NAG Library Routine Document

D06DBF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

D06DBF joins together (restitches) two adjacent, or overlapping, meshes.

2 Specification

```
SUBROUTINE D06DBF (EPS, NV1, NELT1, NEDGE1, COOR1, EDGE1, CONN1, REFT1,    &
                   NV2, NELT2, NEDGE2, COOR2, EDGE2, CONN2, REFT2, NV3,    &
                   NELT3, NEDGE3, COOR3, EDGE3, CONN3, REFT3, ITRACE,      &
                   IWORK, LIWORK, IFAIL)
INTEGER            NV1, NELT1, NEDGE1, EDGE1(3,NEDGE1),                    &
                   CONN1(3,NELT1), REFT1(NELT1), NV2, NELT2, NEDGE2,       &
                   EDGE2(3,NEDGE2), CONN2(3,NELT2), REFT2(NELT2), NV3,     &
                   NELT3, NEDGE3, EDGE3(3,*), CONN3(3,*), REFT3(*),        &
                   ITRACE, IWORK(LIWORK), LIWORK, IFAIL
REAL (KIND=nag_wp) EPS, COOR1(2,NV1), COOR2(2,NV2), COOR3(2,*)
```

3 Description

D06DBF 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 routine 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: EPS – REAL (KIND=nag_wp) *Input*

On entry: the relative precision of the restitching of the two input meshes (see Section 9).

Suggested value: 0.001.

Constraint: EPS > 0.0.

2: NV1 – INTEGER *Input*

On entry: the total number of vertices in the first input mesh.

Constraint: NV1 ≥ 3.

3: NELT1 – INTEGER *Input*

On entry: the number of triangular elements in the first input mesh.

Constraint: NELT1 $\leq 2 \times \text{NV1} - 1$.

4: NEDGE1 – INTEGER *Input*

On entry: the number of boundary edges in the first input mesh.

Constraint: NEDGE1 ≥ 1.

5: COOR1(2, NV1) – REAL (KIND=nag_wp) array *Input*

On entry: COOR1$(1, i)$ contains the x coordinate of the ith vertex of the first input mesh, for $i = 1, 2, \ldots, \mathrm{NV1}$; while COOR1$(2, i)$ contains the corresponding y coordinate.

6: EDGE1(3, NEDGE1) – INTEGER array *Input*

On entry: 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 jth boundary edge. EDGE1$(3, j)$ is a user-supplied tag for the jth boundary edge.

Constraint: $1 \leq \mathrm{EDGE1}(i, j) \leq \mathrm{NV1}$ and $\mathrm{EDGE1}(1, j) \neq \mathrm{EDGE1}(2, j)$, for $i = 1, 2$ and $j = 1, 2, \ldots, \mathrm{NEDGE1}$.

7: CONN1(3, NELT1) – INTEGER array *Input*

On entry: 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 $j = 1, 2, \ldots, \mathrm{NELT1}$.

Constraints:

$1 \leq \mathrm{CONN1}(i, j) \leq \mathrm{NV1}$;
$\mathrm{CONN1}(1, j) \neq \mathrm{CONN1}(2, j)$;
$\mathrm{CONN1}(1, j) \neq \mathrm{CONN1}(3, j)$ and $\mathrm{CONN1}(2, j) \neq \mathrm{CONN1}(3, j)$, for $i = 1, 2, 3$ and $j = 1, 2, \ldots, \mathrm{NELT1}$.

8: REFT1(NELT1) – INTEGER array *Input*

On entry: REFT1(k) contains the user-supplied tag of the kth triangle from the first input mesh, for $k = 1, 2, \ldots, \mathrm{NELT1}$.

9: NV2 – INTEGER *Input*

On entry: the total number of vertices in the second input mesh.

Constraint: NV2 ≥ 3.

10: NELT2 – INTEGER *Input*

On entry: the number of triangular elements in the second input mesh.

Constraint: $\mathrm{NELT2} \leq 2 \times \mathrm{NV2} - 1$.

11: NEDGE2 – INTEGER *Input*

On entry: the number of boundary edges in the second input mesh.

Constraint: NEDGE2 ≥ 1.

12: COOR2(2, NV2) – REAL (KIND=nag_wp) array *Input*

On entry: COOR2$(1, i)$ contains the x coordinate of the ith vertex of the second input mesh, for $i = 1, 2, \ldots, \mathrm{NV2}$; while COOR2$(2, i)$ contains the corresponding y coordinate.

13: EDGE2(3, NEDGE2) – INTEGER array *Input*

On entry: the specification of the boundary edges of the second input mesh. $\mathrm{EDGE2}(1,j)$ and $\mathrm{EDGE2}(2,j)$ contain the vertex numbers of the two end points of the jth boundary edge. $\mathrm{EDGE2}(3,j)$ is a user-supplied tag for the jth boundary edge.

Constraint: $1 \leq \mathrm{EDGE2}(i,j) \leq \mathrm{NV2}$ and $\mathrm{EDGE2}(1,j) \neq \mathrm{EDGE2}(2,j)$, for $i = 1,2$ and $j = 1,2,\ldots,\mathrm{NEDGE2}$.

14: CONN2(3, NELT2) – INTEGER array *Input*

On entry: the connectivity between triangles and vertices of the second input mesh. For each triangle j, $\mathrm{CONN2}(i,j)$ gives the indices of its three vertices (in anticlockwise order), for $i = 1,2,3$ and $j = 1,2,\ldots,\mathrm{NELT2}$.

