Table of Contents

..... 1
Adding Tube2FEM src directories to path 1
Read graph data from input Excel file 2
Detect Edge Nodes 3
Create Vertices, Elements and Radii arrays 4
Create Binary Image from Graph 5
Dilate Binary Image 6
Get centroids of the white voxels in the dilated Image 7
Create Distance Map 7
Save M for visualization in Paraview 8
Save as 3D Tiff 8
Levelset to Isosurface + Remeshing + Smoothing 9
Exporting STL from Patch data 10
Import STL + mesh 10
Slicing and Labelling 10
Remesh open surface mesh 12
Close the hole and labelling 13
Volumetric Meshing using TetGen 14
Write version 2 ASCII .msh mesh format 17
Convert .msh to .xdmf via meshio 17

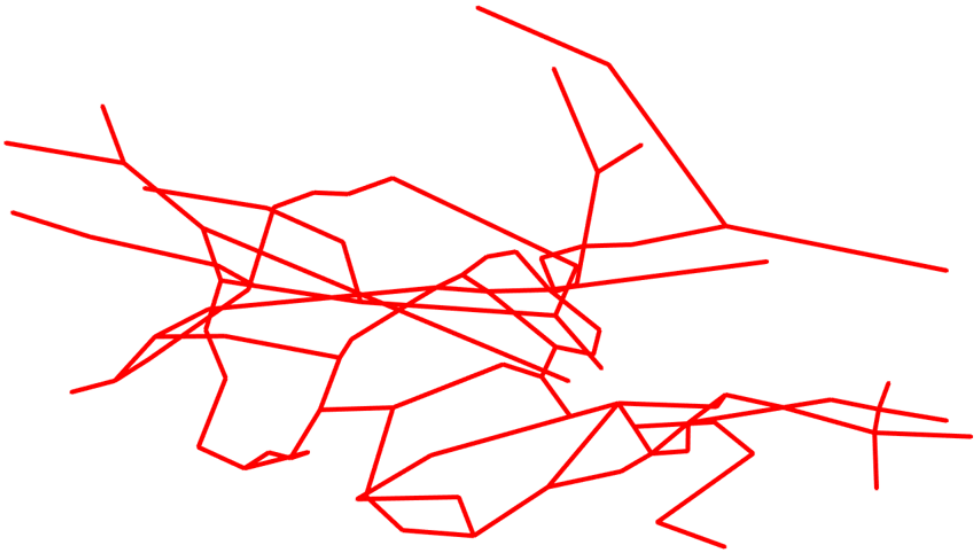

```
clear
clc
close all force
```

Adding Tube2FEM src directories to path

```
CurrentFolder = pwd;
TopFolder = fileparts(pwd);
TopTopFolder = fileparts(fileparts(pwd));
srcFolder = strcat(TopTopFolder,'\src');
finiteElementFolder = strcat(srcFolder,'\FiniteElement');
boundaryConditionsFolder = strcat(srcFolder,'\boundaryConditions');
postprocessingParaViewFolder = strcat(srcFolder,'\postprocessingParaView');
surfaceMeshProcessingFolder = strcat(srcFolder,'\surfaceMeshProcessing');
volumetricMeshFolder = strcat(srcFolder,'\volumetricMesh');
skeletonisationFolder = strcat(srcFolder,'\skeletonisation');

addpath(srcFolder);
addpath(finiteElementFolder);
addpath(boundaryConditionsFolder);
addpath(postprocessingParaViewFolder);
addpath(surfaceMeshProcessingFolder);
addpath(volumetricMeshFolder);
addpath(skeletonisationFolder);
```

