4. The default smoothing method in the script is Laplacian - HC smoothing. Before calling the smoothing method, we can have a look at the surface smoothing function ***sms***, which is a simplified version of ***smoothsurf***.

newnode=sms(node, face, iter, alpha, method)

The input arguments of sms are:

node	Nodal coordinates of the tetrahedral mesh.
face	Mesh surface element list of the tetrahedral mesh.
iter	Number of smoothing iterations.
alpha	Scaler smoothing parameter.
method	Smoothing method: 'laplacian','laplacianhc' and 'lowpass'. By default, it is 'laplacianhc'.

This function returns a matrix, *newnode*, containing updated nodal coordinates of tetrahedral meshes after smoothing. Definition of the smoothing methods are given below:

Laplacian mesh smoothing For each vertex in a mesh, a new position is chosen based on local information (such as the position of neighbours) and the vertex is moved there. In the case that a mesh is topologically a rectangular grid (that is, each internal vertex is connected to four neighbours) then this operation produces the Laplacian of the mesh.
Laplacian-HC mesh smoothing Improved Laplacian filter which prevents the effect of shrinking, while preserving the effect of smoothing. The key idea is to push the vertices obtained in each step of iteration of the Laplacian algorithm back into the direction of the original vertices.
Low pass smoothing An image is smoothed by decreasing the disparity between pixel values by averaging nearby pixels. Using a low pass filter tends to retain the low frequency information within an image while reducing the high frequency information.

For more details on a comparison of the methods, see vR. Bade, H. Haase, B. Preim, "Comparison of Fundamental Mesh Smoothing Algorithms for Medical Surface Models," Simulation and Visualization, pp. 289-304, 2006.

Click the save and run button to apply the Laplacian-HC mesh smoothing method to the desired number of iterations. An example resultant method is shown in Figure 3433 (a).

5. Now, we would like to test *Laplacian* and *Low pass* mesh smoothing methods by using ***smoothsurf*** function, i.e., the full version of the smoothing function.

newnode = smoothsurf(node,mask,conn,iter,alpha,method,beta)

The input arguments of ***smoothsurf*** are:

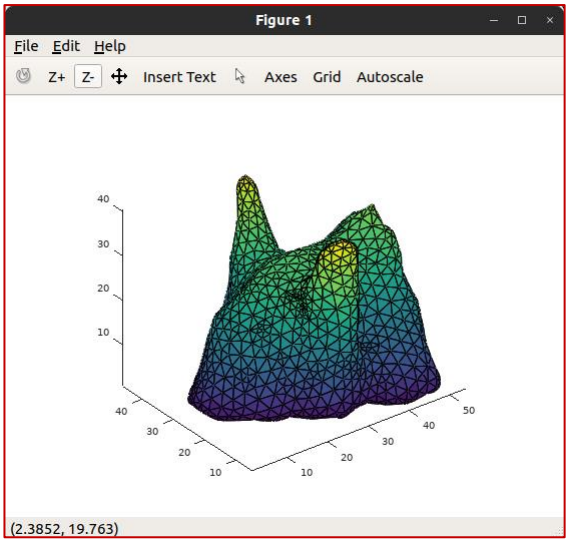
node	Nodal coordinates of the tetrahedral mesh
mask	Flag whether a node is movable: 0 movable, 1 non-movable. If mask=[], it assumes all nodes are movable.
conn	Mesh surface element list of the tetrahedral mesh.
iter	Number of smoothing iterations.
alpha	Scaler smoothing parameter.
method	Smoothing method: 'laplacian','laplacianhc' and 'lowpass'. By default, it is 'laplacianhc'.
beta	Another smoothing parameter.

In the script, the *conn* parameter generates a list of all neighbouring node IDs for node n.

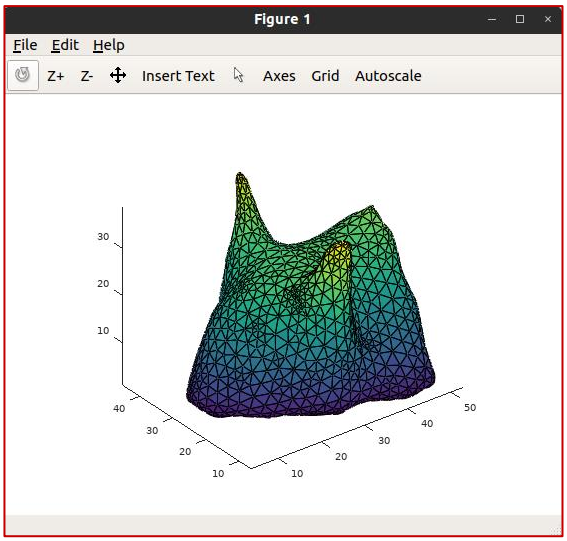
Comment out the line 'node = sms(..)' and uncomment the line below for testing the **Laplacian** smoothing method. Re-run the script. The script will produce a mesh, similar to that shown in Figure 3433 (b).

6. Comment out the line 'node = smoothsurf(.., 'laplacian')' and uncomment the line below for testing **Lowpass** smoothing. Re-run the script. This time the script will produce the mesh in Figure 3433 (c). Do you see any major difference in the resultant mesh? Which method works faster, and which produces a smoothed surface which has retained the most surface details?
7. By amending the number of smoothing iterations, *iter*, investigate how this impacts the resultant meshes. You should discover that there's a 'sweet spot' where fewer iterations do not smooth the mesh adequately and more iterations either cause 'over smoothing' or have little additional benefit for additional computational expense (i.e., it takes longer but has little positive or negative impact).
8. Once you are satisfied that you have an appreciation for the smoothing techniques, run the solution script *tutorial6_smoothing_ex1_sln.m* which will produce four figures. The first three figures will allow you to compare resultant meshes for a specific method with increasing number of iterations, the fourth figure will allow you to compare each method after a fixed number of iterations.

