

# NAG Library Function Document

## nag\_mesh2d\_join (d06dbc)

### 1 Purpose

nag\_mesh2d\_join (d06dbc) joins together (restitches) two adjacent, or overlapping, meshes.

### 2 Specification

```
#include <nag.h>
#include <nagd06.h>

void nag_mesh2d_join (double eps, Integer nv1, Integer nelt1, Integer nedge1,
  const double coor1[], const Integer edge1[], const Integer conn1[],
  const Integer reft1[], Integer nv2, Integer nelt2, Integer nedge2,
  const double coor2[], const Integer edge2[], const Integer conn2[],
  const Integer reft2[], Integer *nv3, Integer *nelt3, Integer *nedge3,
  double coor3[], Integer edge3[], Integer conn3[], Integer reft3[],
  Integer itrace, const char *outfile, NagError *fail)
```

### 3 Description

nag\_mesh2d\_join (d06dbc) 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 Arguments

- |    |                                                                                                     |              |
|----|-----------------------------------------------------------------------------------------------------|--------------|
| 1: | <b>eps</b> – double                                                                                 | <i>Input</i> |
|    | <i>On entry:</i> the relative precision of the restitching of the two input meshes (see Section 8). |              |
|    | <i>Suggested value:</i> 0.001.                                                                      |              |
|    | <i>Constraint:</i> <b>eps</b> > 0.0.                                                                |              |
| 2: | <b>nv1</b> – Integer                                                                                | <i>Input</i> |
|    | <i>On entry:</i> the total number of vertices in the first input mesh.                              |              |
|    | <i>Constraint:</i> <b>nv1</b> ≥ 3.                                                                  |              |
| 3: | <b>nelt1</b> – Integer                                                                              | <i>Input</i> |
|    | <i>On entry:</i> the number of triangular elements in the first input mesh.                         |              |
|    | <i>Constraint:</i> <b>nelt1</b> ≤ 2 × <b>nv1</b> – 1.                                               |              |
| 4: | <b>nedge1</b> – Integer                                                                             | <i>Input</i> |
|    | <i>On entry:</i> the number of boundary edges in the first input mesh.                              |              |
|    | <i>Constraint:</i> <b>nedge1</b> ≥ 1.                                                               |              |

- 5: **coor1** $[2 \times \mathbf{nv1}]$  – const double *Input*  
*On entry:* **coor1** $[2 \times (i - 1)]$  contains the  $x$  co-ordinate of the  $i$ th vertex of the first input mesh, for  $i = 1, 2, \dots, \mathbf{nv1}$ ; while **coor1** $[2 \times (i - 1) + 1]$  contains the corresponding  $y$  co-ordinate.
- 6: **edge1** $[3 \times \mathbf{nedge1}]$  – const Integer *Input*  
*On entry:* the specification of the boundary edges of the first input mesh. **edge1** $[3 \times (j - 1)]$  and **edge1** $[3 \times (j - 1) + 1]$  contain the vertex numbers of the two end points of the  $j$ th boundary edge. **edge1** $[3 \times (j - 1) + 2]$  is a user-supplied tag for the  $j$ th boundary edge.  
*Constraint:*  $1 \leq \mathbf{edge1}[3 \times (j - 1) + i - 1] \leq \mathbf{nv1}$  and **edge1** $[3 \times (j - 1)] \neq \mathbf{edge1}[3 \times (j - 1) + 1]$ , for  $i = 1, 2$  and  $j = 1, 2, \dots, \mathbf{nedge1}$ .
- 7: **conn1** $[3 \times \mathbf{nelt1}]$  – const Integer *Input*  
*On entry:* the connectivity between triangles and vertices of the first input mesh. For each triangle  $j$ , **conn1** $[3 \times (j - 1) + i - 1]$  gives the indices of its three vertices (in anticlockwise order), for  $i = 1, 2, 3$  and  $j = 1, 2, \dots, \mathbf{nelt1}$ .  
*Constraints:*  
 $1 \leq \mathbf{conn1}[3 \times (j - 1) + i - 1] \leq \mathbf{nv1}$ ;  
**conn1** $[3 \times (j - 1)] \neq \mathbf{conn1}[3 \times (j - 1) + 1]$ ;  
**conn1** $[3 \times (j - 1)] \neq \mathbf{conn1}[3 \times (j - 1) + 2]$  and  
**conn1** $[3 \times (j - 1) + 1] \neq \mathbf{conn1}[3 \times (j - 1) + 2]$ , for  $i = 1, 2, 3$  and  $j = 1, 2, \dots, \mathbf{nelt1}$ .
- 8: **reft1** $[\mathbf{nelt1}]$  – const Integer *Input*  
*On entry:* **reft1** $[k - 1]$  contains the user-supplied tag of the  $k$ th triangle from the first input mesh, for  $k = 1, 2, \dots, \mathbf{nelt1}$ .
- 9: **nv2** – Integer *Input*  
*On entry:* the total number of vertices in the second input mesh.  
*Constraint:*  $\mathbf{nv2} \geq 3$ .
- 10: **nelt2** – Integer *Input*  
*On entry:* the number of triangular elements in the second input mesh.  
*Constraint:*  $\mathbf{nelt2} \leq 2 \times \mathbf{nv2} - 1$ .
- 11: **nedge2** – Integer *Input*  
*On entry:* the number of boundary edges in the second input mesh.  
*Constraint:*  $\mathbf{nedge2} \geq 1$ .
- 12: **coor2** $[2 \times \mathbf{nv2}]$  – const double *Input*  
*On entry:* **coor2** $[2 \times (i - 1)]$  contains the  $x$  co-ordinate of the  $i$ th vertex of the second input mesh, for  $i = 1, 2, \dots, \mathbf{nv2}$ ; while **coor2** $[2 \times (i - 1) + 1]$  contains the corresponding  $y$  co-ordinate.
- 13: **edge2** $[3 \times \mathbf{nedge2}]$  – const Integer *Input*  
*On entry:* the specification of the boundary edges of the second input mesh. **edge2** $[3 \times (j - 1)]$  and **edge2** $[3 \times (j - 1) + 1]$  contain the vertex numbers of the two end points of the  $j$ th boundary edge. **edge2** $[3 \times (j - 1) + 2]$  is a user-supplied tag for the  $j$ th boundary edge.  
*Constraint:*  $1 \leq \mathbf{edge2}[3 \times (j - 1) + i - 1] \leq \mathbf{nv2}$  and **edge2** $[3 \times (j - 1)] \neq \mathbf{edge2}[3 \times (j - 1) + 1]$ , for  $i = 1, 2$  and  $j = 1, 2, \dots, \mathbf{nedge2}$ .

