Im2mesh_GUI Tutorial

Jiexian Ma*

This is a tutorial for Im2mesh_GUI v2.18, which is developed by Jiexian Ma. Im2mesh_GUI is a graphical user interface (GUI) version of Im2mesh, an open-source MATLAB/Octave package.[1] It's used to generate finite element mesh based on two-dimensional (2D) segmented multi-phase image. With GUI, Im2mesh becomes much easier to use. User-friendly! It has incorporated many features, such as polyline smoothing and simplification. Through this tutorial, you will understand the workflow and the parameters of Im2mesh_GUI.

Installation

Im2mesh_GUI can be installed in two ways: MATLAB app or standalone desktop application.

For using as MATLAB app, you need to install MATLAB software, Image Processing Toolbox, and Mapping Toolbox. When MATLAB software is opened, you can install Im2mesh_GUI by running "Im2mesh_GUI.mlappinstall". After installation, you should be able to find Im2mesh_GUI in the APPS tab of MATLAB.

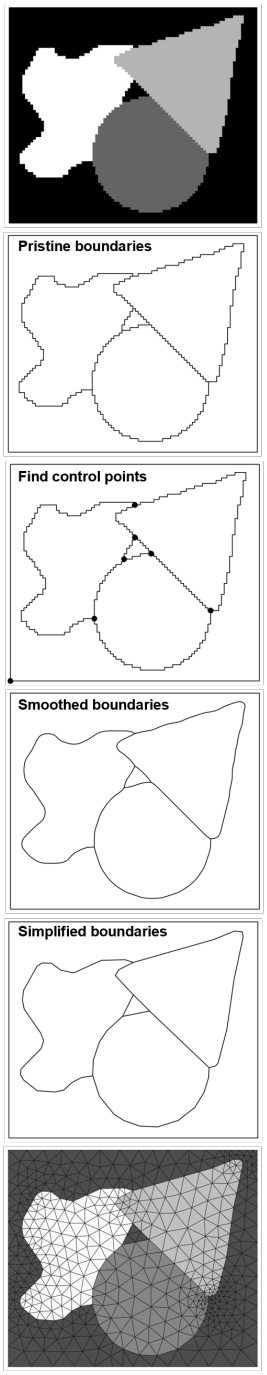
For standalone desktop application, you don't need to install MATLAB software or any MATLAB toolbox. However, you need to install MATLAB Runtime (version 9.11). When you open "Installer_Im2mesh_GUI.exe", the installer will automatically download and install MATLAB Runtime for you. Note that MATLAB Runtime is free. After installation, you can find the application in the following file directory: `C:\Program Files\Im2mesh_GUI\application\Im2mesh_GUI.exe`. When you start the application, it may take a while to initialize. Just be patient.

Workflow

Figure 1 shows the workflow of Im2mesh. The workflow is straightforward. The GUI window has organized according to the workflow (Figure 2). The workflow consists of 9 steps. Some steps are optional: step 2, step 5, and step 7. When running Im2mesh_GUI, you can skip those optional steps if they doesn't work for your image. If you plan to use pixelMesh (pixel-based quadrilateral mesh) as mesh generator, the workflow would be much simpler: step 1-3 and 7-9.

