Im2mesh_GUI Tutorial

Jiexian Ma*

This is a tutorial for Im2mesh_GUI v2.42, 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) multi-phase image. Through this tutorial, you will understand the workflow and the parameters of Im2mesh_GUI.

Contents

*mjx0799@gmail.com

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

1 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 (R2017b or later; version higher than R2018b is preferred), 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. If you find the window of Im2mesh_GUI cannot display completely, you can change the display scaling setting in your operating system to fix that.

For standalone desktop application, you don't need to install MATLAB or any MATLAB toolbox. When you open `Installer_Im2mesh_GUI.exe`, the installer will automatically download and install MATLAB Runtime (version 9.11). After installation, you can find the application in the following file directory: `C:\Program Files \Im2mesh_GUI\application\Im2mesh_GUI.exe`. **When launching the application, please "Run as administrator".** The working folder of standalone desktop app is the installation folder. You can download `Installer_Im2mesh_GUI.exe` from the link below.

`https://mjx888.github.io/others/Installer_Im2mesh_GUI.zip`

2 Workflow

Figure 1 shows the workflow of Im2mesh. The workflow is straightforward. The GUI window has been 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 do not work for your image. If you plan to use pixelMesh as mesh generator, the workflow would be much simpler: Step 1-3 and 7-9.

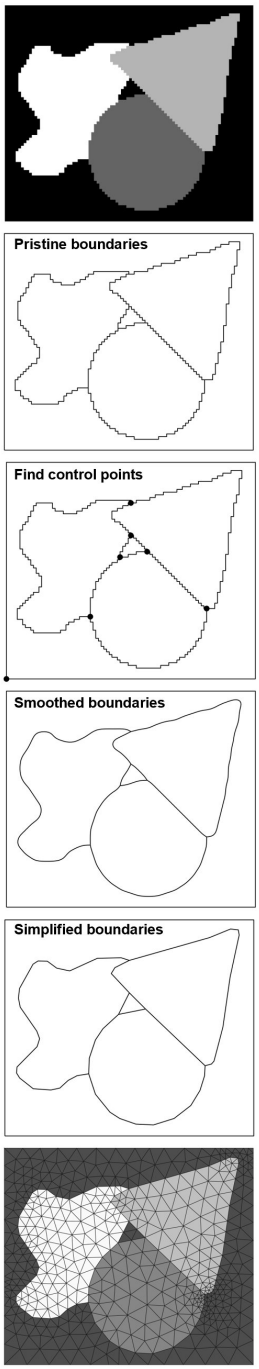
3 Step-by-step Guide

Step 0. Image processing. It is 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 on how to perform image processing in MATLAB:

`https://www.mathworks.com/videos/search.html?q=Image+Processing`
`https://www.mathworks.com/help/images/noise-removal.html`
`https://www.mathworks.com/discovery/image-segmentation.html`
`https://www.mathworks.com/videos/image-segmentation-app-1504291389509.html`
`https://www.mathworks.com/help/images/image-segmentation.html`

You can also use ImageJ to perform digital image processing. There are many videos on how to use ImageJ.

ImageJ - The Basics
ImageJ - Segmentation



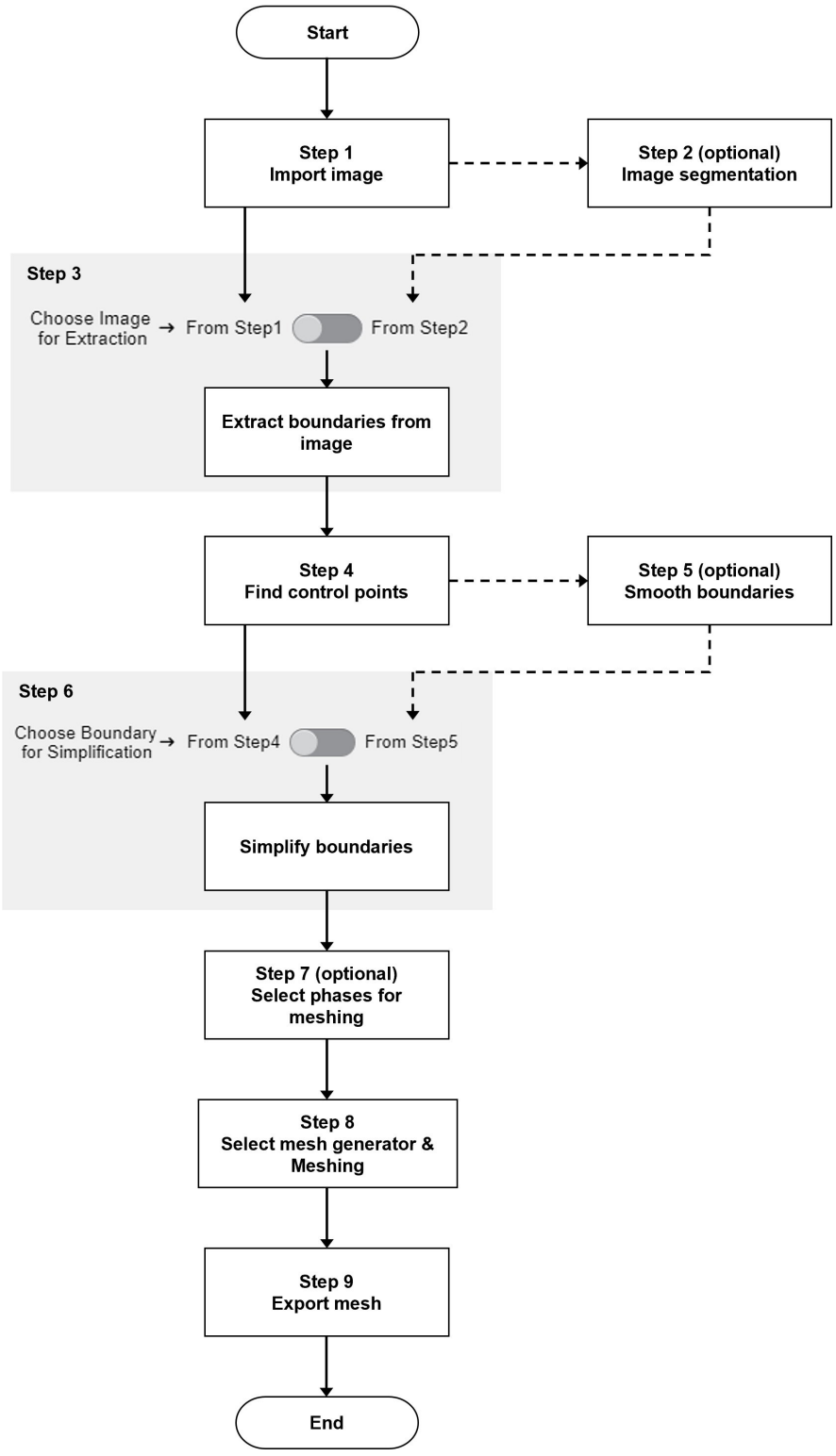


Figure 1: Workflow of Im2mesh

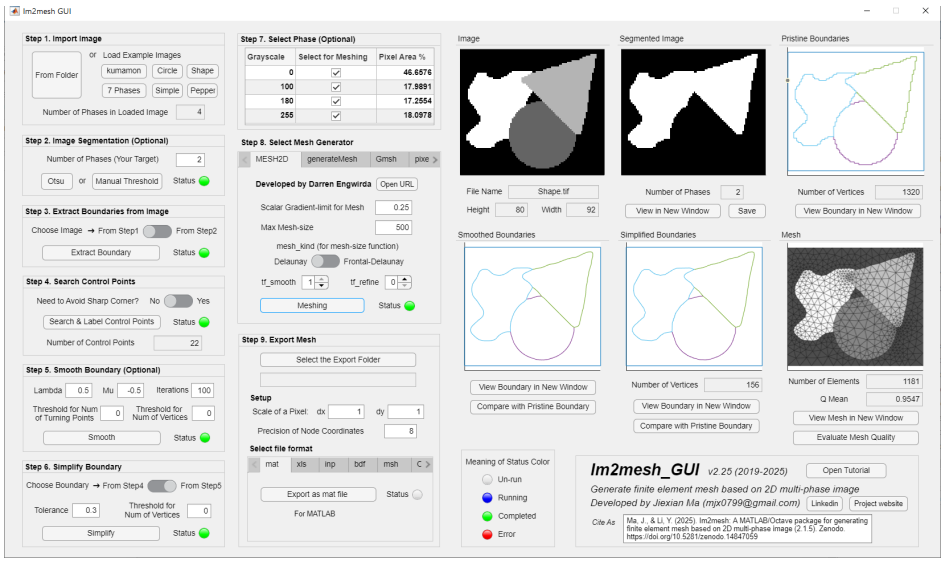


Figure 2: Im2mesh_GUI

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 is 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.

If your image is a grayscale image without segmentation, you can perform image segmentation in Step 2.

