

# Im2mesh Function List and Parameters

## Table of Contents

Functions.....	2
Major functions.....	2
im2mesh.....	2
im2meshBuiltIn.....	3
bounds2mesh.....	3
bounds2meshBuiltIn.....	3
bounds2geo.....	3
im2Bounds.....	3
getExactBounds.....	4
getCtrlPnts.....	4
smoothBounds.....	4
smoothBoundsCCMA.....	4
simplifyBounds.....	4
deltri1.....	4
Functions for plotting.....	5
plotMeshes.....	5
plotBounds.....	5
plotBounds2.....	5
Functions for mesh quality.....	5
tricost.....	5
MeshQualityQuads.....	5
Functions for polygonal boundary.....	5
addIntersectPnts.....	5
addPnt2Bound.....	6
insertMidPnt.....	6
insertEleSizeSeed.....	6
insertBiasedSeed.....	6
Functions for convert variable type.....	6
getPolyNodeEdge.....	6
regroup.....	6
bound2polyshape.....	6
polyshape2bound.....	7
bound2SurfaceLoop.....	7
extractMsh.....	7
Functions for exporting mesh.....	7
getNodeEle.....	7
getInterf.....	7
getBCNode.....	7
printInp2d.....	7
printBdf2d.....	8
printMsh.....	8
printTria.....	8
fixOrdering.....	9
insertNode.....	9
Other functions.....	9
xyRange.....	9

totalNumVertex.....	9
totalNumCtrlPnt.....	9
delZeroAreaPoly.....	9
getPixelPercent.....	9
getPolyShapePercent.....	10
Deprecated functions.....	10
poly2mesh (deprecated).....	10
poly2meshBuiltIn (deprecated).....	10
printGeo (deprecated).....	10
printInp_multiPart (deprecated).....	10
printInp_multiSect (deprecated).....	10
printBdf (deprecated).....	11
Parameters.....	11
Parameters and their default values of function im2mesh.....	11
Parameters and their default values of function im2meshBuiltIn.....	11
tf_avoid_sharp_corner.....	12
lambda.....	12
mu.....	12
iters.....	12
threshold_num_turning.....	12
threshold_num_vert_Smo.....	12
tolerance.....	13
threshold_num_vert_Sim.....	13
grad_limit.....	13
hmax.....	13
mesh_kind.....	13
select_phase.....	13
hgrad.....	14
hmin.....	14
tf_mesh.....	14

## Functions

### Major functions

#### im2mesh

Generate triangular mesh based on grayscale segmented image using MESH2D mesh generator (Darren Engwirda)

```
[ vert, tria, tnum ] = im2mesh( im );    % default setting
[ vert, tria, tnum ] = im2mesh( im, opt );

[ vert, tria, tnum, vert2, tria2 ] = im2mesh( im );
[ vert, tria, tnum, vert2, tria2 ] = im2mesh( im, opt );

[ vert, tria, tnum, vert2, tria2, conn, bounds ] = im2mesh( im );
[ vert, tria, tnum, vert2, tria2, conn, bounds ] = im2mesh( im, opt );
```

If we do not need to generate mesh but we want to check the simplified polygonal boundary

```
opt.tf_mesh = false;
```

```
bounds = im2mesh( im, opt );
```

### im2meshBuiltin

Generate triangular mesh based on grayscale segmented image using matlab built-in function generateMesh

```
[ vert, tria, tnum ] = im2meshBuiltin( im );    % default setting
[ vert, tria, tnum ] = im2meshBuiltin( im, opt );

[ vert, tria, tnum, vert2, tria2 ] = im2meshBuiltin( im );
[ vert, tria, tnum, vert2, tria2 ] = im2meshBuiltin( im, opt );

[ vert, tria, tnum, vert2, tria2, model1, model2 ] = im2meshBuiltin( im );
[ vert, tria, tnum, vert2, tria2, model1, model2 ] = im2meshBuiltin( im, opt );
% model1, model2 - MATLAB PDE model object
```