- 14: **conn2**[ $3 \times \mathbf{nelt2}$ ] – const Integer *Input*  
*On entry:* the connectivity between triangles and vertices of the second input mesh. For each triangle  $j$ , **conn2**[ $3 \times (j - 1) + i - 1$ ] gives the indices of its three vertices (in anticlockwise order), for  $i = 1, 2, 3$  and  $j = 1, 2, \dots, \mathbf{nelt2}$ .  
*Constraints:*  
 $1 \leq \mathbf{conn2}[3 \times (j - 1) + i - 1] \leq \mathbf{nv2}$ ;  
 $\mathbf{conn2}[3 \times (j - 1)] \neq \mathbf{conn2}[3 \times (j - 1) + 1]$ ;  
 $\mathbf{conn2}[3 \times (j - 1)] \neq \mathbf{conn2}[3 \times (j - 1) + 2]$  and  
 $\mathbf{conn2}[3 \times (j - 1) + 1] \neq \mathbf{conn2}[3 \times (j - 1) + 2]$ , for  $i = 1, 2, 3$  and  $j = 1, 2, \dots, \mathbf{nelt2}$ .
- 15: **reft2**[ $\mathbf{nelt2}$ ] – const Integer *Input*  
*On entry:* **reft2**[ $k - 1$ ] contains the user-supplied tag of the  $k$ th triangle from the second input mesh, for  $k = 1, 2, \dots, \mathbf{nelt2}$ .
- 16: **nv3** – Integer \* *Output*  
*On exit:* the total number of vertices in the resulting mesh.
- 17: **nelt3** – Integer \* *Output*  
*On exit:* the number of triangular elements in the resulting mesh.
- 18: **nedge3** – Integer \* *Output*  
*On exit:* the number of boundary edges in the resulting mesh.
- 19: **coor3**[ $\mathit{dim}$ ] – double *Output*  
**Note:** the dimension,  $\mathit{dim}$ , of the array **coor3** must be at least  $2 \times (\mathbf{nv1} + \mathbf{nv2})$ . This may be reduced to **nv3** once that value is known.  
The dimension,  $\mathit{dim}$ , of the array **coor3** must be at least  $2 \times (2 \times (\mathbf{nv1} + \mathbf{nv2}))$ . This may be reduced to **nv3** once that value is known  
*On exit:* **coor3**[ $\times i + 1$ ] will contain the  $x$  co-ordinate of the  $i$ th vertex of the resulting mesh, for  $i = 1, 2, \dots, \mathbf{nv3}$ ; while **coor3**[ $\times i + 2$ ] will contain the corresponding  $y$  co-ordinate.
- 20: **edge3**[ $\mathit{dim}$ ] – Integer *Output*  
**Note:** the dimension,  $\mathit{dim}$ , of the array **edge3** must be at least  $3 \times (\mathbf{nedge1} + \mathbf{nedge2})$ . This may be reduced to **nedge3** once that value is known.  
The dimension,  $\mathit{dim}$ , of the array **edge3** must be at least  $3 \times (3 \times (\mathbf{nedge1} + \mathbf{nedge2}))$ . This may be reduced to **nedge3** once that value is known  
*On exit:* the specification of the boundary edges of the resulting mesh. **edge3**[ $\times (j - 1) + i - 1$ ] 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**[ $\times (j - 1) + 2$ ] 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**[ $\times (j - 1) + 2$ ] 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**[ $\times (j - 1) + 2$ ] will contain the same tag as the corresponding edge belonging to the first and/or the second input mesh.
- 21: **conn3**[ $\mathit{dim}$ ] – Integer *Output*  
**Note:** the dimension,  $\mathit{dim}$ , of the array **conn3** must be at least  $3 \times (\mathbf{nelt1} + \mathbf{nelt2})$ . This may be reduced to **nelt3** once that value is known.

The dimension, *dim*, of the array **conn3** must be at least  $3 \times (3 \times (\mathbf{nelt1} + \mathbf{nelt2}))$ . This may be reduced to **nelt3** once that value is known

*On exit:* the connectivity between triangles and vertices of the resulting mesh. **conn3** $[\times(j-1) + i - 1]$  will give the indices of its three vertices (in anticlockwise order), for  $i = 1, 2, 3$  and  $j = 1, 2, \dots, \mathbf{nelt3}$ .

22: **reft3**[*dim*] – Integer *Output*

**Note:** the dimension, *dim*, of the array **reft3** must be at least **nelt1** + **nelt2**. This may be reduced to **nelt3** once that value is known.

*On exit:* if the two meshes form a partition, **reft3**[ $k - 1$ ] will contain the same tag as the corresponding triangle belonging to the first or the second input mesh, for  $k = 1, 2, \dots, \mathbf{nelt3}$ . If the two meshes overlap, then **reft3**[ $k - 1$ ] 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  $k = 1, 2, \dots, \mathbf{nelt3}$ .

23: **itrace** – Integer *Input*

*On entry:* the level of trace information required from nag\_mesh2d\_join (d06dbc).

**itrace**  $\leq 0$

No output is generated.

**itrace**  $\geq 1$

Details about the common vertices, edges and triangles to both meshes are printed.

24: **outfile** – const char \* *Input*

*On entry:* the name of a file to which diagnostic output will be directed. If **outfile** is **NULL** the diagnostic output will be directed to standard output.

25: **fail** – NagError \* *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry, **nedge1** =  $\langle value \rangle$ .

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

On entry, **nedge2** =  $\langle value \rangle$ .

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

On entry, **nv1** =  $\langle value \rangle$ .

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

On entry, **nv2** =  $\langle value \rangle$ .

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

### NE\_INT\_2

On entry, **nelt1**  $> (2 \times \mathbf{nv1} - 1)$  : **nelt1** =  $\langle value \rangle$ , **nv1** =  $\langle value \rangle$ .

On entry, **nelt2**  $> (2 \times \mathbf{nv2} - 1)$  : **nelt2** =  $\langle value \rangle$ , **nv2** =  $\langle value \rangle$ .

On entry, the endpoints of edge  $j$  in the first mesh have the same index  $i : j = \langle value \rangle$ ,  $i = \langle value \rangle$ .