For RGB images, Im2mesh_GUI will automatically convert the imported RGB image to a 8-bit grayscale image. Therefore, some info in the RGB image will be lost. You may perform color thresholding before importing into Im2mesh.

`https://www.mathworks.com/videos/search.html?q=Color%20Thresholding`

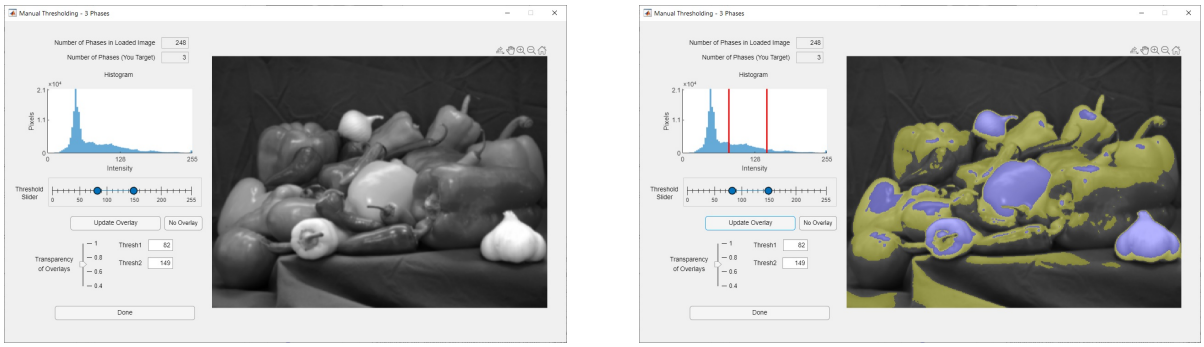
Step 2. Image segmentation. This step is optional. This step is included only for convenience. If you do not need segmentation, just skip this step.

Before pressing button "Otsu" or "Manual Threshold", you need to enter *Number of Phases (Your Target)*. The value of *Number of Phases (Your Target)* has to be smaller than *Number of Phases in Loaded Image*.

- For Otsu's method, *Number of Phases (Your Target)* can be 2-20.
- For manual multi-thresholding, *Number of Phases (Your Target)* can be 2-4 or higher.

Im2mesh_GUI currently supports two image segmentation methods: Otsu's method and manual multi-thresholding, which are the simplest multi-thresholding methods. The GUI of manual thresholding is developed by me. **Note that manual thresholding does not work in MATLAB versions before R2021a.**

Ater pressing button "Manual Threshold", you should be able to see the interface as shown in Figure 3a. Then, you can press button "Update Overlay" to display the overlaid labels (Figure 3b). The initial threshold values are computed based on Otsu's method.



(a) Initialize (b) With overlay

Figure 3: Manual multi-thresholding

In this example (Figure 3), Thresh1 is 82 and Thresh2 is 149. Therefore, grayscale from 0 to 82 is considered as Phase 1, grayscale from 82 to 149 is considered as Phase 2, and grayscale from 149 to 255 is considered as Phase 3. Note that the background (grayscale from 0 to 82) is considered as a phase in the image.

You can drag the threshold slider to manually change threshold values. You can press button "No overlay" and "Update Overlay" alternately to check the effect of thresholding. In addition to changing threshold values by the threshold slider, you can also enter threshold values directly and press "Update Overlay". To change the transparency of overlay, you can drag the transparency slider and then press "Update Overlay".

After you get satisfied results, press button "Done" to return to the main app. After Step 2, you should be able to see the segmented image in the main app. You can save the segmented image to PC by pressing buttom "Save". The segmented image will be save as "im_segmented.tif" in the current working folder of MATLAB.

The implementation of the multi-slider is inspired by the code of Marek Svoboda.[2]

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.[3]

Step 4. Search & label control points. Here, control point is defined as the intersection or meeting point between edges in the images (Figure 4). 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.

I include an option "Need to Avoid Sharp Corners?" in this step (Figure 5). This function has been improved in Im2mesh v 2.30. I may further improve it in the future. 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

[2]Marek Svoboda's answer

[3]https://www.mathworks.com/matlabcentral/fileexchange/72436

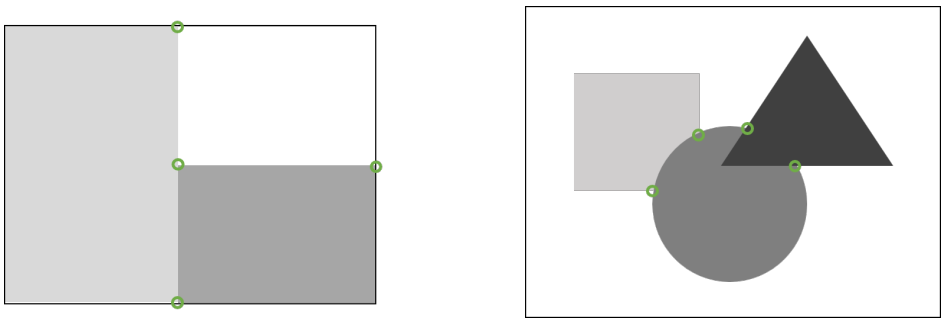


Figure 4: Examples of control points, which are marked with small circles.

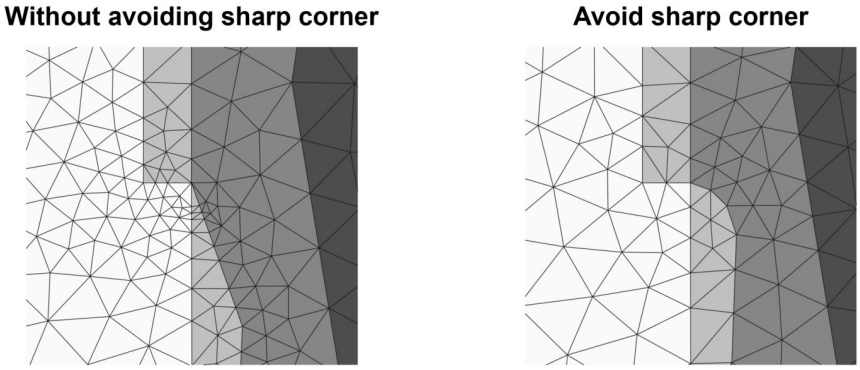


Figure 5: Need to avoid sharp corners?

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.[4]

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.

- *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.

[4]CSE554 lecture slides

• *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, or an array with two elements.

Lambda, *Mu*, and *Iterations* are parameters for Taubin smoothing. The default value is *Lambda* = 0.5, *Mu* = -0.5, *Iterations* = 100. You can use *Lambda* = 0.5 and *Mu* = -0.5 in most cases. **For *Iterations*, you can increase *Iterations* if you want smoother boundary; 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 6). Here, a turning point is defined as a vertex where the plane angle between two connected edges is not 180 degrees (Figure 7). 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 8). Please refer to section 4 of this tutorial to learn about advanced settings for threshold.

