MacMesh for *Salome* user manual

The multi-purpose *Salome* plug-in for regular 2D quadrangle meshing

v. 10 avril 2012

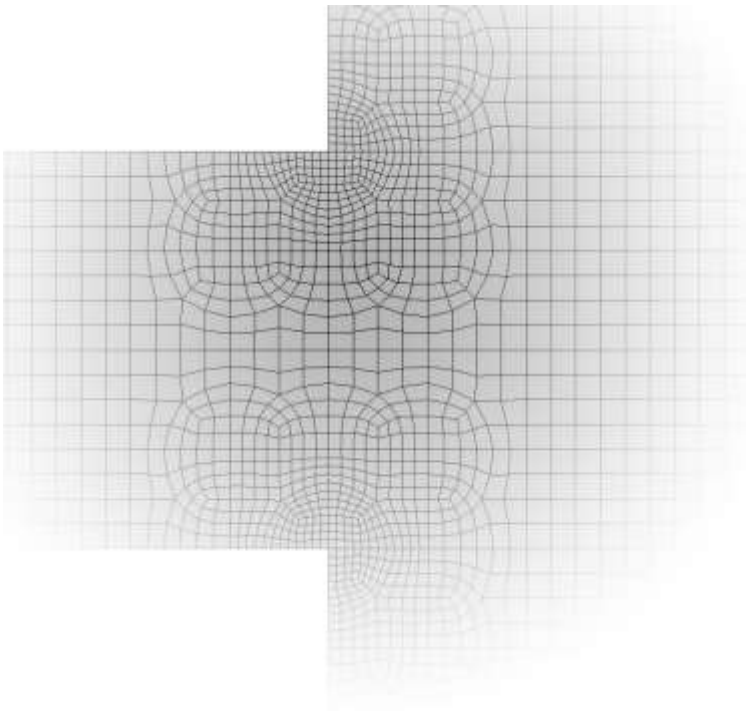

Table of Contents

1. Installation

MacMesh module is distributed as a tar ball archive (.tar.gz) which contains in a single directory all the necessary python code. The user is free to choose the directory of installation as long as the latter is properly specified in the *Salome* scripts calling the *MacMesh* module.

As an example, consider that the module is extracted in :

```
/local00/berro/SalomeFunc/MacMesh
```

In order to call the *MacMesh* module and use the included functions, it is necessary to add its installation path to the environment variable *PATH*. This can be accomplished in python by adding in the header of the *Salome* script, the following commands :

```
sys.path.append('/local00/berro/SalomeFunc/MacMesh/')
from MacObject import *
```

An illustrative example is given in the Example directory inside the tar ball. It represents the creation of a parametric axisymmetric model of a pressure relief valve uniquely by using the MacMesh module.

2. Elementary Objects

The *MacMesh* module is based on 5 elementary objects which are defined in the *MacObject* python class. All supported geometries can be constructed by collating a number of geometrically transformed (rotated and/or translated) elementary objects. The power of using elementary objects for the construction of meshes lies in the possibility to perform elementary level checks on the compatibility of the geometrical objects for producing conforming meshes.

2.1 Box11

Syntax

```
objname = MacObject  ('Box11'                ,
                      [(Xc,Yc) , (DX,DY)]    ,
                      [ Nseg ]  or ['auto']  ,
                      groups = ['S_GR','N_GR','W_GR', 'E_GR'] )
```

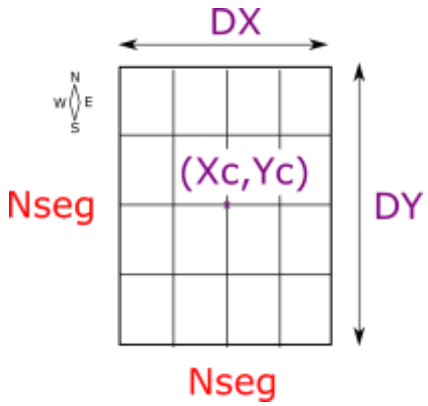

Figure 1: Schematic illustrating the *Box11* elementary shape. The four sides of the box are cut into the same number of segments (Nseg)

Examples

```
Square = MacObject('Box11',[(0.,0.),(20.,20.)],[10])
```

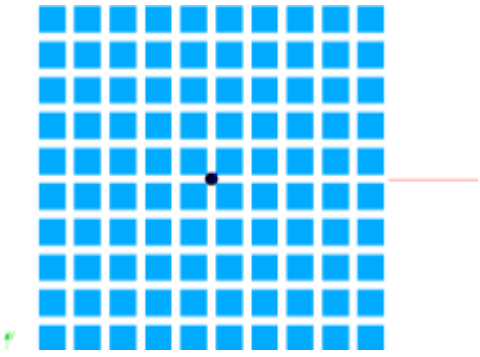

Figure 2: A *Box11* square object centred about the origin (0,0), of a side length = 20 and cut into 10 segments (Nseg = 10) on the X and Y directions.

```
Rectangle = MacObject('Box11',[(0.,0.),(40.,20.)],[10])
```

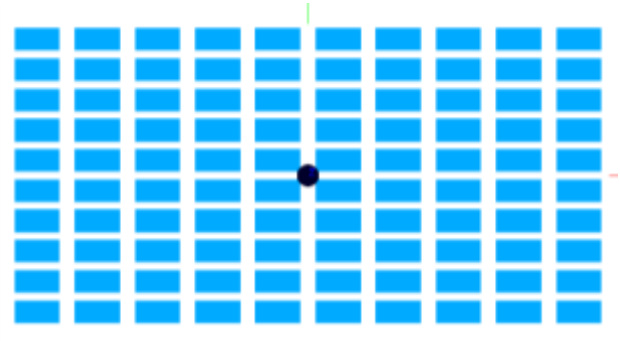

Figure 3: A *Box11* rectangle object centered about the origin (0,0), of an x-dimension of 40 and a y-dimension of 20. The box is cut into 10 segments on each direction (Nseg = 10).

```
SquareWithGroups = MacObject('Box11',[(0.,0.),(20.,20.)],[10],
                             groups=['Bottom','Top',None,None])
PublishGroups()
```