Read graph data from input Excel file

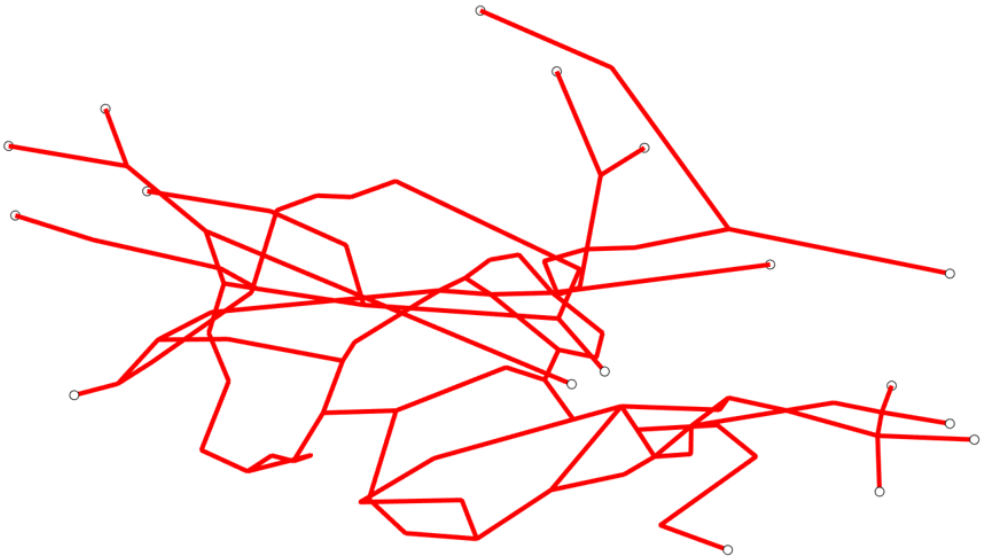
```
[data,txt] = xlsread('Input/SecombRatTum98.xlsx');

% Shifting the graph in the (x,y) space
for i = 1:size(data,1)
    data(i,7) = data(i,7)+30;
    data(i,8) = data(i,8)+30;
    data(i,10) = data(i,10)+30;
    data(i,11) = data(i,11)+30;
end

% Change very small radii to 5 micons minimum value (avoid heavy remeshing)
radii=data(:,5);
radii(radii<=6)=6;
data(:,5)=radii;

% Plot the network graph
cFigure
for i = 1:size(data,1)
     startNode1 = data(i,3);
     endNode1   = data(i,4);
     barRadius1 = data(i,5);
     barLength1 = data(i,6);
     startNodeCoor1 = [data(i,7) data(i,8) data(i,9)];
     endNodeCoor1 = [data(i,10) data(i,11) data(i,12)];
     line([startNodeCoor1(1) endNodeCoor1(1)], ...
         [startNodeCoor1(2) endNodeCoor1(2)], ...
         [startNodeCoor1(3) endNodeCoor1(3)])
     hold on
% Sample each segment depending on its radius for optimisation purposes
    if barRadius1 < 18
        n = round(barLength1/5);
    else
        n = round(barLength1/11);
    end
     V = [startNodeCoor1;endNodeCoor1];
     V =evenlySampleCurve(V,n,'linear',0);
     VN_all(1:n,3*i-2:3*i) = V;
     plotV(V,'r.-','lineWidth',3);
end

view([-45,45])
axisGeom
zoom(1.3)
axis off
hold on
```



Detect Edge Nodes

Detect edge nodes

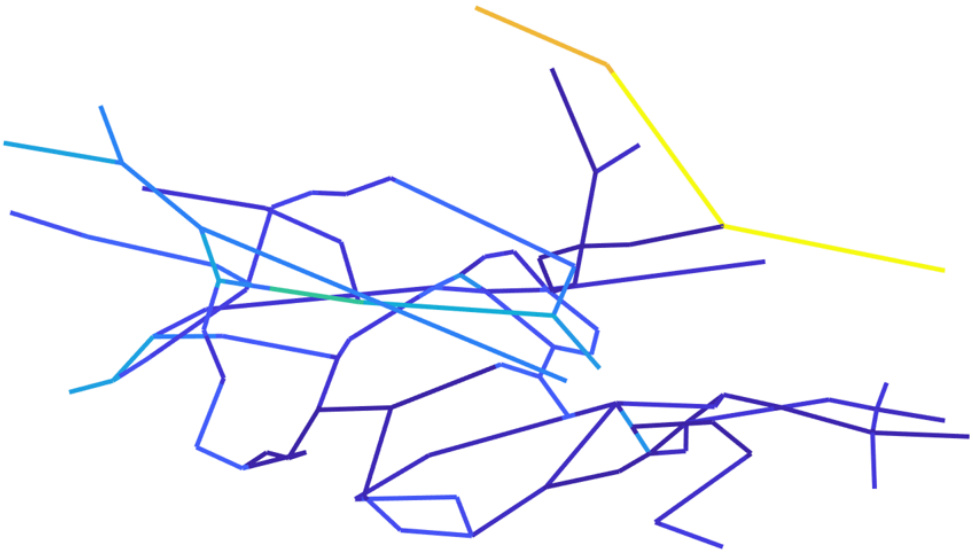
```
[EdgeInput,EdgeInputCoor,labelCount] = EdgeInputDetection(data);
% Detect edge output nodes
[EdgeOutput,EdgeOutputCoor] = EdgeOutputDetection(data,labelCount);
% Edge nodes
EdgePtCoor = [EdgeInputCoor ; EdgeOutputCoor];
```