Constraints:

$1 \leq \mathrm{CONN2}(i,j) \leq \mathrm{NV2}$;
$\mathrm{CONN2}(1,j) \neq \mathrm{CONN2}(2,j)$;
$\mathrm{CONN2}(1,j) \neq \mathrm{CONN2}(3,j)$ and $\mathrm{CONN2}(2,j) \neq \mathrm{CONN2}(3,j)$, for $i = 1,2,3$ and $j = 1,2,\ldots,\mathrm{NELT2}$.

15: REFT2(NELT2) – INTEGER array *Input*

On entry: $\mathrm{REFT2}(k)$ contains the user-supplied tag of the kth triangle from the second input mesh, for $k = 1,2,\ldots,\mathrm{NELT2}$.

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(2, $*$) – REAL (KIND=nag_wp) array *Output*

Note: the second dimension of the array COOR3 must be at least $\mathrm{NV1} + \mathrm{NV2}$.

On exit: $\mathrm{COOR3}(1,i)$ will contain the x coordinate of the ith vertex of the resulting mesh, for $i = 1,2,\ldots,\mathrm{NV3}$; while $\mathrm{COOR3}(2,i)$ will contain the corresponding y coordinate.

20: EDGE3(3, $*$) – INTEGER array *Output*

Note: the second dimension of the array EDGE3 must be at least $\mathrm{NEDGE1} + \mathrm{NEDGE2}$. This may be reduced to NEDGE3 once that value is known.

On exit: the specification of the boundary edges of the resulting mesh. $\mathrm{EDGE3}(i,j)$ will contain the vertex number of the ith end point ($i = 1,2$) of the jth boundary or interface edge.

If the two meshes overlap, $\mathrm{EDGE3}(3,j)$ 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 jth edge is part of the partition interface, then $\mathrm{EDGE3}(3,j)$ 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, $\mathrm{EDGE3}(3,j)$ will contain the same tag as the corresponding edge belonging to the first and/or the second input mesh.

21: CONN3(3, ∗) – INTEGER array *Output*

Note: the second dimension of the array CONN3 must be at least NELT1 + 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(i,j) will give the indices of its three vertices (in anticlockwise order), for $i = 1, 2, 3$ and $j = 1, 2, \ldots,$ NELT3.

22: REFT3(∗) – INTEGER array *Output*

Note: the dimension 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) 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.

23: ITRACE – INTEGER *Input*

On entry: the level of trace information required from D06DBF.

ITRACE ≤ 0
 No output is generated.

ITRACE ≥ 1
 Details about the common vertices, edges and triangles to both meshes are printed on the current advisory message unit (see X04ABF).

24: IWORK(LIWORK) – INTEGER array *Workspace*
25: LIWORK – INTEGER *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D06DBF is called.

Constraint:
LIWORK $\geq 2 \times$ NV1 $+ 3 \times$ NV2 + NELT1 + NELT2 + NEDGE1 + NEDGE2 + 1024.

26: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL = 0 unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL = 1

On entry, EPS ≤ 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 routine 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 routine. 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 routine 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 routine 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.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

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

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

D06DBF is not threaded in any implementation.

9 Further Comments

D06DBF 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 ≥ 1 to get the optimal value for the meshes under consideration.

10 Example

For this routine two examples are presented. There is a single example program for D06DBF, 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 D06DAF). 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 partitioned case the resulting geometry is shown in Figure 1 in Section 10.3 while the restitched mesh is shown in Figure 2 in Section 10.3. In the overlapping case the geometry and mesh are shown in Figure 3 and Figure 4 in Section 10.3.

Example 2 (EX2)

This example restitches three geometries by calling the routine D06DBF twice. The result is a mesh with three partitions. The first geometry is meshed by the Delaunay–Voronoi process (using D06ABF), the second one meshed by an Advancing Front algorithm (using D06ACF), while the third one is the result of a rotation (by $-\pi/2$) of the second one (using D06DAF). The resulting geometry is shown in Figure 5 in Section 10.3 and restitched mesh in Figure 6 in Section 10.3.