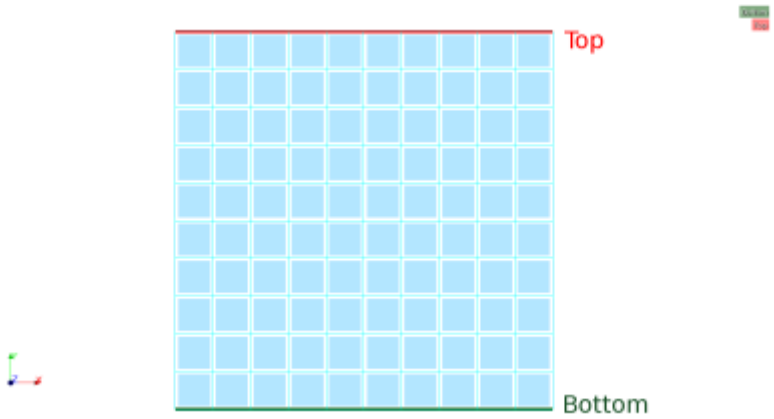

Figure 4: The same *Box11* object as in Figure 2 but with a definition of groups on the upper and lower boundaries.

```
SquareLeft = MacObject('Box11',[(0.,0.),(20.,20.)],[10],
               groups=['Boundary','Boundary','Boundary',None])
SquareRight = MacObject('Box11',[(20.,0.),(20.,20.)],['auto'],
               groups=['Boundary','Boundary',None,'Boundary'])
PublishGroups()
```

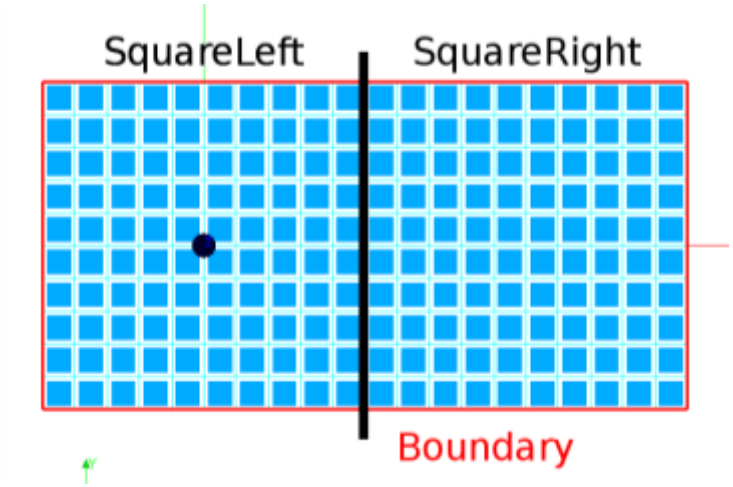

Figure 5: Two *Box11* square objects collated side-by-side where the right square inherits the number of segments (mesh parameters) automatically from the left square.

Box11 is the simplest object that can be thought of : a rectangle/square that is meshed with a constant number of segments on each side. A single mesh parameter (Nseg) is thus needed to define this kind of elementary object. The created 2D elements will have the same aspect ratio as the box itself which can be seen in Figure 3.

The groups option illustrated in Figure 4 and Figure 5 is particularly interesting because it allows to automatically generate *Salome* geometric and mesh groups on the created element. The groups are defined as borders with a known orientation : South (Y-) , North (Y+), West (X-), and East (X+), with this strict order. In case that no particular group is needed on one direction, the user can choose None (and not 'None') as illustrated in Figure 4.

Another important feature is that group names are global and thus may associate elements from different objects (see Figure 5).

Finally, the ['auto'] mesh parameter can be used when an existing object has been created and it is required to inherit mesh parameters from the neighbour(s). The module checks whether a compatible parameter can be found and proceeds with building the object.

The example in Figure 5 provides an excellent illustration of the last two concepts. In this example, two squares are to be built side-by-side where the mesh parameter of the first one is provided ([10]) but that of the second is set to automatic (['auto']). A single group called 'Boundary' containing south, north and west borders of the left square and south, north, and east borders of square 2 is created.

The meshing parameter (Nseg=10), for the second object (right square), was automatically detected and used for its creation. With the PublishGroups() command, the boundary group was successfully created with the specified directional criteria. Note that a unique PublishGroups() call is required at the end of the script as all group information is saved globally for all objects.

This same logic can be applied to all other elementary objects described below as well as to the macros allowing the construction of complicated geometries based on these elementary objects. We will discuss each of them one by one.

2.2 Box42

Syntax

```
objname = MacObject  ('Box42'            ,
                      [(Xc,Yc) , (DX,DY)] ,
                      [ Nseg, Orientation ] ,
                      groups = ['S_GR','N_GR','W_GR', 'E_GR'] )
```

```
Orientation = 'SN' or 'NS' or 'WE' or 'EW'
```

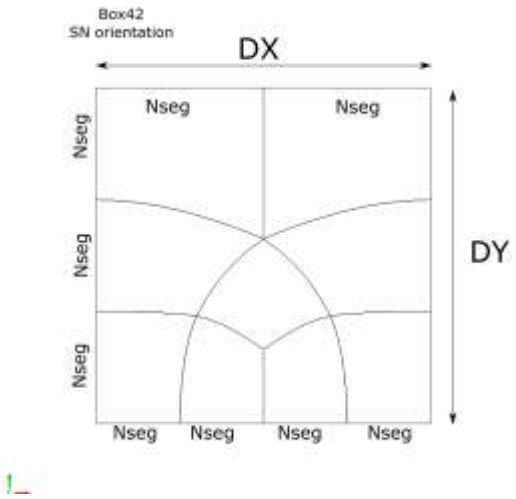

Figure 6: An illustration of the *Box42* elementary object in the SN (south-north) orientation. This object allows to unrefine on a given direction with a factor of 2 (from 4n segments to 2n)