*mjx0799@gmail.com

[1] https://github.com/mjx888/im2mesh



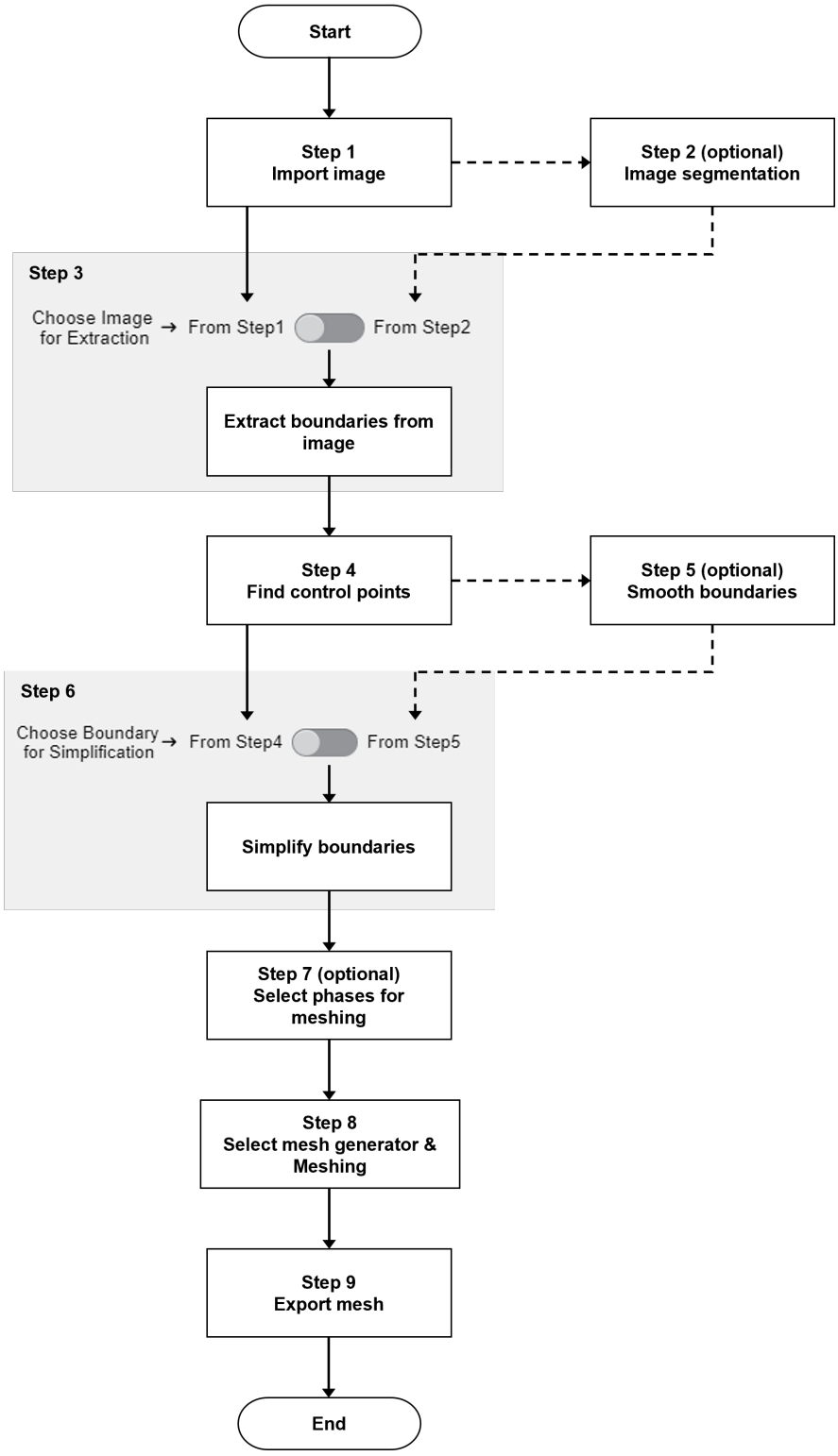


Figure 1: Workflow of Im2mesh

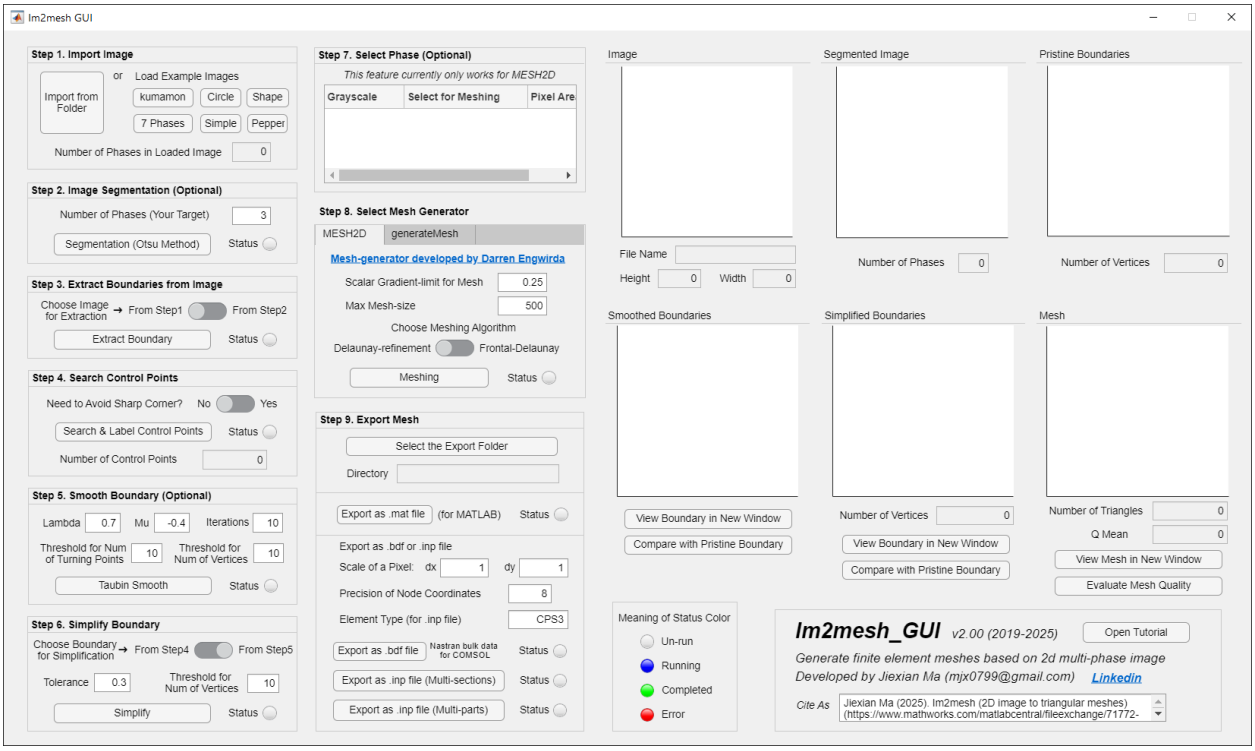


Figure 2: Im2mesh_GUI

Step-by-step Guide

Step 0. Image processing. It's often necessary to perform digital image processing to your raw image. For example, converting to 8-bit grayscale image, noise removal, and image segmentation. You can refer to the following links about how to perform image processing in MATLAB:

`https://www.mathworks.com/help/images/noise-removal.html`
`https://www.mathworks.com/discovery/image-segmentation.html`
`https://www.mathworks.com/help/images/image-segmentation.html`

Step 1. Import image. You can import from your folder, or just load one of the example images. An ideal input image for Im2mesh is a segmented grayscale image. That's because Im2mesh identifies different material phases in an image by their grayscales. Different grayscales correspond to different material phases. If you have 4 levels of grayscale in an image, the resulting mesh will contain 4 phases.

Step 2. Image segmentation. This step is optional. I include this step just for convenience. If you don't need segmentation, just skip this step.

Step 3. Extract polygonal boundaries from image. You can choose your input for this step by clicking the Switch. For example, if your input is from Step 2, just click the

right side of the switch. This step uses a subroutine (`getExactBounds.m`) developed by me to extract the exact boundary from the image.[2]

Step 4. Search & label control points. Here, control point is defined as the intersection or meeting point between edges in the images (Figure 3). These control points will not change their positions (x, y coordinates) in the following steps. This step uses a subroutine (`getCtrlPnts.m`) developed by me to search and label control points.