On entry, the endpoints of edge  $j$  in the second mesh have the same index  $i : j = \langle value \rangle$ ,  $i = \langle value \rangle$ .

On entry, vertices 1 and 2 of triangle  $k$  in the first mesh have the same index  $i : k = \langle value \rangle$ ,  $i = \langle value \rangle$ .

On entry, vertices 1 and 2 of triangle  $k$  in the second mesh have the same index  $i : k = \langle value \rangle$ ,  $i = \langle value \rangle$ .

On entry, vertices 1 and 3 of triangle  $k$  in the first mesh have the same index  $i : k = \langle value \rangle$ ,  $i = \langle value \rangle$ .

On entry, vertices 1 and 3 of triangle  $k$  in the second mesh have the same index  $i : k = \langle value \rangle$ ,  $i = \langle value \rangle$ .

On entry, vertices 2 and 3 of triangle  $k$  in the first mesh have the same index  $i : k = \langle value \rangle$ ,  $i = \langle value \rangle$ .

On entry, vertices 2 and 3 of triangle  $k$  in the second mesh have the same index  $i : k = \langle value \rangle$ ,  $i = \langle value \rangle$ .

#### NE\_INT\_4

On entry,  $\mathbf{conn1}(i, j) < 1$  or  $\mathbf{conn1}(i, j) > \mathbf{nv1}$ , where  $\mathbf{conn1}(i, j)$  denotes  $\mathbf{conn1}[3 \times (j - 1) + i - 1] : \mathbf{conn1}(i, j) = \langle value \rangle$ ,  $i = \langle value \rangle$ ,  $j = \langle value \rangle$ ,  $\mathbf{nv1} = \langle value \rangle$ .

On entry,  $\mathbf{conn2}(i, j) < 1$  or  $\mathbf{conn2}(i, j) > \mathbf{nv2}$ , where  $\mathbf{conn2}(i, j)$  denotes  $\mathbf{conn2}[3 \times (j - 1) + i - 1] : \mathbf{conn2}(i, j) = \langle value \rangle$ ,  $i = \langle value \rangle$ ,  $j = \langle value \rangle$ ,  $\mathbf{nv2} = \langle value \rangle$ .

On entry,  $\mathbf{edge1}(i, j) < 1$  or  $\mathbf{edge1}(i, j) > \mathbf{nv1}$ , where  $\mathbf{edge1}(i, j)$  denotes  $\mathbf{edge1}[3 \times (j - 1) + i - 1] : \mathbf{edge1}(i, j) = \langle value \rangle$ ,  $i = \langle value \rangle$ ,  $j = \langle value \rangle$ ,  $\mathbf{nv1} = \langle value \rangle$ .

On entry,  $\mathbf{edge2}(i, j) < 1$  or  $\mathbf{edge2}(i, j) > \mathbf{nv2}$ , where  $\mathbf{edge2}(i, j)$  denotes  $\mathbf{edge2}[3 \times (j - 1) + i - 1] : \mathbf{edge2}(i, j) = \langle value \rangle$ ,  $i = \langle value \rangle$ ,  $j = \langle value \rangle$ ,  $\mathbf{nv2} = \langle value \rangle$ .

#### NE\_INTERNAL\_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

#### NE\_MESH\_ERROR

A serious error has occurred in an internal call to the restitching routine. Check the input of the two meshes, especially the edges/vertices and/or the triangles/vertices connectivities. Seek expert help.

The function has detected a different number of coincident edges from the two meshes on the partition interface  $\langle value \rangle \langle value \rangle$ . Check the input of the two meshes, especially the edges/vertices connectivity.

The function has detected a different number of coincident triangles from the two meshes in the overlapping zone  $\langle value \rangle \langle value \rangle$ . Check the input of the two meshes, especially the triangles/vertices connectivity.

The function has detected only  $\langle value \rangle$  coincident vertices with a precision  $\mathbf{eps} = \langle value \rangle$ . Either  $\mathbf{eps}$  should be changed or the two meshes are not restitchable.

#### NE\_NOT\_CLOSE\_FILE

Cannot close file  $\langle value \rangle$ .

#### NE\_NOT\_WRITE\_FILE

Cannot open file  $\langle value \rangle$  for writing.

**NE\_REAL**

On entry, **eps** =  $\langle value \rangle$ .  
 Constraint: **eps** > 0.0.

**7 Accuracy**

Not applicable.

**8 Further Comments**

nag\_mesh2d\_join (d06dbc) 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\_mesh2d\_join (d06dbc), 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\_mesh2d\_trans (d06dac)). 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 ( $u_1 = 19.0$ ,  $u_2 = 0.0$ ).

The mesh on the unit square has 400 vertices, 722 triangles and its boundary has 76 edges. In the overlapping case the resulting geometry is shown in Figure 1 and Figure 2. The resulting geometry for the partitioned meshes is shown in Figure 3.

**Example 2 (ex2)**

This example restitches three geometries by calling the function nag\_mesh2d\_join (d06dbc) twice. The result is a mesh with three partitions. The first geometry is meshed by the Delaunay–Voronoi process (using nag\_mesh2d\_delaunay (d06abc)), the second one meshed by an Advancing Front algorithm (using nag\_mesh2d\_front (d06acc)), while the third one is the result of a rotation (by  $-\pi/2$ ) of the second one (using nag\_mesh2d\_trans (d06dac)). The resulting geometry is shown in Figure 4 and Figure 5.

**9.1 Program Text**

```

/* nag_mesh2d_join (d06dbc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagd06.h>

#ifdef __cplusplus
extern "C" {
#endif
static double NAG_CALL fbnd(Integer, double, double, Nag_Comm *);
#ifdef __cplusplus