(a) Laplacian-HC



(b) Laplacian



(c) Lowpass

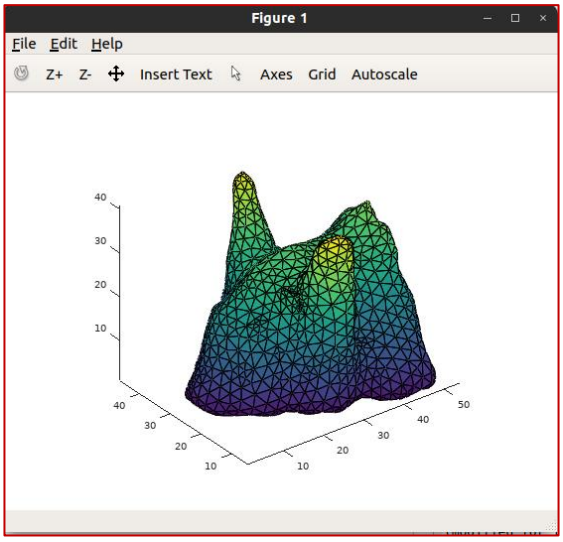


Figure 34. Mesh of the rat head after three different surface smoothing method

Tutorial-VII: Mesh generation from segmented XCT data

Overview:

In this tutorial we will mesh pre-segmented data from a Ti6Al4V tensile sample imaged by XCT at Swansea University. The sample includes a controlled defect and has been manufactured as part of a benchmarking study to cross-compare various IBSim workflows with experimental results collected by digital image correlation (DIC).

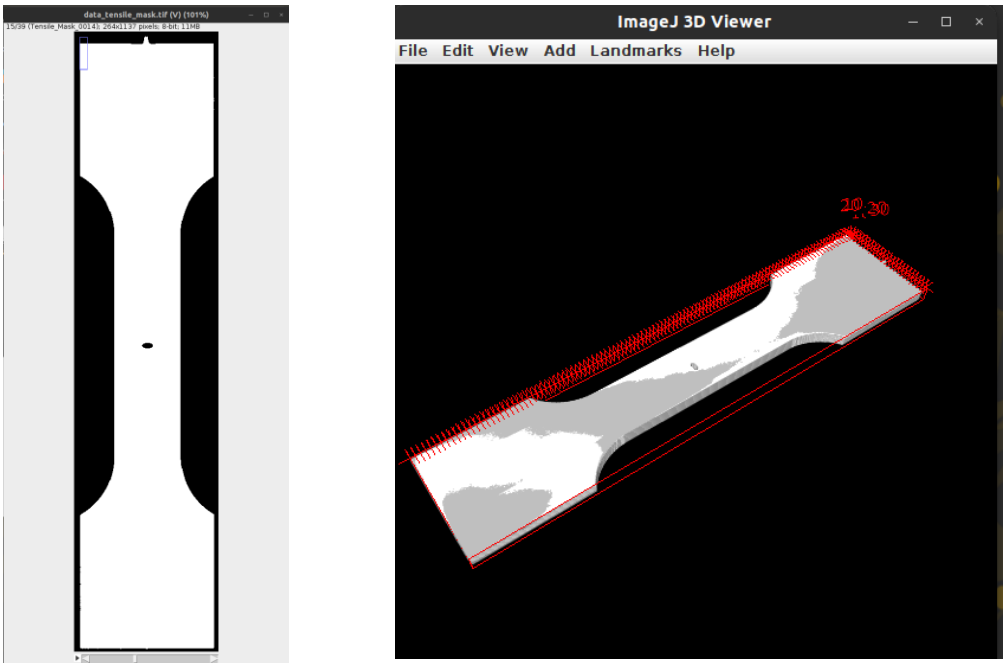
Learning outcomes:

- Consolidate lessons learnt by putting into practice.
- Appreciation for a few aspects of preparing meshes to be 'simulation ready'.

Steps to follow:

1. Provided in the tutorials directory are both the greyscale (**data_tensile_XCT.tif**) and segmented (**data_tensile_mask.tif**) image data files. Import both of these into ImageJ as TIFF Virtual Stacks to familiarise yourself with the data.
2. The grayscale values of the segmented data ranges from 0 to 1. If you have difficulties in seeing the sample in the slices, you adjust the contrast **'Image > Adjust > Brightness/Contrast'**.

   To visualise the segmented volume in 3D, click **'Plugin > 3D Viewer'** and the ImageJ 3D Viewer will open as shown in Figure 3534 b).



(a) (b)

Figure 35. Slice (a) and 3D volume (b) of tensile specimen in ImageJ.

3. Within Octave open the script *tutorial7_vol2mesh_ex3_sln.m* in the editor.
4. Unlike the previous tutorials, this script is ready to be run in its current state. The purpose of this tutorial is to explore various meshing parameters, using the lessons learnt (with a few new items), to obtain a mesh which is appropriate for use with an image-based simulation.

When dealing with larger datasets, one consideration must be the specifications of the available hardware. A compromise must be reached between the mesh's resolution and accuracy with the computational expense. This is usually a balancing act between accuracy and the number of elements within a mesh. However, with appropriate smoothing and mesh refinement/coarsening it can be possible to make effective use of a reduced element count and still achieve accurate meshes.

Part of this control lies with the following parameters which should be set considering the computational power available:

opt.maxnode	User-defined max number of nodes. This parameter can be adjusted according to the memory capacity of your PC.
opt.radbound	Target mesh size bounding sphere radius
maxvol	Max volume of target mesh

5. Thus far, no measurement unit has been used in any of the tutorial data. In this tutorial, the XCT image data has voxels widths of 0.0989 mm (after downsampling the data from its initial resolution for this tutorial). The data can be scaled from one unit per voxel to any measurement units desired by using the *vox_width* parameter.

   The meshing method in this script is set to 'cgalsurf ' and 'lowpass' surface smoothing is implemented.

   For this tutorial, it is recommended to use ParaView to visualise the meshes. However, if you would like to perform a quick check you may uncomment the visualisation section in the script, i.e., 'plotmesh(…)' before clicking the save and run. Figure 3635 shows the mesh generated with the default parameters.

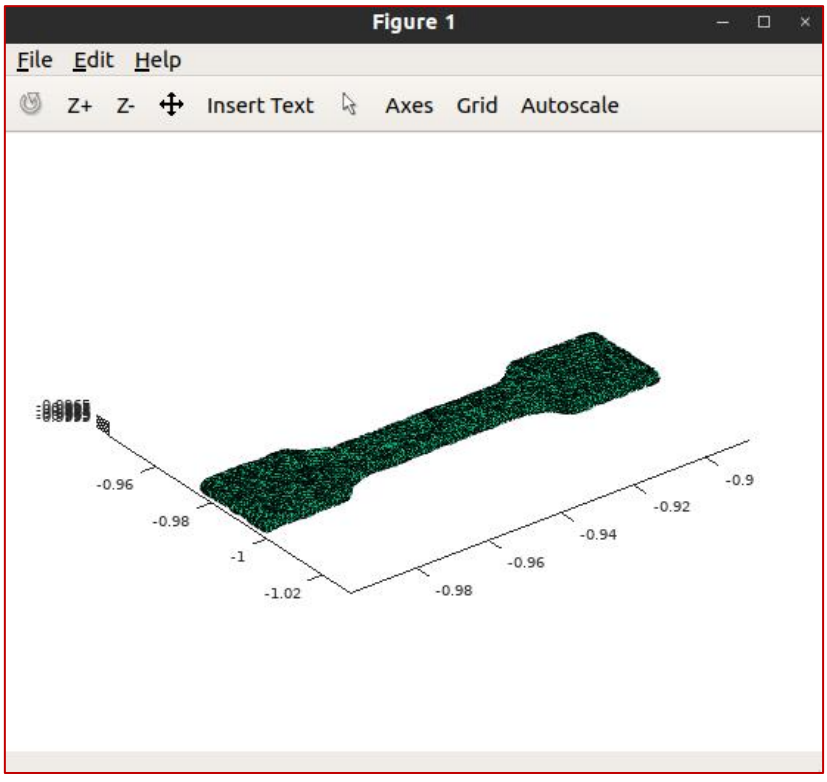


Figure 36. Meshed tensile test specimen after lowpass surface smoothing.