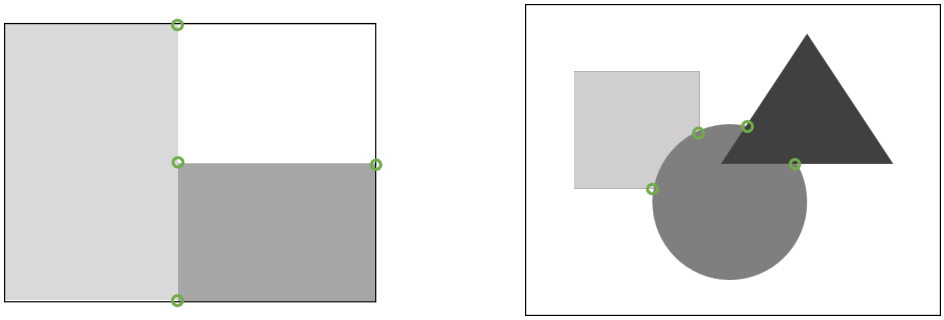


Figure 3: Examples of control points, which are marked with small circles.

I include an option "Need to Avoid Sharp Corners?" in this step. You can set this option to "Yes" in the following cases:

- If you want to avoid sharp corner during boundary smoothing and simplification
- If you find the step of mesh generation has failed for your image
- To avoid bugs

Step 5. Smooth boundary. This step is optional. If your polygonal boundaries don't need smoothing, just skip this step. Note that I found boundary smoothing is beneficial in most cases. This step uses 2D Taubin smoothing method (`taubinSmooth.m`), which can remove noise in polyline without shrinkage. Please refer to the course slide of Dr. Tao Ju if you'd like to learn about Taubin smoothing method.[3]

There are 5 parameters in this step: *Lambda*, *Mu*, *Iterations*, *Threshold for Num of Turning Points*, and *Threshold for Num of Vertices*. The meaning and value range of these parameters are listed as follows.

- *Lambda*: How far each node is moved toward the average position of its neighbours during every second iteration. Type: Float. Range: $0 <$ *Lambda* < 1.
- *Mu*: How far each node is moved opposite the direction of the average position of its neighbours during every second iteration. Type: Float. Range: -1$<$ *Mu* < 0.
- *Iterations*: Number of iterations in Taubin smoothing. If you don't need polyline smoothing, set *Iterations* to 0. Type: Integer. Range: ≥ 0.

[2]https://www.mathworks.com/matlabcentral/fileexchange/72436-getexactbounds-get-exact-boundary
[3]CSE554 lecture slides

• *Threshold for Num of Turning Points*: Threshold value for the number of turning points in a polyline. Only those polylines with number of turning points greater than this threshold will be smoothed. Type: Integer. Range: ≥ 0.

• *Threshold for Num of Vertices*: Threshold value for the number of vertices in a polyline. Only those polylines with number of vertices greater than this threshold will be smoothed. Type: Integer. Range: ≥ 0.

Lambda, *Mu*, and *Iterations* are parameters for Taubin smoothing. The default value is *Lambda* = 0.5, *Mu* = -0.5, *Iterations* = 100. **You can increase *Iterations* if you want smoother boundary. You can reduce *Iterations* if you want rough boundary.**

In this step, there are two thresholds: *Threshold for Num of Turning Points*, and *Threshold for Num of Vertices*. These two parameters are introduced by me to avoid over-smoothing of simple polylines or polygons (Figure 4). Here, a turning point is defined as a vertex where the plane angle between two connected edges is not 180 degrees (Figure 5). By setting a threshold, those polylines with less turning points or less vertices will be skipped in this step so they will not be smoothed (Figure 6). If you want to perform smoothing to all the polylines, set the thresholds to 0.