### bounds2mesh

Generate meshes of parts defined by polygonal boundary. Use MESH2D mesh generator (Darren Engwirda).

```
[vert,tria,tnum] = bounds2mesh( bounds, hmax, grad_limit );
[vert,tria,tnum] = bounds2mesh( bounds, hmax, grad_limit, opt );

[vert,tria,tnum,vert2,tria2] = bounds2mesh( bounds, hmax, grad_limit );
[vert,tria,tnum,vert2,tria2] = bounds2mesh( bounds, hmax, grad_limit, opt );
```

### bounds2meshBuiltin

Generate meshes of parts defined by polygonal boundary. Mesh generator: generateMesh.

```
[vert,tria,tnum] = bounds2meshBuiltin( bounds, hgrad, hmax, hmin);
[vert,tria,tnum,vert2,tria2] = bounds2meshBuiltin( bounds, hgrad, hmax, hmin);
[vert,tria,tnum,vert2,tria2,model1,model2] = bounds2meshBuiltin( bounds, hgrad, hmax, hmin);
```

### bounds2geo

Generate geo file for Gmsh based on polygonal boundary

```
bounds2geo( bounds, path_to_geo )
bounds2geo( bounds, path_to_geo, opt );
```

### im2Bounds

Extract exact polygonal boundaries from grayscale segmented image using getExactBounds.m

```
bounds = im2Bounds( im );
```

### **getExactBounds**

Get the exact boundaries (polygonal) of binary image

```
Bs = getExactBounds( bw );
```

### **getCtrlPnts**

Get control points in polygon boundaries

```
new_bounds = getCtrlPnts( bounds, tf_avoid_sharp_corner, size_im );
```

### **smoothBounds**

Smooth polygon boundaries using 2d Taubin Smoothing (taubinSmooth.m)

```
new_bounds = smoothBounds( bounds, lambda, mu, iters, ...  
                           threshold_num_turning, threshold_num_vert );  
new_bounds = smoothBounds( bounds, lambda, mu, iters, threshold_num_turning );  
new_bounds = smoothBounds( bounds, lambda, mu, iters );  
new_bounds = smoothBounds( bounds, lambda, mu, 0 );    % no smoothing
```

### **smoothBoundsCCMA**

Smooth polygon boundaries using CCMA smoothing algorithm (CCMA.m)

CCMA stand for curvature corrected moving average (<https://github.com/UniBwTAS/ccma>).

```
new_bounds = smoothBoundsCCMA( bounds, w_ma, w_cc, ...  
                              threshold_num_turning, threshold_num_vert );
```

### **simplifyBounds**

Simplify polygon boundaries using Douglas–Peucker algorithm (dpsimplify.m)

```
new_bounds = simplifyBounds( bounds, tolerance, threshold_num_vert );  
new_bounds = simplifyBounds( bounds, tolerance );
```

### **deltri1**

2d Delaunay triangulation with phase information.

```
[vert,conn,tria,tnum] = deltri1( node, edge, part );
```

## Functions for plotting

### plotMeshes

Plot triangular mesh. Also works for quadratic or quadrilateral elements.

```
plotMeshes( vert, ele, tnum ); % multiple phases
plotMeshes( vert, ele );      % one phase
```

### plotBounds

Plot polygon boundaries

```
plotBounds( bounds );
plotBounds( bounds, true ); % show starting and control points
plotBounds( bounds, false, '' ); % multi-color
plotBounds( bounds, false, 'k.-' ); % line spec
plotBounds( bounds, true, 'k.-' ); % line spec
```

### plotBounds2

Plot two polygon boundaries

```
plotBounds2( boundsA, boundsB );
```

## Functions for mesh quality

### tricast

Evaluate mesh quality of triangular mesh

```
tricast(vert,tria,tnum);
```

### MeshQualityQuads

Evaluate mesh quality of quadrilateral mesh

```
[Q,theta] = MeshQualityQuads( ele(:,1:4), vert(:,1), vert(:,2) );
```

## Functions for polygonal boundary

### addIntersectPnts

Search and add intersect points (vertex).

```
bounds = addIntersectPnts( bounds, tolerance );
```

### **addPnt2Bound**

Add points to polygonal boundaries. Check whether points are lying near polygon bounds{i}{j}. If it is, add point to polygon bounds{i}{j}.

```
bounds = addPnt2Bound( points, bounds, tolerance );
```