6. To view the data in ParaView, open the exported data **data_tensile.msh** in as previously shown how. To view the mesh elements, as shown in Figure 3736, use 'Select Display > Representation > Surface with Edges'.

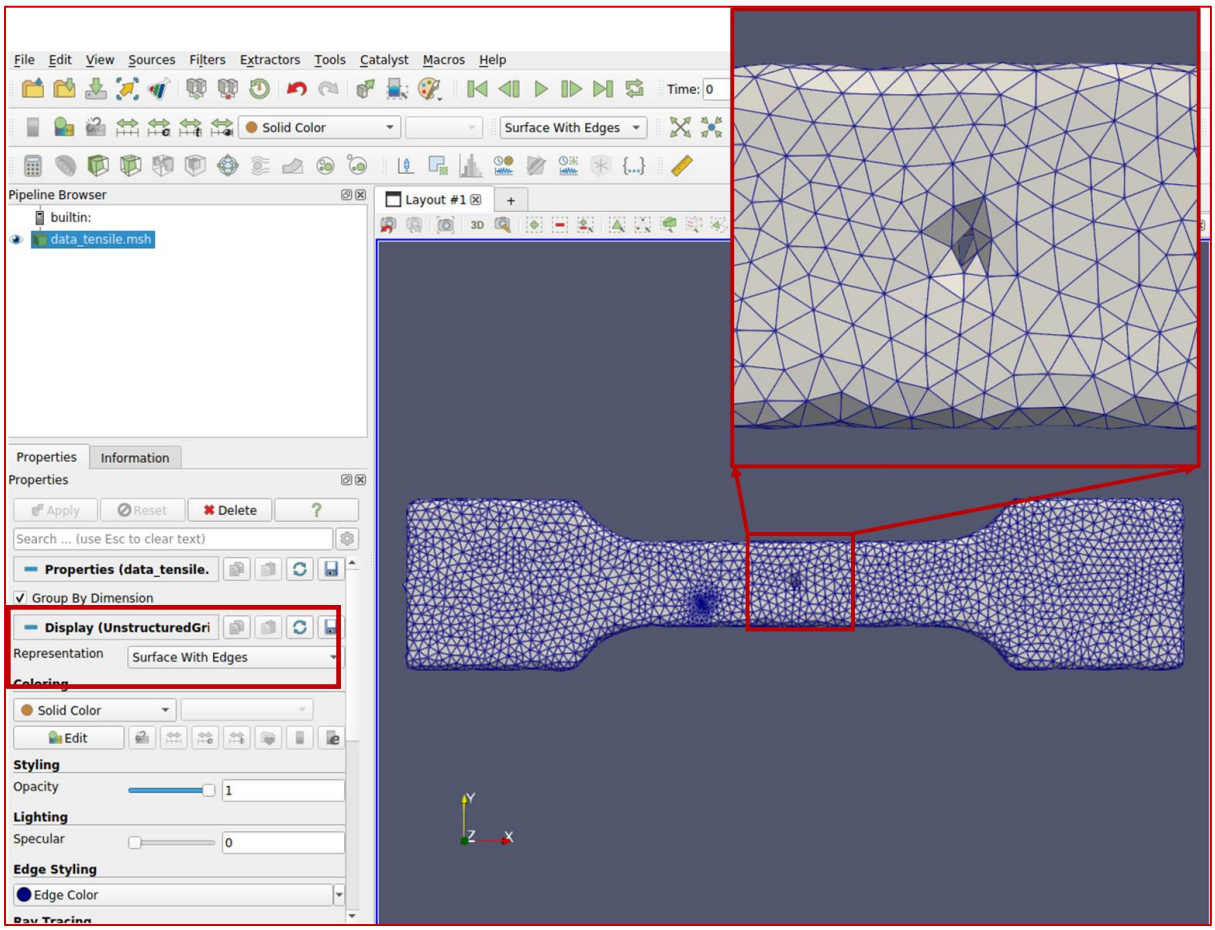


Figure 37. Mesh of the tensile test specimen using the 'cgalmesh' method, visualised in ParaView.

7. Switch from vol2mesh to v2m by un/commenting the appropriate lines and re-run the meshing.

   n Paraview, reload the data file by right clicking '**data_tensile.msh'** within the Pipeline Browser and select 'Reload Files'. The updated mesh (with the default parameters) is shown in Figure 3837. You should see a significant improvement in the mesh's accuracy. By comparing the figures, you can see how, in particular, the shape of the controlled defect is captured more faithfully.

8. Now use your experience to further investigate how changing the meshing parameters impacts the accuracy and element count.

   For this tutorial, it is suggested that you try varying *opt.radbound*, *maxvol*, *its* and meshing method. Keep in mind that even if surface elements appear to be of high quality, the mesh may contain skewed elements internally which will impact the simulation (use the method described in Tutorial V if you would like to investigate this).