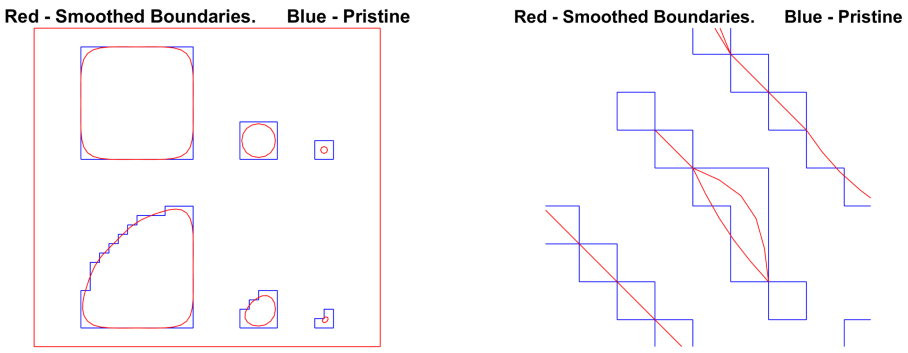


Figure 6: Examples of over-smoothing

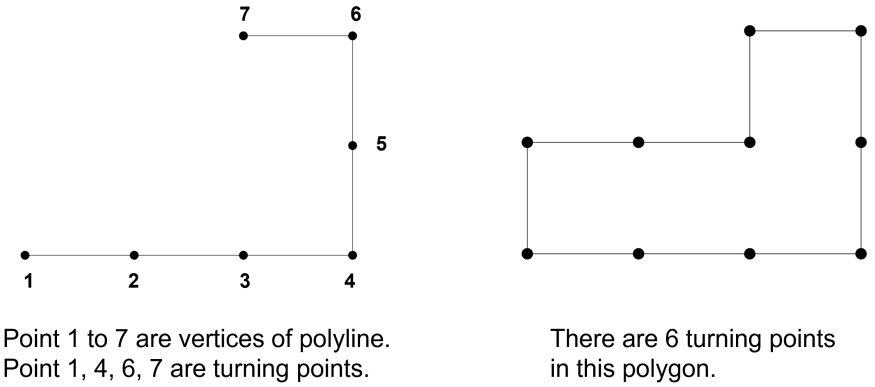


Figure 7: Examples of turning point

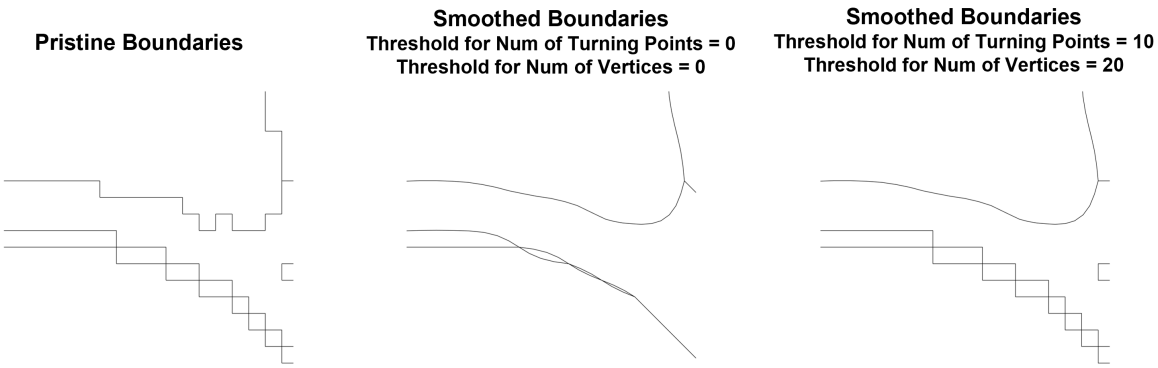


Figure 8: Set thresholds to avoid over-smoothing of simple polylines or polygons

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, or an array with two elements.

If you don't need to simplify polylines, set *Tolerance* to 0 or a small value, such as 1e-10. Note that in some MATLAB version, setting *Tolerance* to 0 may cause error.

Higher *Tolerance* means fewer points are retained in polyline and worse approximation to the pristine boundary. 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 9 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. Please refer to section 4 of this tutorial to learn about advanced settings for threshold.

Once you got the simplified boundaries. You can press button "View Polyline", "View Polyshape", 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 press the button "Simplify" again to obtained new results.

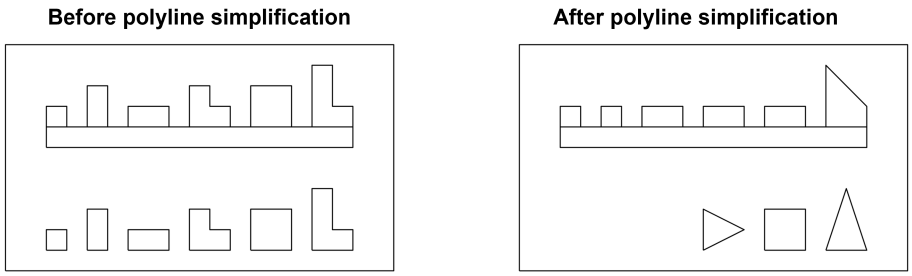


Figure 9: Example of over-simplification

You can setup how the polylines and vertices are displayed when "View Polyline". For example, set "LineSpec" to .-k and set "MarkerSize" to 20, and then press "View Polyline".

You can save the simplified boundary as different data formats.

When pressing button "Polyline", a nested cell array of polylines will be saved as "Polyline.mat" in the current working folder of MATLAB. You can use function plotBounds in Im2mesh package to plot the variable in "Polyline.mat".

When pressing button "Polyshape", polyshapes will be saved as "Polyshape.mat" in the current working folder of MATLAB. You can use the following code to plot polyshapes.

```
figure
hold on; axis equal;
for i = 1: length(psCell)
    plot( psCell{i} );
end
hold off
```

When pressing button "PSLG", a planar straight-line graph (PSLG) will be saved as "PSLG.mat" in the current working folder of MATLAB. There are 3 variables in that file:

```
node - V-by-2 array. x,y coordinates of vertices. Each row is one vertex.
edge - E-by-2 array. Node numbering of two connecting vertices of edges.
       Each row is one edge.
part - cell array. Used to record phase info. part{i} is edge indexes of
       the i-th phase, indicating which edges make up the boundary of
       the i-th phase.
```

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 four mesh generators available in Im2mesh: MESH2D, generateMesh, Gmsh, and pixelMesh.

Each mesh generator has its own parameters or settings. Note that the units of the size parameters I listed below are in pixels. For example, *Max Mesh-size* = 500 means 500 pixels.

MESH2D is an open source mesh generator developed by Dr. Darren Engwirda.[5] It's the default mesh generator of Im2mesh_GUI. It can generate high-quality finite element mesh. You are not required to install MATLAB PDE Toolbox when using MESH2D. If you have questions about the algorithm, please refer to the MESH2D program. If you use MESH2D as mesh generator, you can cite the dissertation of Darren Engwirda.[6] In Im2mesh_GUI, there are 8 parameters for MESH2D. The meanings of them are listed below for your reference.

- *Gradient-limit*: A limit on the gradient of mesh-size function (Figure 10).[7] Type: Float. Range: > 0. Typical value: 0.15 - 0.5.

- *Max Mesh-size*: Maximum mesh edge lengths. This is an approximate upper bound on the mesh edge lengths (Figure 11). Type: Float. Range: > 0.

- *Size at Boundary*: Element size at constraint edges (i.e., polygonal boundary). This parameter is introduced by me to refine mesh near polygonal boundary (Figure 12), which maybe useful in some cases. If you don't need to refine mesh near boundary, you can set *Size at Boundary* to 0 or a value < 0.05 or ≥ 500.

- *mesh_kind*: 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.

- *tf_smooth*: Boolean. Value: 0 or 1. Whether to improve triangulation quality by adjusting the vertex positions and mesh topology (hill-climbing type optimisation). Default value: 1.

- *num_split*: Number of splitting for refining mesh. Each triangle is split into four new sub-triangles. Default value: 0.

- *tf_disp*: Boolean. Value: 0 or 1. Whether to display mesh generation process in MATLAB command window. Set as 0 to silence verbosity. Default value: 1.

- *hinitial*: Initial mesh size when creating mesh-size function. You can ignore this parameter when using GUI.

In some cases with complex input geometry, MESH2D could take a long time to generate mesh. You may use Gmsh instead. It is possible to use Gmsh to generate mesh based on the local-feature-size created by MESH2D. Please refer to section 5 of this tutorial.

generateMesh is a MATLAB built-in function.[8] **It requires installation of MATLAB Partial Differential Equation Toolbox.** The advantage of generateMesh over MESH2D is the speed. Im2mesh v 2.40 has fixed the crash issue when using generateMesh.

There are 3 parameters for generateMesh: *Mesh Growth Rate*, *Max Mesh Edge Length*, and *Min Mesh Edge Length*.

