# NAG Toolbox

# nag_mesh_2d_join (d06db)

## 1 Purpose

nag_mesh_2d_join (d06db) joins together (restitches) two adjacent, or overlapping, meshes.

## 2 Syntax

```
[nv3, nelt3, nedge3, coor3, edge3, conn3, reft3, ifail] = nag_mesh_2d_join
(coor1, edge1, conn1, reft1, coor2, edge2, conn2, reft2, itrace, 'eps', eps,
'nv1', nv1, 'nelt1', nelt1, 'nedge1', nedge1, 'nv2', nv2, 'nelt2', nelt2,
'nedge2', nedge2)
[nv3, nelt3, nedge3, coor3, edge3, conn3, reft3, ifail] = d06db(coor1, edge1,
conn1, reft1, coor2, edge2, conn2, reft2, itrace, 'eps', eps, 'nv1', nv1,
'nelt1', nelt1, 'nedge1', nedge1, 'nv2', nv2, 'nelt2', nelt2, 'nedge2', nedge2)
```

## 3 Description

nag_mesh_2d_join (d06db) joins together two adjacent, or overlapping, meshes. If the two meshes are adjacent then vertices belonging to the part of the boundary forming the common interface should coincide. If the two meshes overlap then vertices and triangles in the overlapping zone should coincide too.

This function is partly derived from material in the MODULEF package from INRIA (Institut National de Recherche en Informatique et Automatique).

## 4 References

None.

## 5 Parameters

### 5.1 Compulsory Input Parameters

1: **coor1**$(\mathbf{2}, \mathbf{nv1})$ – REAL (KIND=nag_wp) array

**coor1**$(1, i)$ contains the $x$ coordinate of the $i$th vertex of the first input mesh, for $i = 1, 2, \ldots,$ **nv1**; while **coor1**$(2, i)$ contains the corresponding $y$ coordinate.

2: **edge1**$(\mathbf{3}, \mathbf{nedge1})$ – INTEGER array

The specification of the boundary edges of the first input mesh. **edge1**$(1, j)$ and **edge1**$(2, j)$ contain the vertex numbers of the two end points of the $j$th boundary edge. **edge1**$(3, j)$ is a user-supplied tag for the $j$th boundary edge.

*Constraint*: $1 \leq$ **edge1**$(i, j) \leq$ **nv1** and **edge1**$(1, j) \neq$ **edge1**$(2, j)$, for $i = 1, 2$ and $j = 1, 2, \ldots,$ **nedge1**.

3: **conn1**$(\mathbf{3}, \mathbf{nelt1})$ – INTEGER array

The connectivity between triangles and vertices of the first input mesh. For each triangle $j$, **conn1**$(i, j)$ gives the indices of its three vertices (in anticlockwise order), for $i = 1, 2, 3$ and $j = 1, 2, \ldots,$ **nelt1**.

*Constraints*:

$1 \leq \mathbf{conn1}(i,j) \leq \mathbf{nv1}$;
$\mathbf{conn1}(1,j) \neq \mathbf{conn1}(2,j)$;
$\mathbf{conn1}(1,j) \neq \mathbf{conn1}(3,j)$ and $\mathbf{conn1}(2,j) \neq \mathbf{conn1}(3,j)$, for $i = 1,2,3$ and $j = 1,2,\ldots,\mathbf{nelt1}$.

4: **reft1**(**nelt1**) – INTEGER array

**reft1**$(k)$ contains the user-supplied tag of the $k$th triangle from the first input mesh, for $k = 1,2,\ldots,\mathbf{nelt1}$.

5: **coor2**(**2**, **nv2**) – REAL (KIND=nag_wp) array

**coor2**$(1,i)$ contains the $x$ coordinate of the $i$th vertex of the second input mesh, for $i = 1,2,\ldots,\mathbf{nv2}$; while **coor2**$(2,i)$ contains the corresponding $y$ coordinate.

6: **edge2**(**3**, **nedge2**) – INTEGER array

The specification of the boundary edges of the second input mesh. **edge2**$(1,j)$ and **edge2**$(2,j)$ contain the vertex numbers of the two end points of the $j$th boundary edge. **edge2**$(3,j)$ is a user-supplied tag for the $j$th boundary edge.

*Constraint*: $1 \leq \mathbf{edge2}(i,j) \leq \mathbf{nv2}$ and $\mathbf{edge2}(1,j) \neq \mathbf{edge2}(2,j)$, for $i = 1,2$ and $j = 1,2,\ldots,\mathbf{nedge2}$.

7: **conn2**(**3**, **nelt2**) – INTEGER array

The connectivity between triangles and vertices of the second input mesh. For each triangle $j$, **conn2**$(i,j)$ gives the indices of its three vertices (in anticlockwise order), for $i = 1,2,3$ and $j = 1,2,\ldots,\mathbf{nelt2}$.