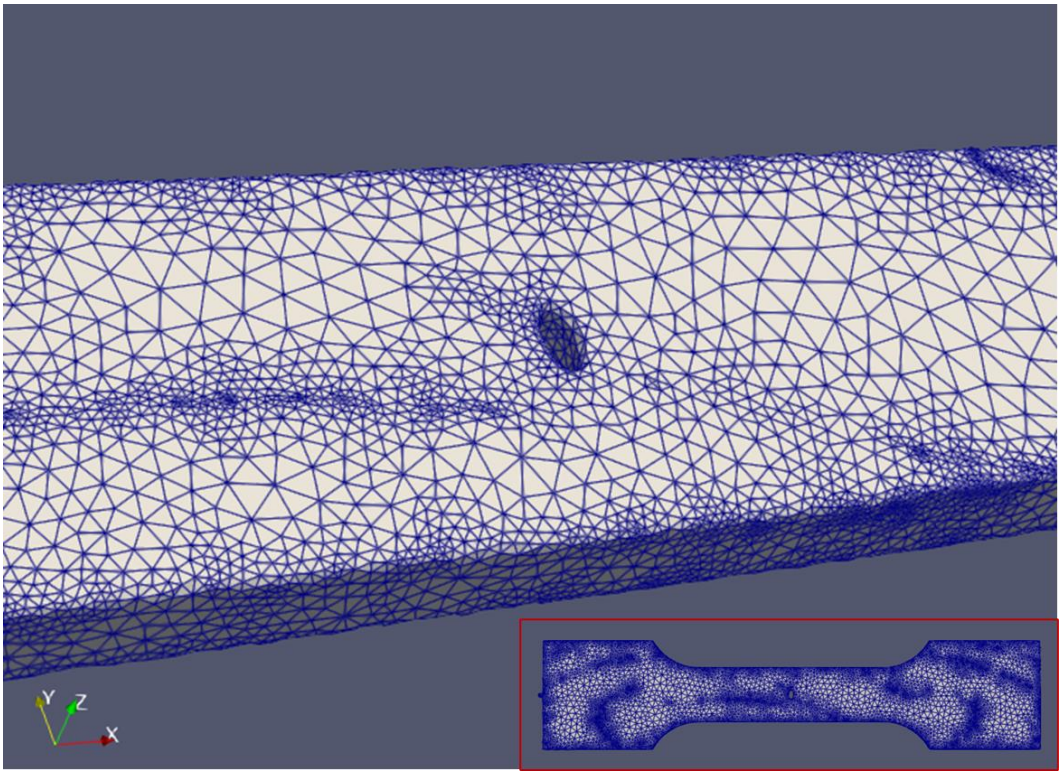


Figure 38. The tensile test specimen after remeshing with the 'cgalmesh' method.

Tutorial-VIII: Volumetric mesh generation from multiple levelsets

Overview:

In this final tutorial, we will look at combining two separate pre-segmented binary images. That is, we will aim to mesh multiple levelsets of a human head, consisting of the external surface **data_head.tiff** and internal brain tissue **data_brain.tiff**.

Learning outcomes:

- Appreciation of some challenges when combining images which haven't been cleaned, as with the previous tutorial data (i.e., resolving issues with intersecting element).
- Meshing multiple levelsets whilst combining two binary images.
- Visualising multiple phases in ParaView.

Steps to follow:

1. Within Octave open the script *tutorial8_vol2mesh_ex3_sln.m* in the editor.

   As in previous tutorials, the image data, **data_head.tiff** and **data_brain.tiff,** can be found in the working directory.
2. In the script, the section dedicated to loading of the image files has been already set up. The data is loaded into the *head* and *brain* matrices.

   Both datasets are later overlaid into a new matrix, *cleanimgfull*. There are multiple ways of achieving this. Considering that the two input datasets are binary (i.e., zeros and ones) and that the brain dataset lies completely within the head, the approach used here is to directly add the two matrices. This results in a matrix where the voxel values for the background are still zero, are still one for the head, but the brain voxels (which overlays the head) are now two.

   By setting opt.radbound parameters for each dataset, the head and brain are meshed with different element sizes. This is a slightly different approach to that which was shown in Tutorial V but has the same outcome.

   Without modifying run the script by clicking the save and run button.

   In Octave's *Command Window*, an error message (Figure 3938) pointing at intersecting subfaces will appear. This error results from holes in the images which need to be filled by an additional operation.

```
Error:  Invalid PLC.
  Two subfaces (10501, 80705, 81028) and (82567, 52341, 82218)
  are found intersecting each other.
  Hint:  Use -d switch to find all intersecting facets.
```

Figure 39. Error message due to intersecting facets while meshing multiple levelsets.

   To initiate the hole-filling operation, activate the ***fillholes3D*** function by removing '%' from the lines starting 'head = fillholes3d(..)' and 'brain=fillholes3d(...)'. The script will take a few minutes to run. Once complete it will export the mesh data as **data_head.msh** into your working directory.

3. To visualise the mesh, import **data_head.msh** in ParaView and click *Apply*. Ignore the error message. Change the visualisation style as before, i.e.,
   'Properties > Representation > Surface With Edges'.
4. Adjust the opacity to differentiate the head and brain:
   'Properties > Styling > Opacity'