[5] https://github.com/dengwirda/mesh2d

[6] http://hdl.handle.net/2123/13148

[7] http://persson.berkeley.edu/pub/persson04gradlim.pdf

[8] https://www.mathworks.com/help/pde/ug/pde.pdemodel.generatemesh.html

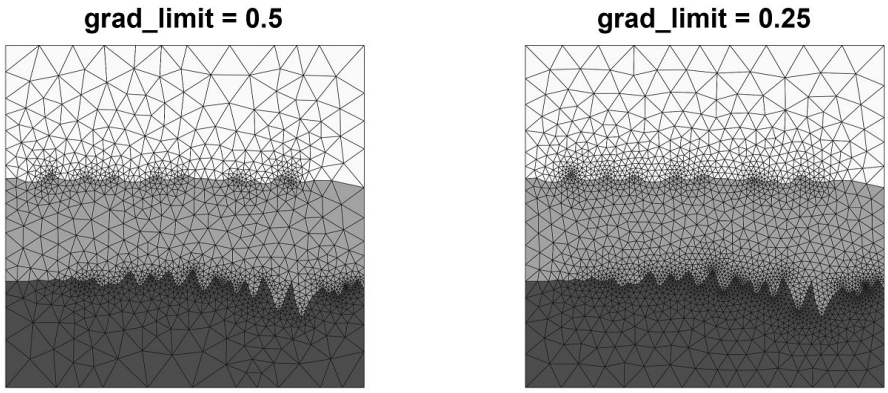


Figure 10: Effect of *Gradient-limit*

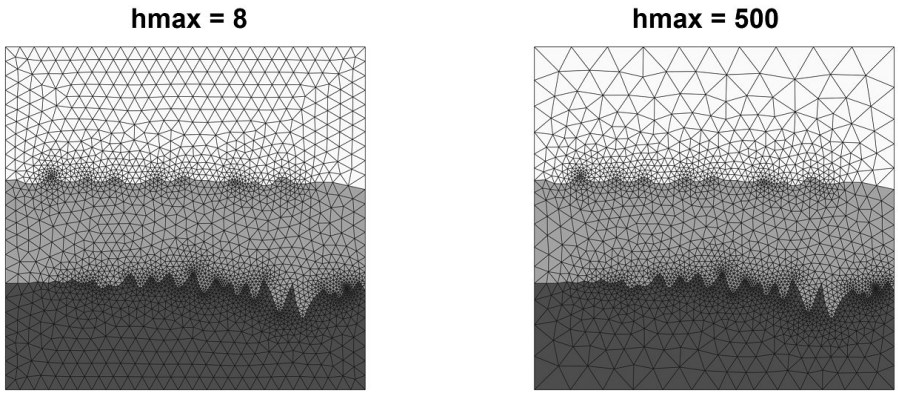


Figure 11: Effect of *Max Mesh-size*

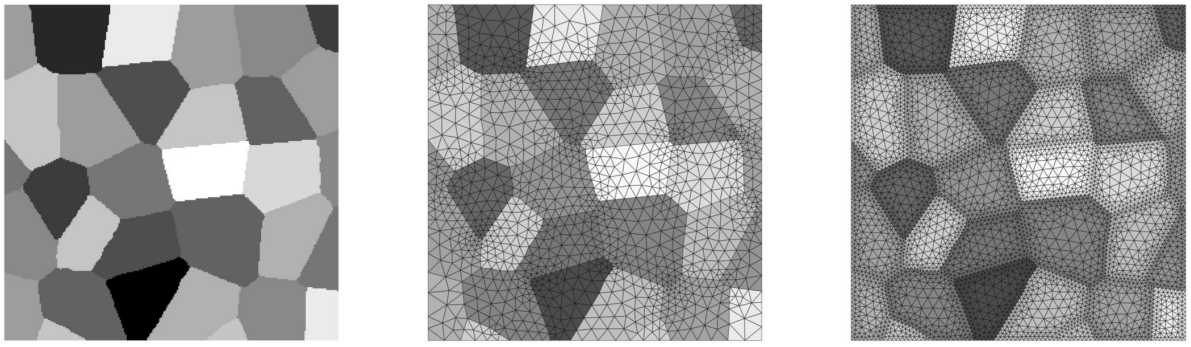


Figure 12: Refine mesh near polygonal boundary

• *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.

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

Gmsh is a powerful open-source finite element mesh generator. Please refer to section 5 of this tutorial to learn how to use Gmsh in Im2mesh.

pixelMesh is 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 5,700 vertices in the simplified boundaries, MESH2D took 50-70 seconds. For 15,000 vertices in the simplified boundaries, MESH2D took 250-350 seconds.

What to do if the mesh generation has failed for your images? Try different mesh generators or try one of the following settings.

• Set the thresholds in Step 5 or 6 to a higher value (such as 4 to 10), and re-run Step 5 to Step 8

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

Once you obtain the mesh, 3 quantities will be displayed under the mesh plot. *N tria* is the number of triangles in the mesh. *N quad* is the number of quadrangles or quadrilaterals. *Q mean* is the mean value of mesh quality.

You can press button "View Mesh" to view mesh in a new window. There are a few style settings when plotting the mesh, including *colormap*, *mode*, *wid*, *alpha*, *beta*, *tf_gs*, and *Q thresh*. You can adjust these parameters to get good figures for visualization or publication. Once you change these style parameters, you need to press button "Update" or "View Mesh" to get the updated plot.

• *colormap*. The coding of *colormap* is as follows. 0: grayscale, 1: lines, 2: parula, 3: turbo, 4: jet, 5: hot, 6: cool, 7: summer, 8: winter, 9: bone, 10: pink.

• *mode*: Display mode. When *mode* = Slow, faces and edges will be plotted. When *mode* = Fast, only edges will be plotted. If the mesh data is huge, *mode* = Fast is significantly faster.

• *wid*: Line width of the plotted edges. Positive value.

• *alpha*: Edge line transparency. A scalar value in range [0,1]. When the mesh data is huge, I usually set *alpha* to 0.2 - 0.3.

• *beta*: Brightness adjustment of colormap. Scalar value in range [-1, 1]. The colors brighten when $beta > 0$. The colors darken when $beta < 0$. The magnitude of the color change is proportional to the magnitude of *beta*.

• *tf_gs*: Whether to use graphics smoothing when plotting. For some computer hardwares, un-check *tf_gs* may plot mesh faster.

- *Q thresh*: Threshold for mesh quality. This parameter is used to display poor quality elements. Those elements with mesh quality lower than *Q thresh* will be highlighted. When the mesh data is huge, I usually set *colormap* to 0 to facilitate the visualization of poor elements.

If you plan to plot mesh via coding, please take a look at function `plotMeshes` in Im2mesh package. Function `plotMeshes` has been significantly improved in Im2mesh v2.42. It can plot mesh with the settings I listed above.

You can press button "Mesh Quality" to evaluate mesh quality. "Mesh Quality" use a subroutine (`tricost.m`) written by Darren Engwirda to evaluate mesh quality. "Mesh Quality" has 3 outputs (Figure 13). 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.

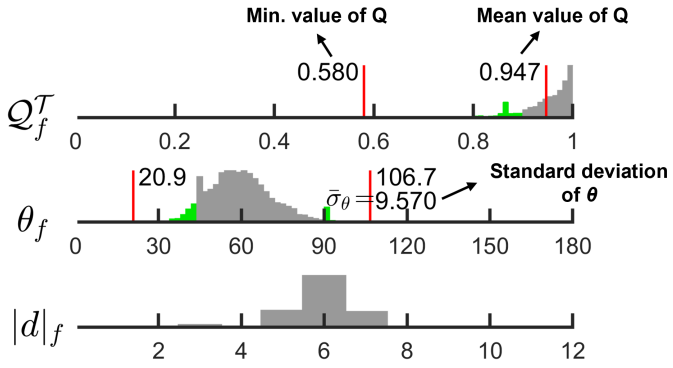


Figure 13: An example of mesh quality. A few statistics can be obtained from the plot.

If the quality of the generated mesh is not good, you can change parameters of Step 8 and press the button "Meshing" again to obtain new results. You can 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.

In some cases, fine-tuning the settings for boundary smoothing and simplification could be useful (Figure 14). For example, in stress analysis, we prefer smoother boundaries and no loss of detail between different material phases (Figure 14g,h).

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

- *Tolerance* in Step 6.

- *Gradient-limit* in Step 8 if you use MESH2D.

- *Mesh Growth Rate* and *Min Mesh Edge Length* in Step 8 if you use generateMesh.