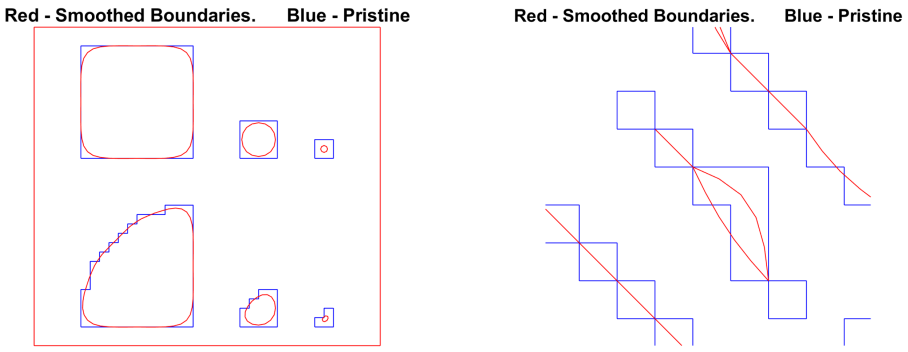


Figure 4: Examples of over-smoothing

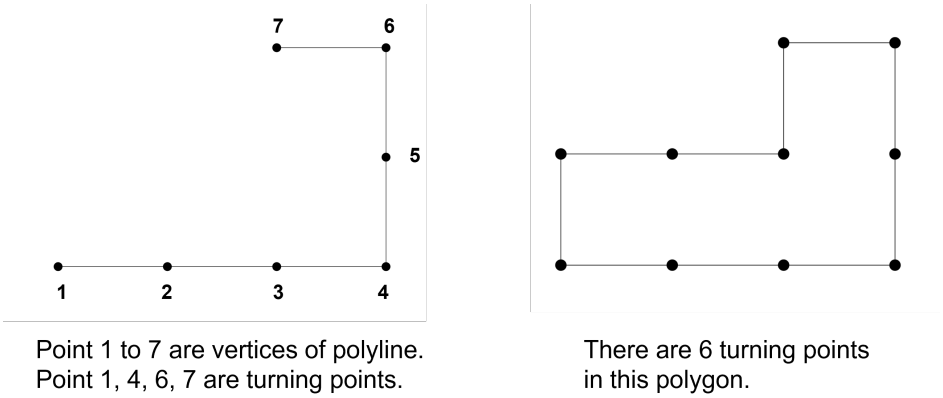


Figure 5: Examples of turning point

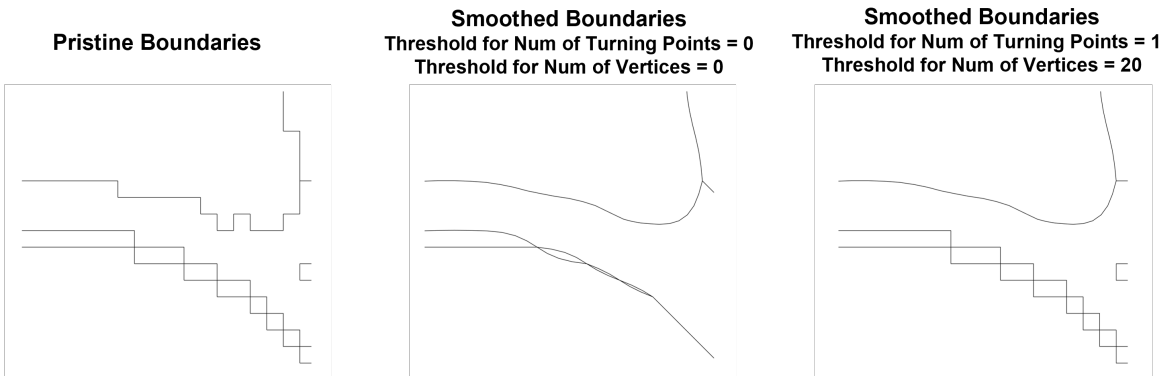


Figure 6: Set thresholds to avoid over-smoothing of simple polylines or polygons

Step 6. Simplify boundary. You can choose your input for this step by clicking the Switch. For example, if your input is from Step 5, just click the right side of the switch. This step uses Douglas-Peucker algorithm to simplify polylines. The subroutine (`dpsimplify.m`) is written by Wolfgang Schwanghart. There are 2 parameters in this step: *Tolerance*, and *Threshold for Num of Vertices*.

- *Tolerance*: The maximum allowable deviation of a vertex from the simplified curve. It's for Douglas-Peucker algorithm. Type: Float. Range: ≥ 0.

- *Threshold for Num of Vertices*: Threshold value for the number of vertices in a polyline. Only those polylines with number of vertices greater than this threshold will be simplified. Type: Integer. Range: ≥ 0.

Higher *Tolerance* means fewer points are retained in polyline and worse approximation to the pristine boundary. If you don't need to simplify polylines, set *Tolerance* to 0. The reasonable value of *Tolerance* depends on your workflow and your image.

- When you're using the boundaries from Step 4 as input, the typical value of *Tolerance* is 0.8 - 2.

- When you're using the boundaries from Step 5 as input, the typical value of *Tolerance* is 0.2 - 0.8.

In this step, there is a parameter called *Threshold for Num of Vertices*. This parameter is used to avoid over-simplification of polyline. Figure 7 shows examples of over-simplification. Some of the over-simplified polygons even have zero area. By setting a threshold, those polylines with less vertices will be skipped in this step so they will not be simplified. If you want to perform simplification to all the polylines, set the threshold to 0.

Once you got the simplified boundaries. You can click button "View Boundary in New Window" or "Compare with Pristine Boundary" and zoom in to check the results. If the resulted boundary is not ideal, you can change parameters of Step 6 and click the button "Simplify" again to obtained new results.

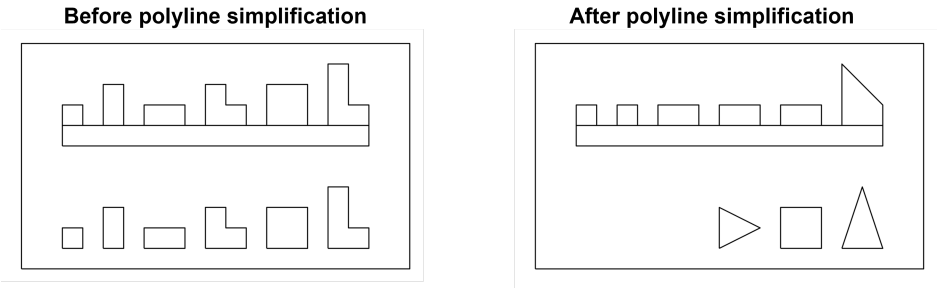


Figure 7: Example of over-simplification

Step 7. Select phase. All the phases are selected in the default case. If you don't need certain phases for meshing, un-check them in the table.

Step 8. Select mesh generator & meshing. There are three mesh generators available in Im2mesh: MESH2D, generateMesh, and pixelMesh.

MESH2D is a mesh generator developed by Darren Engwirda.[4] It's the default mesh generator of Im2mesh_GUI. It can generate high-quality finite element mesh. You are not required to install any MATLAB Toolbox when using MESH2D. There are 3 parameters for MESH2D: *Gradient-limit*, *Max Mesh-size*, and *Meshing Algorithm*.

- *Gradient-limit*: A limit on the gradient of mesh-size function (Figure 8).[5] Type: Float. Range: > 0. Typical value: 0.2 - 0.5.

- *Max Mesh-size*: Maximum mesh edge lengths. This is an approximate upper bound on the mesh edge lengths (Figure 9). Type: Float. Range: > 0.

- *Meshing Algorithm*: Method used to create mesh-size functions based on an estimate of the "local-feature-size" associated with a polygonal domain. Value: Delaunay-refinement or Frontal-Delaunay.

generateMesh is a MATLAB built-in function.[6] It requires installation of MATLAB Partial Differential Equation Toolbox. I developed a subroutine (`poly2meshBuiltIn.m`) to make the mesh generation for multi-parts become possible. There are 3 parameters for generateMesh: *Mesh Growth Rate*, *Max Mesh Edge Length*, and *Min Mesh Edge Length*.

- *Mesh Growth Rate*. The rate at which the mesh transitions between regions of different edge size. Type: Float. Range: $1 \leq$ *Mesh Growth Rate* ≤ 2. Typical value: 1.2 - 1.5.