*Constraints*:

$1 \leq \mathbf{conn2}(i,j) \leq \mathbf{nv2}$;
$\mathbf{conn2}(1,j) \neq \mathbf{conn2}(2,j)$;
$\mathbf{conn2}(1,j) \neq \mathbf{conn2}(3,j)$ and $\mathbf{conn2}(2,j) \neq \mathbf{conn2}(3,j)$, for $i = 1,2,3$ and $j = 1,2,\ldots,\mathbf{nelt2}$.

8: **reft2**(**nelt2**) – INTEGER array

**reft2**$(k)$ contains the user-supplied tag of the $k$th triangle from the second input mesh, for $k = 1,2,\ldots,\mathbf{nelt2}$.

9: **itrace** – INTEGER

The level of trace information required from nag_mesh_2d_join (d06db).

**itrace** $\leq 0$
No output is generated.

**itrace** $\geq 1$
Details about the common vertices, edges and triangles to both meshes are printed on the current advisory message unit (see nag_file_set_unit_advisory (x04ab)).

## 5.2 Optional Input Parameters

1: **eps** – REAL (KIND=nag_wp)

*Default*: 0.001.

The relative precision of the restitching of the two input meshes (see Section 9).

*Constraint*: **eps** $> 0.0$.

2: **nv1** – INTEGER

*Default*: the dimension of the array **coor1**.

The total number of vertices in the first input mesh.

*Constraint*: **nv1** $\geq 3$.

3: **nelt1** – INTEGER

*Default*: the dimension of the arrays **conn1**, **reft1**. (An error is raised if these dimensions are not equal.)

The number of triangular elements in the first input mesh.

*Constraint*: **nelt1** $\leq 2 \times$ **nv1** $- 1$.

4: **nedge1** – INTEGER

*Default*: the dimension of the array **edge1**.

The number of boundary edges in the first input mesh.

*Constraint*: **nedge1** $\geq 1$.

5: **nv2** – INTEGER

*Default*: the dimension of the array **coor2**.

The total number of vertices in the second input mesh.

*Constraint*: **nv2** $\geq 3$.

6: **nelt2** – INTEGER

*Default*: the dimension of the arrays **conn2**, **reft2**. (An error is raised if these dimensions are not equal.)

The number of triangular elements in the second input mesh.

*Constraint*: **nelt2** $\leq 2 \times$ **nv2** $- 1$.

7: **nedge2** – INTEGER

*Default*: the dimension of the array **edge2**.

The number of boundary edges in the second input mesh.

*Constraint*: **nedge2** $\geq 1$.

## 5.3 Output Parameters

1: **nv3** – INTEGER

The total number of vertices in the resulting mesh.

2: **nelt3** – INTEGER

The number of triangular elements in the resulting mesh.

3: **nedge3** – INTEGER

The number of boundary edges in the resulting mesh.

4: **coor3**$(\mathbf{2}, :)$ – REAL (KIND=nag_wp) array

The second dimension of the array **coor3** will be **nv1** $+$ **nv2**.

**coor3**$(1,i)$ will contain the $x$ coordinate of the $i$th vertex of the resulting mesh, for $i = 1,2,\ldots,$ **nv3**; while **coor3**$(2,i)$ will contain the corresponding $y$ coordinate.

5: **edge3**$(3,:)$ – INTEGER array

The second dimension of the array **edge3** will be **nedge1** + **nedge2**. This may be reduced to **nedge3** once that value is known.

The specification of the boundary edges of the resulting mesh. **edge3**$(i,j)$ will contain the vertex number of the $i$th end point ($i = 1,2$) of the $j$th boundary or interface edge.

If the two meshes overlap, **edge3**$(3,j)$ will contain the same tag as the corresponding edge belonging to the first and/or the second input mesh.

If the two meshes are adjacent,

(i) if the $j$th edge is part of the partition interface, then **edge3**$(3,j)$ will contain the value $1000 \times k_1 + k_2$ where $k_1$ and $k_2$ are the tags for the same edge of the first and the second mesh respectively;

(ii) otherwise, **edge3**$(3,j)$ will contain the same tag as the corresponding edge belonging to the first and/or the second input mesh.

6: **conn3**$(3,:)$ – INTEGER array

The second dimension of the array **conn3** will be **nelt1** + **nelt2**. This may be reduced to **nelt3** once that value is known.

The connectivity between triangles and vertices of the resulting mesh. **conn3**$(i,j)$ will give the indices of its three vertices (in anticlockwise order), for $i = 1,2,3$ and $j = 1,2,\ldots,$ **nelt3**.

7: **reft3**$(:)$ – INTEGER array

The dimension of the array **reft3** will be **nelt1** + **nelt2**. This may be reduced to **nelt3** once that value is known

If the two meshes form a partition, **reft3**$(k)$ will contain the same tag as the corresponding triangle belonging to the first or the second input mesh, for $k = 1,2,\ldots,$ **nelt3**. If the two meshes overlap, then **reft3**$(k)$ will contain the value $1000 \times k_1 + k_2$ where $k_1$ and $k_2$ are the user-supplied tags for the same triangle of the first and the second mesh respectively, for $k = 1,2,\ldots,$ **nelt3**.