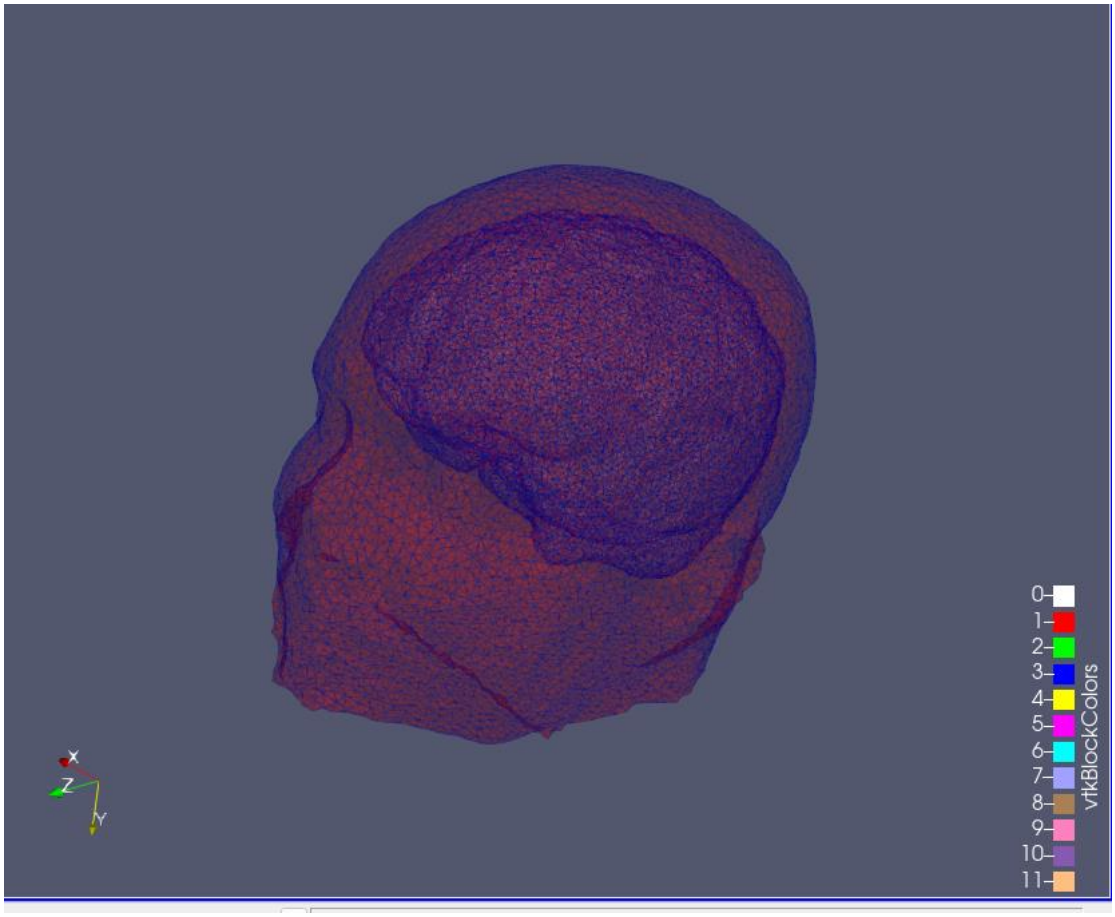


Figure 40. Meshed head and brain in ParaView.

5. To isolate the brain region of the mesh from the rest, create a block filter:
   'Filters > Alphabetical > Extract Block'
   a filter '**ExtractBlock1'** will appear in the Pipeline Browser.

   A block structure will appear in the Properties tab. Under volumes, region_1 and region_2 (alternatively, they may be called sample and PhysicalGroup2) are the head and brain regions, respectively.

   Select the first entry in the list under volumes and click *Apply*. i.e.,:
   'Properties (ExtractBlock1) > Blocks > 3-Volumes > region_1'

   The mesh for the brain region is then shown in isolation from the global mesh, as shown in Figure 4140. You may wish to repeat the above step for the other volume region which will allow you to control the visualisation options for each region independently, e.g., have the brain visualised as opaque whilst the head is semi-transparent.

6. Go back the script in Octave, amend opt.radbound e.g., create a coarser mesh in the head and rerun the script. After the meshing has completed, reload the mesh file in ParaView.

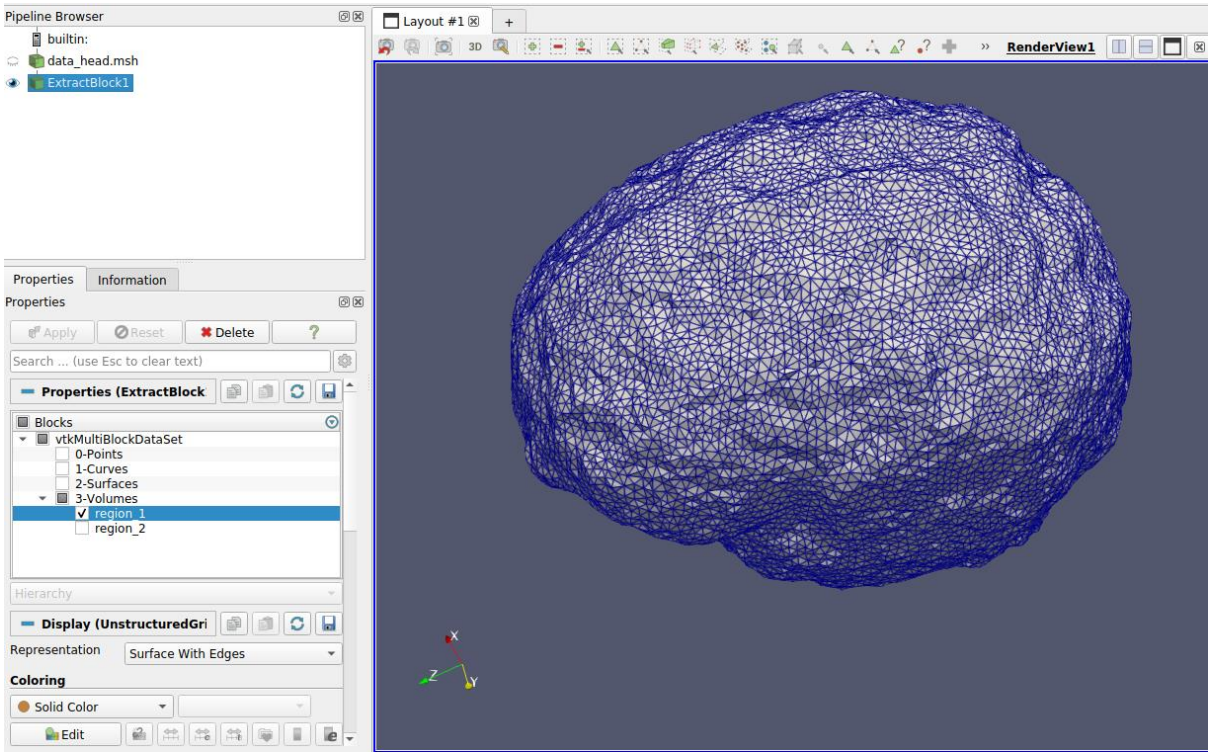


Figure 41. Isolated brain tissue in ParaView.

Appendix

tutorial1_surf2mesh_ex1

tutorial1_surf2mesh_ex1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from surface patches and bounding box
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% load the sample data
%% f and v store the surface patch faces and nodes
%load <filename.mat>;

%% Visualise the imported data
%% v(:,1) points to the first column within the array
%trisurf(f,v(:,1),v(:,2),v(:,3)); axis equal;

%% Perform mesh generation
%[node, elem, face] = surf2mesh(<v>, <f>, <p0>, <p1>, <keepratio>, <maxvol>, <
regions>, <holes>, <forcebox>);

%% Visualize the resulting mesh
%plotmesh(node,face(:,1:3)); axis equal;
```

tutorial1_surf2mesh_ex1_sln.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from surface patches and bounding box
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% load the sample data
%% f and v store the surface patch faces and nodes
load data_tube_surface.mat;

%% Visualise the imported data
%% v(:,1) points to the first column within the array
trisurf(f,v(:,1),v(:,2),v(:,3)); axis equal;

%% Perform mesh generation
[node,elem,face]=surf2mesh(v,f,[1 1 1],[100 100 100],0.1,25);

%% Visualize the resulting mesh
plotmesh(node,face(:,1:3)); axis equal;
```

tutorial2_vol2mesh_ex1

tutorial2_vol2mesh_ex1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from binarized volumetric image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% load the sample data
%% img_vol is a volumetric image such as an X-ray or MRI image
load <fname>;
%% Rename variable for consistency with other tutorials.
img_vol = volimage; clear("volimage");