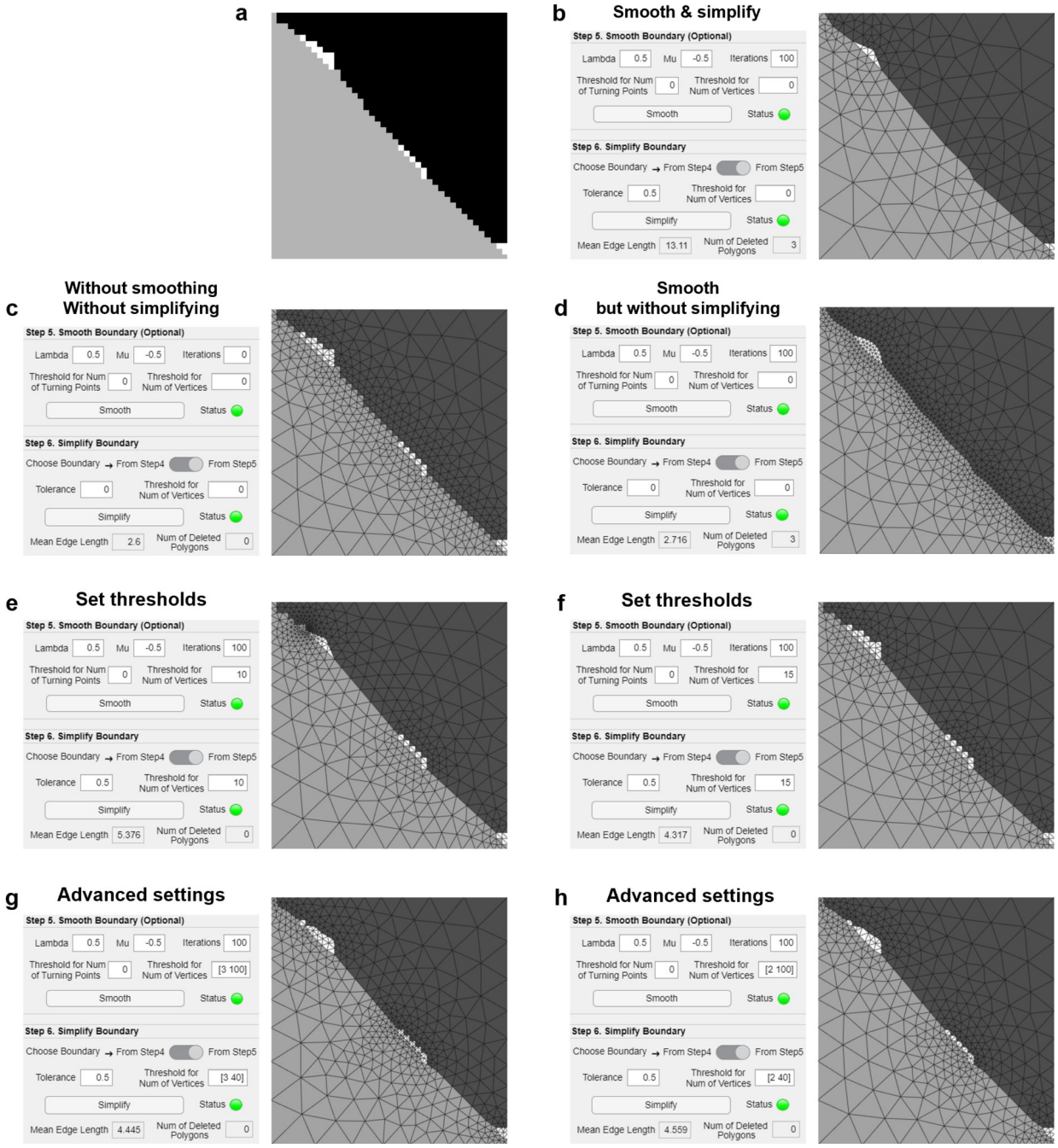


Figure 14: Effect of settings on the mesh

My suggestion for choosing the value of these critical parameters:

- First, try different *Tolerance* value in an ascending order and press 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 formats.

- `mat` file (MATLAB)
- PDE model object (MATLAB)
- `xls` file (Excel)
- `inp` file (Abaqus)
- `bdf` file (Nastran bulk data, compatible with COMSOL)
- `msh` file (Gmsh)
- Other formats (via Gmsh)

Before exporting file, you need to setup parameters: *dx* and *precision_ nodecoor*.

dx is the physical dimensions (scales) of a pixel. The value of *dx* 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 $dx = 0.11$.

precision_ nodecoor is the number of digits to the right of the decimal point when writing node coordinates to `inp` file, `bdf` file, and `msh` file

`MAT` file

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

- `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.

• `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, which corresponds to physical surface tag in Gmsh. It's a 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.

When the element is quadrilateral, the meanings are the same. Just the number of columns in `eleL` and `eleQ` will increase.

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

Function `plotMeshes` can be downloaded from: `https://github.com/mjx888/im2mesh/blob/main/Im2mesh_Matlab/plotMeshes.m`

MATLAB PDE model object

Im2mesh can export mesh as MATLAB PDE model objects (with linear or quadratic elements).[9] If you are familiar with MATLAB PDE Toolbox, you can use PDE model object to solve PDE.[10] [11]

`XLS` file

In the exported `XLS` file, there are 5 sheets: `vertL`, `eleL`, `pcode`, `vertQ`, `eleQ`. Their meanings have been explained above.

`INP` 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**

[9] https://www.mathworks.com/help/pde/ug/pde.pdemodel.html
[10] https://www.mathworks.com/help/pde/pde-problem-setup.html
[11] https://www.mathworks.com/help/pde/ug/solve-problems-using-pdemodel-objects.html

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.

Im2mesh v 2.40 will automatically export node sets at the boundary and at the interface, which can be useful when defining boundary condition. If you do not need these node sets, you can un-check them. You can also define customized node sets using function printInp2d in Im2mesh package.

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 (Figure 15).

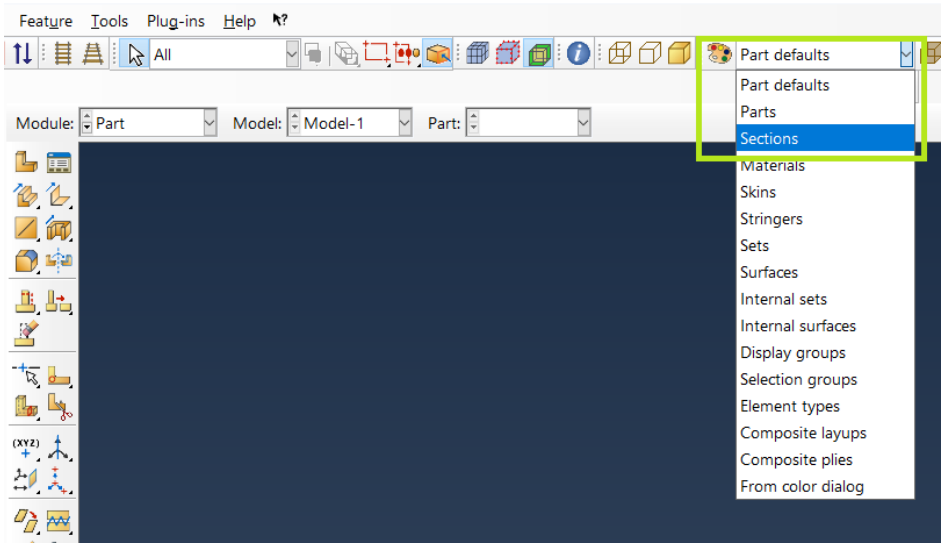


Figure 15: Abaqus

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. If you prefer multiple parts, feel free to let me know.

`msh` file

`msh` file is the Gmsh mesh file format. Here, a function (`printMsh.m`) developed by me is used to write mesh data to `msh` file.[12] Function printMsh currently only supports 2D triangular mesh with linear elements. However, function printMsh can be modified to support quadratic element. If you prefer quadratic element, feel free to send me a brief email to let me know.

MSH file format version is 4.1. I have tested the msh file generated by function printMsh in Gmsh 4.13.1. I didn't see any errors. If you prefer older MSH file format, such as version 1 or 2, you may save it as version 1 or 2 via Gmsh. Please refer to Gmsh manual about details.

[12]https://www.mathworks.com/matlabcentral/fileexchange/180415

In the generated msh file "test_linear.msh", physical surface tags are used to label different material phases. The k-th physical surface is the k-th material phase in the image.

You can open the `msh` file in Gmsh. Then, you may define boundary node set or element set in Gmsh and export mesh as other file formats.

Other formats

The generated mesh can be exported as other formats via Gmsh. Gmsh supports a lot of mesh file formats. Here, I created an interface to simplify the workflow of file format conversion.

Before starting, you need to download Gmsh (`https://gmsh.info`) and un-zip. Then, in the panel "Other formats" (within Step 9), you need to "Locate gmsh.exe". I have tested with MATLAB 2021b and gmsh 4.13.1 on Windows system. I'm not sure whether it works on Linux or macOS. On Linux or macOS, you may try to locate other executable files of gmsh.

Then, you can set *File Name Suffix*. Please refer to page 24 about the export format supported by Gmsh. For example, if you want STL surface file, set *File Name Suffix* to stl.

When you press button "Export to File", Im2mesh will generate "test_linear.msh" in the export folder and then send the msh file to Gmsh to generate a file with targeted format. You should be able to see a file named "test_linear_gmsh" created in the export folder. If your mesh has a huge number of elements and you want to export as a few different formats, I suggest you un-check "New msh File".

If "Export to File" fails, you will see error messages in the MATLAB command window.

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. You can also import the `bdf` file into COMSOL as geometry and then use COMSOL to generate mesh.