8: **ifail** – INTEGER

**ifail** $= 0$ unless the function detects an error (see Section 5).

## 6 Error Indicators and Warnings

Errors or warnings detected by the function:

**ifail** $= 1$

On entry, **eps** $\leq 0.0$,
or **nv1** $< 3$,
or **nelt1** $> 2 \times$ **nv1** $- 1$,
or **nedge1** $< 1$,
or **edge1**$(i,j) < 1$ or **edge1**$(i,j) >$ **nv1** for some $i = 1,2$ and $j = 1,2,\ldots,$ **nedge1**,
or **edge1**$(1,j) =$ **edge1**$(2,j)$ for some $j = 1,2,\ldots,$ **nedge1**,
or **conn1**$(i,j) < 1$ or **conn1**$(i,j) >$ **nv1** for some $i = 1,2,3$ and $j = 1,2,\ldots,$ **nelt1**,
or **conn1**$(1,j) =$ **conn1**$(2,j)$ or **conn1**$(1,j) =$ **conn1**$(3,j)$ or
**conn1**$(2,j) =$ **conn1**$(3,j)$ for some $j = 1,2,\ldots,$ **nelt1**,
or **nv2** $< 3$,
or **nelt2** $> 2 \times$ **nv2** $- 1$,
or **nedge2** $< 1$,

or **edge2**$(i,j) < 1$ or **edge2**$(i,j) >$ **nv2** for some $i = 1,2$ and $j = 1,2,\ldots,$**nedge2**,
or **edge2**$(1,j) =$ **edge2**$(2,j)$ for some $j = 1,2,\ldots,$**nedge2**,
or **conn2**$(i,j) < 1$ or **conn2**$(i,j) >$ **nv2** for some $i = 1,2,3$ and $j = 1,2,\ldots,$**nelt2**,
or **conn2**$(1,j) =$ **conn2**$(2,j)$ or **conn2**$(1,j) =$ **conn2**$(3,j)$ or
**conn2**$(2,j) =$ **conn2**$(3,j)$ for some $j = 1,2,\ldots,$**nelt2**,
or $liwork < 2 \times$ **nv1** $+ 3 \times$ **nv2** $+$ **nelt1** $+$ **nelt2** $+$ **nedge1** $+$ **nedge2** $+ 1024$.

**ifail** $= 2$

Using the input precision **eps**, the function has detected fewer than two coincident vertices between the two input meshes. You are advised to try another value of **eps**; if this error still occurs the two meshes are probably not stitchable.

**ifail** $= 3$

A serious error has occurred in an internal call to the restitching function. You should check the input of the two meshes, especially the edge/vertex and/or the triangle/vertex connectivities. If the problem persists, contact NAG.

**ifail** $= 4$

The function has detected a different number of coincident triangles from the two input meshes in the overlapping zone. You should check the input of the two meshes, especially the triangle/vertex connectivities.

**ifail** $= 5$

The function has detected a different number of coincident edges from the two meshes on the partition interface. You should check the input of the two meshes, especially the edge/vertex connectivities.

**ifail** $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

**ifail** $= -399$

Your licence key may have expired or may not have been installed correctly.

**ifail** $= -999$

Dynamic memory allocation failed.

## 7 Accuracy

Not applicable.

## 8 Further Comments

nag_mesh_2d_join (d06db) finds all the common vertices between the two input meshes using the relative precision of the restitching argument **eps**. You are advised to vary the value of **eps** in the neighbourhood of 0.001 with **itrace** $\geq 1$ to get the optimal value for the meshes under consideration.

## 9 Example

For this function two examples are presented. There is a single example program for nag_mesh_2d_join (d06db), with a main program and the code to solve the two example problems given in Example 1 (EX1) and Example 2 (EX2).

**Example 1 (EX1)**

This example involves the unit square $[0,1]^2$ meshed uniformly, and then translated by a vector $\vec{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$ (using nag_mesh_2d_transform_affine (d06da)). This translated mesh is then restitched with the original mesh. Two cases are considered:

(a) overlapping meshes ($u_1 = 15.0$, $u_2 = 17.0$),

(b) partitioned meshes ($u_1 = 19.0$, $u_2 = 0.0$).

**Example 2 (EX2)**

This example restitches three geometries by calling the function nag_mesh_2d_join (d06db) twice. The result is a mesh with three partitions. The first geometry is meshed by the Delaunay–Voronoi process (using nag_mesh_2d_gen_delaunay (d06ab)), the second one meshed by an Advancing Front algorithm (using nag_mesh_2d_gen_front (d06ac)), while the third one is the result of a rotation (by $-\pi/2$) of the second one (using nag_mesh_2d_transform_affine (d06da)).