%% Visualise a slice from the segmented volume data
%% Replace 'zslice_number' with # between 1 and the max. number of slices in
%% z. Alternatively, do this in the 1st or 2nd argument to slice in the x or
%% y planes e.g., img_vol(:,<yslice_number>,:).
img_slice = squeeze(img_vol(:,:,<zslice_number>));
imagesc(img_slice); axis equal;

%% perform mesh generation
%[node,elem,face]=vol2mesh(<img>,<ix>,<iy>,<iz>,<opt>,<maxvol>,<dofix>)

%% visualize the resulting mesh
%plotmesh(node,face); axis equal;

%% use the alternative 'simplify' method: first create voxel-based surface
%% mesh, and then resample it to the desired density. this method does not
%% guarantee to be free of self-intersecting elements, as 'cgalsurf' promises.
%[node,elem,face]=vol2mesh(<img>,<ix>,<iy>,<iz>,<opt>,<maxvol>,<dofix>,<method
>,<isovalues>)

%% visualize the resulting mesh
%plotmesh(node,face); axis equal;

%% use the alternative 'cgalmesh' method. This will call cgalmesher to process
%% labled volume (i.e. segmented data) to produce surfaces and tet mesh
%% in a single run.
%clear opt
%opt.radbound=2;
%[node,elem,face]=v2m(<img>,<isovalues>,<opt>,<maxvol>,<method>);

%% visualize the resulting mesh
%plotmesh(node,face); axis equal;
```

tutorial2_vol2mesh_ex1_sln.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from binarized volumetric image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% load the sample data
%% img_vol is a volumetric image such as an X-ray or MRI image
load data_rat_head.mat;
%% Rename variable for consistency with other tutorials.
img_vol = volimage; clear("volimage");

%% Visualise a slice from the segmented volume data
%% Replace 'zslice_number' with # between 1 and the max. number of slices in
%% z. Alternatively, do this in the 1st or 2nd argument to slice in the x or
%% y planes e.g., img_vol(:,<yslice_number>,:).
%img_slice = squeeze(img_vol(:,:,22));
%imagesc(img_slice); axis equal;

%% perform mesh generation
[node,elem,face]=vol2mesh(img_vol,1:size(img_vol,1),1:size(img_vol,2),...
                          1:size(img_vol,3),2,2,1);

%% visualize the resulting mesh
figure;
plotmesh(node,face); axis equal;

%% use the alternative 'simplify' method: first create voxel-based surface
%% mesh, and then resample it to the desired density. this method does not
%% guarantee to be free of self-intersecting elements, as 'cgalsurf' promises.
[node,elem,face]=vol2mesh(img_vol,1:size(img_vol,1),1:size(img_vol,2),...
                         1:size(img_vol,3),0.1,2,1,'simplify');

%% visualize the resulting mesh
figure;
plotmesh(node,face); axis equal;

%% use the alternative 'cgalmesh' method. This will call cgalmesher to process
%% labled volume (i.e. segmented data) to produce surfaces and tet mesh
%% in a single run.
clear opt;
opt.radbound=2;
[node,elem,face]=v2m(uint8(img_vol),0.5,opt,100,'cgalmesh');

%% visualize the resulting mesh
figure;
plotmesh(node,face); axis equal;
```

tutorial3_grayscale_ex1

tutorial3_grayscale_ex1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from greyscale volumetric image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% Read in image file (number of slices must be known)
num_slice=<number_of_slices>;
fname=<file_name>;
for i=1:num_slice
  img_slice=imread(fname,i);
  img_vol(:,:,i)=img_slice;
end

%% Create mesh
[node,elem,face]=v2m(<img>,<isovalues>,<opt>,<maxvol>,<method>)

%% Visualise mesh
plotmesh(node,face); axis equal;
```

tutorial3_grayscale_ex1_sln.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from greyscale volumetric image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% Read in image file (number of slices must be known)
num_slice=105;
fname="data_UKAEA_MB.tif";
for i=1:num_slice
  img_slice=imread(fname,i);
  img_vol(:,:,i)=img_slice;
end

%% Create mesh
[node,elem,face]=v2m(img_vol,128,5,100,'cgalsurf');

%% Visualise mesh
plotmesh(node,face); axis equal;
```

tutorial3_grayscale_ex2_sln.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from greyscale volumetric image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% Read in image file (number of slices must be known)
num_slice=200;
fname="data_KFoam_200pixcube.tif"
for i=1:num_slice
  img_slice=imread(fname,i);
  img_vol(:,:,i)=img_slice;
end

%% Create mesh
[node,elem,face]=v2m(img_vol,23,5,100,'cgalsurf');

%% Visualise mesh
plotmesh(node,face); axis equal;
```

tutorial4_labelling_ex1

tutorial4_labelling_ex1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from multi-mask segmented volumetric image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% First open image in ImageJ to view.
%% Greyscale values only range from 0-5, might need to manually change
contrast.

%% Read in image file (number of slices must be known)
num_slice=<number_of_slices>;
fname=<file_name>;
for i=1:num_slice
  img_slice=imread(fname,i);
  img_vol(:,:,i)=img_slice;
end

%% mesh the domain with different sizing options
%% Try each of these in turn
maxvol='1';                 %% a single scalar sets cell size for all labels
%maxvol='1=2:2=1:3=2:4=1';  %%
%maxvol='2:1:2:1';          %% same as maxvol='1=2:2=1:3=2:4=1'
%maxvol='3=2:1:0.5';        %% same as maxvol='3=2:4=1:5=0.5'

%% Mesh and plot
[no,el]=v2m(img_vol,[],5,maxvol,'cgalmesh');
figure;
%plotmesh(no,el);               %% This plots the whole volume mesh
plotmesh(no(:,1:3),el,'x-y<0'); %% This plots a portion to view the x-section
```

tutorial4_labelling_ex1_sln.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from multi-mask segmented volumetric image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% First open image in ImageJ to view.
%% Greyscale values only range from 0-5, might need to manually change
contrast.