- *Max Mesh Edge Length*: An approximate upper bound on the mesh edge lengths. Type: Float. Range: > 0.

[4] https://github.com/dengwirda/mesh2d
[5] http://persson.berkeley.edu/pub/persson04gradlim.pdf
[6] https://www.mathworks.com/help/pde/ug/pde.pdemodel.generatemesh.html

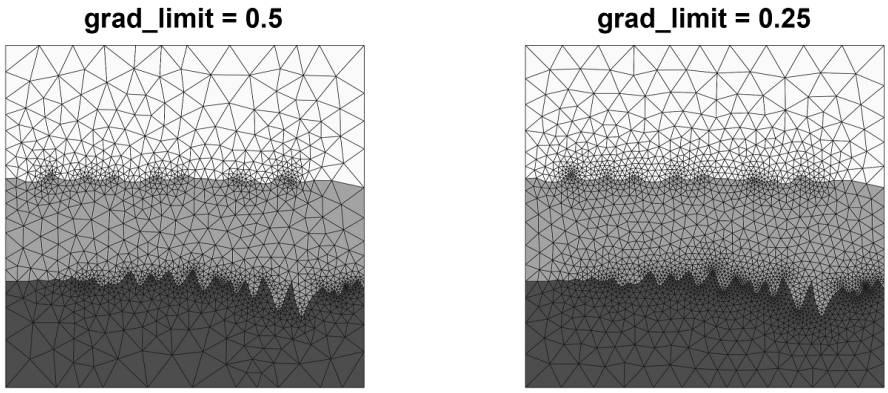


Figure 8: Effect of *Gradient-limit*

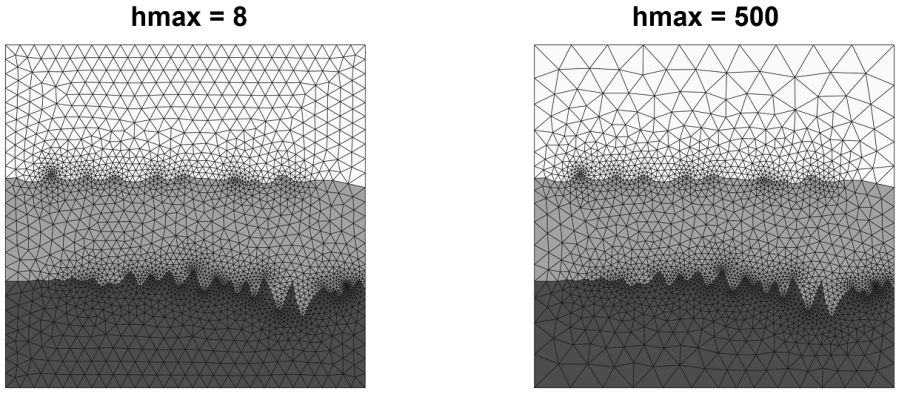


Figure 9: Effect of *Max Mesh-size*

• *Min Mesh Edge Length*: An approximate lower bound on the mesh edge lengths. Type: Float. Range: ≥ 0.

pixelMesh is a function used to generate pixel-based quadrilateral mesh. It's written by me many years ago. It's for some special applications.

The step of meshing is the most time-consuming step in Im2mesh. The computation time of meshing is dependent on the number of vertices in the simplified boundaries. I ran tests on my desktop computer with 4-phase images and with option "Need to Avoid Sharp Corners?" = Yes. For 15,000 vertices in the simplified boundaries, MESH2D took 40-50 seconds. For 80,000 vertices in the simplified boundaries, MESH2D took 300-450 seconds.

What to do if the mesh generation has failed for your images?

• Set the option "Need to Avoid Sharp Corners?" to Yes, and re-run Step 4 to Step 8

• Set the thresholds in Step 5 or 6 to a higher value (e.g., 20), and re-run Step 5 to Step 8

Once you obtain the mesh from MESH2D or generateMesh. You can click button "View Mesh in New Window" or "Evaluate Mesh Quality" to check the results. "Evaluate

Mesh Quality" use a subroutine (`tricost.m`) written by Darren Engwirda to evaluate mesh quality. "Evaluate Mesh Quality" has 3 outputs (Figure 10). The definition of Q and θ are as follows.

- Q: Area-length measure of a triangle. $Q = \frac{4\,A}{\sqrt{3}\,l_{rms}^2}$, where A is the area of a triangle and l_{rms} is the root-mean-squared edge length in a triangle. Q achieve a score of 1 for ideal elements. Q approaches zero as an element distorts toward degeneracy.
- θ: The plane angle between adjacent edges.

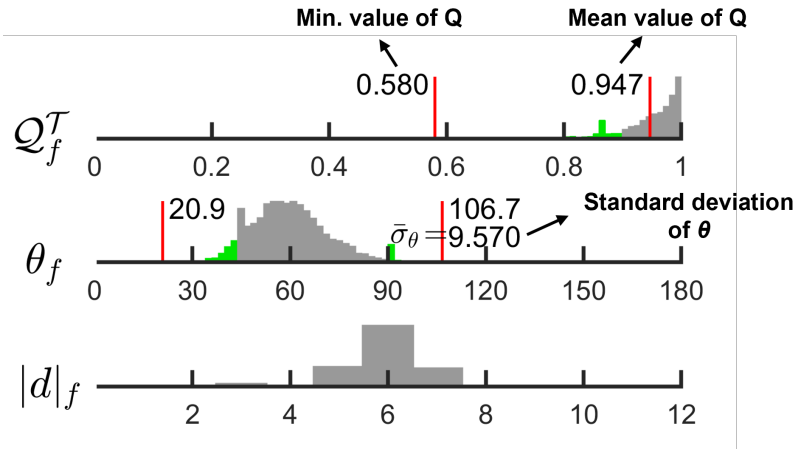


Figure 10: An example of mesh quality. A few statistics can be obtained from the plot.