### **insertMidPnt**

Repeatedly inserts midpoints between vertices of a polyline.

```
xyNew = insertMidPnt( xy, iters );
```

### **insertEleSizeSeed**

insert equally spaced seeds to polyline (edges).

```
xyNew = insertEleSizeSeed( xy, targetLen );
```

### **insertBiasedSeed**

Inserts biased points between vertices of an edge

```
xyNew = insertBiasedSeed( xy, iters, ratio );
```

## **Functions for convert variable type**

### **getPolyNodeEdge**

Get nodes and edges of polygonal boundary

```
[ poly_node, poly_edge ] = getPolyNodeEdge( bounds );
```

### **regroup**

Organize cell array poly\_node, poly\_edge into array nodeU, edgeU and cell array part for MESH2D. Array nodeU, edgeU and cell array part is planar straight-line graph (PSLG).

```
[ nodeU, edgeU, part ] = regroup( poly_node, poly_edge );
```

### **bound2polyshape**

Convert a cell array of polygonal boundaries to a cell array of polyshape objects.

```
p = bound2polyshape( bounds );
```

### **polyshape2bound**

Convert a cell array of polyshape objects to a cell array of polygonal boundaries.

```
bounds = polyshape2bound( p );
```

### **bound2SurfaceLoop**

Convert a cell array of polygonal boundaries to a nesting cell array for storing multiple loops (Gmsh).

```
[ phaseLoops, vertex, edge ] = bound2SurfaceLoop( bounds );
```

### **extractMsh**

Extract nodes and elements from struct variable 'msh'. Struct variable 'msh' is generated by Gmsh.

```
[vert,ele,tnum] = extractMsh( msh );
```

## **Functions for exporting mesh**

### **getNodeEle**

Get node coordinates and elements from mesh

```
[ nodecoor_list, nodecoor_cell, ele_cell ] = getNodeEle( vert, tria, tnum );
```

### **getInterf**

Find nodes at the interface between different phases.

```
interfnod_cell = getInterf( nodecoor_cell );
```

### **getBCNode**

Find nodes at the boundary.

```
[ xmin_node_cell, xmax_node_cell, ...  
  ymin_node_cell, ymax_node_cell ] = getBCNode( nodecoor_cell );
```

### **printInp2d**

Write 2d finite element mesh (nodes and elements) to inp file (Abaqus).

The exported inp file will have a model with one part, which contains multiple sections. Each section corresponds to one material phase in the mesh.

Works for linear and quadratic element.

Works for triangular and quadrilateral element.

```
printInp2d( vert, ele );
printInp2d( vert, ele, [], [], [], file_name );
printInp2d( vert, ele, tnum );
printInp2d( vert, ele, tnum, [], precision );
printInp2d( vert, ele, tnum, ele_type, precision );
printInp2d( vert, ele, tnum, ele_type, precision, file_name );
```

### **printBdf2d**

Write 2d finite element mesh (nodes and elements) to bdf file (Nastran bulk data, compatible with COMSOL).

Works for linear triangular and linear quadrilateral element.

Not work for quadratic element.

```
printBdf2d( vert, ele );
printBdf2d( vert, ele, [], [], [], file_name );
printBdf2d( vert, ele, tnum );
printBdf2d( vert, ele, tnum, [], precision );
printBdf2d( vert, ele, tnum, [], precision, file_name );
```

### **printMsh**

Write 2d finite element mesh (nodes and elements) to msh file. msh is Gmsh mesh file format. MSH file format version: 4.1. Test in software Gmsh 4.13.1

printMsh only works for 2d trangles & linear element.

```
printMsh( vert, ele );
printMsh( vert, ele, [], [], [], file_name );
printMsh( vert, ele, tnum );
printMsh( vert, ele, [], [], precision );
printMsh( vert, ele, tnum, [], precision );
printMsh( vert, ele, tnum, [], precision );
printMsh( vert, ele, tnum, [], precision, file_name );
```

### **printTria**

Print nodes and elements into file 'test.node' and 'test.ele'. Only support triangular element with 3 nodes.