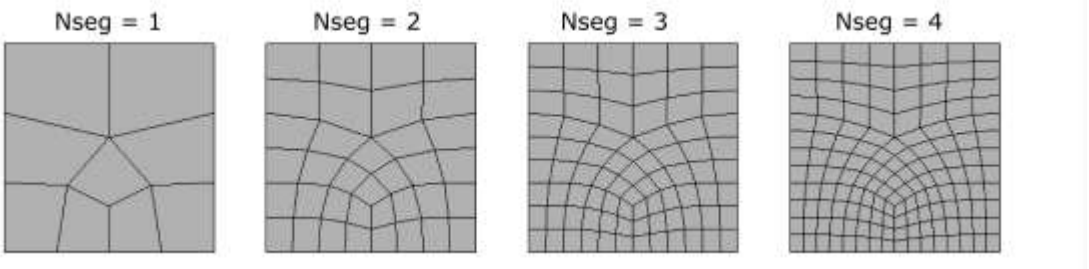

Figure 7: A *Box42* object with 4 different values for the Nseg mesh parameter.

2.3 BoxAng32

Syntax

```
objname = MacObject  ('BoxAng32'          ,
                      [(Xc,Yc) , (DX,DY)]  ,
                      [ Nseg, Orientation ],
                      groups = ['S_GR','N_GR','W_GR', 'E_GR'] )
```

```
Orientation = 'NE' or 'NW' or 'SE' or 'SW'
```

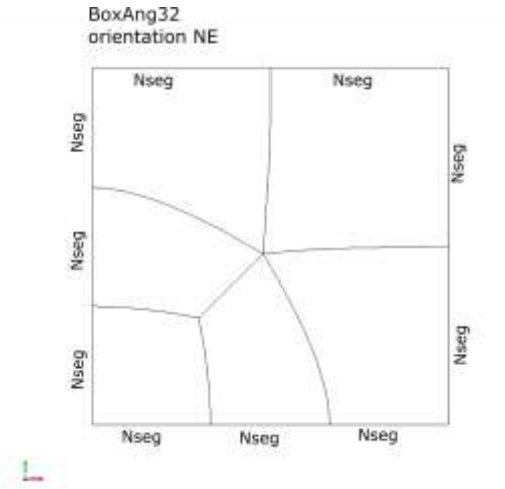

Figure 8: Illustration of the *BoxAng32* elementary object with a NE (north-east) orientation. This object can be used to radially unrefine with a ratio of 3:2.

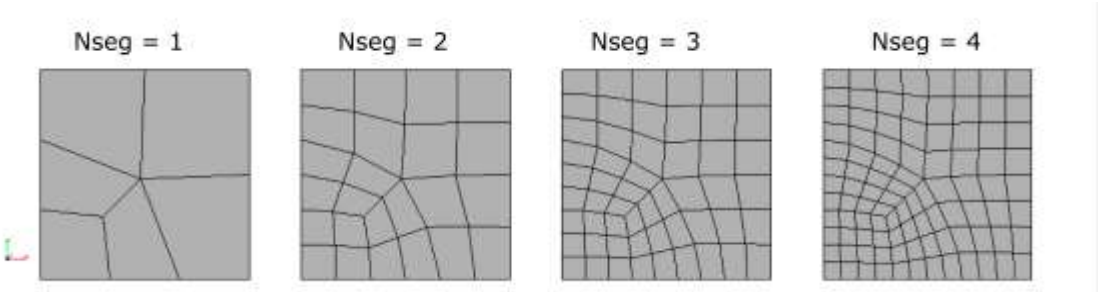

Figure 9: A *BoxAng32* object with different values for the Nseg mesh parameter.

2.4 CompBox

Syntax

```
objname = MacObject  ('CompBox'     ,
                      [(Xc,Yc) , (DX,DY)] ,
                      [ Nseg ]    or ['auto'] ,
                      groups = ['S_GR','N_GR','W_GR', 'E_GR'] )
```