10.1 Program Text

```
!   D06DBF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.
    Module d06dbfe_mod

!     D06DBF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                           :: fbnd
!     .. Parameters ..
      Integer, Parameter, Public       :: meshout = 7, nin = 5, nout = 6,      &
                                          nvb1 = 19
      Logical, Parameter, Public       :: pmesh = .False.
    Contains
```

```
      Function fbnd(i,x,y,ruser,iuser)

!       .. Function Return Value ..
        Real (Kind=nag_wp)             :: fbnd
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: x, y
        Integer, Intent (In)           :: i
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: ruser(*)
        Integer, Intent (Inout)        :: iuser(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: radius2, x0, y0
!       .. Executable Statements ..
        fbnd = 0.0_nag_wp

        Select Case (i)
        Case (1)

!         inner circle

          x0 = 0.0_nag_wp
          y0 = 0.0_nag_wp
          radius2 = 1.0_nag_wp
          fbnd = (x-x0)**2 + (y-y0)**2 - radius2
        Case (2)

!         outer circle

          x0 = 0.0_nag_wp
          y0 = 0.0_nag_wp
          radius2 = 5.0_nag_wp
          fbnd = (x-x0)**2 + (y-y0)**2 - radius2
        End Select

        Return

      End Function fbnd
    End Module d06dbfe_mod
    Program d06dbfe

!     D06DBF Example Main Program

!     .. Use Statements ..
      Use d06dbfe_mod, Only: nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Executable Statements ..
      Write (nout,*) 'D06DBF Example Program Results'

      Call ex1

      Call ex2

    Contains
      Subroutine ex1

!       .. Use Statements ..
        Use nag_library, Only: d06daf, d06dbf, nag_wp, x01aaf
        Use d06dbfe_mod, Only: meshout, nout, nvb1, pmesh
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: ddx, ddy, dx, eps, pi2, pi4, r_i,    &
                                          r_j
        Integer                        :: i, ifail, imax, itrace, itrans, j,   &
                                          jmax, jtrans, k, ktrans, l, liwork,  &
                                          lrwork, nedge1, nedge2, nedge3,      &
                                          nelt1, nelt2, nelt3, ntrans, nv1,    &
                                          nv2, nv3
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable :: coor1(:,:), coor2(:,:), coor3(:,:), &
                                          rwork(:), trans(:,:)
        Integer, Allocatable           :: conn1(:,:), conn2(:,:), conn3(:,:),  &
```

```
                                    edge1(:,:), edge2(:,:), edge3(:,:),  &
                                    itype(:), iwork(:), reft1(:),        &
                                    reft2(:), reft3(:)
!       .. Intrinsic Procedures ..
        Intrinsic                :: real, sin
!       .. Executable Statements ..
        Write (nout,*)
        Write (nout,*) 'Example 1'
        Write (nout,*)

        imax = nvb1 + 1
        jmax = imax
        nedge1 = 2*(imax-1) + 2*(jmax-1)
        nedge2 = nedge1
        nedge3 = nedge1 + nedge2
        ntrans = 1
        lrwork = 12*ntrans

!       Allocate for mesh : coordinates and connectivity of the 1st domain,
!                           2nd translated domain and restitched domain.

        nv1 = (nvb1+1)**2
        nelt1 = 2*nvb1*nvb1
        nv2 = nv1
        nv3 = nv1 + nv2
        nelt2 = nelt1
        nelt3 = nelt1 + nelt2
        liwork = 2*nv1 + 3*nv2 + nelt1 + nelt2 + nedge1 + nedge2 + 1024
        Allocate (coor1(2,nv1),coor2(2,nv2),coor3(2,nv3),conn1(3,nelt1),       &
          conn2(3,nelt2),conn3(3,nelt3),reft1(nelt1),reft2(nelt2),             &
          reft3(nelt3),edge1(3,nedge1),edge2(3,nedge2),edge3(3,nedge3),        &
          itype(ntrans),trans(6,ntrans),rwork(lrwork),iwork(liwork))

!       Set up interior mesh as small perturbations on regular grid
!       with regularity on the boundary of square on [0,1]x[0,1].
        dx = 1.0_nag_wp/real(nvb1,kind=nag_wp)
        pi2 = x01aaf(dx) + x01aaf(dx)
        pi4 = pi2 + pi2
        k = 0
        Do j = 1, jmax
          ddy = real(j-1,kind=nag_wp)*dx
          Do i = 1, imax
            k = k + 1
            ddx = real(i-1,kind=nag_wp)*dx
            coor1(1,k) = ddx + dx*0.05_nag_wp*sin(pi4*ddx)*sin(pi4*ddy)
            coor1(2,k) = ddy + dx*0.05_nag_wp*sin(pi2*ddx)*sin(pi2*ddy)
          End Do
        End Do

!       Triangulate using skew-diagonals on grid squares
        k = 0
        l = 0
        Do i = 1, nvb1
          Do j = 1, nvb1
            l = l + 1
            k = k + 1
            conn1(1,k) = l
            conn1(2,k) = l + 1
            conn1(3,k) = l + nvb1 + 2
            k = k + 1
            conn1(1,k) = l
            conn1(2,k) = l + nvb1 + 2