Precision is number of digits behind decimal point, for node coordinates

```
printTria( vert, tria, tnum, precision );
```



### **fixOrdering**

Fix node ordering in each elements of 2D finite element mesh. Node ordering in a element should be counterclockwise.

```
ele = fixOrdering( vert, ele );
```

### **insertNode**

Inserts midpoints into all edges to form quadratic elements. Works for triangular and quadrilateral element

```
[vertU, triaU] = insertNode(vert, tria);
```

## **Other functions**

### **xyRange**

Get the range of x y coordinate in polygonal boundary. Input argument can be a nested cell array of polygonal boundary or a cell array of polyshape.

```
[xminG,xmaxG,yminG,ymaxG] = xyRange( inarg );
```

### **totalNumVertex**

Calculate the total number of vertices in all polygonal boundaries

```
num_vert = totalNumVertex( bounds );
```

### **totalNumCtrlPnt**

Calculate the total number of control points in all polygonal boundaries. Each polygon has at least one ccontrol point (i.e., the starting vertex).

```
num_ctrlp = totalNumCtrlPnt( bounds );
```

### **delZeroAreaPoly**

Delete polygon with zero area

```
bounds = delZeroAreaPoly( bounds );
```

### **getPixelPercent**

Calculate the area percentage of each grayscale in image

```
percent_pixel = getPixelPercent( im );
```

### **getPolyShapePercent**

Calculate the area percentage of each phase in polygonal boundaries

```
percent_polyarea = getPolyShapePercent( bounds );
```

## **Deprecated functions**

### **poly2mesh (deprecated)**

Generate meshes of parts defined by polygons using MESH2D mesh generator (Darren Engwirda)

```
[vert,tria,tnum] = poly2mesh( poly_node, poly_edge, hmax, mesh_kind, grad_limit );  
  
[vert,tria,tnum,vert2,tria2] = poly2mesh( poly_node, poly_edge, hmax, mesh_kind, grad_limit );  
[vert,tria,tnum,vert2,tria2] = poly2mesh( poly_node, poly_edge, hmax, mesh_kind, grad_limit, op
```

### **poly2meshBUILTIn (deprecated)**

Generate meshes of parts defined by polygons using matlab built-in function generateMesh.

```
[vert,tria,tnum] = poly2meshBUILTIn( poly_node, poly_edge, pcell, hgrad, hmax, hmin );  
[vert,tria,tnum,vert2,tria2] = poly2meshBUILTIn( poly_node, poly_edge, ...  
                                              pcell, hgrad, hmax, hmin );
```

### **printGeo (deprecated)**

Print geo file (Gmsh input file format).

```
printGeo( C, point, line, opt, file_name );
```

### **printInp\_multiPart (deprecated)**

Print the nodes and elements into Inp file 'test\_multi\_parts.inp', test in software Abaqus. Each phase corresponds to one part in Abaqus.

```
printInp_multiPart( nodecoor_cell, ele_cell, ele_type, precision );  
printInp_multiPart( nodecoor_cell, ele_cell, ele_type, precision, file_name );
```

### **printInp\_multiSect (deprecated)**

Print the nodes and elements into Inp file 'test\_multi\_sections.inp', test in software Abaqus. One part with multiple sections. Each phase corresponds to one section in Abaqus.

```
printInp_multiSect( nodecoor_list, ele_cell, ele_type, precision );
printInp_multiSect( nodecoor_list, ele_cell, ele_type, precision, file_name );
```

### **printBdf (deprecated)**

Print the nodes and elements into Inp file 'test.bdf'

```
printBdf( nodecoor_list, ele_cell, precision );
printBdf( nodecoor_list, ele_cell, precision, file_name );
```

## **Parameters**

### **Parameters and their default values of function im2mesh**

```
opt.tf_avoid_sharp_corner = false;
opt.lambda = 0.5;
opt.mu = -0.5;
opt.iters = 100;
opt.threshold_num_turning = 0;
opt.threshold_num_vert_Smo = 0;
opt.tolerance = 0.3;
opt.threshold_num_vert_Sim = 0;
opt.select_phase = [];
opt.grad_limit = 0.25;
opt.hmax = 500;
opt.mesh_kind = 'del aunay';
opt.tf_mesh = true;
```

