

**Ani2D Version 2.0**

**Ani2D-AFT: Flexible Mesh Generator  
Using Advanced Front Technique**

**User's Guide for libaft2D.a  
February 10, 2007**

1997-2007

## 1 Basic features of the library

The C package Ani2D-AFT is a part of the package Ani2D. Ani2D-AFT was developed by Alexander Danilov under the supervision of Yuri Vassilevski. It is designated for generating triangular meshes in arbitrary 2D domains.

The basic features of library *libaft2D.a* are listed below.

**Domain type** : single or multiple component, simply or multiply connected finite domains

**Boundary type** : piece-wise smooth

**Domain data input** : set of linear/curvilinear intervals representing the boundary, or the boundary mesh

**Number of mesh elements** : non-limited

**Data format** : double precision (for real values) or integer (for integer numbers) arrays. Enumeration starts from 1.

## 2 Analytical representation of the boundary

```
external userboundary
call registeruserfn( userboundary )
...
call aft2dboundary( Nbv, bv, Nbl, bl, bltail, h,
&          nv, vrt,
&          nt, tri, material,
&          nb, bnd,
&          nc, crv, iFNC )
```

### Input

The piece-wise smooth boundary is represented in terms of the union of a finite number of intervals. Each interval is a smooth curve. It may be split into several subcurves. End points of the intervals are denoted as V-points. Each simply connected subdomain (component of the domain) has a boundary composed of the given intervals. The input domain is specified by three arrays. Array *bv* describes all V-points, whereas arrays *bl*, *bltail* describe the intervals. *Nbv* is the number of V-points, *i*th pair of the array *bv*(2,*Nbv*) are coordinates of the *i*th V-point,  $i = 1, \dots, Nbv$ . *Nbl* is the number of intervals, *i*th subset with 7 elements of the integer array *bl*(7,*Nbl*) describes the *i*th interval,  $i = 1, \dots, Nbl$ . Each integer from the 7-element subset specifies:

1. The index of the V-point at which the interval begins
2. The index of the V-point at which the interval terminates

3. It is 0 if the interval is linear, otherwise it is a natural number defining the type of parametrization in the user defined function `userboundary` in the file `crv_model.c`.
4. A dummy integer.
5. The label of the interval. It does not affect the mesh generation. All the mesh edges from the interval will inherit this number.
6. The index of the subdomain, for which the interval is a boundary part.
7. The slit marker. It is 0 if the interval is the outer part of the domain boundary, otherwise, it is the index of the other subdomain sharing the interval with the subdomain with the 6th element index.

The  $i$ th pair of the array `bltail(2,Nbl)` is the two zeros if the interval is linear, otherwise the pair contains two parameters corresponding the starting point and the terminal point of the interval. The parameter passed to the interval parametrization provides the actual coordinates of the V-point.

The rules for interval specification are as follows:

1. When moving along the interval from the starting point to the terminal point, the subdomain is located at the right. In other terms, the intervals are given clock-wise.
2. For slit intervals, the subdomain with the 6th element index must be located at the right.
3. For slit intervals where both sharing subdomains coincide, the order of the V-points is arbitrary.
4. Coordinates of V-points whose coordinates are defined in parametric functions using the data of `bltail(2,Nbl)`, may be different from the respective entries of the array `bv`. The latter entries are dummy in this case.

Boundary parametrization is to be defined by the user in the file `crv_model.c`. The name of the user routine must be registered in the library before the call of `aft2dboundary`:

```
external userboundary
call registeruserfn( userboundary )
...
call aft2dboundary( ... )
```

Here a function `userboundary` is registered and used in meshing the boundary. An example of the user defined function describing the complement of a wing to the unit square is attached (`crv_model.c`).

The generator produces a quasi-uniform mesh of the given mesh size `h`.

## Output

The number of mesh nodes is  $nv$ , their Cartesian coordinates are stored in the array  $vrt(2,*)$ . The number of mesh triangles is  $nt$ , the connectivity list of triangles is stored in the array  $tri(3,*)$ , the triangle materials (labels) are  $material(*)$ . The number of mesh boundary edges is  $nb$ . The first two columns of  $bnd(4,*)$  are node indexes of the boundary edges. The third column of  $bnd(4,*)$  is the parametrization identifier: 0 for non-parametrized edge (linear segment),  $n > 0$  for parametrized edge with parametrization data in  $crv(*,n)$ ,  $iFNC(n)$ . The fourth column of  $bnd(4,*)$  is a boundary label/identifier. The number of curved (parametrized) boundary edges is  $nc$ , their parametrization is stored in the respective two entries of  $crv(2,*)$  and an entry of  $iFNC(*)$ . The  $j$ th parametrized edge has a parameter for its starting point  $crv(1,j)$  and a parameter for its terminal point  $crv(2,j)$ , as well as an identifier  $iFNC(j)$  of the function to be used in the computation of Cartesian coordinates of interior points of the edge.

### 3 Grid representation of the boundary

```

call aft2dfront(
&          Nbr, brd, Nvr, vbr,
&          nv, vrt,
&          nt, tri, material,
&          nb, bnd)

```

#### Input

The domain boundary is represented in terms of its discretization. Total number of the mesh edges in the boundary mesh is denoted by  $Nbr$ , the number of nodes is denoted by  $Nvr$ . The  $i$ th pair of the array  $vbr(2,Nvr)$  contains the coordinates of  $i$ th boundary node. The  $i$ th pair of the array  $brd(2,Nbr)$  contains the node indexes of the starting and terminal points of the edge.

There are two possibilities to represent the boundary mesh.

The first method ( $Nbr > 0$ ). The boundary nodes and edges are stored in an arbitrary order. While passing along an edge from the starting point to the terminal point, the subdomain must be located at the right.

The second method ( $Nbr = 0$ ). Only boundary nodes are used to represent the boundary, but their order is important. In such a case,  $brd$  is dummy. The rules for the boundary grid specification are as follows:

1. The boundary is a union of loops. The loops are stored in  $vbr$  in the sequential order.
2. In each loop the first node and the last node must be identical. This fact is used to determine different loops.
3. When passing to the next node within one loop, the subdomain has to be located at the right.
4. Loops can touch each other in any place, in this case the shared node(s) must be mentioned in each loop.

The resulting mesh will have the trace at the boundary matching to the boundary grid `vbr`, `brd`. The mesh size is governed by the distance to the boundary: the farther from the boundary, the coarser mesh is.

Once the boundary discretization is defined, there is no need in a user defined function of boundary parametrization. No function must be written and registered in the library.

Two illustrative examples of using the initial front data may be found in directory `examples`.

### **Output**

The number of mesh nodes is `nv`, their Cartesian coordinates are stored in the array `vrt(2,*)`. The number of mesh triangles is `nt`, the connectivity list of triangles is stored in the array `tri(3,*)`, the triangle materials (labels) are `material(*)`. The number of mesh boundary edges is `nb`. The first two columns of `bnd(4,*)` are node indexes of the boundary edges. The third column of `bnd(4,*)` is dummy. The fourth column of `bnd(4,*)` is a boundary label/identifier.