            conn1(3,k) = l + nvb1 + 1
          End Do
          l = l + 1
        End Do

        reft1(1:nelt1) = 1
        reft2(1:nelt2) = 2

!       Define the edges of the boundary
```

```
        Do i = 1, nvb1
          edge1(1,i) = i
          edge1(2,i) = i + 1
        End Do
        Do i = 1, nvb1
          edge1(1,nvb1+i) = i*imax
          edge1(2,nvb1+i) = (i+1)*imax
        End Do
        Do i = 1, nvb1
          edge1(1,2*nvb1+i) = imax*jmax - i + 1
          edge1(2,2*nvb1+i) = imax*jmax - i
        End Do
        Do i = 1, nvb1
          edge1(1,3*nvb1+i) = (jmax-i)*imax + 1
          edge1(2,3*nvb1+i) = (jmax-i-1)*imax + 1
        End Do
        edge1(3,1:nedge1) = 1

        If (pmesh) Then
!         Print interior mesh of single square
          Do i = 1, nelt1
            Write (meshout,99997) coor1(1,conn1(1,i)), coor1(2,conn1(1,i))
            Write (meshout,99997) coor1(1,conn1(2,i)), coor1(2,conn1(2,i))
            Write (meshout,99997) coor1(1,conn1(3,i)), coor1(2,conn1(3,i))
            Write (meshout,99997) coor1(1,conn1(1,i)), coor1(2,conn1(1,i))
            Write (meshout,*)
          End Do
          Write (meshout,*)
        End If

        Do ktrans = 1, 2

!         Translation of the 1st domain to obtain the 2nd domain
!         KTRANS = 1 leading to a domains (4x2) overlapping
!         KTRANS = 2 leading to a domains partition

          If (ktrans==1) Then
            itrans = nvb1 - 4
            jtrans = nvb1 - 2
          Else
            itrans = nvb1
            jtrans = 0
          End If

          itype(1:ntrans) = (/1/)
          r_i = real(itrans,kind=nag_wp)/real(nvb1,kind=nag_wp)
          r_j = real(jtrans,kind=nag_wp)/real(nvb1,kind=nag_wp)
          trans(1,1:ntrans) = (/r_i/)
          trans(2,1:ntrans) = (/r_j/)
          itrace = 0

          ifail = 0
          Call d06daf(nv2,nedge2,nelt2,ntrans,itype,trans,coor1,edge1,conn1,   &
            coor2,edge2,conn2,itrace,rwork,lrwork,ifail)

          edge2(3,1:nedge2) = 2

!         Call to the restitching driver

          itrace = 0
          eps = 1.E-2_nag_wp

          ifail = 0
          Call d06dbf(eps,nv1,nelt1,nedge1,coor1,edge1,conn1,reft1,nv2,nelt2,  &
            nedge2,coor2,edge2,conn2,reft2,nv3,nelt3,nedge3,coor3,edge3,conn3, &
            reft3,itrace,iwork,liwork,ifail)

          If (pmesh) Then

!           Output the overlapping or partitioned mesh
```

```
          Write (nout,99998) nv3, nelt3, nedge3

          Do i = 1, nelt3
            Write (meshout,99997) coor3(1,conn3(1,i)), coor3(2,conn3(1,i))
            Write (meshout,99997) coor3(1,conn3(2,i)), coor3(2,conn3(2,i))
            Write (meshout,99997) coor3(1,conn3(3,i)), coor3(2,conn3(3,i))
            Write (meshout,99997) coor3(1,conn3(1,i)), coor3(2,conn3(1,i))
            Write (meshout,*)
          End Do
          Write (meshout,*)

          Do k = 1, nelt3
            Write (nout,99996) conn3(1:3,k), reft3(k)
          End Do

          Do k = 1, nedge3
            Write (nout,99998) edge3(1:3,k)
          End Do
        Else

          If (ktrans==1) Then
            Write (nout,*)                                                 &
              'The restitched mesh characteristics in the overlapping case'
          Else
            Write (nout,*)                                                 &
              'The restitched mesh characteristics in the partition case'
          End If

          Write (nout,99999) 'nv    =', nv3
          Write (nout,99999) 'nelt  =', nelt3
          Write (nout,99999) 'nedge =', nedge3
        End If
      End Do

99999 Format (1X,A,I6)
99998 Format (1X,3I10)
99997 Format (2(2X,E13.6))
99996 Format (1X,4I10)
    End Subroutine ex1
    Subroutine ex2

!     .. Use Statements ..
      Use nag_library, Only: d06abf, d06acf, d06baf, d06caf, d06daf, d06dbf, &
                             f16dnf, nag_wp
      Use d06dbfe_mod, Only: fbnd, meshout, nin, nout, pmesh
!     .. Local Scalars ..
      Real (Kind=nag_wp)              :: eps
      Integer                         :: i, ifail, itrace, j, k, liwork,      &
                                         lrwork, maxind, maxval, ncomp,       &
                                         nedge1, nedge2, nedge3, nedge4,      &
                                         nedge5, nedmx, nelt1, nelt2, nelt3,  &
                                         nelt4, nelt5, nlines, npropa, nqint, &
                                         ntrans, nv1, nv2, nv3, nv4, nv5,     &
                                         nvb1, nvb2, nvfix, nvint, nvmax,     &
                                         sdcrus
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable :: coor1(:,:), coor2(:,:), coor3(:,:), &
                                         coor4(:,:), coor5(:,:), coorch(:,:), &
                                         crus(:,:), rate(:), rwork(:),        &