### **Parameters and their default values of function im2meshBuiltIn**

```
opt.tf_avoid_sharp_corner = false;
opt.lambda = 0.5;
opt.mu = -0.5;
opt.iters = 100;
opt.threshold_num_turning = 0;
opt.threshold_num_vert_Smo = 0;
opt.tolerance = 0.3;
opt.threshold_num_vert_Sim = 0;
opt.select_phase = [];
opt.hgrad = 1.25;
opt.hmax = 500;
opt.hmin = 1;
```

**tf\_avoid\_sharp\_corner**

Type: boolean.

For getCtrlPnts.

Meaning: Whether to avoid sharp corner when simplifying polygon.

**lambda**

Type: Float. Range:  $0 < \text{Lambda} < 1$ .

For smoothBounds.

Meaning: How far each node is moved toward the average position of its neighbours during every second iteration.

**mu**

Type: Float. Range:  $-1 < \text{Mu} < 0$ .

For smoothBounds.

Meaning: How far each node is moved opposite the direction of the average position of its neighbours during every second iteration.

**iters**

Type: Integer. Range:  $\geq 0$ .

For smoothBounds.

Meaning: Number of iterations in Taubin smoothing. If you don't need polyline smoothing, set Iterations to 0.

**threshold\_num\_turning**

Type: Integer. Range:  $\geq 0$ .

For smoothBounds.

Meaning: 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.

**threshold\_num\_vert\_Smo**

Type: Integer. Range:  $\geq 0$ .

For smoothBounds.

Meaning: Threshold value for the number of vertices in a polyline. Only those polylines with number of vertices greater than this threshold will be smoothed.

#### **tolerance**

Type: Float. Range:  $\geq 0$ .

For simplifyBounds.

Meaning: The maximum allowable deviation of a vertex from the simplified curve. It's for Douglas-Peucker algorithm. If you don't need to simplify polylines, set tolerance to 0 or a small value, such as  $1e-10$

#### **threshold\_num\_vert\_Sim**

Type: Integer. Range:  $\geq 0$ .

For simplifyBounds.

Meaning: Threshold value for the number of vertices in a polyline. Only those polylines with number of vertices greater than this threshold will be simplified.

#### **grad\_limit**

Type: Float. Range:  $> 0$ . Typical value: 0.2 - 0.5.

For poly2mesh & MESH2D.

Meaning: Gradient-limit, a limit on the gradient of mesh-size function.

#### **hmax**

Type: Float. Range:  $> 0$ .

For poly2mesh & MESH2D.

Meaning: Maximum mesh edge lengths. This is an approximate upper bound on the mesh edge lengths.

#### **mesh\_kind**

Value: 'delaunay' or 'delfront'

For poly2mesh & MESH2D.

Meaning: Meshing algorithm used to create mesh-size functions based on an estimate of the "local-feature-size" associated with a polygonal domain. 'delaunay' means Delaunay-refinement. 'delfront' means Frontal-Delaunay.

#### **select\_phase**

Type: vector

Meaning: Select certain phases in image for meshing. If 'select\_phase' is [], all the phases will be chosen.

'select\_phase' is an index vector for sorted grayscales (ascending order) in an image. For example, an image with grayscales of 40, 90, 200, 240, 255. If u're interested in 40, 200, and 240, then set 'select\_phase' as [1 3 4]. Those phases corresponding to grayscales of 40, 200, and 240 will be chosen to perform meshing.

### **hgrad**

Type: Float. Range:  $1 \leq \text{Mesh Growth Rate} \leq 2$ . Typical value: 1.2 - 1.5.

For poly2meshBuiltIn & generateMesh

Meaning: Mesh growth rate, the rate at which the mesh transitions between regions of different edge size.

### **hmin**

Type: Float. Range:  $\geq 0$ .

For poly2meshBuiltIn & generateMesh

Meaning: Min mesh edge length, an approximate lower bound on the mesh edge lengths.

### **tf\_mesh**

Boolean.

Meaning: Whether to mesh. If true, meshing. Else, no meshing and return boundary