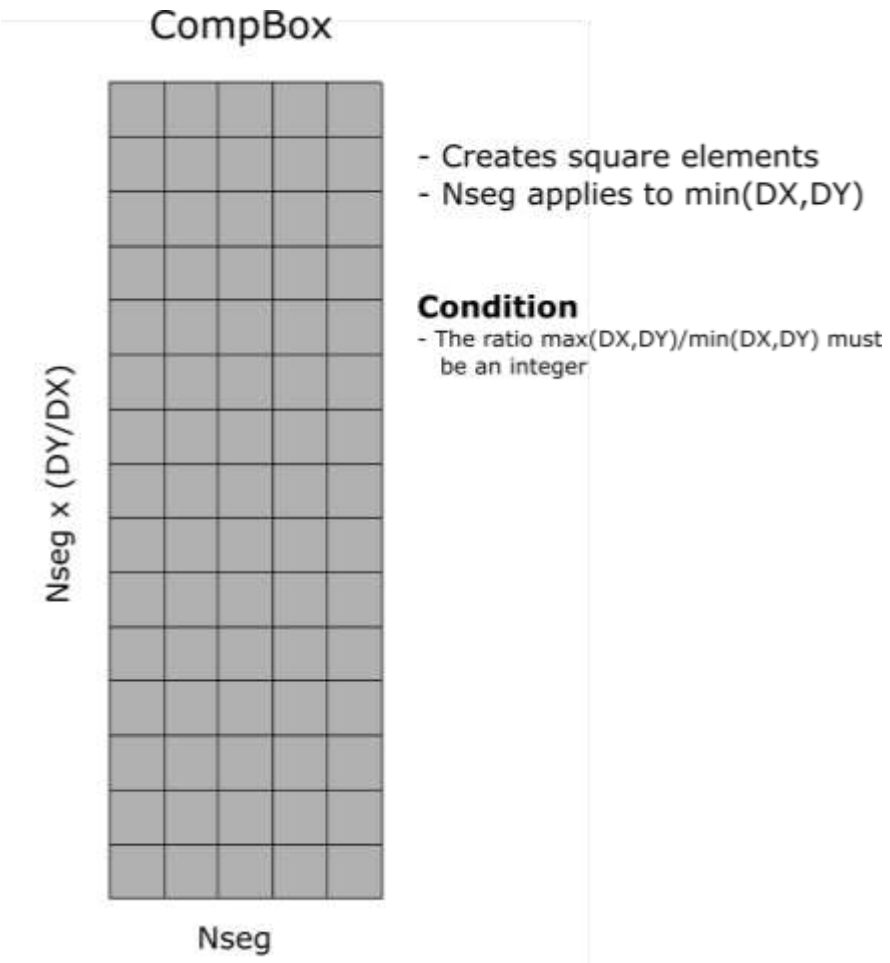

Figure 10: Illustration of the *CompBox* elementary object with Nseg = 5 and DY>DX. This object is used to create a square or a rectangle that is based on perfect square mesh elements. This evidently requires a condition on the ratio between the x and y dimensions are specified on the figure.

2.5 CompBoxF

Syntax

```
objname = MacObject  ('CompBoxF'   ,
                      [(Xc,Yc) , (DX,DY)] ,
                      [ (NsegX,NsegY) ] or ['auto'] ,
                      groups = ['S_GR','N_GR','W_GR', 'E_GR'] )
```

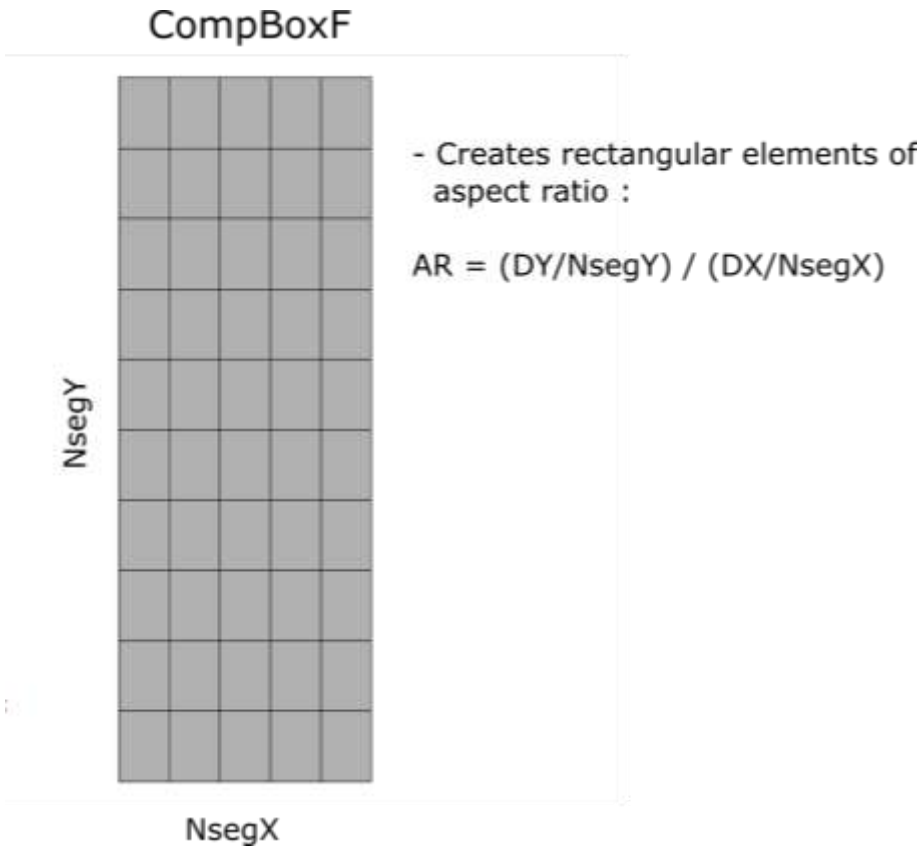

Figure 11: Illustration of the *CompBoxF* (F corresponding to Free) elementary object which, unlike *CompBox* does not require a condition on the ratio of DX/DY.

Note that if the mesh parameter is set to ['auto'], the module determines the NsegX and NsegY from neighbouring objects. If information is missing for one of the directions, let's say the y-direction, NsegY is calculated automatically as to render an optimal aspect ratio close to 1.

2.6 QuartCyl

Syntax

```
objname = MacObject ('QuartCyl'          ,
                     [(Xc,Yc) , (DX,DY)]      ,
                     [Nseg, Direction, PitchRatio ],
                     groups = ['S_GR','N_GR','W_GR', 'E_GR','Circle'])
```

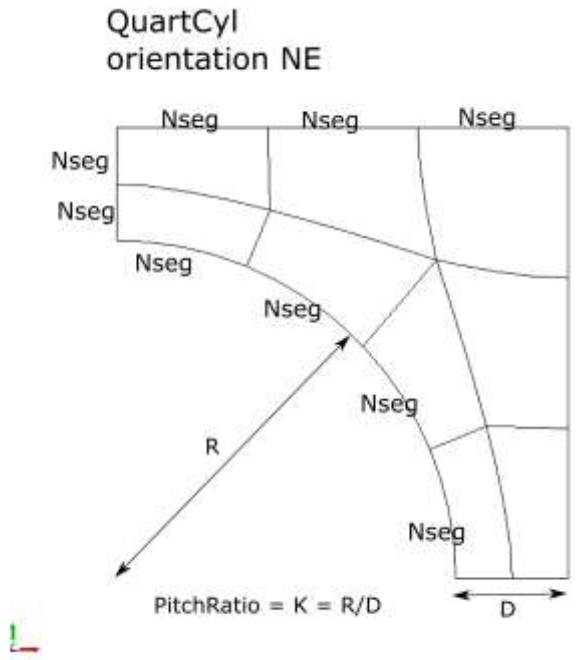

Figure 12: Illustration of the *QuartCyl* object with a NE (north-east) orientation. This object allows creating a quarter cylinder linked to a square or a rectangle.