Create Vertices, Elements and Radii arrays

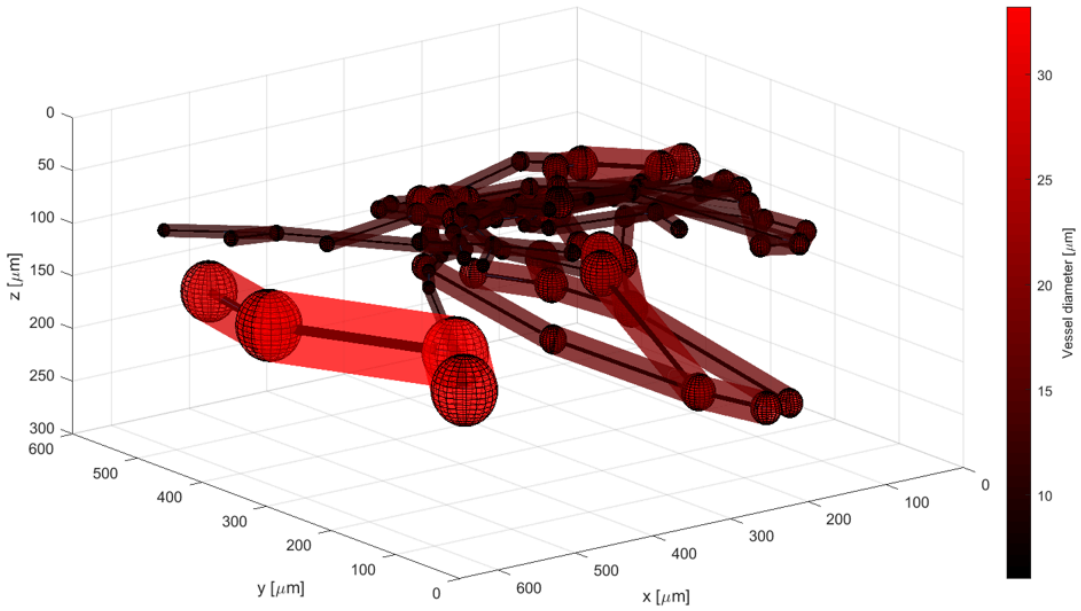
```
CellArray_V = cell(1,size(data,1));
CellArray_E = cell(1,size(data,1));
CellArray_R = cell(1,size(data,1));

for i = 1:size(data,1)
    barRadius1 = data(i,5);
    barLength1 = data(i,6);
    if barRadius1 < 18
        n = round(barLength1/5);
    else
        n = round(barLength1/11);
    end
    V = VN_all(1:n,3*i-2:3*i);
    E = [(1:size(V,1)-1)' (2:size(V,1))'];
    barRadius = data(i,5);
    CellArray_V{1,i} = V;
    CellArray_E{1,i} = E;
    CellArray_R{1,i} = linspace(barRadius,barRadius,size(V,1))';

end

barRadii = CellArray_R{1,1};
for i = 2:size(data,1)
    barRadii = [barRadii; CellArray_R{1,i}];
end
[E,V_Skel]=joinElementSets(CellArray_E,CellArray_V);
[E,V_Skel,ind1]=mergeVertices(E,V_Skel);
barRadii=barRadii(ind1);
```

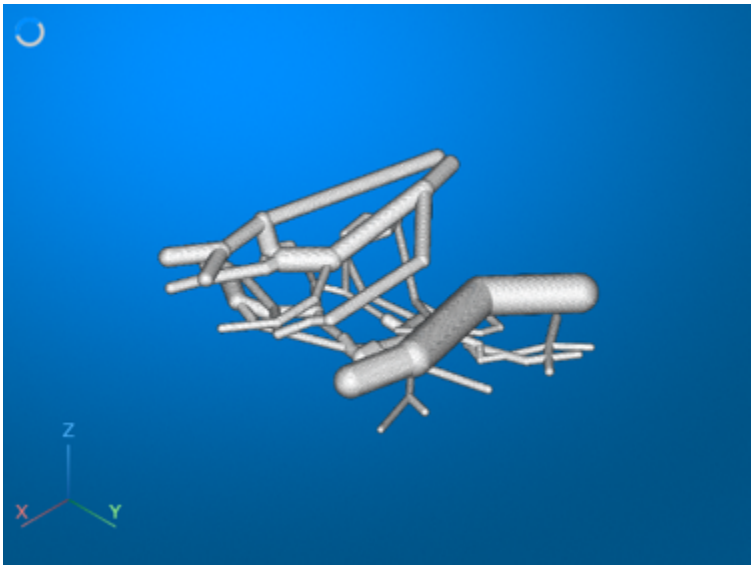
```
% Square it for computational purposes
barRadii = barRadii.^2;

% Plot graph + radii field
cFigure;
gpatch(E,V_Skel,'none',sqrt(barRadii),0,3);
axisGeom;
zoom(1.3)
axis off
drawnow;
```



Create Binary Image from Graph

```
binaryNetwork = binaryImageFromGraph(data);
v= volshow(binaryNetwork);
```