## 9.1 Program Text

```
     function d06db_example

fprintf('d06db example results\n\n');

d06db_ex1;
d06db_ex2;

function d06db_ex1

  fprintf('Example 1\n\n');
  n = 20;
  coor1 = zeros(2,n^2);
  dx = [0:1/(n-1):1];
  for j = 1:n
    coor1(1,(j-1)*n+1:j*n) = dx;
    coor1(2,(j-1)*n+1:j*n) = dx(j);
  end
  edge1 = ones(3, 4*n-4, nag_int_name);
  ind = nag_int(1:n-1);
  edge1(1, 1:(4*n-4)) = [ind,n*ind,n^2+1-ind,n^2+1-n*ind];
  edge1(2, 1:(4*n-5)) = edge1(1, 2:(4*n-4));
  edge1(2, 4*n-4)     = edge1(1, 1);
  conn1 = zeros(3, 2*(n-1)^2, nag_int_name);

  k = 0;
  l = nag_int(0);
  for i=1:n-1
    for j=1:n-1
       l = l + 1;
       k = k + 1;
       conn1(1, k) = l;
       conn1(2, k) = l + 1;
       conn1(3, k) = l + n + 1;
       k = k + 1;
       conn1(1, k) = l;
       conn1(2, k) = l + n + 1;
       conn1(3, k) = l + n;
    end
    l = l + 1;
  end
  reft1 = ones(2*(n-1)^2, 1, nag_int_name);
  reft2 = reft1;
  reft2(:) = nag_int(2);
  itype = [nag_int(1)];
  itrace = nag_int(1);

  for ktrans = 1:2
    if ktrans==2
```

```
      % Transform the first domain to obtain an overlapping second domain
      a = (n-5)/(n-1);
      b = (n-3)/(n-1);
      trans = [a; b; 0; 0; 0; 0];
      % plot boundary for partitioned geometry
      fig3 = figure;
      plot([0 1  1 0 0], [0 0 1 1 0], 'black', ...
           [a a+1 a+1 a a],[b b b+1 b+1 b],'black')
      axis off
      axis equal
      title('Figure 3: Boundary of overlapping squares');
      fig4 = figure;
    else
      % Domains partition
      trans = [1; 0; 0; 0; 0; 0];
      % plot boundary for partitioned geometry
      fig1 = figure;
      plot([0 2  2 0 0],[0 0 1 1 0],'black',[1 1],[0, 1],'black')
      axis off
      axis equal
      title('Figure 1: Boundary of partitioned squares');
      fig2 = figure;
    end
    [coor1, edge1, conn1, coor2, edge2, conn2, ifail] = ...
    d06da( ...
           itype, trans, coor1, edge1, conn1, itrace);

    % Restitch the meshes
    [nv3, nelt3, nedge3, coor3, edge3, conn3, reft3, ifail] = ...
    d06db( ...
           coor1, edge1, conn1, reft1, coor2, edge2, conn2, reft2, itrace);

    % Plot the result
    triplot(transpose(double(conn3(:,1:nelt3))), coor3(1,:), coor3(2,:));
    axis equal;
    axis off;
    if ktrans==2
      title ('Figure 4: mesh for two overlapping squares');
    else
      title ('Figure 2: mesh for two partitioned squares');
    end

  end
  % print(fig1,'-dpng','-r75','d06db_fig1.png');
  % print(fig1,'-deps','-r75','d06db_fig1.eps');
  % print(fig2,'-dpng','-r75','d06db_fig2.png');
  % print(fig2,'-deps','-r75','d06db_fig2.eps');
  % print(fig3,'-dpng','-r75','d06db_fig3.png');
  % print(fig3,'-deps','-r75','d06db_fig3.eps');
  % print(fig4,'-dpng','-r75','d06db_fig4.png');
  % print(fig4,'-deps','-r75','d06db_fig4.eps');

function d06db_ex2

  fprintf('\n\nExample 2\n\n');

  % First Geometry
  % ------------------------------------------------------------
  nlines = 9;
  % Characteristic points of the boundary mesh
  coorch = [ 2   2   1  -1  -2.2361   0.0000   0   0   0.0000;
            -1   1   0   0   0.0000  -2.2361  -1   1   2.2361];
  % Lines of the boundary mesh
  n10 = nag_int(10);
  line = [n10 10 10 10 10 10 10 10 10;
            1  2  9  5  6  3  8  4  7;
            2  9  5  6  1  8  4  7  3;
            0  2  2  2  2  1  1  1  1];
  rate = ones(nlines,1);
  % number of connected components
  ncomp = 2;
```

```
nlcomp = nag_int([5; -4]);
lcomp = ones(nlines,1,nag_int_name);
lcomp(1:nlcomp(1))        = [1   2   3   4   5];
lcomp(nlcomp(1)+1:nlines) = [9   8   7   6];

% Generate boundary mesh
nvmax = nag_int(700);
nedmx = nag_int(200);
itrace = nag_int(0);
crus = [0; 0];
[nvb1, coor1, nedge1, edge1, user, ifail] = ...