In addition to the regular group definitions on each direction, it is possible to define for the *QuartCyl* object, a fifth group corresponding to the interior circle boundary. The pitch ratio is defined as the ratio between the circle's radius R and D, the gap between circle and the box's boundary as shown in Figure 12.

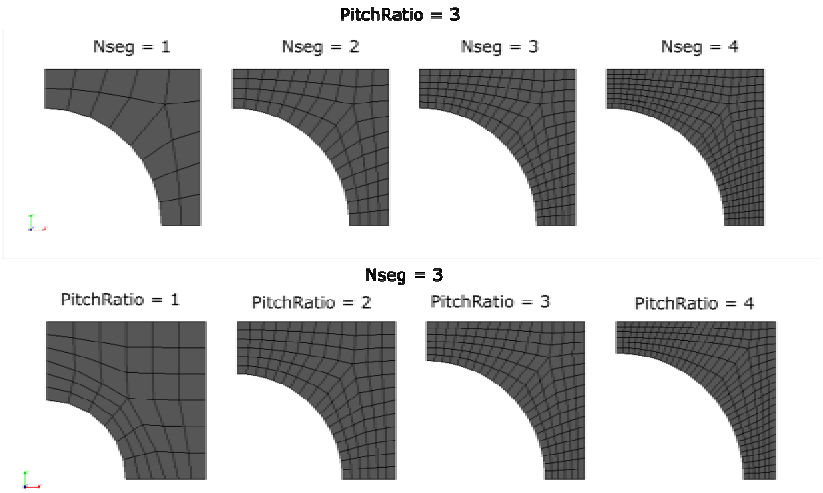

Figure 13: A *QuartCyl* object (oriented NE) with different values for the *PitchRatio* and the *Nseg* parameters.

Note that when the pitch ratio increases, the refinement automatically increases in order to yield an optimal aspect ratio in the constitutive elements of the object. In fact, the number of cuts on the circumference of the quarter circle (4 in the first figure) depends on the *PitchRatio* parameter because a quadrangle of the final mesh has a side length of $a0=(D/2)$ on one side and $a1=(\pi R/2*Ncuts)$ on the side of the circle. When *D* decreases (that is *PitchRatio* increases), a bigger number for *Ncuts* is chosen so that the ratio $a0/a1$ remain close to 1 and the final 2D mesh elements are as close to squares as possible.

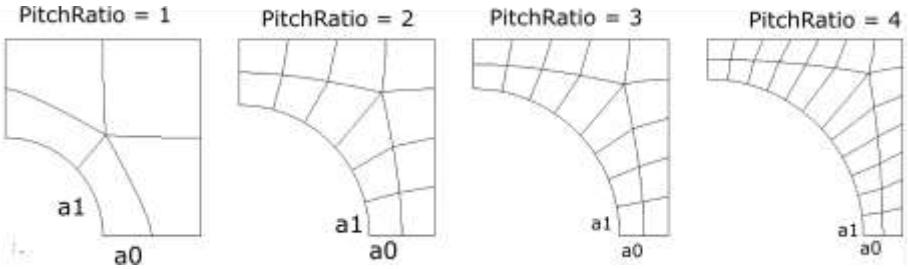

Figure 14: A schematic showing the influence of PitchRatio on the number of cuts. The purpose being in all cases to reproduce mesh elements with aspect ratios of nearly 1 (squares)

3. Macros

Macros are common combinations of the previously defined elementary objects. They actually make use of the elementary objects to produce geometries and meshes that make physical sense. For example, the *Cylinder* macro creates a 2D cylinder (a circle) embedded inside a rectangle by combining 4 *QuartCyl* objects. Macros also deal with the definition of groups on a high-level and translate the information to their constitutive objects.

Unlike elementary objects that are included in the MacObject.py class, macros are defined in separate files and thus need to be imported separately in the script. For example :

```
from Cylinder import *              # Cylinder macro
from SharpAngle import *            # SharpAngIn and SharpAngOut macros
from CentralUnrefine import *       # CentralUnrefine macro
from CompositeBox import *          # CompositeBox macro
```

3.1 Cylinder

Syntax

```
objname = Cylinder( Xc, Yc, D, DX, DY, LocalMeshing,
          groups = ['S_GR','N_GR','W_GR','E_GR','InnerCircle_GR'] )
```