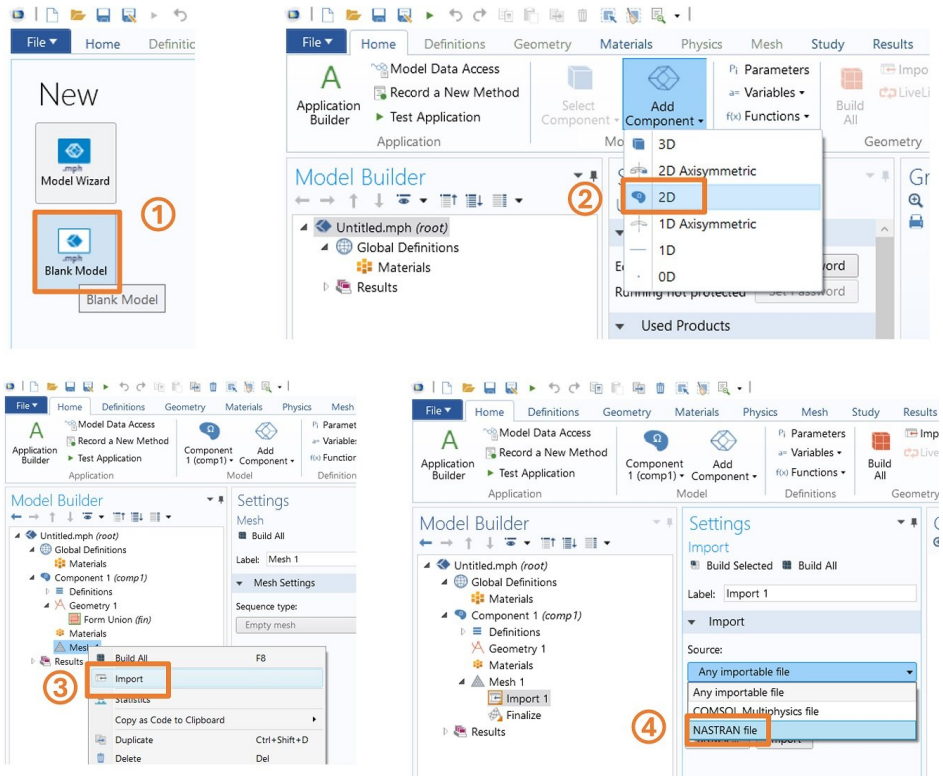


Figure 16: Import `bdf` file into COMSOL

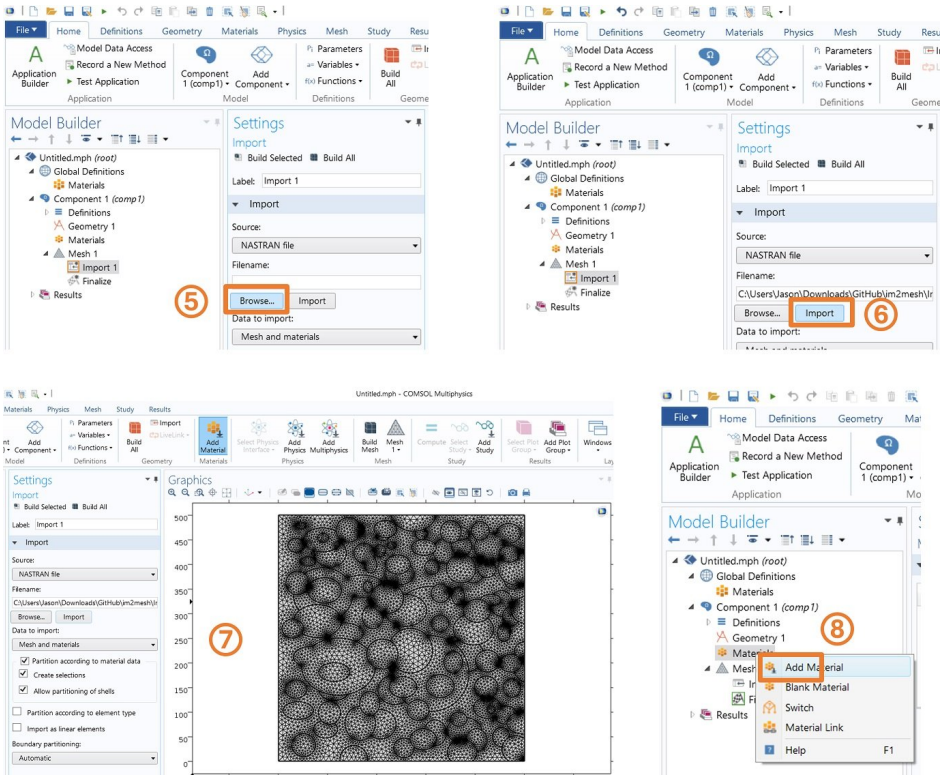


Figure 17: Import `bdf` file into COMSOL

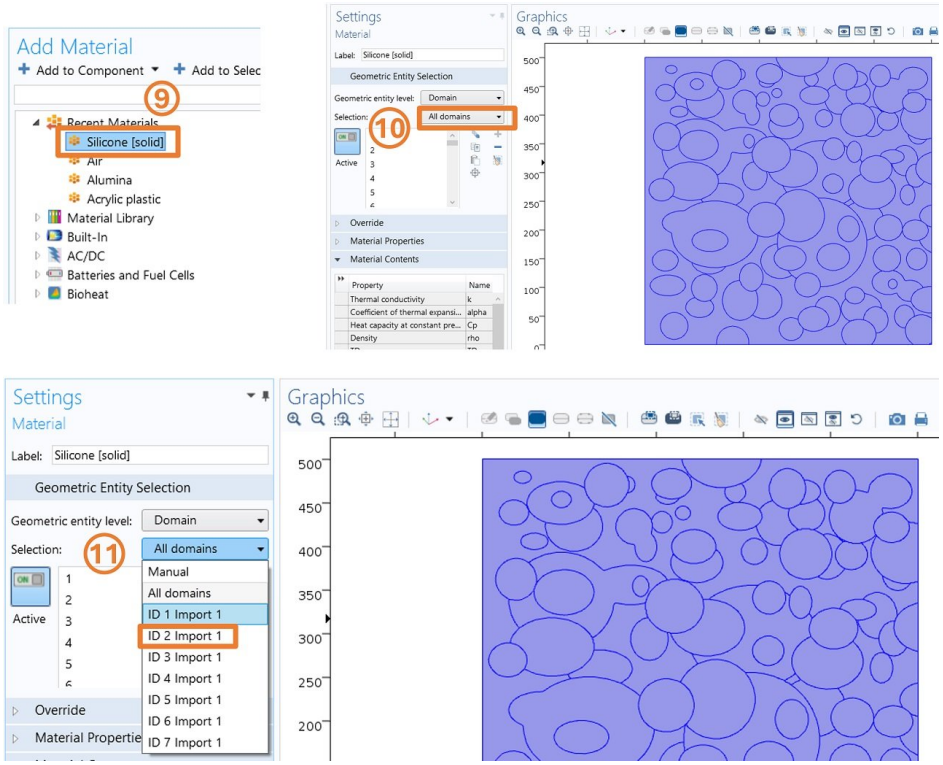


Figure 18: Import `bdf` file into COMSOL

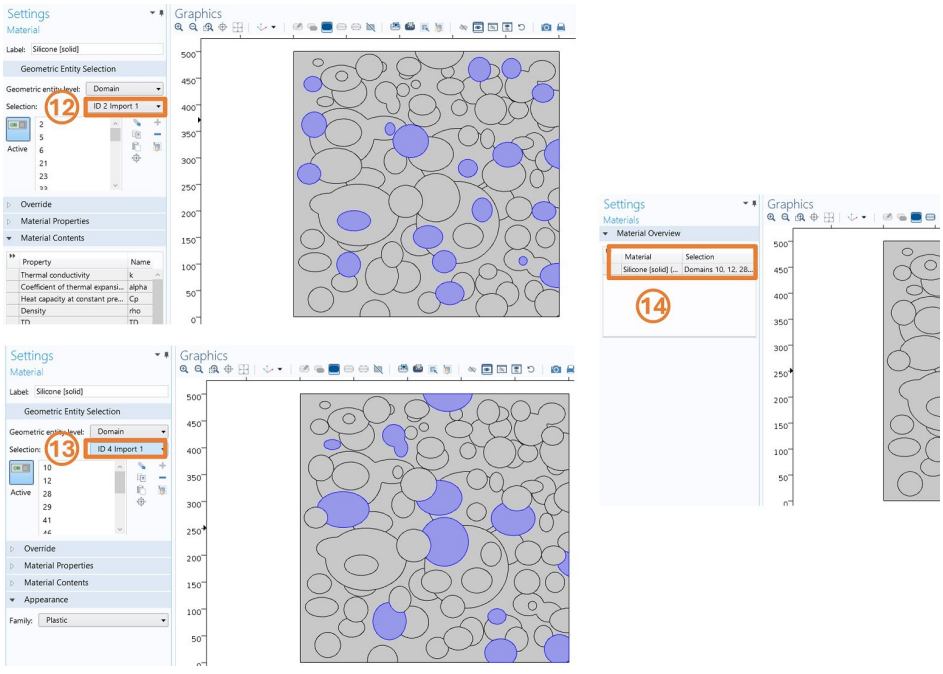


Figure 19: Import `bdf` file into COMSOL

4 Advanced Settings

In the previous section, We noticed that there exists over-smoothing and over-simplification issues. Although we can avoid these issues by setting thresholds, the resulted boundary has rough geometry (Figure 8 and Figure 14f), which could be problematic for stress analysis. In some cases, we may prefer smoother boundary to reduce stress concentration. Therefore, advanced settings for thresholds are introduced in Im2mesh vesion 2.40 to handle this situation (Figure 14g,h).

Threshold for Num of Vertices in Step 5 can be set as an integer or an array with two elements. Typical values are [2, 100], [3, 100], [4, 100].

- When the *Threshold for Num of Vertices* is set to an integer, polylines with fewer vertices than the threshold will not be smoothed (Figure 20a).

- When the *Threshold for Num of Vertices* is set to an array with two elements (e.g. [lower threshold, upper threshold]), the actual number of iterations during polyline smoothing is chosen based on a piecewise linear function (Figure 20b,c). For example, *Threshold for Num of Vertices* = [5,20] and *Iterations* = 100 means that polylines with fewer than 5 vertices will not be smoothed, those with more than 20 vertices will be smoothed using 100 iterations, and those with 5 to 20 vertices will be smoothed using an interpolated number of iterations based on their vertex count.