```

```

}
#endif

static int ex1(FILE *fpin, FILE *fpout, char *outfile);
static int ex2(FILE *fpin, FILE *fpout, char *outfile);

#define EDGE1(I, J)  edge1[3*((J) -1)+(I) -1]
#define EDGE2(I, J)  edge2[3*((J) -1)+(I) -1]
#define EDGE3(I, J)  edge3[3*((J) -1)+(I) -1]
#define EDGE4(I, J)  edge4[3*((J) -1)+(I) -1]
#define EDGE5(I, J)  edge5[3*((J) -1)+(I) -1]
#define CONN1(I, J)  conn1[3*((J) -1)+(I) -1]
#define CONN2(I, J)  conn2[3*((J) -1)+(I) -1]
#define CONN3(I, J)  conn3[3*((J) -1)+(I) -1]
#define CONN4(I, J)  conn4[3*((J) -1)+(I) -1]
#define CONN5(I, J)  conn5[3*((J) -1)+(I) -1]
#define COOR1(I, J)  coor1[2*((J) -1)+(I) -1]
#define COOR2(I, J)  coor2[2*((J) -1)+(I) -1]
#define COOR3(I, J)  coor3[2*((J) -1)+(I) -1]
#define COOR4(I, J)  coor4[2*((J) -1)+(I) -1]
#define COOR5(I, J)  coor5[2*((J) -1)+(I) -1]
#define TRANS(I, J)  trans[6*((J) -1)+(I) -1]
#define LINED(I, J)  lined[4*((J) -1)+(I) -1]
#define COORCH(I, J) coorch[2*(J-1)+I-1]
#define COORUS(I, J) coorus[2*(J-1)+I-1]

int main(int argc, char *argv[])
{
    FILE      *fpin, *fpout;
    char      *outfile = 0;
    Integer    exit_status_ex1 = 0;
    Integer    exit_status_ex2 = 0;
    NagError   fail;

    INIT_FAIL(fail);

    /* Check for command-line IO options */
    fpin = nag_example_file_io(argc, argv, "-data", NULL);
    fpout = nag_example_file_io(argc, argv, "-results", NULL);
    (void) nag_example_file_io(argc, argv, "-nag_write", &outfile);

    fprintf(fpout, "nag_mesh2d_join (d06dbc) Example Program Results\n");
    exit_status_ex1 = ex1(fpin, fpout, outfile);
    exit_status_ex2 = ex2(fpin, fpout, outfile);

    if (fpin != stdin) fclose(fpin);
    if (fpout != stdout) fclose(fpout);
    return (exit_status_ex1 == 0 && exit_status_ex2 == 0 )? 0 : 1;
}

int ex1(FILE *fpin, FILE *fpout, char *outfile)
{
    const Integer nvmax = 900, nedmx = 200, neltmx = 2*nvmax+5, ntrans = 1,
               mode = 0;
    double      eps;
    Integer      exit_status, i, imax, itrace, itrans, jmax, jtrans, k, ktrans;
    Integer      nedge1, nedge2, nedge3, nelt1, nelt2, nelt3, nv1, nv2, nv3,
               reftk;
    Integer      imaxml, jmaxml, ind;
    char         pmesh[2];
    double      *coor1 = 0, *coor2 = 0, *coor3 = 0, *trans = 0;
    Integer      *conn1 = 0, *conn2 = 0, *conn3 = 0, *edge1 = 0, *edge2 = 0;
    Integer      *edge3 = 0, *itype = 0, *reft1 = 0, *reft2 = 0, *reft3 = 0;
    NagError     fail;

    INIT_FAIL(fail);
    exit_status = 0;

    fprintf(fpout, "\nExample 1\n\n");

    /* Skip heading in data file */

```

```

fscanf(fpin, "%*[\n] ");
fscanf(fpin, "%*[\n] ");

/* Read the mesh: coordinates and connectivity of the 1st domain */

fscanf(fpin, "%ld", &nv1);
fscanf(fpin, "%ld", &nelt1);
fscanf(fpin, "%*[\n] ");

nv2 = nv1;
nelt2 = nelt1;

imax = 20;
jmax = imax;
imaxml = imax - 1;
jmaxml = jmax - 1;
nedge1 = 2*(imaxml + jmaxml);
nedge2 = nedge1;

/* Allocate memory */

if (!(coor1 = NAG_ALLOC(2*nv1, double)) ||
    !(coor2 = NAG_ALLOC(2*nv2, double)) ||
    !(coor3 = NAG_ALLOC(2*nvmax, double)) ||
    !(trans = NAG_ALLOC(6*ntrans, double)) ||
    !(conn1 = NAG_ALLOC(3*nelt1, Integer)) ||
    !(conn2 = NAG_ALLOC(3*nelt2, Integer)) ||
    !(conn3 = NAG_ALLOC(3*neltmx, Integer)) ||
    !(edge1 = NAG_ALLOC(3*nedge1, Integer)) ||
    !(edge2 = NAG_ALLOC(3*nedge2, Integer)) ||
    !(edge3 = NAG_ALLOC(3*nedmx, Integer)) ||
    !(itype = NAG_ALLOC(ntrans, Integer)) ||
    !(ref1 = NAG_ALLOC(nelt1, Integer)) ||
    !(ref2 = NAG_ALLOC(nelt2, Integer)) ||
    !(ref3 = NAG_ALLOC(neltmx, Integer)))
{
    fprintf(fpout, "Allocation failure\n");
    exit_status = -1;
    goto END;
}

for (i = 1; i <= nv1; ++i)
{
    fscanf(fpin, "%lf", &COOR1(1, i));
    fscanf(fpin, "%lf", &COOR1(2, i));
    fscanf(fpin, "%*[\n] ");
}

for (k = 1; k <= nelt1; ++k)
{
    fscanf(fpin, "%ld", &CONN1(1, k));
    fscanf(fpin, "%ld", &CONN1(2, k));
    fscanf(fpin, "%ld", &CONN1(3, k));
    fscanf(fpin, "%ld", &reftk);
    fscanf(fpin, "%*[\n] ");

    ref1[k-1] = 1;
    ref2[k-1] = 2;
}

fscanf(fpin, " ' %1s '", pmesh);
fscanf(fpin, "%*[\n] ");

/* the edges of the boundary */

ind = 0;

for (i = 1; i <= imaxml; ++i)
{
    ++ind;
}