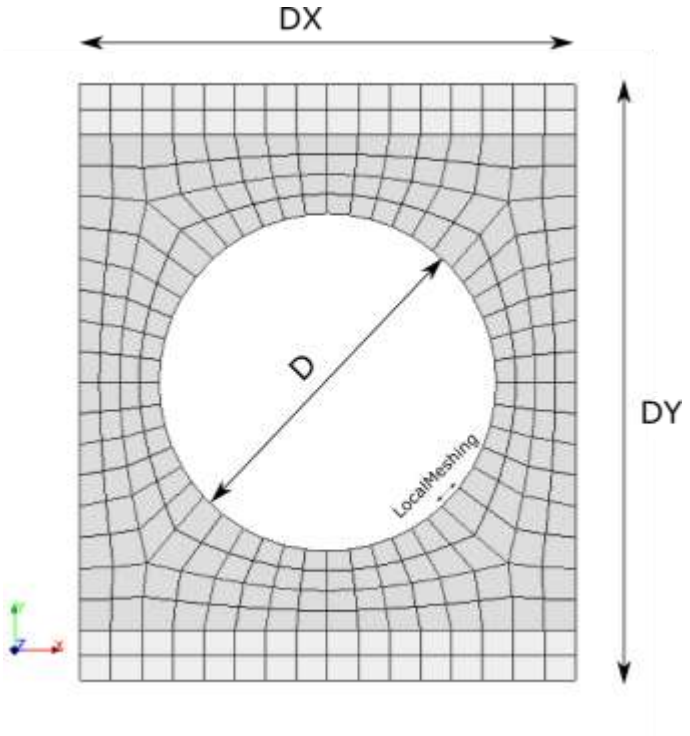

Figure 15: Illustration of the *Cylinder* macro that works by combining four identical *QuartCyl* elementary object with different orientations.

LocalMeshing is defined in distance units as a first prescription for the finest refinement near the cylinder (see Figure 15). Note that in most cases, the exact refinement can not be achieved due to the discrete nature of the meshing concept. In these cases, the cylinder macro module calculates an optimized value of the local refinement and displays it on the output screen.

For example, let's suppose that we need to create a cylinder of *D=10 mm*, *DX = 15 mm*, and *DY = 18 mm*, centered about the origin (see Figure 15 for geometry). This can be achieved with the following simple command :

```
Cylinder ( 0. , 0., 10., 15., 18., LocalMeshing)
```

Let's suppose *LocalMeshing* is set to *1mm*. The program proceeds as follows. It calculates the cylindre's circumference $P = 31.415\ mm$. This value is then divided by the number of initial cuts needed for an optimal aspect ratio. *Ncuts* is calculated using the relationship:

```
Ncuts = 4 x floor ( [ πD ] / [ min( DX,DY) – D  ] )
```

In this example, $D = 10$, and $min(DX,DY) = 15$. Then $Ncuts = 4\ x\ floor\ (10\pi / 5) = 4\ x\ floor\ (2\pi) = 24$

Thus the minimum local refinement is $P/Ncuts = 1.309\ mm$. The purpose is then to find the smallest integer « *i* » that satisfies : $1.309/i < LocalMeshing$ where $LocalMeshing = 1$ in this case.

This gives : $i > 1.309$, hence $i = 2$. The possible local meshing is thus : $1.309 / 2 = 0.655\ mm$.

This value is returned by the function which displays the following:

```
A local pitch ratio of K = 2.0 will be used.
Possible Local meshing is : 0.654498469498 This value is returned by
this function for your convenience.

Initializing object No. 1
Generating quarter cylinder
Successfully created

Initializing object No. 2
Generating quarter cylinder
Successfully created

Initializing object No. 3
Generating quarter cylinder
Successfully created

 (.........)

Initializing object No. 6
Generating composite box
Successfully created
```

Notice that the macro creates 6 objects instead of only 4 which correspond to the *QuartCyl* elementary objects. The last two actually are needed because the required Y dimension of the box is bigger than its X dimension. In order to complete the gap, the macro automatically creates two *CompBoxF* that extend from the *Cylindre* boundaries to the outer limits of the box.

3.2 SharpAngleOut

Syntax

```
objname = SharpAngleOut ( Xo, Yo, DX, DY, DLocal,
                          LocalMeshing, Orientation, NLevels
                          groups = ['InnerH_GR','InnerV_GR',
                                    'OuterH_GR','OuterV_GR',
                                    'Inlet_GR' ,'Outlet_GR'] )
```

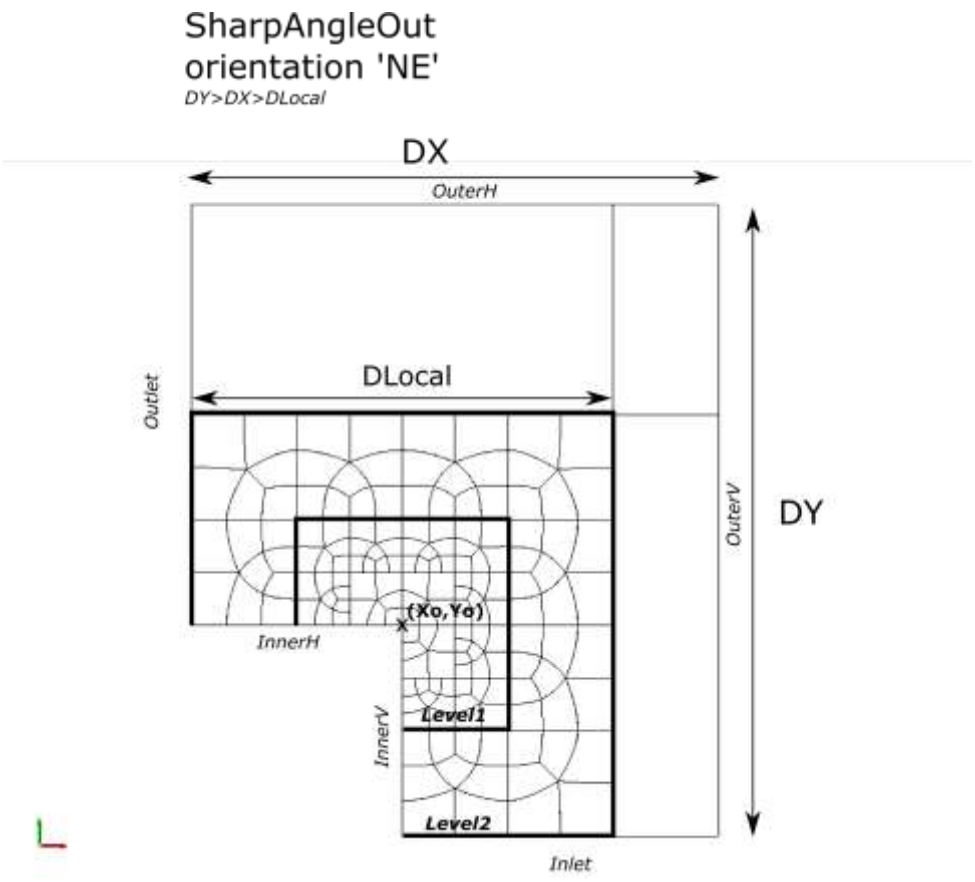

Figure 16: Illustration of a *SharpAngleOut* macro with a NE (north-east) orientation and a parameter *Nlevels* = 2.