If the resulted mesh is not ideal, you can change parameters of Step 8 and click the button "Meshing" again to obtained new results. You may also go back to Step 5 or 6 to change parameters. But once you do that, you need to re-run all the steps after Step 5 or 6 to get the updated results.

What are the critical parameters that affect the number of triangular mesh?

- *Tolerance* in Step 6.
- *Gradient-limit* in Step 8, if you are using MESH2D.
- *Mesh Growth Rate* and *Min Mesh Edge Length* in Step 8, if you are using generateMesh.

My suggestion for choosing the value of these critical parameters:

- First, try different *Tolerance* value in an ascending order and click button "Compare with Pristine Boundary" to view the corresponding simplified boundaries. Identify the largest *Tolerance* value that provides an acceptable approximation to the pristine boundaries. With a larger *Tolerance* value, the number of generated triangles in Step 8 will decrease.
- After that, try different *Gradient-limit* and evaluate the corresponding mesh quality Q. Increasing the *Gradient-limit* will reduce the number of generated triangles. However, a higher *Gradient-limit* will also lower the mesh quality Q. You may perform a mesh convergence analysis to find the largest acceptable *Gradient-limit* for your cases.

Step 9. Export mesh. Im2mesh_GUI supports exporting linear element and quadratic element. You can export mesh as the following files.

- `mat` file (MATLAB)
- `xls` file (Excel)
- `inp` file (Abaqus)
- `bdf` file (Nastran bulk data, compatible with COMSOL)

`MAT` file

If you use MESH2D or pixelMesh as mesh generator, the exported `mat` file would have the following variables: `vertL`, `eleL`, `pcode`, `vertQ`, `eleQ`.

If you use generateMesh as mesh generator, the exported `mat` file would have the following variables: `vertL`, `eleL`, `pcode`, `vertQ`, `eleQ`, `meshL`, `meshQ`.

- `vertL`, `eleL`, `pcode` define a model with linear elements.
- `vertQ`, `eleQ`, `pcode` define a model with quadratic elements.

When the element is triangular, the meanings of variables in the `mat` file are listed as follows. When the element is quadrilateral, the meanings are the same. Just the number of columns in `eleL` and `eleQ` will increase.

- `vertL`: Mesh nodes (for linear element). It's a Nn-by-2 matrix, where Nn is the number of nodes in the mesh. Each row of `vertL` contains the x, y coordinates for that mesh node.
- `eleL`: Mesh elements (for linear element). For triangular elements, it's a Ne-by-3 matrix, where Ne is the number of elements in the mesh. Each row in `eleL` contains the indices of the nodes for that mesh element.
- `pcode`: Label of phase. Ne-by-1 array, where Ne is the number of elements

  ```
  pcode(j,1) = k; means the j-th element belongs to the k-th phase.
  ```

- `vertQ`: Mesh nodes (for quadratic element). It's a Nn-by-2 matrix.
- `eleQ`: Mesh elements (for quadratic element). For triangular elements, it's a Ne-by-6 matrix.
- `meshL`, `meshQ`: MATLAB FEMesh objects (with linear or quadratic elements).[7]

After loading the `mat` file into MATLAB, you can use the following function to plot the mesh. It works for linear and quadratic element. Syntax: `plotMeshes(vertL,eleL,pcode);` or `plotMeshes(vertQ,eleQ,pcode);`

[7] https://www.mathworks.com/help/pde/ug/pde.femesh.html

```
function plotMeshes( vert, ele, pcode )
% plotMeshes: plot mesh
    % check input --------------------------------
    ele_wid = size(ele,2);
    if ele_wid == 3         % linear triangle
        range_vec = 1:3;
    elseif ele_wid == 6     % quadratic triangle
        range_vec = [1 4 2 5 3 6];
    elseif ele_wid == 4     % linear quadrilateral
        range_vec = 1:4;
    elseif ele_wid == 8     % quadratic quadrilateral
        range_vec = [1 5 2 6 3 7 4 8];
    else
        error("ele - wrong size")
    end

    % plot mesh  ---------------------------------
    figure; hold on; axis image off;
    tvalue = unique( pcode );
    num_phase = length( tvalue );

    % setup color
    if num_phase == 1
        col = 0.98;
    elseif num_phase > 1
        col = 0.3: 0.68/(num_phase-1): 0.98;
    else
        error("num_phase < 1")
    end

    % use function patch to plot
    for i = 1: num_phase
        current_phase = tvalue(i);
        patch( 'faces',ele( pcode==current_phase, range_vec ), ...
            'vertices',vert, ...
            'facecolor',[ col(i), col(i), col(i) ], ...
            'edgecolor',[.1,.1,.1] ...
            );
    end
    hold off
end
```

If you're familiar with MATLAB Partial Differential Equation Toolbox, you can use MATLAB FEMesh object to solve PDE in MATLAB.[8]

[8] https://www.mathworks.com/help/pde/index.html

`XLS` file

In the exported `XLS` file, there are 6 sheets. Sheet1 is empty. The other sheets are `vertL`, `eleL`, `pcode`, `vertQ`, `eleQ`. Their meanings have been explained above.

`INP` and `BDF` file

Before exporting `inp` file and `bdf` file, you need to setup some parameters, such as *dx*, *dy*, *Precision of Node Coordinates*, and *Element Type*.

dx and *dy* are the scales of a pixel. *dx* is the column direction, and *dy* is the row direction. In most cases, *dx* and *dy* are equal. The value of *dx* and *dy* depends on the scale of your image and the unit in your finite element simulation. For example, the scale of your image is 0.11 mm/pixel, you can set $dx = 0.11$ and $dy = 0.11$.

Precision of Node Coordinates is the number of digits to the right of the decimal point when writing node coordinates to `inp` file and `bdf` file

When exporting `inp` file, you need to specify *Element Type*. You can check the Abaqus manual to find out which element type is suitable for your simulation. Im2mesh will automatically export two `inp` files: one for a model with linear elements and the other for a model with quadratic elements. Typically, models with quadratic elements are better. If you encounter issues when doing simulation with quadratic elements, please contact me. It may be a bug. If you use pixelMesh as mesh generator, please set the Element Type to quadrilateral elements in Abaqus; otherwise, error will occur when importing the `inp` file into Abaqus.