d06ba( ...
       coorch, line, @fbnd, crus, rate, nlcomp, lcomp, nvmax, ...
       nedmx, itrace);

% Generate mesh using Delaunay-Voronoi method
weight = [];
npropa = nag_int(1);
[nv1, nelt1, coor1, conn1, ifail] = ...
d06ab( ...
       nvb1, edge1, coor1, weight, npropa, itrace, 'nedge', nedge1);

% Smooth
numfix = nag_int([0]);
nqint = nag_int(0);
[coor1, ifail] = d06ca( ...
                        coor1, edge1, conn1, numfix, itrace, nqint, ...
                        'nv', nv1, 'nelt', nelt1, 'nedge', nedge1, ...
                        'nvfix', nag_int(0));

% Second Geometry
% ------------------------------------------------------------
nlines = 4;
% Characteristic points of the boundary mesh
coorch = [ 2   6   6   2;
          -1  -1   1   1];
% Lines of the boundary mesh
n19 = nag_int(19);
line = [n19 10 19 10;
          1  2  3  4;
          2  3  4  1;
          0  0  0  0];
rate = ones(nlines,1);
% number of connected components
ncomp = 1;
nlcomp = [nag_int(4)];
lcomp = nag_int([1   2   3   4]);

% Generate boundary mesh
[nvb2, coor2, nedge2, edge2, user, ifail] = ...
d06ba( ...
       coorch, line, @fbnd, crus, rate, nlcomp, lcomp, nvmax, ...
       nedmx, itrace);
% Generate mesh using Advancing Front method
[nv2, nelt2, coor2, conn2, ifail] = ...
d06ac( ...
       nvb2, edge2, coor2, weight, itrace, 'nedge', nedge2);

% Third Geometry (rotation and translation of second)
% ------------------------------------------------------------
itype = [nag_int(3)];
trans = [6; -1; -90; 0; 0; 0];
[coor3, edge3, conn3, coor3, edge3, conn3, ifail] = ...
d06da( ...
       itype, trans, coor2, edge2, conn2, itrace, 'nv', nv2, ...
       'nelt', nelt2, 'nedge', nedge2);

% ------------------------------------------------------------
% Combine 3 meshes
% ------------------------------------------------------------
% Restitch 1 and 2 to form mesh 4
```

```
  reft1(1:nelt1) = nag_int(1);
  reft2(1:nelt2) = nag_int(2);
  [nv4, nelt4, nedge4, coor4, edge4, conn4, reft4, ifail] = ...
  d06db( ...
         coor1, edge1, conn1, reft1, coor2, edge2, conn2, reft2, itrace, ...
         'nv1', nv1, 'nelt1', nelt1, 'nedge1', nedge1, ...
         'nv2', nv2, 'nelt2', nelt2, 'nedge2', nedge2);

  % Restitch 3 and 4 to form mesh 5
  reft3(1:nelt2) = nag_int(3);
  [nv5, nelt5, nedge5, coor5, edge5, conn5, reft5, ifail] = ...
  d06db( ...
         coor4, edge4, conn4, reft4, coor3, edge3, conn3, reft3, itrace, ...
         'nv1', nv4, 'nelt1', nelt4, 'nedge1', nedge4, ...
         'nv2', nv2, 'nelt2', nelt2, 'nedge2', nedge2);

  fprintf(' The restitched mesh characteristics\n');
  fprintf(' nv    = %5d\n', nv5);
  fprintf(' nelt  = %5d\n', nelt5);
  fprintf(' nedge = %5d\n', nedge5);

  % Plot boundary
  fig5 = figure;
  a = asin(1/sqrt(5));
  t1 = linspace(a, 2*pi-a, 100);
  t2 = linspace(0, 2*pi, 100);
  hold on
  r = sqrt(5);
  plot(r*cos(t1),r*sin(t1),'black');
  r = 1;
  plot(r*cos(t2),r*sin(t2),'black');
  plot([2 6 6 2 2],[-1 -1 1 1 -1],'black');
  plot([6 8 8 6 6],[-1 -1 3 3 -1],'black');
  axis off
  axis equal
  title('Figure 5: Boundary and Interior Interfaces for Key Shape');

  % Plot mesh
  fig6 = figure;
  triplot(transpose(double(conn5(:,1:nelt5))), coor5(1,:), coor5(2,:));
  axis equal;
  axis off;
  title ('Figure 6: mesh of keyshape');
  % print(fig5,'-dpng','-r75','d06db_fig5.png');
  % print(fig5,'-deps','-r75','d06db_fig5.eps');
  % print(fig6,'-dpng','-r75','d06db_fig6.png');
  % print(fig6,'-deps','-r75','d06db_fig6.eps');

function [result, user] = fbnd(i, x, y, user)
  result = 0;
  if (i == 1)
    % inner circle
    result = x^2 + y^2 - 1;
  elseif (i == 2)
    % outer circle
    result = x^2 + y^2 - 5;
  end
```