As can be seen on Figure 16, this interesting macro is based on *Box42*, *Box32*, and *CompBoxF* elementary objects. Such a macro is used for example when a special refinement is needed near geometrical corners.

Several parameters allows to control the way this macro is constructed as shown in Figure 16. *Xo* and *Yo* are the coordinates of the corner point. *DX* and *DY* represent the extents on the x and y direction of the macro. *DLocal* represents the internal size of the refinement box, it can also be set to 'auto'. In the latter case, *DLocal* is given the value of *min(DX,DY)*. The *LocalMeshing* parameter is given in distance units and is similar to that for the *Cylindre* macro in the sense that it gives a prescription of the maximum local refinement which in this case is near *(Xo,Yo)* at the corner's center. We will get back later on the method of its precise calculation.

Orientation allows setting the corner's direction, from inner -> outer.

NLevels is another very important parameter as it controls the number of "snake loops" used for the refinement.

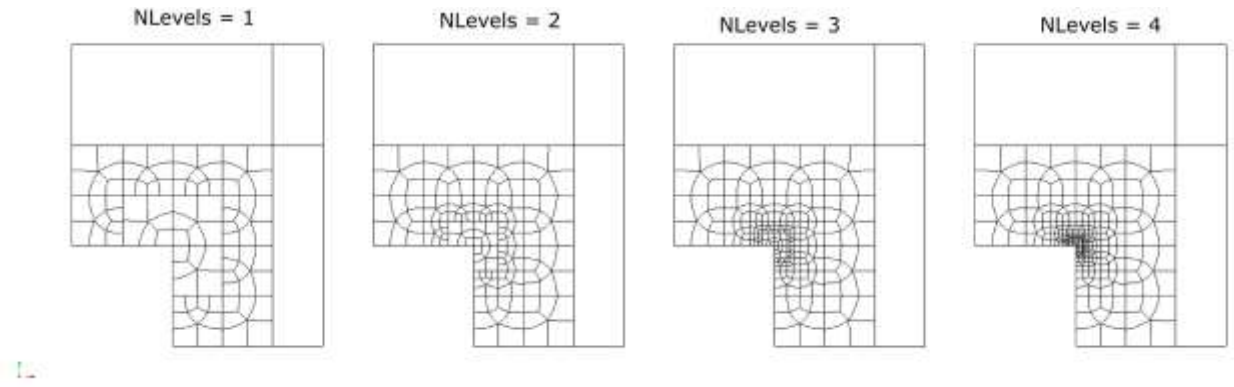

Figure 17: A "skeleton" representation of the cuts perfomed for the *SharpAngleOut* macro with four different values of the NLevels parameter.

As shown in Figure 17, an increased number of *NLevels* results generally in an increased unrefinement rate over *DLocal*. If a progressive unrefinement is required as in the cases where interesting phenomena originate at a corner but then may propagate across the domain, it is preferred to stick with a low number of levels and in extending *DLocal* as long as possible. However, in static type problems where phenomena are localized, an increase in *NLevels* permits achieving a high local resolution.

As for the groups that can be saved with this macro, the notion of orientation is completely different from that for elementary objects because of the particular form of the angle. Six groups can be defined and correspond to the 6 boundaries of the SharpAngleOut macro as illustrated on Figure 16.

In order to illustrate the way local meshing criterion is determined, consider the example case in which it is required to create an angle where the unrefinement extends over *8mm* (*DLocal*), the final box dimensions are required to be *10mm* and *12mm* on the x and y directions respectively. Moreover, it is preferred that the local refinement is just lower than *0.2 mm*.

We will test four values for *Nlevels*, from 1 to 4.

The geometrical cutting method imposes a lowest refinement which depends on *DLocal* and *NLevels*, it is defined with the formula :

```
d0 = Dlocal / (2^(Nlevels+2) x 3 ) # Highest local refinement
```

This gives the following values for *d0* in this case (*Dlocal = 8mm*)

	Nlevels = 1	**Nlevels = 2**	**Nlevels = 3**	**Nlevels = 4**
d0	1/3 = 0,3333	1/6 = 0,1666	1/12 = 0,0833	1/24 = 0,04166

As with the cylinder, it follows that we look for the smallest integer "*i*" that satisfies the inequation :

```
Real Meshing = d0/i < LocalMeshing
```

In this case, if *LocalMeshing* is set to *0.2mm*, *i = 1* is OK for all cases of *NLevels* except *Nlevels=1 for which i=2*. The possible real meshing distances are then :

	Nlevels = 1	Nlevels = 2	Nlevels = 3	Nlevels = 4
Real Meshing < 0,2 mm	0,1666	1/6 = 0,1666	1/12 = 0,0833	1/24 = 0,04166

This shows that an excessive refinement may be obtained when a large number of *NLevels* is used. Note also that the real meshing value is displayed on the screen when this macro is used:

```
Possible Local meshing is : 0.166666666667 This value is returned by
this function for your convenience
```

Finally, here is a snapshot of the meshes generated in the previous example.