                                         trans(:,:), weight(:)
      Real (Kind=nag_wp)              :: ruser(1)
      Integer, Allocatable            :: conn1(:,:), conn2(:,:), conn3(:,:),  &
                                         conn4(:,:), conn5(:,:), edge1(:,:),  &
                                         edge2(:,:), edge3(:,:), edge4(:,:),  &
                                         edge5(:,:), itype(:), iwork(:),      &
                                         lcomp(:), lined(:,:), nlcomp(:),     &
                                         numfix(:), reft1(:), reft2(:),       &
                                         reft3(:), reft4(:), reft5(:)
      Integer                         :: iuser(1)
!     .. Intrinsic Procedures ..
      Intrinsic                       :: abs
```

```
!       .. Executable Statements ..
        Write (nout,*)
        Write (nout,*) 'Example 2'
        Write (nout,*)

!       Skip headings in data file
        Read (nin,*)
        Read (nin,*)

!       Build and mesh two domains and rotate/translate second to create
!       third.

!       ----------------------------------------------
!       1st domain: Annulus with straight right edge.
!         First two points are end of straight edge.
!         The mesh for inner circle is defined by four NWSE points with mesh
!             points on the quarter-circle between each pair (fbnd, i=1).
!         The mesh for the incomplete outer circle  is defined by three NWS
!             points and the edge points, mesh points computed using fbnd,
!             i=2.

!       The number of lines (1+4+4).
        Read (nin,*) nlines, nvmax, nedmx
        Allocate (coor1(2,nvmax),edge1(3,nedmx),lined(4,nlines),lcomp(nlines), &
          coorch(2,nlines),rate(nlines))

!       Characteristic points of the boundary geometry.
        Read (nin,*) coorch(1,1:nlines)
        Read (nin,*) coorch(2,1:nlines)

!       The Lines of the boundary mesh
        Read (nin,*)(lined(1:4,j),rate(j),j=1,nlines)

!       Allocate workspace for d06baf.

!       sdcrus = 0 in this case.
        sdcrus = 0
        Do i = 1, nlines
          If (lined(4,i)<0) Then
            sdcrus = sdcrus + lined(1,i) - 2
          End If
        End Do
!       Get max(LINED(1,:)) for computing lrwork
        Call f16dnf(nlines,lined,4,maxind,maxval)
        liwork = 8*nlines + nvmax + 3*nedmx + 3*sdcrus
        lrwork = 2*(nlines+sdcrus) + 2*maxval*nlines
        Allocate (crus(2,sdcrus),rwork(lrwork),iwork(liwork))

!       The number of connected components (outer circle/edge and inner
!       circle)
        Read (nin,*) ncomp
        Allocate (nlcomp(ncomp))

!       Read the lines comprising each connected component
        j = 1
        Do i = 1, ncomp
          Read (nin,*) nlcomp(i)
          k = j + abs(nlcomp(i)) - 1
          Read (nin,*) lcomp(j:k)
          j = k + 1
        End Do

        itrace = 0
!       Call to the 2D boundary mesh generator
        ifail = 0
        Call d06baf(nlines,coorch,lined,fbnd,crus,sdcrus,rate,ncomp,nlcomp,    &
          lcomp,nvmax,nedmx,nvb1,coor1,nedge1,edge1,itrace,ruser,iuser,rwork,  &
          lrwork,iwork,liwork,ifail)

        Deallocate (rwork,iwork)
```

```
!       Generate mesh using Delaunay-Voronoi method

!       Initialize mesh control parameters and allocate workspace.
        itrace = 0
        npropa = 1
        nvint = 0
        lrwork = 12*nvmax + 15
        liwork = 6*nedge1 + 32*nvmax + 2*nvb1 + 78
        Allocate (weight(nvint),rwork(lrwork),iwork(liwork),                  &
          conn1(3,2*nvmax+5))

!       Call to the 2D Delaunay-Voronoi mesh generator
        ifail = 0
        Call d06abf(nvb1,nvint,nvmax,nedge1,edge1,nv1,nelt1,coor1,conn1,       &
          weight,npropa,itrace,rwork,lrwork,iwork,liwork,ifail)

        Deallocate (rwork,iwork)

!       Call the smoothing routine
        nvfix = 0
        nqint = 10
        lrwork = 2*nv1 + nelt1
        liwork = 8*nelt1 + 2*nv1
        Allocate (numfix(nvfix),rwork(lrwork),iwork(liwork))
        ifail = 0
        Call d06caf(nv1,nelt1,nedge1,coor1,edge1,conn1,nvfix,numfix,itrace,    &
          nqint,iwork,liwork,rwork,lrwork,ifail)

        Deallocate (rwork,iwork,coorch,lined,lcomp,rate,nlcomp,crus)

!       ----------------------------------------------
!       2nd domain: a rectangle (4 by 2) abutting straight edge of 1st domain.

        Read (nin,*) nlines
        Allocate (lined(4,nlines),lcomp(nlines),coorch(2,nlines),rate(nlines), &
          coor2(2,nvmax),edge2(3,nedmx))

!       Characteristic points of the boundary geometry
        Read (nin,*) coorch(1,1:nlines)
        Read (nin,*) coorch(2,1:nlines)

!       The Lines of the boundary mesh
        Read (nin,*)(lined(1:4,j),rate(j),j=1,nlines)

!       sdcrus = 0 again here.
        sdcrus = 0