```

```

    EDGE1(1, ind) = i;
    EDGE1(2, ind) = i + 1;
    EDGE1(3, ind) = 1;
}

for (i = 1; i <= jmaxm1; ++i)
{
    ++ind;
    EDGE1(1, ind) = i*imax;
    EDGE1(2, ind) = (i+1)*imax;
    EDGE1(3, ind) = 1;
}

for (i = 1; i <= imaxm1; ++i)
{
    ++ind;
    EDGE1(1, ind) = imax*jmax - i + 1;
    EDGE1(2, ind) = imax*jmax - i;
    EDGE1(3, ind) = 1;
}

for (i = 1; i <= jmaxm1; ++i)
{
    ++ind;
    EDGE1(1, ind) = (jmax - i)*imax + 1;
    EDGE1(2, ind) = (jmax - i - 1)*imax + 1;
    EDGE1(3, ind) = 1;
}

for (ktrans = 0; ktrans < 2; ++ktrans)
{
    /* Translation of the 1st domain to obtain the 2nd domain */
    /* KTRANS = 0 leading to a domain overlapping */
    /* KTRANS = 1 leading to a domain partition */

    if (ktrans == 0)
    {
        itrans = imax - 5;
        jtrans = jmax - 3;
    }
    else
    {
        itrans = imax - 1;
        jtrans = 0;
    }

    itype[0] = 1;
    TRANS(1, 1) = (double) itrans/(imax - 1.0);
    TRANS(2, 1) = (double) jtrans/(jmax - 1.0);
    itrace = 0;

    /* nag_mesh2d_trans (d06dac).
     * Generates a mesh resulting from an affine transformation
     * of a given mesh
     */
    if (outfile) fclose(fpout);
    nag_mesh2d_trans(mode, nv2, nedge2, nelt2, ntrans, itype, trans, coor1,
                    edge1, conn1, coor2, edge2, conn2, itrace, outfile,
                    &fail);
    if (outfile && !(fpout = fopen(outfile, "a")))
    {
        exit_status = 2;
        goto END;
    }
    if (fail.code == NE_NOERROR)
    {
        for (i = 1; i <= nedge2; ++i) EDGE2(3, i) = 2;

        /* Call to the restitching driver */

        itrace = 0;

```

```

eps = 0.01;

/* nag_mesh2d_join (d06dbc).
 * Joins together two given adjacent (possibly overlapping)
 * meshes
 */
if (outfile) fclose(fpout);
nag_mesh2d_join(eps, nv1, nelt1, nedge1, coor1, edge1, conn1, reft1,
                nv2, nelt2, nedge2, coor2, edge2, conn2, reft2, &nv3,
                &nelt3, &nedge3, coor3, edge3, conn3, reft3, itrace,
                outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
    exit_status = 2;
    goto END;
}
if (fail.code == NE_NOERROR)
{
    if (pmesh[0] == 'N')
    {
        if (ktrans == 0)
        {
            fprintf(fpout, "The restitched mesh characteristics\n");
            fprintf(fpout, "in the overlapping case\n");
        }
        else
        {
            fprintf(fpout, "in the partition case\n");
        }
        fprintf(fpout, " nv      =%6ld\n", nv3);
        fprintf(fpout, " nelt   =%6ld\n", nelt3);
        fprintf(fpout, " nedge =%6ld\n", nedge3);
    }
    else if (pmesh[0] == 'Y')
    {
        /* Output the mesh to view it using the
         NAG Graphics Library */

        fprintf(fpout, " %10ld%10ld%10ld\n", nv3,
                nelt3, nedge3);

        for (i = 1; i <= nv3; ++i)
            fprintf(fpout, " %15.6e %15.6e\n",
                    COOR3(1, i), COOR3(2, i));

        for (k = 1; k <= nelt3; ++k)
            fprintf(fpout, " %10ld%10ld%10ld"
                    "%10ld\n", CONN3(1, k), CONN3(2, k),
                    CONN3(3, k), reft3[k - 1]);

        for (k = 1; k <= nedge3; ++k)
            fprintf(fpout, " %10ld%10ld%10ld\n",
                    EDGE3(1, k), EDGE3(2, k), EDGE3(3, k));
    }
    else
    {
        fprintf(fpout, "Problem with the printing option Y or N\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    fprintf(fpout, "Error from nag_mesh2d_join (d06dbc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}
}
else
{

```

```

        fprintf(fpout, "Error from nag_mesh2d_trans (d06dac).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
    }
}

END:
if (coor1) NAG_FREE(coor1);
if (coor2) NAG_FREE(coor2);
if (coor3) NAG_FREE(coor3);
if (trans) NAG_FREE(trans);
if (conn1) NAG_FREE(conn1);
if (conn2) NAG_FREE(conn2);
if (conn3) NAG_FREE(conn3);
if (edge1) NAG_FREE(edge1);
if (edge2) NAG_FREE(edge2);
if (edge3) NAG_FREE(edge3);
if (itype) NAG_FREE(itype);
if (reft1) NAG_FREE(reft1);
if (reft2) NAG_FREE(reft2);
if (reft3) NAG_FREE(reft3);

return exit_status;
}

int ex2(FILE *fpin, FILE *fpout, char *outfile)
{
    const Integer nvmax = 900, nedmx = 200, neltnx = 2*nvmax+5,
                ntrans = 1, nus = 0, nvint = 0, nvfix = 0;
    double      eps;
    Integer      exit_status, i, itrace, j, k, l, ncomp, nedge1, nedge2, nedge3;
    Integer      nedge4, nedge5, nelt1, nelt2, nelt3, nelt4, nelt5, nlines;
    Integer      npropa, nqint, nv1, nv2, nv3, nv4, nv5, nvb1, nvb2, mode;
    char         pmesh[2];
    double       *coor1 = 0, *coor2 = 0, *coor3 = 0, *coor4 = 0, *coor5 = 0;
    double       *coorch = 0, *coorus = 0, *rate = 0, *trans = 0, *weight = 0;
    Integer      *conn1 = 0, *conn2 = 0, *conn3 = 0, *conn4 = 0, *conn5 = 0;
    Integer      *edge1 = 0, *edge2 = 0, *edge3 = 0, *edge4 = 0, *edge5 = 0;
    Integer      *itype = 0, *lcomp = 0, *lined = 0, *nlcomp = 0, *numfix = 0;
    Integer      *reft1 = 0, *reft2 = 0, *reft3 = 0, *reft4 = 0, *reft5 = 0;
    NagError     fail;
    Nag_Comm     comm;

    INIT_FAIL(fail);
    exit_status = 0;

    fprintf(fpout, "\nExample 2\n\n");

    /* Skip heading in data file */

    fscanf(fpin, "%*[\n] ");

    /* Build the mesh of the 1st domain */
    /* Initialise boundary mesh inputs: */
    /* the number of line and of the characteristic points of */
    /* the boundary mesh */

    fscanf(fpin, "%ld", &nlines);
    fscanf(fpin, "%*[\n] ");

    /* Allocate memory */

    if (!(coor1 = NAG_ALLOC(2*nvmax, double)) ||
        !(coor2 = NAG_ALLOC(2*nvmax, double)) ||
        !(coor3 = NAG_ALLOC(2*nvmax, double)) ||
        !(coor4 = NAG_ALLOC(2*nvmax, double)) ||
        !(coor5 = NAG_ALLOC(2*nvmax, double)) ||
        !(coorch = NAG_ALLOC(2*nlines, double)) ||
        !(coorus = NAG_ALLOC(1, double)) ||
        !(rate = NAG_ALLOC(nlines, double)) ||