Threshold for Num of Vertices in Step 6 has the similar meaning (Figure 20d). Typical values are [2, 40], [3, 40], [4, 40].

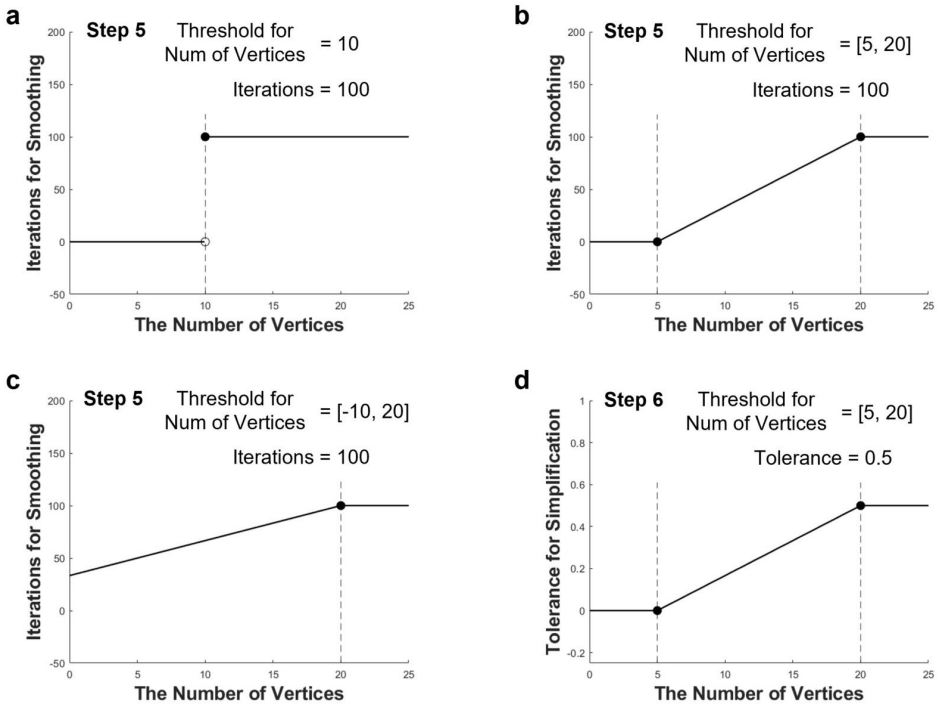


Figure 20: Advanced settings for thresholds

5 Gmsh

Before starting, you need to download Gmsh (`https://gmsh.info`) and un-zip. Then, in the interface of Im2mesh_GUI, you need to "Locate gmsh.exe". I have tested with MATLAB 2021b and gmsh 4.13.1 on Windows system. I'm not sure whether it works on Linux or macOS. On Linux or macOS, you may try to locate other executable files of gmsh. Each time you restart Im2mesh_GUI, you will have to "Locate gmsh.exe" again. To save some time, you can "Set as default". When you press "Set as default", a text file named "path_to_gmsh.txt" will be created in the install folder of Im2mesh_GUI. The location of gmsh.exe is stored in this text file.

In Im2mesh_GUI, there are a few parameters for Gmsh. Some of their meanings are listed in the manual of Gmsh 4.13.1 (`https://gmsh.info/doc/texinfo/gmsh.html`). I list them below for your reference.

- *MeshSizeMin*: Minimum mesh element size. Default value of Gmsh: 0.
- *MeshSizeMax*: Maximum mesh element size. Default value of Gmsh: 1e+22.
- *Algorithm*: 2D mesh algorithm (1: MeshAdapt, 2: Automatic, 3: Initial mesh only, 5: Delaunay, 6: Frontal-Delaunay, 7: BAMG, 8: Frontal-Delaunay for Quads, 9: Packing of Parallelograms, 11: Quasi-structured Quad). Default value of Gmsh: 6. Please refer to section 1.2.1 in Gmsh manual about mesh algorithm. You may also refer to "An introduction to Mesh Generation".[13]
- *ElementOrder*: 1 or 2.
- *RecombineAll*: Boolean. Value: 0 or 1. Whether to apply recombination algorithm to all surfaces, ignoring per-surface spec. Default value of Gmsh: 0.
- *RecombinationAlgorithm*: Mesh recombination algorithm (0: simple, 1: blossom, 2: simple full-quad, 3: blossom full-quad). Default value of Gmsh: 1.
- *ScalingFactor*: Global scaling factor applied to the saved mesh. Default value of Gmsh: 1. Because we work with images, we can set *ScalingFactor* as the physical dimensions (scales) of a pixel in the image. For example, the scale of your image is 0.11 mm per pixel, you can set *ScalingFactor* = 0.11.
- *grad_mode*: Gradient mode. Integer. Value: 0, 1, 2. This is a parameter introduced

  When *grad_mode* is 0, mesh size field is not specified.

  When *grad_mode* is 1, the element size linearly increases as the distance from constraint edges (i.e., polygonal boundary) increases. Element size = *SizeAtBound* + *SizeSlope* × distance from edges. This is inspired by the code of Dorian Depriester.[14]

  When *grad_mode* is 2, the mesh size field is created based on the local-feature-size function of MESH2D.

[13] https://perso.uclouvain.be/vincent.legat/documents/meca2170/meshGenerationBook.pdf
[14] https://www.mathworks.com/matlabcentral/fileexchange/71469-mtex2gmsh

- *SizeAtBound*: Element size at constraint edges (i.e., polygonal boundary).

- *SizeSlope*: Slope in mesh size field.

- *num_split*: Number of splitting for refining mesh. For example, each triangle is split into four new sub-triangles. Default value: 0

How to set parameters:

- To generate triangular mesh, you can set *Algorithm* to 1-7 and set *RecombineAll* to 0.

- To generate unstructured quadrilateral mesh, you can set *Algorithm* to 8 and set *RecombineAll* to 1. You can try different *RecombinationAlgorithm*.

- To generate quasi-structured quadrilateral mesh, you can set *Algorithm* to 11 and set *RecombineAll* to 0.

- When *grad_mode* is 0, you can adjust the value of *MeshSizeMax* to achieve better mesh quality. Typical value of *MeshSizeMax* is 4 to 20.

- When *grad_mode* is 1, you can adjust *SizeAtBound* and *SizeSlope* to achieve better mesh quality.

- When *grad_mode* is 2, you can set *SizeAtBound* to a larger value (e.g., 10-500) and adjust *SizeSlope*.

After setting parameters, you can press "Generate geo File" and a `geo` file (gmshTemp.geo) will be generated in your current working folder of MATLAB. `geo` file is a text file that contains Gmsh's own scripting language. It's used to define geometry and mesh generation instructions for Gmsh.

After `geo` file is generated, you can set up *verbosity* and *num_thresh*.

- *verbosity*: Level of information printed on the terminal and the message console (0: silent except for fatal errors, 1: +errors, 2: +warnings, 3: +direct, 4: +information, 5: +status, 99: +debug). Default value of Gmsh: 5.

- *num_thresh*: Maximum number of threads used by Gmsh when compiled with OpenMP support. For parallel computing during mesh generation. When *num_thresh* is 0, use system default, i.e. OMP_NUM_THREADS. Default value of Gmsh: 1.

  Usually, I set *num_thresh* as 1 or 0. On my desktop computer (GUI with *Algorithm*=6), when *grad_mode* is 0 or 2, *num_thresh*=0 is faster; when *grad_mode* is 1, *num_thresh*=1 is faster.

Then, you can press "Send geo File to Gmsh" to generate mesh. Once meshed, you will find an m file (gmshTemp.m) in your current working folder and a struct variable (msh) in your MATLAB workspace.

If you change the parameters for mesh generation, you will have to do "Generate geo File" and "Send geo File to Gmsh" again. An alternative approach is to open the `geo` file in a text editor and modify it manually, and then press "Send geo File to Gmsh".

After you got the generated mesh, you can check "Q mean". For quadrilateral element, Im2mesh_GUI use a subroutine (`MeshQualityQuads.m`) written by Allan P. Engsig-Karup to evaluate mesh quality.[15] Here, Q is defined as $prod(1 - abs((pi/2 - angles)/(pi/2)))$, where angles are the interior angles of the quadrilateral. $0 <= Q <= 1$, where $Q = 1$ for rectangles (good quality) and $Q = 0$ for quadrilaterals with triangular shape (four unique vertices assumed). Since I'm not quite familiar with the quality evaluation of quad mesh, I may add other quality parameters later. If you have any suggestions regarding quality evaluation of quad mesh, feel free to send me a brief email.

When you generate unstructured quadrilateral mesh, there may exist some triangles that are failed to recombine into quad mesh. Please be cautious. To remove those triangles from the quad mesh, you can set *MeshSizeMax* to a smaller value and mesh again. It should help.