        Do i = 1, nlines
          If (lined(4,i)<0) Then
            sdcrus = sdcrus + lined(1,i) - 2
          End If
        End Do

!       Generate initial mesh using Delaunay-Voronoi method

        liwork = 8*nlines + nvmax + 3*nedmx + 3*sdcrus
        Call f16dnf(nlines,lined,4,maxind,maxval)
        lrwork = 2*(nlines+sdcrus) + 2*maxval*nlines

        Allocate (crus(2,sdcrus),rwork(lrwork),iwork(liwork))

!       The number of connected components (1 rectangle)
        Read (nin,*) ncomp
        Allocate (nlcomp(ncomp))

!       Four lines of rectangle
        j = 1
        Do i = 1, ncomp
          Read (nin,*) nlcomp(i)
          k = j + abs(nlcomp(i)) - 1
          Read (nin,*) lcomp(j:k)
          j = k + 1
```

```
      End Do

      itrace = 0
!     Call to the 2D boundary mesh generator
      ifail = 0
      Call d06baf(nlines,coorch,lined,fbnd,crus,sdcrus,rate,ncomp,nlcomp,    &
        lcomp,nvmax,nedmx,nvb2,coor2,nedge2,edge2,itrace,ruser,iuser,rwork,  &
        lrwork,iwork,liwork,ifail)

      Deallocate (rwork,iwork,weight)

!     Remesh 2nd domain using the advancing front method

!     Initialize mesh control parameters
      itrace = 0
      nvint = 0
      lrwork = 12*nvmax + 30015
      liwork = 8*nedge2 + 53*nvmax + 2*nvb2 + 10078
      Allocate (weight(nvint),rwork(lrwork),iwork(liwork),                   &
        conn2(3,2*nvmax+5))

!     Call to the 2D Advancing front mesh generator
      ifail = 0
      Call d06acf(nvb2,nvint,nvmax,nedge2,edge2,nv2,nelt2,coor2,conn2,       &
        weight,itrace,rwork,lrwork,iwork,liwork,ifail)

      Deallocate (rwork,iwork)

!     ----------------------------------------------
!     3rd domain: rotation and translation of the 2nd domain mesh

      ntrans = 1
      lrwork = 12*ntrans
      Allocate (rwork(lrwork),itype(ntrans),trans(6,ntrans),coor3(2,nv2),    &
        edge3(3,nedge2),conn3(3,nelt2))

      itype(1:ntrans) = (/3/)
      trans(1,1:ntrans) = (/6.0_nag_wp/)
      trans(2,1:ntrans) = (/-1.0_nag_wp/)
      trans(3,1:ntrans) = (/-90.0_nag_wp/)
      itrace = 0

      ifail = 0
      Call d06daf(nv2,nedge2,nelt2,ntrans,itype,trans,coor2,edge2,conn2,     &
        coor3,edge3,conn3,itrace,rwork,lrwork,ifail)

      Deallocate (rwork)

!     ----------------------------------------------
!     Combine Meshes

      nv3 = nv2
      nelt3 = nelt2
      nedge3 = nedge2

      nv4 = nv1 + nv2
      nelt4 = nelt1 + nelt2
      nedge4 = nedge1 + nedge2
      liwork = 2*nv1 + 3*nv2 + nelt1 + nelt2 + nedge1 + nedge2 + 1024
      Allocate (iwork(liwork),coor4(2,nv4),edge4(3,nedge4),conn4(3,nelt4),   &
        reft4(nelt4),reft1(nelt1),reft2(nelt2))

!     Restitching of the mesh 1 and 2 to form the mesh 4
      reft1(1:nelt1) = 1
      reft2(1:nelt2) = 2
      eps = 1.E-3_nag_wp
      itrace = 0
      ifail = 0
      Call d06dbf(eps,nv1,nelt1,nedge1,coor1,edge1,conn1,reft1,nv2,nelt2,    &
        nedge2,coor2,edge2,conn2,reft2,nv4,nelt4,nedge4,coor4,edge4,conn4,   &
        reft4,itrace,iwork,liwork,ifail)
```

```
        Deallocate (iwork)

        nv5 = nv4 + nv3
        nelt5 = nelt4 + nelt3
        nedge5 = nedge4 + nedge3
        liwork = 2*nv4 + 3*nv3 + nelt4 + nelt3 + nedge4 + nedge3 + 1024
        Allocate (iwork(liwork),coor5(2,nv5),edge5(3,nedge5),conn5(3,nelt5),   &
          reft5(nelt5),reft3(nelt3))

!       Restitching of the mesh 3 and 4 to form the mesh 5
        reft3(1:nelt3) = 3
        itrace = 0
        ifail = 0
        Call d06dbf(eps,nv4,nelt4,nedge4,coor4,edge4,conn4,reft4,nv3,nelt3,    &
          nedge3,coor3,edge3,conn3,reft3,nv5,nelt5,nedge5,coor5,edge5,conn5,   &
          reft5,itrace,iwork,liwork,ifail)

        If (pmesh) Then
!         Output the mesh
          Do i = 1, nelt5
            Write (meshout,99997) coor5(1,conn5(1,i)), coor5(2,conn5(1,i))
            Write (meshout,99997) coor5(1,conn5(2,i)), coor5(2,conn5(2,i))
            Write (meshout,99997) coor5(1,conn5(3,i)), coor5(2,conn5(3,i))
            Write (meshout,99997) coor5(1,conn5(1,i)), coor5(2,conn5(1,i))
            Write (meshout,*)
          End Do
          Write (meshout,*)

          Write (nout,99998) nv5, nelt5, nedge5
          Do i = 1, nv5
            Write (nout,99997) coor5(1:2,i)
          End Do
          Do k = 1, nelt5
            Write (nout,99996) conn5(1,k), conn5(2,k), conn5(3,k), reft5(k)
          End Do
          Do k = 1, nedge5
            Write (nout,99998) edge5(1:3,k)
          End Do
        Else
          Write (nout,*) 'The restitched mesh characteristics'
          Write (nout,99999) 'nv    =', nv5
          Write (nout,99999) 'nelt  =', nelt5
          Write (nout,99999) 'nedge =', nedge5
        End If

99999   Format (1X,A,I6)
99998   Format (1X,3I10)
99997   Format (2(2X,E13.6))
99996   Format (1X,4I10)
      End Subroutine ex2
    End Program d06dbfe
```