Dilate Binary Image

Define a 3D structuring element for dilation

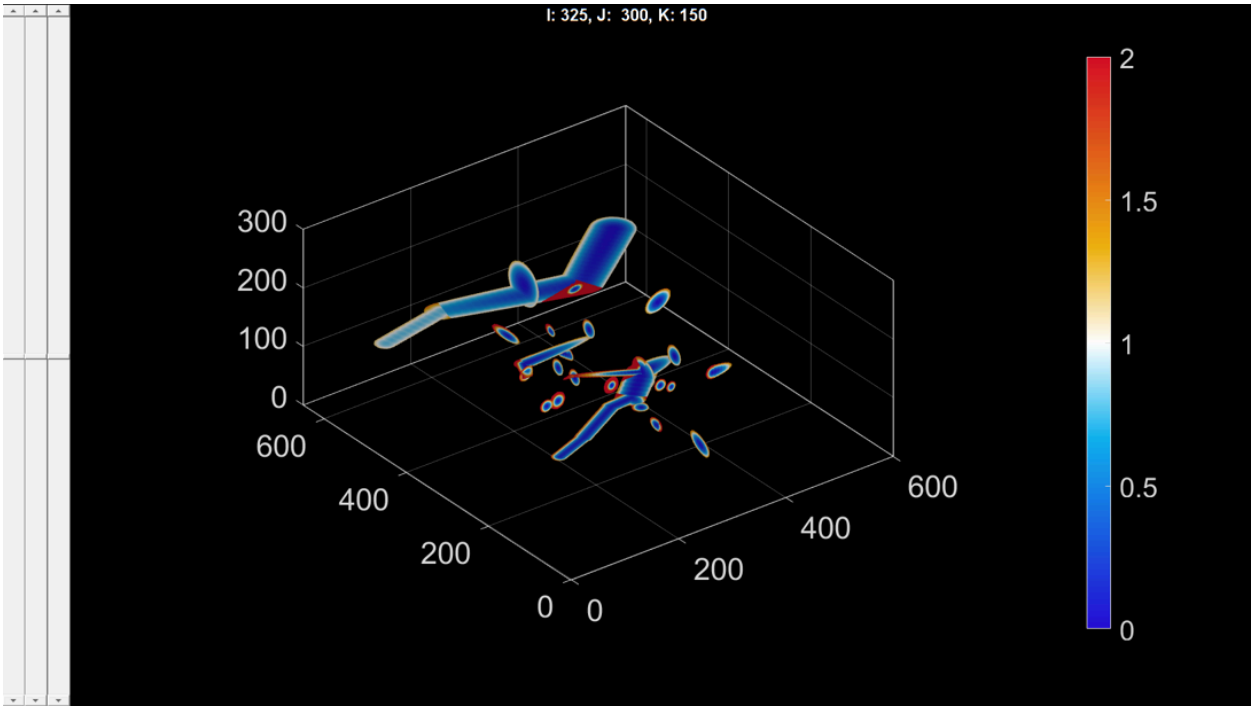
```
se = strel('sphere', 3);

% Perform a dilation on the 3D image (1 dilation is enough)
dilatedImage = imdilate(binaryNetwork, se);
pause(2)
%v2 = volshow(dilatedImage);
```

Get centroids of the white voxels in the dilated Image

```
stats = regionprops3(dilatedImage, 'Centroid', 'VoxelIdxList');
[row,col,slice] = ind2sub(size(dilatedImage),stats.VoxelIdxList{255,1});
V_Dilated = [row,col,slice];
```

Create Distance Map

```
pointSpacing=1;
voxelSize=pointSpacing*ones(1,3);
contourLevel=1;
maxX = 50*(ceil(max(data(:,7))/50.)+1);
maxY = 50*(ceil(max(data(:,8))/50.)+1);
maxZ = 50*(ceil(max(data(:,9))/50.)+1);
x=1:voxelSize:maxX;
y=1:voxelSize:maxY;
z=1:voxelSize:maxZ;
[X,Y,Z]=ndgrid(x,y,z);
V_grid=[X(:) Y(:) Z(:)];

indVec = stats.VoxelIdxList{255,1};

% Get Distance between Voxel Centroids and Nodes on Graph
[M_Dilated,ind_Dilated]=minDistMod(V_Dilated,V_Skel);

M_grid = nan*zeros(size(V_grid,1),1);
M_grid(indVec) = M_Dilated;
indMin = 190*ones(size(V_grid,1),1);
indMin(indVec) = ind_Dilated;

M_grid=M_grid./barRadii(indMin);
M=reshape(M_grid,size(X));

% Visualisation
vizStruct.colormap=warmcold(250);
vizStruct.clim=[0 contourLevel*2];
sv3(fliplr(M),voxelSize,vizStruct);
camlight headlight;
```



Save M for visualization in Paraview

```
cd Input
binaryDir = fullfile('Binary'); % Ensure correct path

if isfolder(binaryDir)  % Use isfolder() instead of exist()
    rmdir(binaryDir, 's'); % Remove directory and its contents
    fprintf('Deleted existing Binary directory and its contents.\n');
end


% Create a new "Binary" directory
mkdir(binaryDir);
cd(binaryDir);
for k=1:size(M,3)
  tiff = M(:, :, k);
  outputFileName = sprintf('M%d.tiff', k);
  imwrite(tiff,outputFileName,'WriteMode', 'append','Compression','none')
end
cd ../..
```

Deleted existing Binary directory and its contents.