## 9.2 Program Results

```
     d06db example results

Example 1

 Transformation   1: translation
  translation vector:   1.000       0.000

 Final transformation matrix y = A*x + b:
    1.000       0.000          1.000
    0.000       1.000          0.000
```

```
 Transformation   1: translation
  translation vector:  0.7895      0.8947

 Final transformation matrix y = A*x + b:
    1.000       0.000         0.7895
    0.000       1.000         0.8947

Example 2

 The restitched mesh characteristics
 nv    =   643
 nelt  =  1133
 nedge =   171
```

**Figure 1: Boundary of partitioned squares**

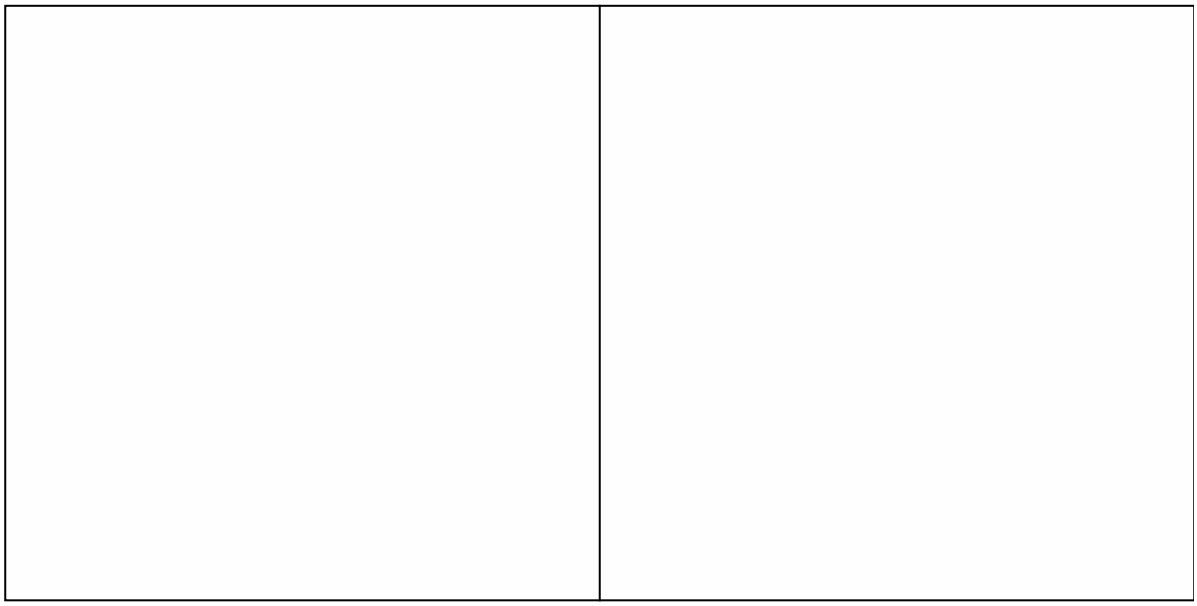

**Figure 2: mesh for two partitioned squares**

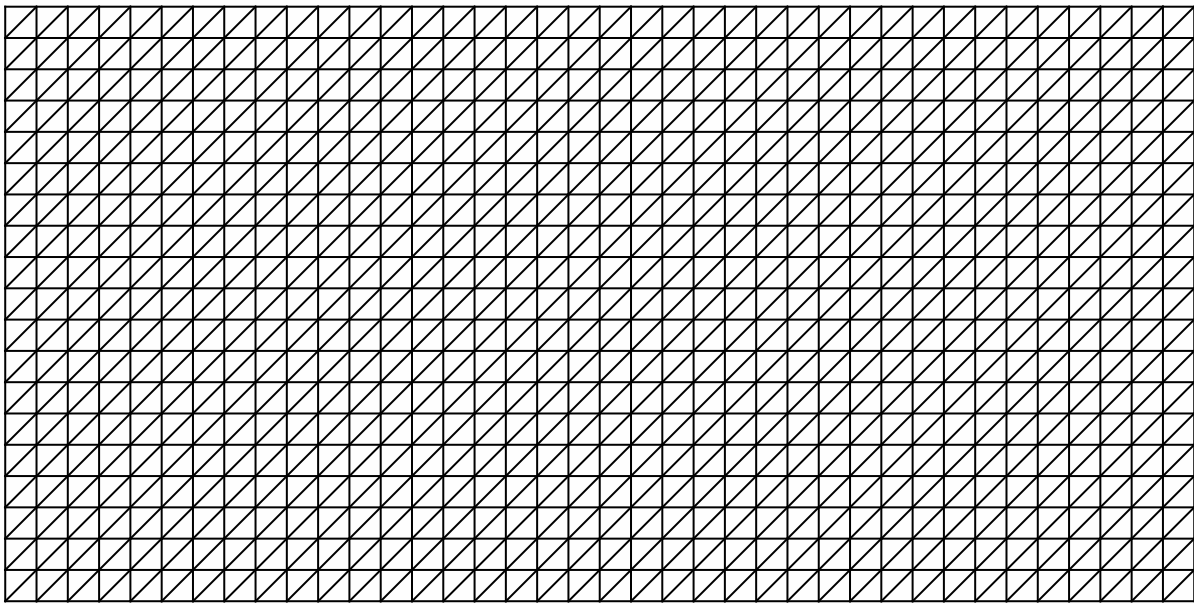