%% Read in image file (number of slices must be known)
num_slice=51;
fname="data_hemispherequadrant.tif";
for i=1:num_slice
  img_slice=imread(fname,i);
  img_vol(:,:,i)=img_slice;
end

%% mesh the domain with different sizing options
figure;
maxvol='1';
[no,el]=v2m(img_vol,[],5,maxvol,'cgalmesh');
subplot(221);
plotmesh(no(:,1:3),el,'x-y<0');
title('a single scalar sets cell size for all labels');

maxvol='1=2:2=1:3=2:4=1';
[no,el]=v2m(img_vol,[],5,maxvol,'cgalmesh');
subplot(222);
plotmesh(no(:,1:3),el,'x-y<0');
title(sprintf('maxvol is "%s"',maxvol));

maxvol='2:1:2:1';
[no,el]=v2m(img_vol,[],5,maxvol,'cgalmesh');
subplot(224);
plotmesh(no(:,1:3),el,'x-y<0');
title(sprintf('maxvol is "%s", same as above',maxvol));

maxvol='3=2:1:0.5';
[no,el]=v2m(img_vol,[],5,maxvol,'cgalmesh');
subplot(223);
plotmesh(no(:,1:3),el,'x-y<0');
title(sprintf('maxvol is "%s", same to 3=2:4=1:5=0.5',maxvol));
```

tutorial5_vol2mesh_ex2

tutorial5_vol2mesh_ex2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from 2D image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

fname=<file_name>;
ext_in=<extension_type>;
f_in=strcat(fname,ext_in);

%% load iso2mesh 2D image
img=imread(f_in);
%% flip image left to right
img=fliplr(img);

%% extrude image by repeating over 30 slices
%% also take reverse by using '1-img'
fullimg=repmat(1-img,[1,1,<number_of_slices>]);

%% create volumetric tet mesh from the 3D image
clear opt
opt.keepratio=0.1; % option only useful when vol2mesh uses 'simplify' method
opt.radbound=5;    % set target surface mesh element bounding sphere radius
[node,elem,face]=vol2mesh(fullimg,1:size(fullimg,1),1:size(fullimg,2),1:size(f
ullimg,3),opt,100,1);

%% visualise mesh
hb=plotmesh(node,face); axis equal;
view(-90.5,-72);

%% Notice the floating 'island' in the centre of the letter 'O'.
%% This would potentially cause issues for the FEA solver.
%% To ensure no islands, we will add a 'background' to the text.
%% Uncomment the line below and move it after the 'fullimg' line above,
%% This appends an additional 30 slices of block colour to the 3D image.
%fullimg(:,:,31:60)=repmat(ones(size(img)),[1,1,30]);

%% Export mesh so that it can be used in other software (e.g. FEA simulation).
%% Combines the original filename with the new extension to give new filename.
%% Can be amended to custom name if desired.
%ext_out=".msh";
%f_out=strcat(fname,ext_out);
%elem(elem==0)=1;
%savemsh(node,elem,f_out);
```

tutorial5_vol2mesh_ex2_sln.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from 2D image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

fname="data_iso2mesh_bar";
ext_in=".tif";
f_in=strcat(fname,ext_in);

%% load iso2mesh 2D image
img=imread(f_in);
%% flip image left to right
img=fliplr(img);

%% extrude image by repeating over 30 slices
%% also take reverse by using '1-img'
fullimg=repmat(1-img,[1,1,30]);
fullimg(:,:,31:60)=repmat(ones(size(img)),[1,1,30]);

%% create volumetric tetrahedral mesh from the 3D image
clear opt
opt.keepratio=0.1; % option only useful when vol2mesh uses 'simplify' method
opt.radbound=5;    % set target surface mesh element bounding sphere radius
[node,elem,face]=vol2mesh(fullimg,1:size(fullimg,1),1:size(fullimg,2),1:size(f
ullimg,3),opt,100,1);

%% visualise mesh
hb=plotmesh(node,face); axis equal;
view(-90.5,-72);

%% Export mesh so that it can be used in other software (e.g. FEA simulation).
%% Combines the original filename with the new extension to give new filename.
%% Can be amended to custom name if desired.
ext_out=".msh";
f_out=strcat(fname,ext_out);
elem(elem==0)=1;
savemsh(node,elem,f_out);
```

tutorial6_smoothing_ex1

tutorial6_smoothing_ex1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for surface smoothing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% load the sample data
load data_rat_head.mat;

%% Set number of smoothing iterations
its=<number_of_iterations>;

%% perform mesh generation
[node,face]=v2s(volimage,0.5,2,'cgalmesh');
face=face(:,1:3);

%% Generate list of all neighboring node ID for node n
conn=meshconn(face(:,1:3),size(node,1));

%% apply smoothing
for i=1:its
  fprintf('Iteration %i\n', i);
  %% Try the different smoothing methods
  node=sms(node,face(:,1:3),1,0.5);                % Laplacian+HC smoothing
  %node=smoothsurf(node,[],conn,1,0.5,'laplacian'); % Laplacian smoothing
  %node=smoothsurf(node,[],conn,1,0.5,'lowpass');   % Lowpass smoothing

end

plotmesh(node,face(:,1:3)); axis equal;

%% Be sure to try the 'solution' script which plots the mesh at increasing
%% number of iterations and the range of methods in a single figure for
%% easier comparison.
```

tutorial6_smoothing_ex1_sln.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for surface smoothing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% load the sample data
load data_rat_head.mat;

%% Set number of smoothing iterations
its=12;
print_list=[4 8 12];

%% perform mesh generation
[node,face]=v2s(volimage,0.5,2,'cgalmesh');
face=face(:,1:3);

%% Set parameters for figures
p0=min(node);
p1=max(node);
rownum=2;
colnum=2;
figure;
subplot(rownum,colnum,1);

%% Plot unsmoothed mesh
plotmesh(node,face(:,1:3));
if(~isoctavemesh)
  title({'Laplacian+HC Smoothing Test','no smoothing'});
else
  title('Laplacian+HC - no smoothing');
end
axis equal;
set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])

%=============================================================
% apply Laplacian+HC smoothing
%=============================================================

n1=node;
j=1;
for i=1:its
  %% apply Laplacian+HC mesh smoothing
  n1=sms(n1,face(:,1:3),1,0.5);
  %% Set to plot on particular iterations
  print_check=0;
  print_check=ismember(i,print_list);
```

```
  if (print_check==1)
    subplot(rownum,colnum,j+1);
    plotmesh(n1,face(:,1:3));
    title(['iter=' num2str(i)]);
    axis equal;
    set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])
    j=j+1;
  end
end

sm1=n1;

%============================================================
% apply Laplacian smoothing
%============================================================

%% Plot unsmoothed mesh
figure;
subplot(rownum,colnum,1);
plotmesh(node,face(:,1:3));
if(~isoctavemesh)
        title({'Laplacian Smoothing Test','no smoothing'});
else
        title('Laplacian - no smoothing');
end
axis equal;
set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])