The exported `inp` file would have a model with one part, which contains multiple sections. Each section corresponds to one phase in your image. After importing the `inp` file into Abaqus, you can view different sections (please check Figure 11).

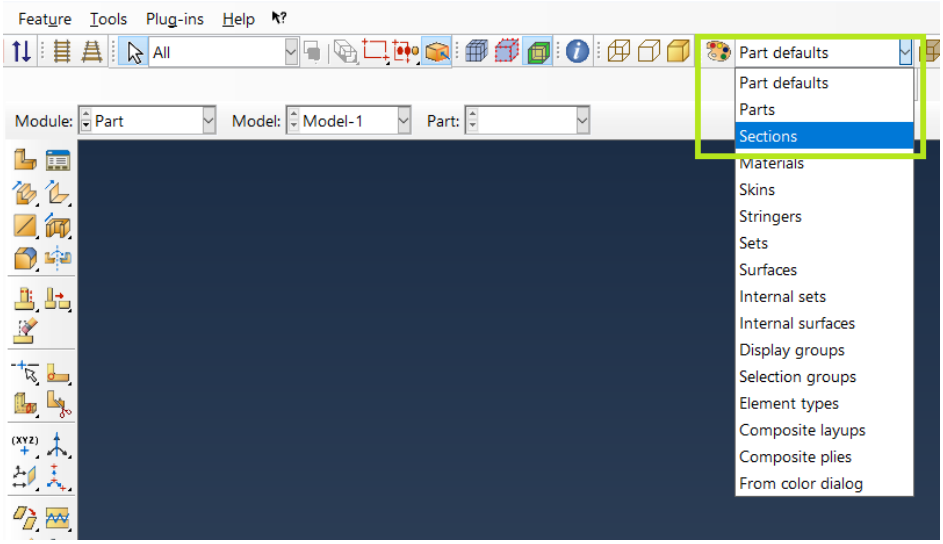


Figure 11: Abaqus

It's possible to export the mesh as a model with multiple parts, where each part corresponds to one phase in the image. However, that function (written by me in 2018) maybe outdated so I didn't incorporate it into Im2mesh_GUI.

How to import `BDF` file into COMSOL

Please refer to the following figures about how to import `bdf` file into COMSOL and assign material properties to different phases.