**Figure 3: Boundary of overlapping squares**

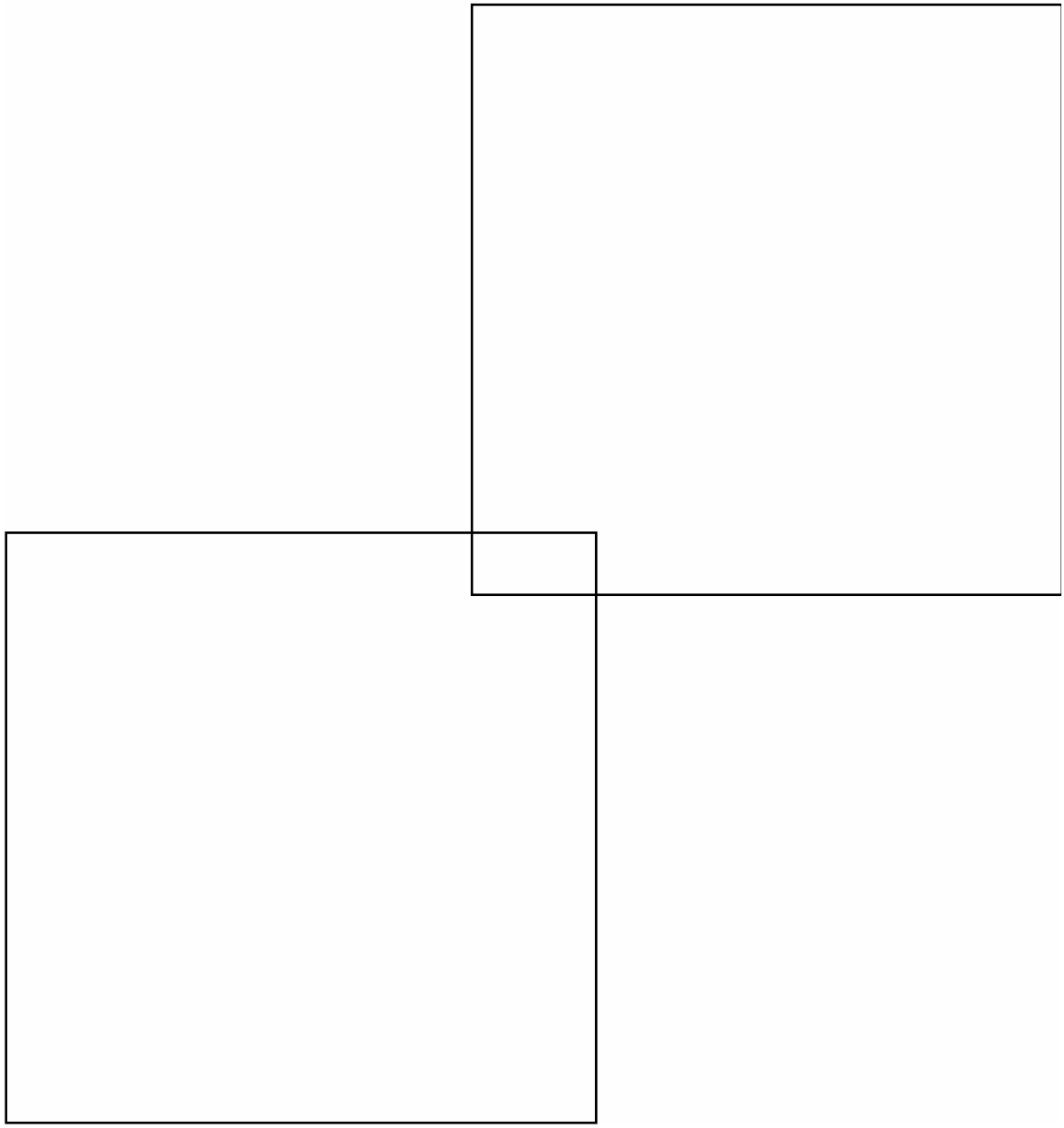

**Figure 4: mesh for two overlapping squares**

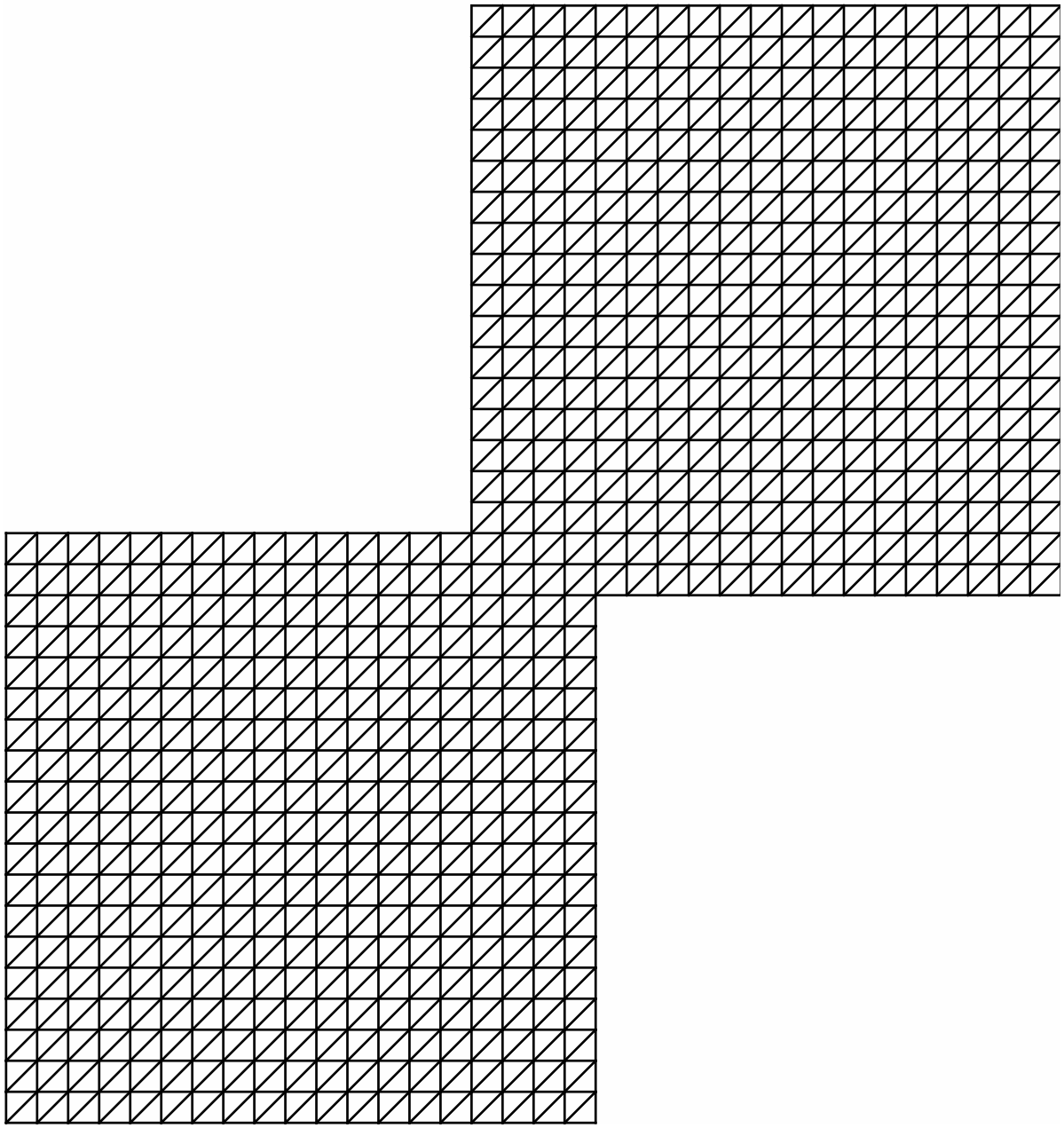

**Figure 5: Boundary and Interior Interfaces for Key Shape**

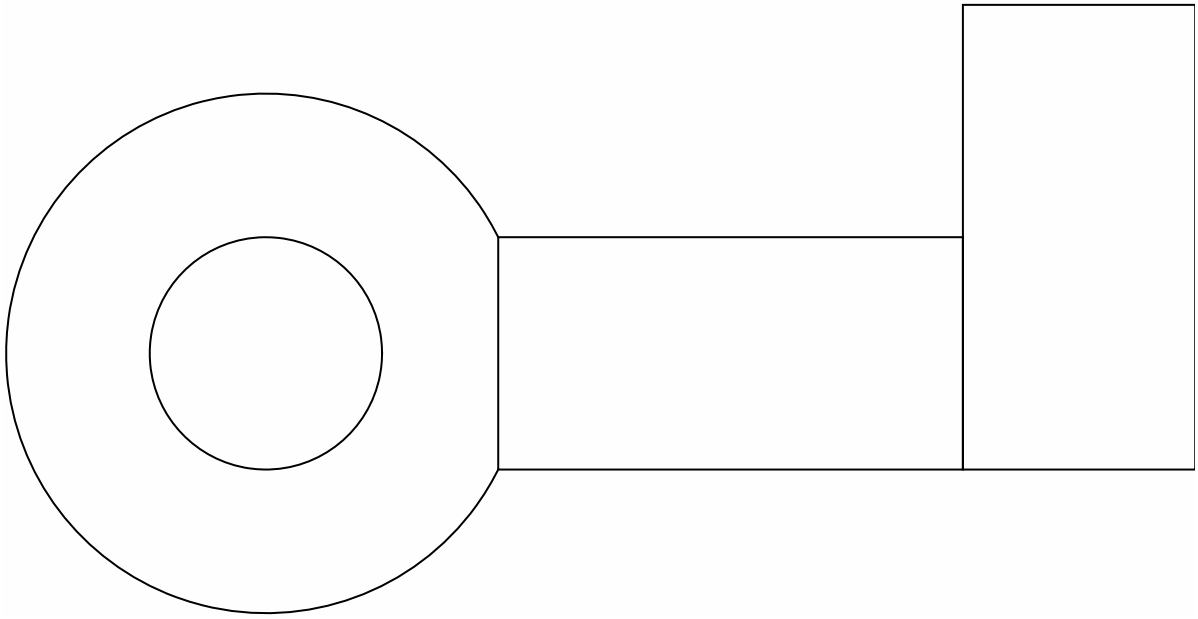

**Figure 6: mesh of keyshape**