Save as 3D Tiff

```
inputPath = strcat(CurrentFolder,'\Input\Binary');
inputPath = strrep(inputPath, '\', '/');

outputPath = strcat(CurrentFolder,'\Input\');
```

```
outputPath = strrep(outputPath, '\', '/');
pyrunfile("save3DTiff.py",inputPath=inputPath, outputPath=outputPath)
```

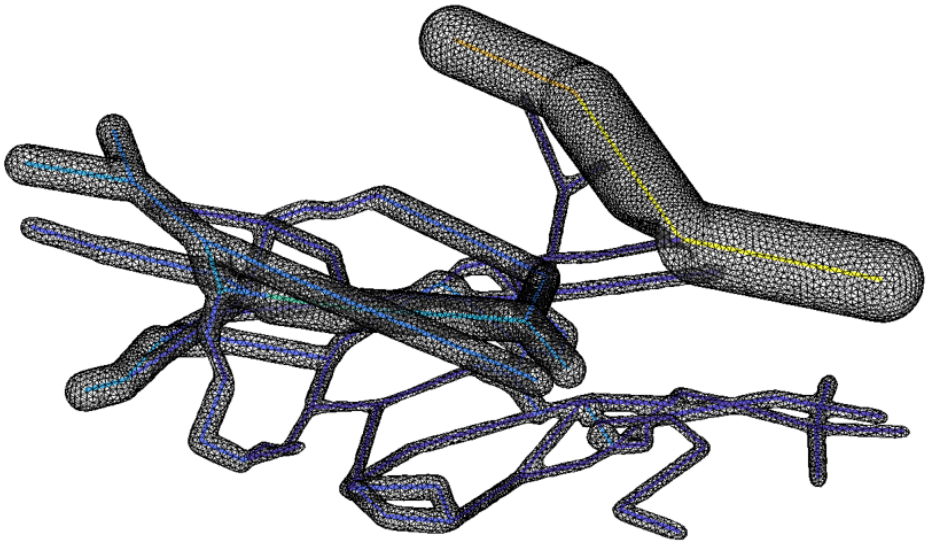
```
C:/Users/homeuser/Documents/GitHub/Tube2FEM/caseStudies/
caseStudy3_SecombNetwork/Input/Binary
Current working directory: C:\Users\homeuser\Documents\GitHub\Tube2FEM
\caseStudies\caseStudy3_SecombNetwork\Input\Binary
TIFF Image Shape: (650, 600)
```

Levelset to Isosurface + Remeshing + Smoothing

```
% Extract Isosurface from levelset (distance map) + Clean Patch
imOrigin=min(V_Dilated,[],1)-voxelSize/2;
controlPar_isosurface.nSub=[1 1 1];%round(max(v)/2./v);
controlPar_isosurface.capOpt=1; %Option to cap open ended surfaces
controlPar_isosurface.voxelSize=voxelSize;
controlPar_isosurface.contourLevel=contourLevel;
[Fi,Vi]=levelset2isosurface(M,controlPar_isosurface); %Get iso-surface
Fi_sorted=sort(Fi,2);
logicInvalid=any(diff(Fi_sorted,1,2)==0,2);
Fi=Fi(~logicInvalid,:);
[Fi,Vi]=patchCleanUnused(Fi,Vi);
Vi=Vi(:,[2 1 3]);

[Fi,Vi]=triSurfRemoveThreeConnect(Fi,Vi);
[Fi,Vi]=patchCleanUnused(Fi,Vi);

% Remeshing using Geogram
optionStructGG.pointSpacing= pointSpacing*6;
[Fi,Vi]=ggremesh(Fi,Vi,optionStructGG);

% Smoothing using Humphrey Classes method
Eb=patchBoundary(Fi,Vi);
controlPar_smooth.Method='HC';
controlPar_smooth.Alpha=0.1;
controlPar_smooth.Beta=0.5;
controlPar_smooth.n=150;
controlPar_smooth.RigidConstraints=unique(Eb(:));

[Vi]=patchSmooth(Fi,Vi,[],controlPar_smooth);
Eb=patchBoundary(Fi,Vi);

% Visualisation
cFigure;
gpatch(Fi,Vi,'w','k',0.2);
axisGeom;
camlight headlight;
drawnow;
hold on
gpatch(E,V_Skel,'none',sqrt(barRadii),0,3);
drawnow
```

```
zoom(1.3)
axis off
```

Warning: The patch_area function is depricated. Use patchArea instead.



Exporting STL from Patch data

```
cd Mesh
patch2STL('fullNetworkOptimised.stl',Vi,Fi,[],'Network');
```

Warning: The trinorm function is depricated. Use patchNormal instead.

Import STL + mesh

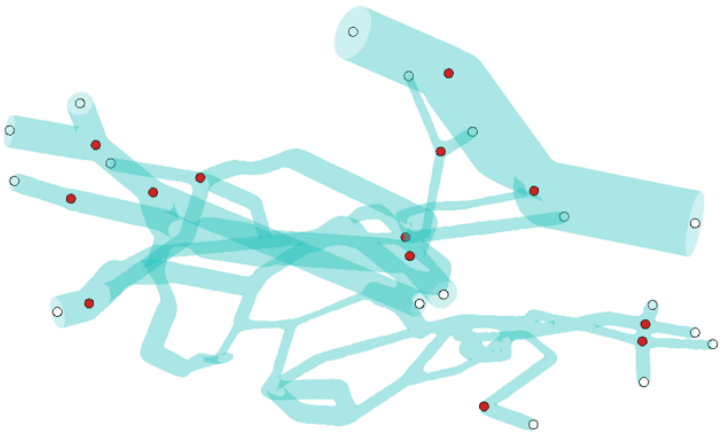
```
fileName = 'fullNetworkOptimised.stl';
[stlStruct] = import_STL(fileName);

F=stlStruct.solidFaces{1};
V=stlStruct.solidVertices{1};
% Merging nodes (nodes are not merged in stl)
[F,V]=mergeVertices(F,V);
```