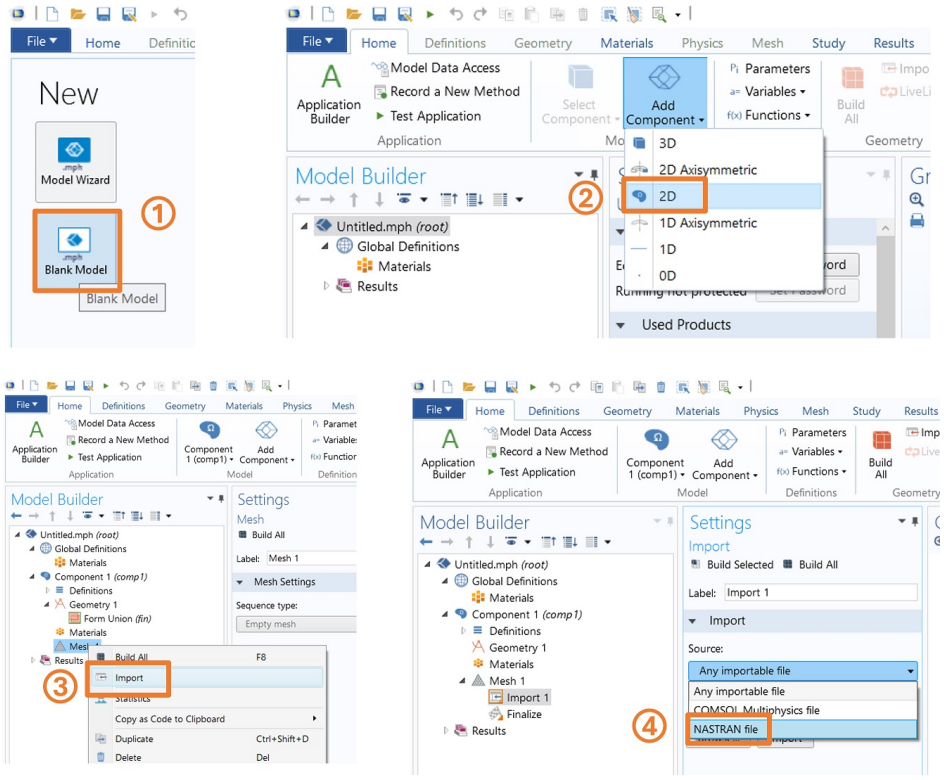


Figure 12: Import `bdf` file into COMSOL

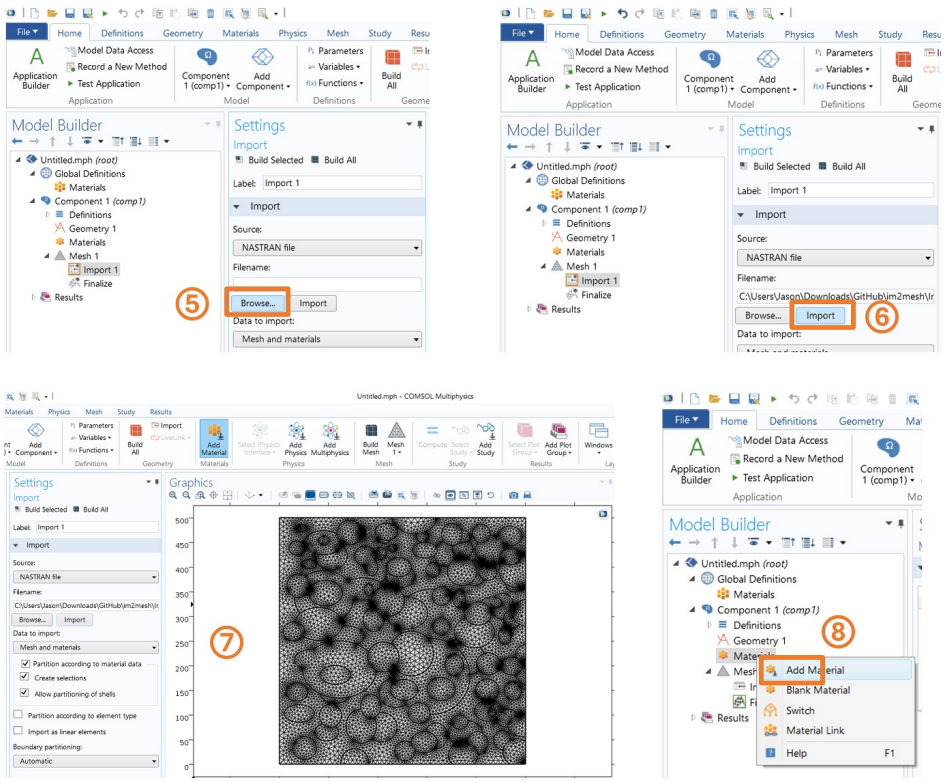


Figure 13: Import `bdf` file into COMSOL

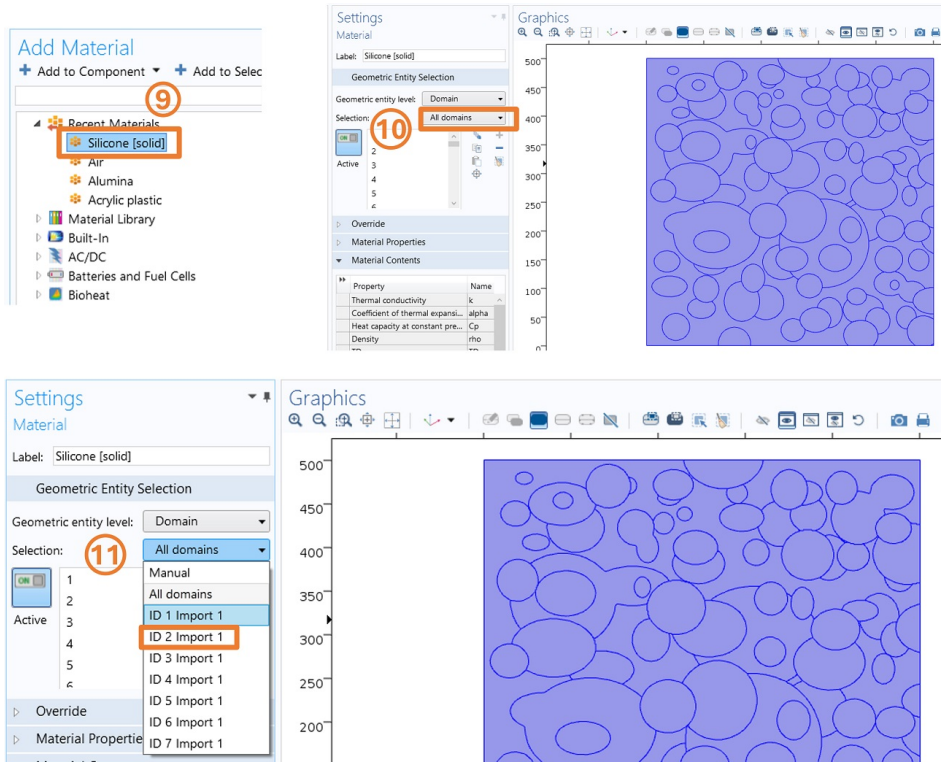


Figure 14: Import `bdf` file into COMSOL

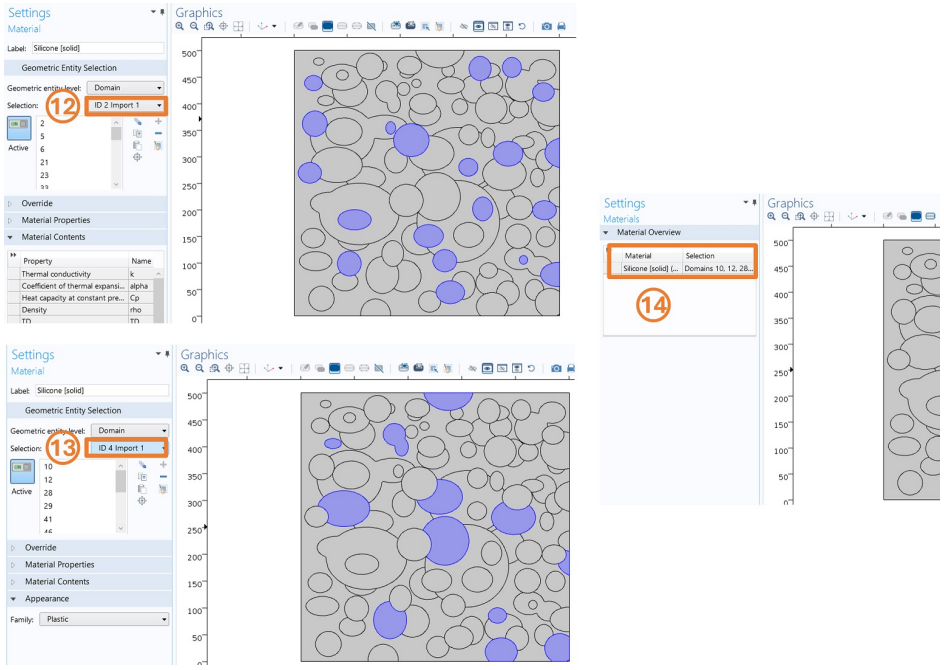


Figure 15: Import `bdf` file into COMSOL