If you're not satisfied with the mesh quality, you can open the `gmshTemp.geo` file in Gmsh and improve it. Specify a mesh size field may improve the mesh quality. You can add more mesh nodes in Gmsh. There are a lot of videos on how to use Gmsh. For example:

`https://www.youtube.com/watch?v=-wm2LzCFvQQ`

`https://www.youtube.com/watch?v=GL6GUoIwSdU`

Once you're ok with the mesh, you can export it by pressing "Export to Current Folder". You can specify export format by setting *File Name Suffix for Output*, which corresponds to Mesh.Format in Gmsh. Gmsh support a lot of output formats. I list below for your reference.

```
msh (Gmsh), inp (Abaqus), key (LSDYNA),
celum, cgns, diff (Diffpack 3D),
unv (I-deas Universal), ir3 (Iridum), med,
mesh (INRIA Medit), mail (CEA Triangulation),
m (Matlab), bdf (Nastran Bulk Data File),
off (Object File Format), p3d (Plot3D Structured Mesh),
stl (STL surface), wrl (VRML surface), vtk (VTK),
dat (Tochnog), ply2 (PLY2 Surface), su2,
neu (GAMBIT Neutral File), x3d, _0000.rad (RADIOSS BLOCK)
```

For example, if you want STL surface file, set *File Name Suffix for Output* to stl. If you want MATLAB file, set *File Name Suffix for Output* to m. Please refer to the manual of Gmsh if you have questions about the format. Once you press "Export to Current Folder", a file named "gmshExport" will be created in your MATLAB current working folder.

Note that Gmsh offers a lot of file export settings, such as exporting certain boundary node set. You can open the `gmshTemp.geo` file in Gmsh to explore these options.

If you plan to work with the matlab file (m file) generated by Gmsh, please be cautious. You may need to adjust the node order in the mesh element. Please refer to demo13 in Im2mesh package about how to do that.

[15] https://www.mathworks.com/matlabcentral/fileexchange/33108

demo13: `https://mjx888.github.io/im2mesh_demo_html/demo13.html`

6 Edit Polygonal Boundary and Refine Mesh

It is possible to edit the polygonal boundary before mesh generation. We can achieve many things by editing the boundary (Figure 21). For instance, we can add a notch or other shape to the geometry; we can add mesh seeds/nodes to the boundary. Please refer to demo14-16 in Im2mesh package about how to edit polygonal boundary before mesh generation.

Im2mesh can refine mesh locally (Figure 22). Please refer to demo17 in Im2mesh package.

If you plan to edit polygonal boundary or refine mesh, the workflow shown in Figure 23 could be helpful.

demo14: `https://mjx888.github.io/im2mesh_demo_html/demo14.html`
demo15: `https://mjx888.github.io/im2mesh_demo_html/demo15.html`
demo16: `https://mjx888.github.io/im2mesh_demo_html/demo16.html`
demo17: `https://mjx888.github.io/im2mesh_demo_html/demo17.html`

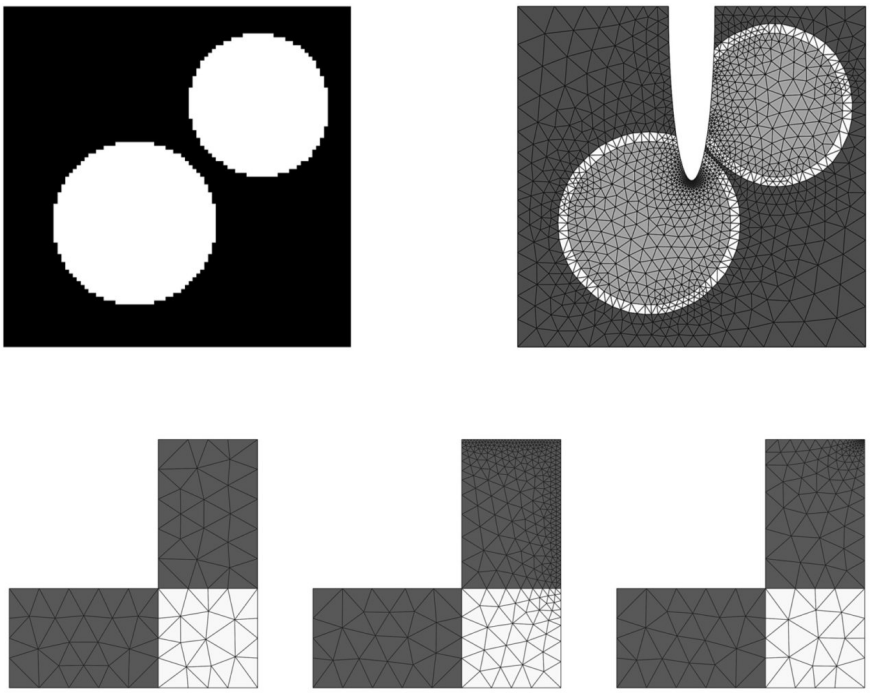


Figure 21: Edit polygonal boundary

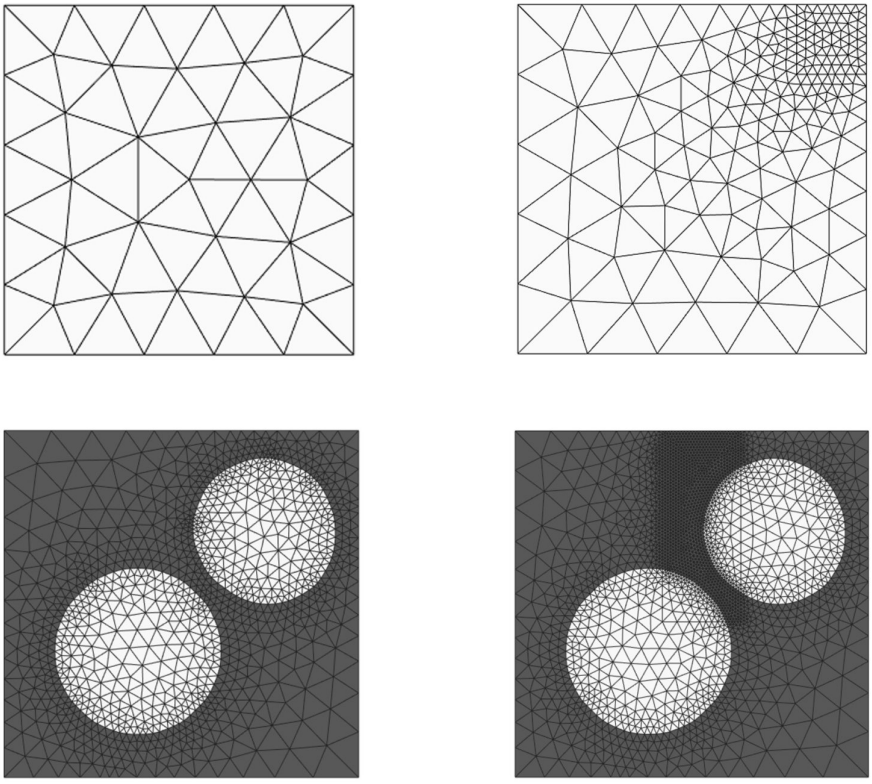


Figure 22: Refine mesh locally

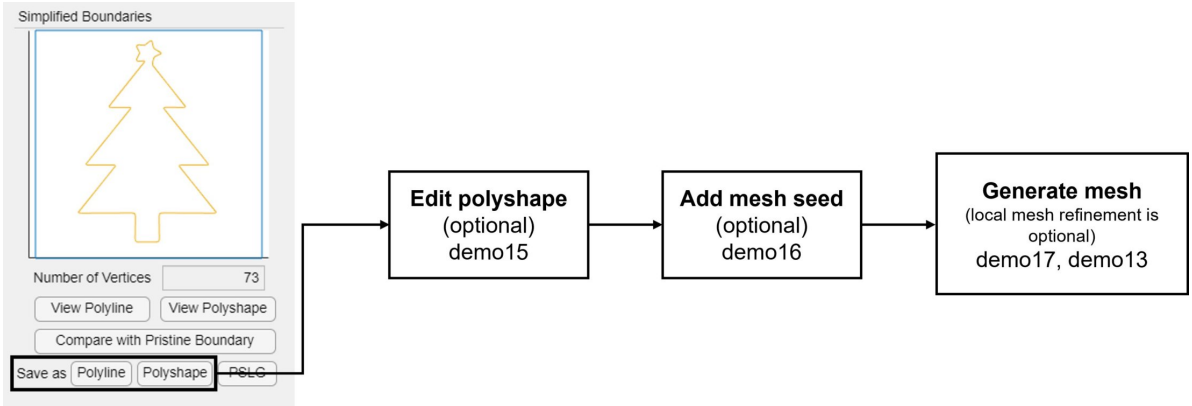


Figure 23: A workflow