Slicing and Labelling

```
dirVec = zeros(size(EdgePtCoor,1),3);
indAll = [1:size(EdgePtCoor,1)];
notWorking = [];
flag = ~ismember(indAll,notWorking);
```

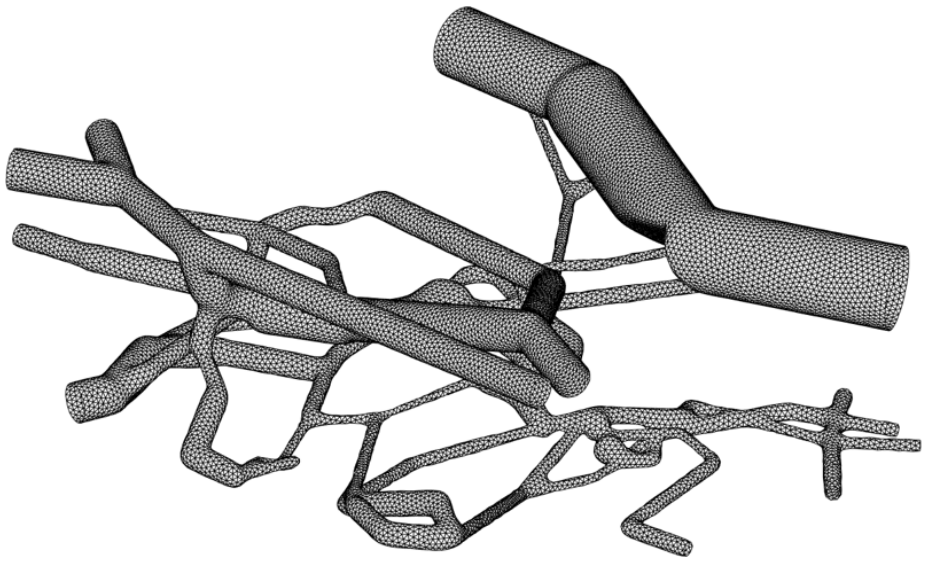
```
index = find(flag);
cFigure

for i =1:size(EdgePtCoor,1)

    i;
    indexi = index(i);
    C=[];
    snapTolerance=mean(patchEdgeLengths(F,V))/100;
    Pin=[EdgePtCoor(indexi,6),EdgePtCoor(indexi,7),EdgePtCoor(indexi,8)];
    Pi = [EdgePtCoor(indexi,3),EdgePtCoor(indexi,4),EdgePtCoor(indexi,5)];
    n= Pin - Pi;
    % Extract the Surface Mesh Faces and Vertices that are inside
    % a 4*radius Sphere having its center (Pi)
    [D]=minDist(V,Pi);
    radius = EdgePtCoor(indexi,9);
    logicDist=D<4*radius;
    logicCut=any(logicDist(F),2);
    Fa= F(logicCut,:);
    Fb = F(~logicCut,:);

    hold on
    axisGeom
    scatter3(Pi(1),Pi(2),Pi(3),'MarkerEdgeColor','k', ...
        'MarkerFaceColor',[1. 1. 1.])
    scatter3(Pin(1),Pin(2),Pin(3),'MarkerEdgeColor','k', ...
        'MarkerFaceColor',[1. 0. 0.])

    % Check if Fa is composed of multiple disconnected surfaces
    a=(120/180)*pi;
    G=patchFeatureDetect(Fa,V,a);

    if max(G)~=1
        uniqueG = unique(G);
        sizeUniqueG = size(uniqueG,1);
        count = zeros(sizeUniqueG,1);
            for k = 1:sizeUniqueG
                count(k) = sum(G==uniqueG(k));
            end
        maxCount = index(count == max(count));
        Fb = [Fb;Fa(G~=maxCount,:)];
        Fa = Fa(G==maxCount,:);
    end

    % Slicing - Slicing plane is defined by point (Pcut) and a Normal (n)
    Pcut = 0*Pin + 1*Pi;
    [Fa,Va,~,logicSide]=triSurfSlice(Fa,V,C,Pcut,n,snapTolerance);

    % Attaching the Processed Surface back to the Main Network Mesh
    [Fa,Va]=patchCleanUnused(Fa(~logicSide,:),Va);
    [F1,V1]=joinElementSets({Fa,Fb},{Va,V});
    [F,V]=mergeVertices(F1,V1);
end
```

```
C = ones(size(F,1),1);
patch('faces',F,'vertices',V,'FaceColor','flat','CData',C, ...
    'FaceAlpha',0.2,'EdgeColor','none');
axisGeom
axis off
zoom(1.3)
```