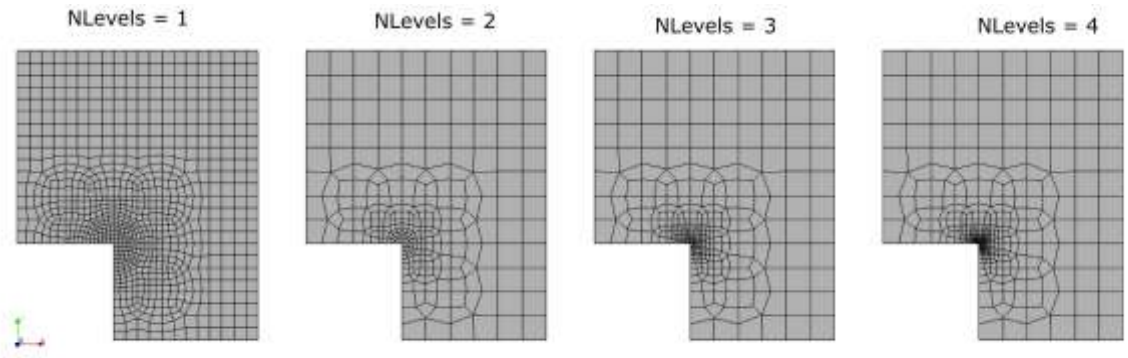

Figure 18: snapshot of the obtained meshes with the SharpAngOut macro for different values of the NLevels parameter.

It is obvious that a better quality mesh is obtained with *NLevels = 1* compared to *Nlevels=2* even though the local meshing is exactly the same (see table). However, this comes at the cost of 800 face elements v.s only 272 for the case of *NLevels = 2*.

3.3 SharpAngleIn

Syntax

```
objname = SharpAngleIn ( Xo, Yo, DX, DY, DLocal,
                         LocalMeshing, Orientation, NLevels
                         groups = ['S_GR','N_GR','W_GR', 'E_GR'])
```

Note that the orientation of the SharpAngleIn is outwards from the refinement corner.

3.4 CompositeBox

Syntax

```
objname = CompositeBox ( Xo, Yo, DX, DY,
                         groups = ['S_GR','N_GR','W_GR', 'E_GR'])
```

An apparently simple object that regroups all possibilities for inserting meshed boxes in any situation. This macro inherits meshing parameters from the bounding objects even if there are several objects on one direction or over several directions.

In practice, unless we are creating the first object in the domain (in that case we have to define a mesh parameter), it is always convenient to use a composite box. Note that an extra macro can be used in case CompositeBox presents errors, it is called CompositeBoxF and uses the same exact syntax and group options.

3.5 CentralUnrefine

Syntax

```
objname = CentralUnrefine ( Xo, Yo, DX, DY, Orientation,
                         groups = [(1),(2),(3),(4),(5),(6)])
```

```
Orientation = 'SN', 'NS', 'EW', or 'WE'
```

This macro is based on Box42 and Box32 elementary objects. Xo and Yo represent the coordinates of the center of the unrefinement neck while DX and DY represent the extension of the unrefinement on X and Y as shown in Figure 19.

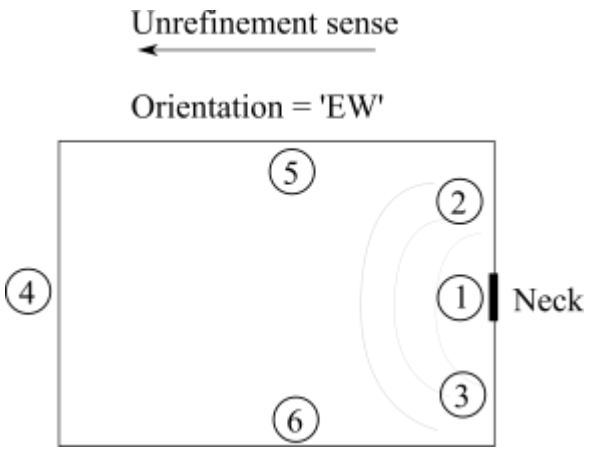

Figure 19: schematic of the CentralUnrefine object with an 'EW' orientation. The circled numbers represent the order of the group names that can be defined for this object. This macro needs to inherit its meshing properties from a 'neck' object as shown in the figure.

4. Miscellaneous user-tools

4.1 PublishGroup.py

```
Salome_mesh = PublishGroups()
```

In addition to publishing geometric and mesh groups, the PublishGroups function of the PublishGroup module returns the salome smesh object corresponding to the final mesh. This is practical if further script manipulations are necessary like extruding or revolving the 2D mesh.

```
ExtrudeMesh( Salome_mesh,
         -->  Direction     = [Vx,Vy,Vz],
         -->  Distance      = D,
         -->  NSteps        = N,
         -->  Scale         = r                    )
```

This command allows easy extrusion and rescaling of a published mesh. Only the mesh object is obligatory for this command, defaults for Direction, Distance, NSteps, and Scale are : [0,0,1] (z-axis), D=1, N=1, and r=1 (no scaling). Note that this command recovers the group names as specified in the original mesh and creates one that defines the volume.

```
RevolveMesh(  Salome_mesh,
         -->  Direction     = [Vx,Vy,Vz],
         -->  Center        = [Cx,Cy,Cz],
         -->  AngleDeg      = AlphaDeg,
```

```
        //OR AngleRad      = AlphaRad,
        -->  Scale         = r                    )
```

This command allows easy revolving and rescaling of a published mesh for axisymmetric studies. Only the mesh object is obligatory for this command, defaults for Direction, Center, AngleDeg, AngleRad and Scale are : [0,0,1] (z-axis), [0,0,0] (origin), Alpha=10 degrees or equivalent in radians, and finally r=1 (no scaling). This command recovers the group names as specified in the original mesh and creates a new one that defines the volume.