```

```

!(trans = NAG_ALLOC(6*ntrans, double)) ||
!(weight = NAG_ALLOC(1, double)) ||
!(conn1 = NAG_ALLOC(3*neltmx, Integer)) ||
!(conn2 = NAG_ALLOC(3*neltmx, Integer)) ||
!(conn3 = NAG_ALLOC(3*neltmx, Integer)) ||
!(conn4 = NAG_ALLOC(3*neltmx, Integer)) ||
!(conn5 = NAG_ALLOC(3*neltmx, Integer)) ||
!(edge1 = NAG_ALLOC(3*nedmx, Integer)) ||
!(edge2 = NAG_ALLOC(3*nedmx, Integer)) ||
!(edge3 = NAG_ALLOC(3*nedmx, Integer)) ||
!(edge4 = NAG_ALLOC(3*nedmx, Integer)) ||
!(edge5 = NAG_ALLOC(3*nedmx, Integer)) ||
!(itype = NAG_ALLOC(ntrans, Integer)) ||
!(lcomp = NAG_ALLOC(nlines, Integer)) ||
!(lined = NAG_ALLOC(4*nlines, Integer)) ||
!(numfix = NAG_ALLOC(1, Integer)) ||
!(reft1 = NAG_ALLOC(2*nvmax+5, Integer)) ||
!(reft2 = NAG_ALLOC(2*nvmax+5, Integer)) ||
!(reft3 = NAG_ALLOC(2*nvmax+5, Integer)) ||
!(reft4 = NAG_ALLOC(2*nvmax+5, Integer)) ||
!(reft5 = NAG_ALLOC(2*nvmax+5, Integer))
{
  fprintf(fpout, "Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* Characteristic points of the boundary geometry */
for (j = 1; j <= nlines; ++j)
{
  fscanf(fpin, "%lf", &COORCH(1, j));
}
fscanf(fpin, "%*[\n] ");

for (j = 1; j <= nlines; ++j)
{
  fscanf(fpin, "%lf", &COORCH(2, j));
}
fscanf(fpin, "%*[\n] ");

/* The lines of the boundary mesh */
for (j = 1; j <= nlines; ++j)
{
  for (i = 1; i <= 4; ++i) fscanf(fpin, "%ld", &LINED(i, j));
  fscanf(fpin, "%lf", &rate[j - 1]);
}
fscanf(fpin, "%*[\n] ");

/* The number of connected components */
/* on the boundary and their data */
fscanf(fpin, "%ld", &ncomp);
fscanf(fpin, "%*[\n] ");

/* Allocate memory */
if (!(nlcomp = NAG_ALLOC(ncomp, Integer)))
{
  fprintf(fpout, "Allocation failure\n");
  exit_status = -1;
  goto END;
}

j = 1;

for (i = 1; i <= ncomp; ++i)
{
  fscanf(fpin, "%ld", &nlcomp[i - 1]);
  fscanf(fpin, "%*[\n] ");
}

```

```

        l = j + abs(nlcomp[i - 1]) - 1;
        for (k = j; k <= l; ++k) fscanf(fpin, "%ld", &lcomp[k - 1]);
        fscanf(fpin, "%*[\n] ");
        j += abs(nlcomp[i - 1]);
    }

    itrace = 0;

    /* Call to the 2D boundary mesh generator */

    /* nag_mesh2d_bound (d06bac).
     * Generates a boundary mesh
     */
    if (outfile) fclose(fpout);
    nag_mesh2d_bound(nlines, coorch, lined, fbnd, coorus, nus, rate, ncomp,
                    nlcomp, lcomp, nvmax, nedmx, &nvb1, coor1, &nedge1,
                    edge1, itrace, outfile, &comm, &fail);
    if (outfile && !(fpout = fopen(outfile, "a")))
    {
        exit_status = 2;
        goto END;
    }
    if (fail.code != NE_NOERROR)
    {
        fprintf(fpout, "Error from nag_mesh2d_bound (d06bac).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
    }

    /* Generate mesh using Delaunay-Voronoi method */

    /* Initialise mesh control parameters */

    itrace = 0;
    npropa = 1;

    /* nag_mesh2d_delaunay (d06abc).
     * Generates a two-dimensional mesh using a Delaunay-Voronoi
     * process
     */
    if (outfile) fclose(fpout);
    nag_mesh2d_delaunay(nvb1, nvint, nvmax, nedge1, edge1, &nv1, &nelt1, coor1,
                       conn1, weight, npropa, itrace, outfile, &fail);
    if (outfile && !(fpout = fopen(outfile, "a")))
    {
        exit_status = 2;
        goto END;
    }
    if (fail.code != NE_NOERROR)
    {
        fprintf(fpout, "Error from nag_mesh2d_delaunay (d06abc).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
    }

    for (k = 1; k <= nelt1; ++k) reft1[k - 1] = 1;

    /* Call the smoothing routine */

    nqint = 10;
    /* nag_mesh2d_smooth (d06cac).
     * Uses a barycentering technique to smooth a given mesh
     */
    if (outfile) fclose(fpout);
    nag_mesh2d_smooth(nv1, nelt1, nedge1, coor1, edge1, conn1, nvfix, numfix,
                     itrace, outfile, nqint, &fail);
    if (outfile && !(fpout = fopen(outfile, "a")))
    {
        exit_status = 2;
    }