10.2 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 2
  9      700      200                  : 1st geometry nlines(m), nvmax, nedmx
  2.0000   2.0000   1.0000
 -1.0000  -2.2361   0.0000
  0.0000   0.0000   0.0000             : coorch(1,1:m)
 -1.0000   1.0000   0.0000
  0.0000   0.0000  -2.2361
 -1.0000   1.0000   2.2361             : coorch(2,1:m)
 10        1        2        0     1.0
 10        2        9        2     1.0
 10        9        5        2     1.0
```

```
10        5        6        2     1.0
10        6        1        2     1.0
10        3        8        1     1.0
10        8        4        1     1.0
10        4        7        1     1.0
10        7        3        1     1.0 : line(1:4,j),rate(j), j=1,m
 2                                    : contours (outer+inner)
 5                                    :  lines in contour 1: outer
 1        2        3        4     5
-4                                    :  lines in contour 2: inner
 9        8        7        6

 4                                    : 2nd geometry nlines(m)
 2.0      6.0      6.0      2.0       : coorch(1,1:m)
-1.0     -1.0      1.0      1.0       : coorch(2,1:m)
19        1        2        0     1.0
10        2        3        0     1.0
19        3        4        0     1.0
10        4        1        0     1.0 : line(1:4,j),rate(j), j=1,m
 1                                    : contours (rectangle)
 4                                    :  lines in contour 1
 1        2        3        4
```

10.3 Program Results