%% Generate list of all neighboring node ID for node n
conn=meshconn(face(:,1:3),size(node,1));

n1=node;
j=1;
for i=1:its
  %% apply Laplacian mesh smoothing
  n1=smoothsurf(n1,[],conn,1,0.5,'laplacian');
  %% Set to plot on particular iterations
  print_check=0;
  print_check=ismember(i,print_list);
  if (print_check==1)
    subplot(rownum,colnum,j+1);
    plotmesh(n1,face(:,1:3));
    title(['iter=' num2str(i)]);
    axis equal;
    set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])
    j=j+1;
  end
end
```

```
sm2=n1;

%============================================================
% apply Low-pass smoothing
%============================================================

%% Plot unsmoothed mesh
figure;
subplot(rownum,colnum,1);
plotmesh(node,face(:,1:3));
if(~isoctavemesh)
        title({'Low-pass Smoothing Test','no smoothing'});
else
        title('Low-pass - no smoothing');
end
axis equal;
set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])

%% Generate list of all neighboring node ID for node n
conn=meshconn(face(:,1:3),size(node,1));

n1=node;
j=1;
for i=1:its
  %% apply lowpass mesh smoothing
  n1=smoothsurf(n1,[],conn,1,0.5,'lowpass');
  %% Set to plot on particular iterations
  print_check=0;
  print_check=ismember(i,print_list);
  if (print_check==1)
    subplot(rownum,colnum,j+1);
    plotmesh(n1,face(:,1:3));
    title(['iter=' num2str(i)]);
    axis equal;
    set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])
    j=j+1;
  end
end

sm3=n1;

%% Plot all 3 smoothed meshes at final iteration
figure;
subplot(rownum,colnum,1);
plotmesh(node,face(:,1:3));
if(~isoctavemesh)
        title({'Low-pass Smoothing Test','no smoothing'});
```

```
else
        title('No smoothing');
end
axis equal;
set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])

subplot(rownum,colnum,2);
plotmesh(sm1,face(:,1:3));
title(['Laplacian+HC,iter=12']);
axis equal;
set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])

subplot(rownum,colnum,3);
plotmesh(sm2,face(:,1:3));
title(['Laplacian,iter=12']);
axis equal;
set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])

subplot(rownum,colnum,4);
plotmesh(sm3,face(:,1:3));
title(['Low-pass,iter=12']);
axis equal;
set(gca,'xlim',[p0(1),p1(1)],'ylim',[p0(2),p1(2)],'zlim',[p0(3),p1(3)])
```

tutorial7_vol2mesh_ex3

tutorial7_vol2mesh_ex3_sln.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script for mesh generation from tensile test sample
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%% First open image in ImageJ to view.
%% Both the XCT and segmented (mask) data are provided.
%% For the segmented data, the greyscale values only range from 0-1,
%% You might need to manually change contrast to see the data.

%% Read in image file (number of slices must be known)
fprintf('Reading in data..\n');
num_slice=39;
fname="data_tensile";
ext_in="_mask.tif";
f_in=strcat(fname,ext_in);
for i=1:num_slice
  img_slice=imread(f_in,i);
  img_vol(:,:,i)=img_slice;
end
img_vol=uint8(img_vol);

%% Set meshing parameters
opt.maxnode=100000;  % this sets the max. # of nodes, use to avoid running out
 of RAM
opt.radbound=10;     % set the target surface mesh element bounding sphere rad
ius
maxvol=500;          % set the target element maxvol

%% Scale data
vox_width=0.0000989; % this scales the data, use vox width in desired units
%% This is transforms the data with a matrix, using the diagonal elements only
%% (as in this example) performs linear scaling.
opt.A=[vox_width 0 0; 0 vox_width 0; 0 0 vox_width];
opt.B=[0 0 0]';


%% Mesh the volume
fprintf('Meshing..\n');

%%=============================================================================
%% Test both of these methods & try varying the meshing parameters above
%%=============================================================================
```

```
[node,elem,face]=vol2mesh(img_vol,1:size(img_vol,1),1:size(img_vol,2),1:size(i
mg_vol,3),opt,maxvol,1);
%[node,elem,face]=v2m(img_vol,0.5,opt,maxvol,'cgalmesh');

%% Perform surface smoothing
%% Try varying the number of iterations
fprintf('Smoothing..\n');
conn=meshconn(face(:,1:3),size(node,1));
its=6;
for i=1:its
  fprintf('Iteration %i\n', i);
  node=smoothsurf(node,[],conn,1,0.5,'lowpass');
end

%% Visualise mesh
%plotmesh(node,face); axis equal;
%% Rather than visualising in matlab/octave, try using ParaView

%% Export the mesh
fprintf('Exporting the mesh..\n');
ext_out=".msh";
%% Combines the original filename with the new extension to give new filename.
%% Can be amended to custom name if desired.
f_out=strcat(fname,ext_out);
elem(elem==0)=1;
savemsh(node(:,1:3),elem,f_out);
```

tutorial8_vol2mesh_ex4

tutorial8_vol2mesh_ex4_sln.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   demo script to create volumetric mesh from multiple levelsets
%   from combingin two binary segmented images (head and brain)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

%===============================================================================
%  Load the pre-segmented images
%===============================================================================

%% load full head image (T1 MRI scan)
fprintf(1,'loading binary head image...\n');
for i=1:256
  head(:,:,i)=imread('data_head.tif',i);
end

%% load segmented brain images (by freesurfer recon_all)
fprintf(1,'loading binary brain image...\n');
for i=1:256
  brain(:,:,i)=imread('data_brain.tif',i);
end

%===============================================================================
%  Clean then combine the images
%===============================================================================

fprintf(1,'filling holes in the volumetric images...\n');

%% First, try running the script without the hole filling process.
%% Note how the meshing fails due to intersecting elements.

%% fill holes in the head image and create the canonical binary volume
%% this may take a few minutes for a 256x256x256 volume
%head=fillholes3d(logical(head>0),10);
%brain=fillholes3d(logical(brain>0),10);

%% add brain image as additional segment
cleanimgfull=head+(brain>0);

%===============================================================================
%  Mesh the image and export the output
%===============================================================================

%% create volumetric tetrahedral mesh from the two-layer 3D images
%% this may take another few minutes for a 256x256x256 volume
```

```
%% by default, vol2mesh uses 'cgalsurf' method, which requires the following.
%% set unique values for each 'mask', try changing the values independently.
opt(1).radbound=4; % head surface element size bound
opt(2).radbound=2; % brain surface element size bound
opt(1).side='lower'; %
opt(2).side='lower'; %

%% Mesh the combined volume
[node,elem,face]=vol2mesh(cleanimgfull,1:size(cleanimgfull,1),1:size(cleanimgf
ull,2),1:size(cleanimgfull,3),opt,100,1);

%% Export the mesh
elem(elem==0)=1;
savemsh(node(:,1:3),elem,"data_head.msh");
```