```

```

    goto END;
}
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_mesh2d_smooth (d06cac).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}

/* Build the mesh of the 2nd domain */

fscanf(fpin, "%ld", &nlines);
fscanf(fpin, "%*[\n] ");

/* Characteristic points of the boundary geometry */

for (j = 1; j <= nlines; ++j) fscanf(fpin, "%lf", &COORCH(1, j));
fscanf(fpin, "%*[\n] ");

for (j = 1; j <= nlines; ++j) fscanf(fpin, "%lf", &COORCH(2, j));
fscanf(fpin, "%*[\n] ");

/* The lines of the boundary mesh */

for (j = 1; j <= nlines; ++j)
{
    for (i = 1; i <= 4; ++i) fscanf(fpin, "%ld", &LINED(i, j));
    fscanf(fpin, "%lf", &rate[j - 1]);
}
fscanf(fpin, "%*[\n] ");

/* The number of connected components */
/* to the boundary and their data */

fscanf(fpin, "%ld", &ncomp);
fscanf(fpin, "%*[\n] ");

j = 1;
for (i = 1; i <= ncomp; ++i)
{
    fscanf(fpin, "%ld", &nlcomp[i - 1]);
    fscanf(fpin, "%*[\n] ");

    for (k = j; k <= j+abs(nlcomp[i-1])-1; ++k)
        fscanf(fpin, "%ld", &lcomp[k - 1]);
    fscanf(fpin, "%*[\n] ");
    j += abs(nlcomp[i-1]);
}

fscanf(fpin, " ' %ls '", pmesh);
fscanf(fpin, "%*[\n] ");

itrace = 0;

/* Call to the 2D boundary mesh generator */

/* nag_mesh2d_bound (d06bac), see above. */
if (outfile) fclose(fpout);
nag_mesh2d_bound(nlines, coorch, lined, fbnd, coorus, nus, rate, ncomp,
                nlcomp, lcomp, nvmax, nedmx, &nvb2, coor2, &nedge2, edge2,
                itrace, outfile, &comm, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
    exit_status = 2;
    goto END;
}
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_mesh2d_bound (d06bac).\n%s\n",
            fail.message);
}

```

```

        exit_status = 1;
        goto END;
    }

/* Generate mesh using the advancing front method */

itrace = 0;

/* nag_mesh2d_front (d06acc).
 * Generates a two-dimensional mesh using an Advancing-front
 * method
 */
if (outfile) fclose(fpout);
nag_mesh2d_front(nvb2, nvint, nvmax, nedge2, edge2, &nv2, &nelt2, coor2,
                conn2, weight, itrace, outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
    {
        exit_status = 2;
        goto END;
    }
if (fail.code != NE_NOERROR)
    {
        fprintf(fpout, "Error from nag_mesh2d_front (d06acc).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
    }

for (k = 1; k <= nelt2; ++k) reft2[k - 1] = 2;

/* Rotation of the 2nd domain mesh */
/* to produce the 3rd mesh domain */

itype[0] = 3;
TRANS(1, 1) = 6.0;
TRANS(2, 1) = -1.0;
TRANS(3, 1) = -90.0;
itrace = 0;
mode = 0;

/* nag_mesh2d_trans (d06dac), see above. */
if (outfile) fclose(fpout);
nag_mesh2d_trans(mode, nv2, nedge2, nelt2, ntrans, itype, trans, coor2,
                edge2, conn2, coor3, edge3, conn3, itrace, outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
    {
        exit_status = 2;
        goto END;
    }
if (fail.code != NE_NOERROR)
    {
        fprintf(fpout, "Error from nag_mesh2d_trans (d06dac).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
    }

nv3 = nv2;
nelt3 = nelt2;
nedge3 = nedge2;

for (k = 1; k <= nelt3; ++k) reft3[k - 1] = 3;

/* Restitching meshes 1 and 2 to form mesh 4 */

eps = 0.001;
itrace = 0;

/* nag_mesh2d_join (d06dbc), see above. */
if (outfile) fclose(fpout);
nag_mesh2d_join(eps, nv1, nelt1, nedge1, coor1, edge1, conn1, reft1, nv2,

```

```

        nelt2, nedge2, coor2, edge2, conn2, reft2, &nv4, &nelt4,
        &nedge4, coor4, edge4, conn4, reft4, itrace, outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
    exit_status = 2;
    goto END;
}
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_mesh2d_join (d06dbc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}

/* Restitching meshes 3 and 4 to form mesh 5 */

itrace = 0;

/* nag_mesh2d_join (d06dbc), see above. */
if (outfile) fclose(fpout);
nag_mesh2d_join(eps, nv4, nelt4, nedge4, coor4, edge4, conn4, reft4, nv3,
               nelt3, nedge3, coor3, edge3, conn3, reft3, &nv5, &nelt5,
               &nedge5, coor5, edge5, conn5, reft5, itrace, outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
    exit_status = 2;
    goto END;
}
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_mesh2d_join (d06dbc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}

if (pmesh[0] == 'N')
{
    fprintf(fpout, "The restitched mesh characteristics\n");
    fprintf(fpout, " nv      =%6ld\n", nv5);
    fprintf(fpout, " nelt   =%6ld\n", nelt5);
    fprintf(fpout, " nedge  =%6ld\n", nedge5);
}
else if (pmesh[0] == 'Y')
{
    /* Output the mesh to view it using the NAG Graphics Library */

    fprintf(fpout, " %10ld%10ld%10ld\n", nv5, nelt5, nedge5);

    for (i = 1; i <= nv5; ++i)
        fprintf(fpout, "  %15.6e  %15.6e\n",
                COOR5(1, i), COOR5(2, i));

    for (k = 1; k <= nelt5; ++k)
        fprintf(fpout, "%10ld%10ld%10ld%10ld\n",
                CONN5(1, k), CONN5(2, k), CONN5(3, k), reft5[k - 1]);

    for (k = 1; k <= nedge5; ++k)
        fprintf(fpout, " %10ld%10ld%10ld\n",
                EDGE5(1, k), EDGE5(2, k), EDGE5(3, k));
}
else
{
    fprintf(fpout, "Problem with the printing option Y or N\n");
}

END:
if (coor1) NAG_FREE(coor1);
if (coor2) NAG_FREE(coor2);
if (coor3) NAG_FREE(coor3);