```
D06DBF Example Program Results

Example 1

The restitched mesh characteristics in the overlapping case
nv    =   785
nelt  =  1428
nedge =   152
The restitched mesh characteristics in the partition case
nv    =   780
nelt  =  1444
nedge =   133

Example 2

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

Example Program
Figure 1: Boundary and Interior Interface of Partitioned Squares

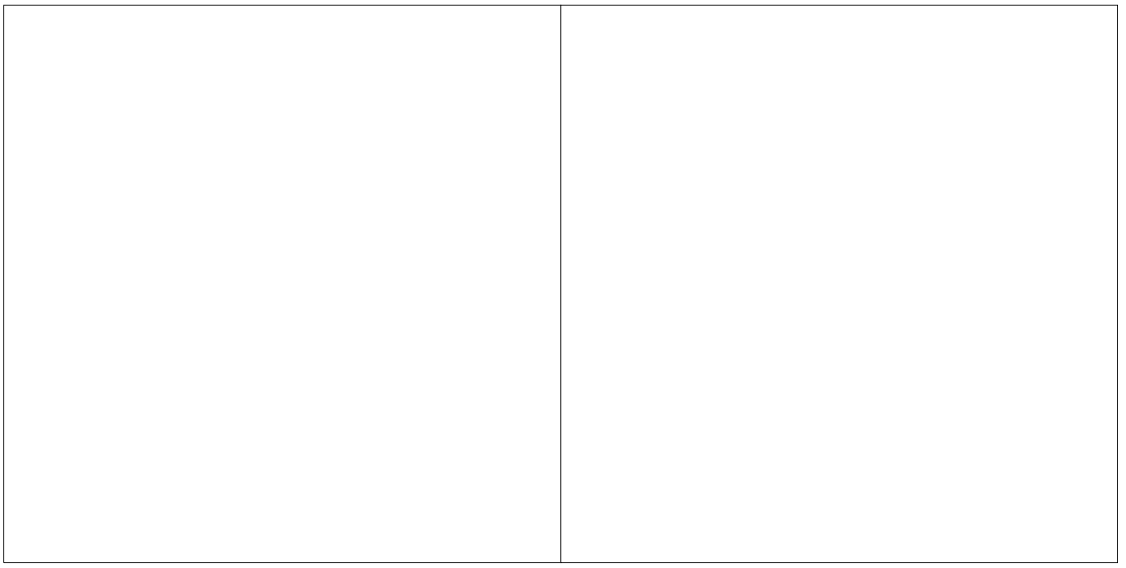

Figure 2: Interior Mesh of Partitioned Squares

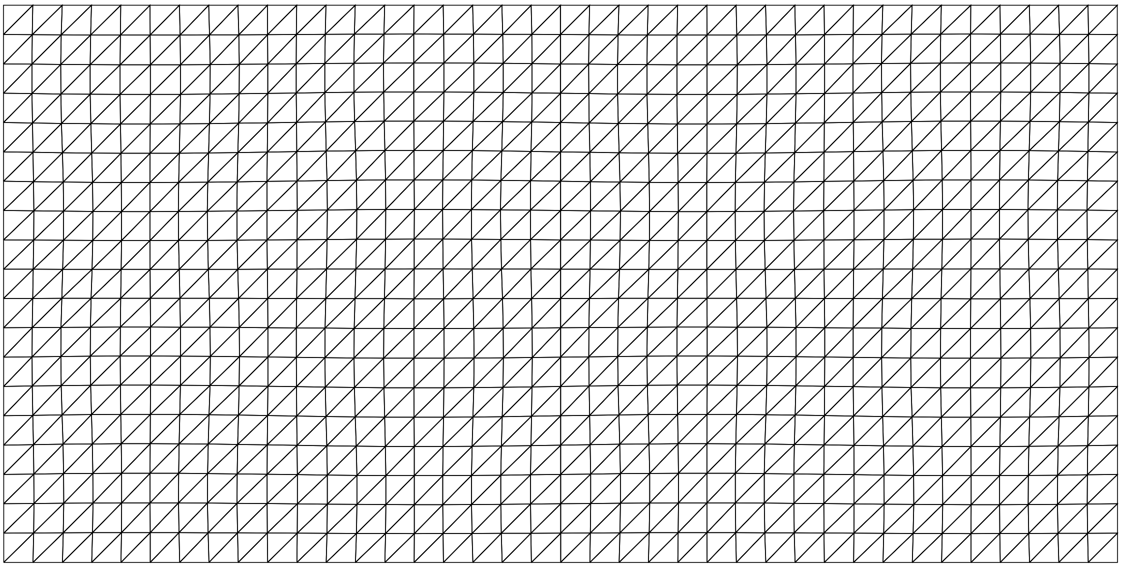

Figure 3: Boundary and Interior Interface of Overlapping Squares

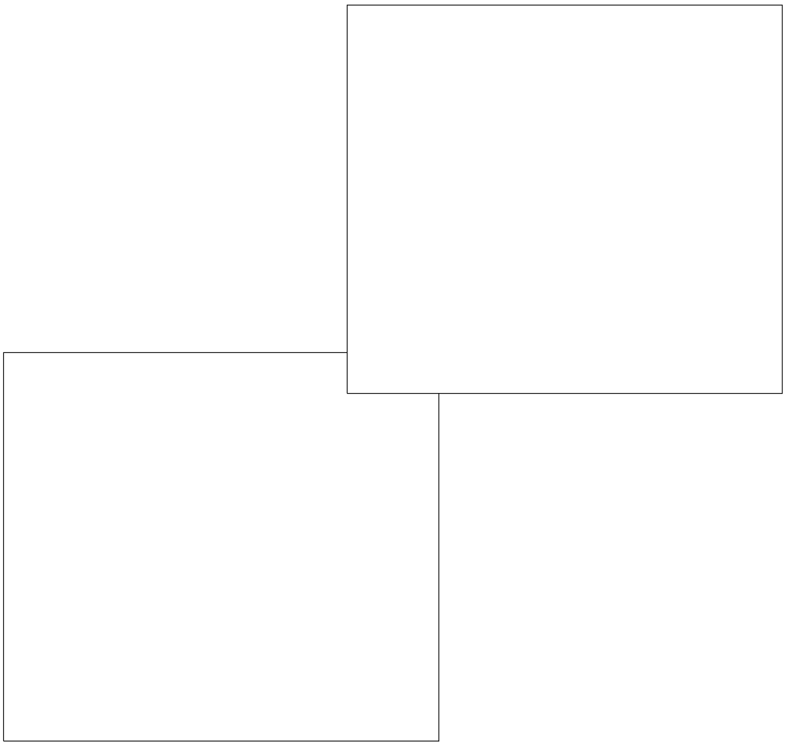

Figure 4: Interior Mesh of Overlapping Squares

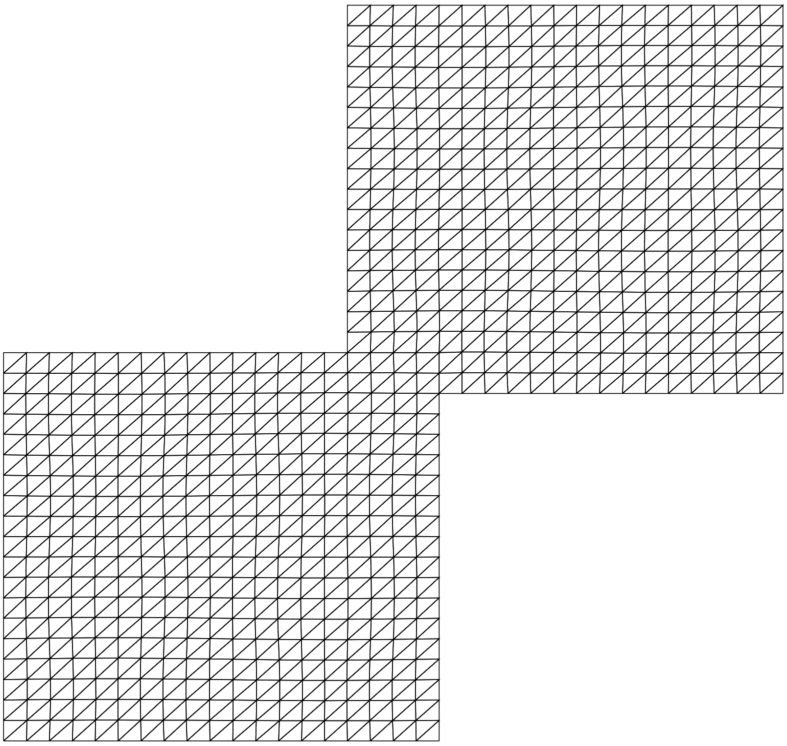

Figure 5: Boundary and Interior Interfaces for Key Shape

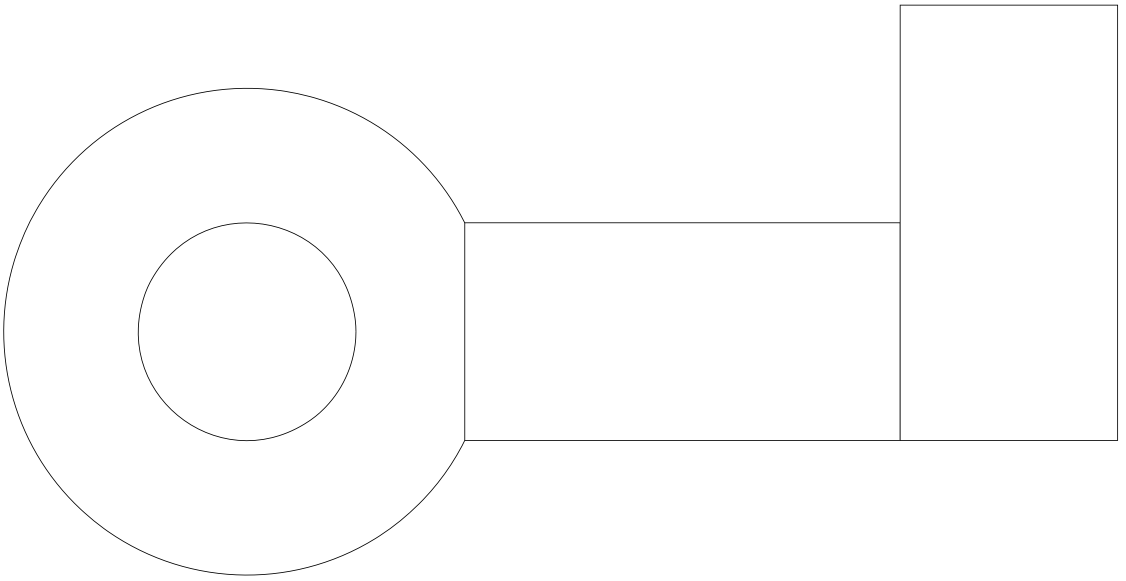

Figure 6: Interior Mesh of KeyShape