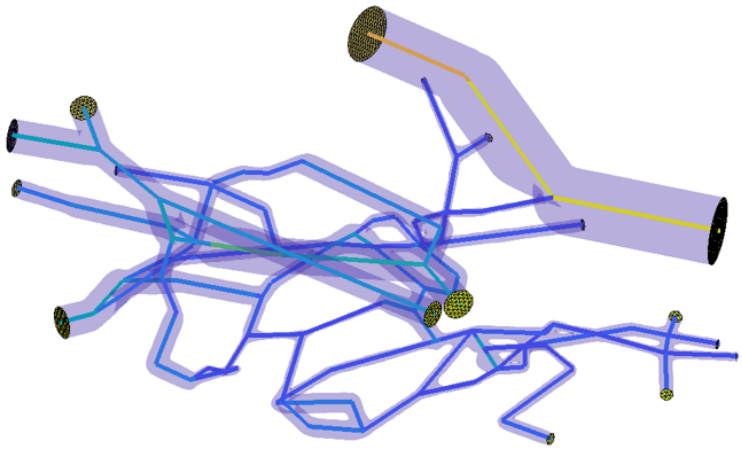
Remesh open surface mesh

```
nb_pts = 25000; % going below 25000 nodes will remove a vessel (Careful)
optionStruct2.nb_pts = nb_pts;
[F,V] = ggremesh(F,V,optionStruct2);
cFigure;
gpatch(F,V,'w','k');
axisGeom;
axis off
zoom(1.3)
```



Close the hole and labelling

```
cFigure
C = ones(size(F,1),1);
[F,V,C]=closeHolesAndLabelling(F,V,C);
patch('faces',F,'vertices',V,'FaceColor','flat','CData',C, ...
    'FaceAlpha',0.2,'EdgeColor','none');
axisGeom
gpatch(E,V_Skel,'none',sqrt(barRadii),0,3);
axis off
zoom(1.3)
```



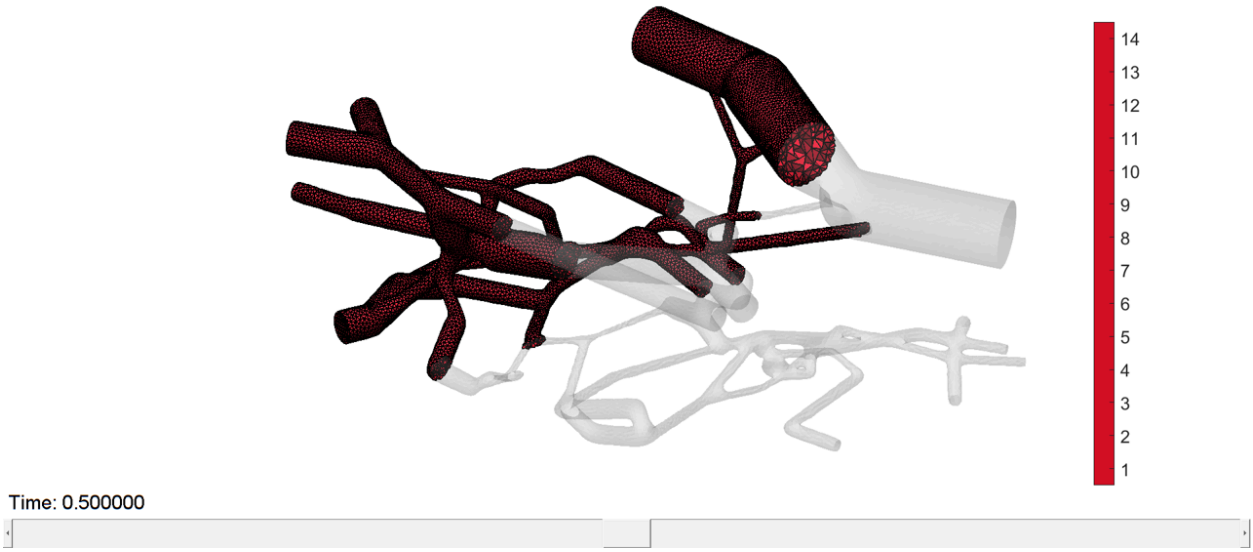
Volumetric Meshing using TetGen

```
V_regions=getInnerPoint(F,V); %Define region points
V_holes=[]; %Define hole points
[regionTetVolumes]=tetVolMeanEst(F,V); %Volume estimate for regular tets
stringOpt='-pq1.2AaY';
modelName = 'test';

% Mesh using TetGen
% Create Tetgen input structure
inputStruct.stringOpt=stringOpt; %Tetgen options
inputStruct.Faces=F; %Boundary faces
inputStruct.Nodes=V; %Nodes of boundary
inputStruct.faceBoundaryMarker=C;
inputStruct.regionPoints=V_regions; %Interior points for regions
inputStruct.holePoints=V_holes; %Interior points for holes
inputStruct.regionA=regionTetVolumes; %Desired tet volume for each region
inputStruct.modelName = modelName;

% Run TetGen
[meshOutput]=runTetGen(inputStruct);

% Access mesh output structure
E=meshOutput.elements; %The elements
VLung=meshOutput.nodes; %The vertices or nodes
CE=meshOutput.elementMaterialID; %Element material or region id
FLung=meshOutput.facesBoundary; %The boundary faces
Cb=meshOutput.boundaryMarker; %The boundary markers

% Visualization
```

```
hf=cFigure;
hold on;
% Creating single-domain volumetric mesh
meshOutput.elementMaterialID = ones(size(meshOutput.elementMaterialID,1),1);

% Visualizing using meshView
optionStruct.hFig=[hf];
optionStruct.hFig.WindowState = "maximized";
meshView(meshOutput,optionStruct);
% title('Tetrahedral mesh','FontSize',fontSize);
fontSize=15;
axisGeom(gca,fontSize);
gdrawnow;
axis off
camlight('left')
zoom(1.3)
% print(gcf,'SecombVolumetricMesh.png','-dpng','-r600');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--- TETGEN Tetrahedral meshing --- 25-Apr-2025 06:39:18

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--- Writing SMESH file --- 25-Apr-2025 06:39:18

smeshName =

    'C:\Program Files\MATLAB\GIBBON-master\data\temp\test.smesh'

----> Adding node field
----> Adding facet field
----> Adding holes specification
----> Adding region specification
--- Done --- 25-Apr-2025 06:39:18

runString =

    '"C:\Program Files\MATLAB\GIBBON-master\lib_ext\tetGen\win64\tetgen.exe" -
pq1.2AaY "C:\Program Files\MATLAB\GIBBON-master\data\temp\test.smesh"'

--- Running TetGen to mesh input boundary--- 25-Apr-2025 06:39:18
Opening C:\Program Files\MATLAB\GIBBON-master\data\temp\test.smesh.
Delaunizing vertices...
Delaunay seconds:  0.165
Creating surface mesh ...
Surface mesh seconds:  0.037
Recovering boundaries...
Boundary recovery seconds:  0.125
Removing exterior tetrahedra ...
Spreading region attributes.
Exterior tets removal seconds:  0.062
Recovering Delaunayness...
Delaunay recovery seconds:  0.04
Refining mesh...
```

```
Refinement seconds:  0.848
Optimizing mesh...
Optimization seconds:  0.083

Writing C:\Program Files\MATLAB\GIBBON-master\data\temp\test.1.node.
Writing C:\Program Files\MATLAB\GIBBON-master\data\temp\test.1.ele.
Writing C:\Program Files\MATLAB\GIBBON-master\data\temp\test.1.face.
Writing C:\Program Files\MATLAB\GIBBON-master\data\temp\test.1.edge.

Output seconds:  0.43
Total running seconds:  1.79

Statistics:

  Input points: 25503
  Input facets: 51064
  Input segments: 76595
  Input holes: 0
  Input regions: 1

  Mesh points: 59974
  Mesh tetrahedra: 290409
  Mesh faces: 606350
  Mesh faces on exterior boundary: 51064
  Mesh faces on input facets: 51064
  Mesh edges on input segments: 76595
  Steiner points inside domain: 34471

--- Done --- 25-Apr-2025 06:39:20

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--- Importing TetGen files --- 25-Apr-2025 06:39:20
--- Done --- 25-Apr-2025 06:39:20
```