```

```

if (coor4) NAG_FREE(coor4);
if (coor5) NAG_FREE(coor5);
if (coorch) NAG_FREE(coorch);
if (coorus) NAG_FREE(coorus);
if (rate) NAG_FREE(rate);
if (trans) NAG_FREE(trans);
if (weight) NAG_FREE(weight);
if (conn1) NAG_FREE(conn1);
if (conn2) NAG_FREE(conn2);
if (conn3) NAG_FREE(conn3);
if (conn4) NAG_FREE(conn4);
if (conn5) NAG_FREE(conn5);
if (edge1) NAG_FREE(edge1);
if (edge2) NAG_FREE(edge2);
if (edge3) NAG_FREE(edge3);
if (edge4) NAG_FREE(edge4);
if (edge5) NAG_FREE(edge5);
if (itype) NAG_FREE(itype);
if (lcomp) NAG_FREE(lcomp);
if (lined) NAG_FREE(lined);
if (nlcomp) NAG_FREE(nlcomp);
if (numfix) NAG_FREE(numfix);
if (ref1) NAG_FREE(ref1);
if (ref2) NAG_FREE(ref2);
if (ref3) NAG_FREE(ref3);
if (ref4) NAG_FREE(ref4);
if (ref5) NAG_FREE(ref5);

return exit_status;
}

double NAG_CALL fbnd(Integer i, double x, double y, Nag_Comm *pcomm)
{
    double radius2, x0, y0, ret_val;

    ret_val = 0.0;
    switch (i)
    {
        case 1:

            /* inner circle */

            x0 = 0.0;
            y0 = 0.0;
            radius2 = 1.0;
            ret_val = (x-x0)*(x-x0) + (y-y0)*(y-y0) - radius2;
            break;

        case 2:

            /* outer circle */

            x0 = 0.0;
            y0 = 0.0;
            radius2 = 5.0;
            ret_val = (x-x0)*(x-x0) + (y-y0)*(y-y0) - radius2;
            break;
    }

    return ret_val;
}

```

## 9.2 Program Data

**Note 1:** since the data file for this example is quite large only a section of it is reproduced in this document. The full data file is distributed with your implementation.

D06DBF Example Program Data

Example 1

```

      400      722      :NV1 NELT1
0.000000E+00 0.000000E+00
      .
      .
0.100000E+01 0.100000E+01      :COOR1(1:2,1:NV1)
      1      2      22      0
      .
      .
      379      400      399      0 : (CONN1(:,K), REFT, K = 1,...,NELT1)
'N'      : Printing option 'Y' or 'N'

```

## 9.3 Program Results

nag\_mesh2d\_join (d06dbc) Example Program Results

Example 1

The restitched mesh characteristics  
in the overlapping case

nv = 785

nelt = 1428

nedge = 152

in the partition case

nv = 780

nelt = 1444

nedge = 133

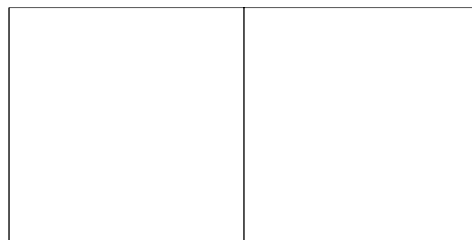
Example 2

The restitched mesh characteristics

nv = 643

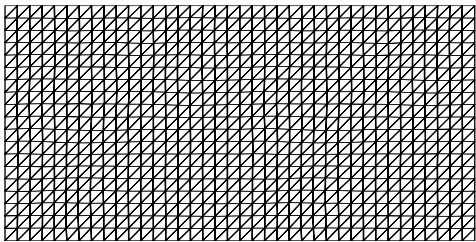
nelt = 1133

nedge = 171

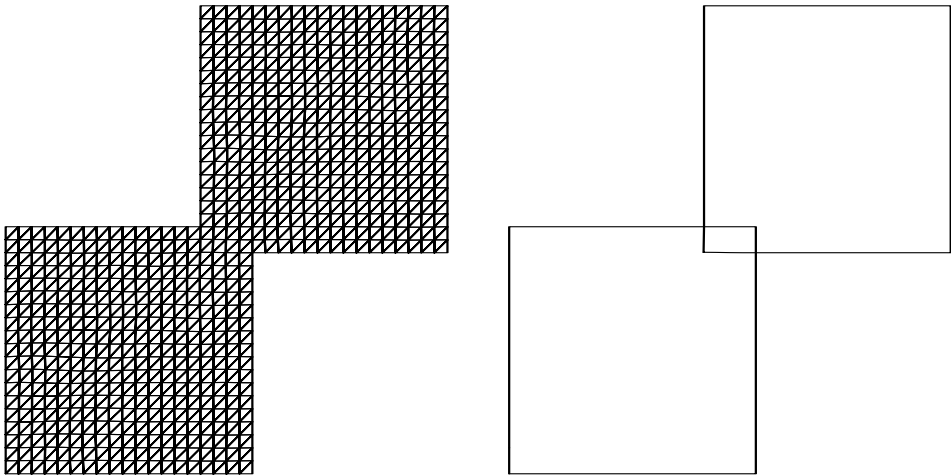


**Figure 1**

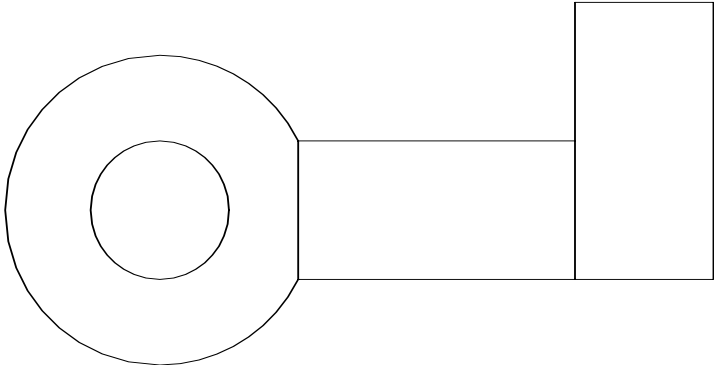
The boundary and the interior interfaces of the two partitioned squares geometry



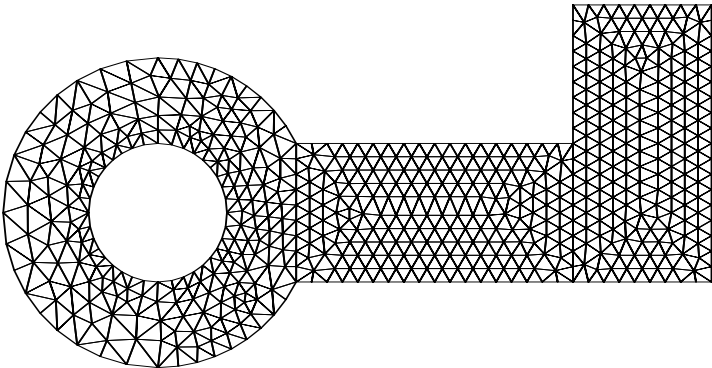
**Figure 2**  
The interior mesh of the two partitioned squares geometry



**Figure 3**  
The boundary and the interior interfaces (left); the interior mesh (right) of the two overlapping squares geometry



**Figure 4**  
The boundary and the interior interfaces of the double restitched geometry



**Figure 5**  
The interior mesh of the double restitched geometry

---