Write version 2 ASCII .msh mesh format

```
GmshFileName = 'SecombGmshFormat.msh';
[status] = writeGmsh(GmshFileName,meshOutput);
```

Gmsh Version 2 ASCII written

Convert .msh to .xdmf via meshio

```
meshPath = strcat(CurrentFolder,'\Mesh');
meshPath = strrep(meshPath, '\', '/');
% Split the path into parts
pathParts = strsplit(meshPath, '/');
% Reconstruct the path without the first directory
newMeshPath = strjoin(pathParts(2:end), '/');
% Get WSL path
rootPath = "../../../../../../../../../../../../../../../../../../../mnt/c/";
wslMeshPath = strcat(rootPath,newMeshPath,"/meshioConvertMesh.py");
wslMeshPath= sprintf('"%s"', wslMeshPath);

fid1 = fopen('meshInfo.txt','wt');
meshPath1 = sprintf('"%s"', strcat(rootPath,newMeshPath));
fprintf(fid1,strcat('meshPath=',meshPath1));
fprintf(fid1,'\n');
GmshFileName= sprintf('"%s"', GmshFileName);
fprintf(fid1,strcat('GmshFileName=',GmshFileName));
fclose(fid1);

% Create a script combining meshInfo.txt
% and the general ConvertGmshToXdmf.py from src
script1 = fileread('meshInfo.txt');
script2 = fileread('../../../src/volumetricMesh/ConvertGmshToXdmf.py');

fid2 = fopen('meshioConvertMesh.py','wt');
fprintf(fid2,script1);
fprintf(fid2,'\n');
fprintf(fid2,script2);
fprintf(fid2,'\n');
fclose(fid2);

% Run Python on WSL
wslPath = '"C:\Windows\System32\wsl.exe"';
runString = strcat(wslPath,' python3',append(' ',wslMeshPath));
[runStatus,runOut]=system(runString,'-echo');
```

Current working directory: /mnt/c/Users/homeuser/Documents/GitHub/Tube2FEM/caseStudies/caseStudy3_SecombNetwork/Mesh

Published with MATLAB® R2023a