Intersector 2.5

Sam Landier, Christophe Benoit, Stephanie Peron, Luis Bernardos
- Onera -

1 Intersector: Mesh intersection module

1.1 Preamble

This module provides pre ans post processing services relying on mesh-intersection computations on arbitrary polyhedral meshes.
It also gives auxiliary functions that transform topologically and geometrically polyhedral meshes which are useful in the process of mesh generation by intersection.

A mesh can be stored as an array (as defined in the Converter documentation) or in a zone node of a CGNS/python tree (pyTree).

This module is part of Cassiopee.

When using the array interface, import the Intersector module:

```
import Intersector as XOR (array version)
or
import Intersector.PyTree as XOR (PyTree version)
```

1.2 Main Functions

XOR.conformUnstr: conformizes a TRI or BAR soup (i.e. a set of elements not necessarily connected as a mesh) by detecting and solving all the collisions between elements.
Colliding elements are cut to get a conformal set. Mixed type BAR/TRI are not handled.

Parameter	**Meaning**	
s1	input mesh (BAR or TRI). If s2 is 'None' self-intersections are solved over s1.	
[s2]	optional : conformize s1 taking into account collisions that might occur with s2	
tol	merging tolerance when points (exisiting or computed by intersections) are too close.	
left_or_right	Tells the function what to ouput : the transformed s1 (left operand), s2(right operand) or both.	
itermax	Number of intersection/merging iterations. 10 is the default value.	
Parameter values	**Meaning**	
tol¿0.	use the input value as an absolute merging tolerance.	
tol=0. (default)	Computes the tolerance as 5tol¡0.	Consider this input value (must be between 0. and 1.) as a ratio to apply to the min edge length to get the tolerance.
left_or_right=0	Output s1	
left_or_right=1	Output s2.	
left_or_right=2 (default)	Output both s1 and s2.	

Tip: set itermax to 1. to improve speed and the Delaunay kernel robustness. The result might have poorer quality triangles though.

```
B = XOR.conformUnstr(s1, s2=None, tol=0., left_or_right=2, itermax=10)
```

(See: conformUnstr.py) (See: conformUnstrPT.py)

XOR.booleanUnion: performs a boolean union of two TRI-surfaces:

```
b = XOR.booleanUnion(a1, a2, tol=0.)
```

(See: booleanUnion.py) (See: booleanUnionPT.py)

XOR.booleanMinus: performs a boolean difference of two TRI-surfaces:

```
b = XOR.booleanMinus(a1, a2, tol=0.)
```

(See: booleanMinus.py) (See: booleanMinusPT.py)

XOR.booleanIntersection: performs a boolean intersection of two TRI-surfaces:

```
b = XOR.booleanIntersection(a1, a2, tol=0.)
```

(See: booleanIntersection.py) (See: booleanIntersectionPT.py)

XOR.intersection: returns the 'BAR' contour defining the intersection between two TRI-surfaces:

```
b = XOR.intersection(a1, a2, tol=0.)
```

(See: intersection.py) (See: intersectionPT.py)

XOR.XcellN: computes the cell nature field of a background mesh (bgm) in an overset configuration : similarily to the blanCells functions, the input maskMesh are volume meshes that hide bgm. The computed celln is accurate, giving a floating value ranging from 0. (fully masked) to 1. (fully visible).
The input grids (bgm and makingMesh) are defined by coords located at nodes as a list of arrays.

Warning: location of celln must be located at centers.
Warning: In order to set the celln to 0. inside blanking bodies, you need to create BCWall type boundaries on the body faces.

```
celln = XOR.XcellN(bgm, celln, maskMesh)
```

(See: XcellNPT.py)

1.3 Auxiliary Functions

XOR.triangulateExteriorFaces: Triangulates any external polygon on a polyhedral mesh (NGON format)

```
b = XOR.triangulateExteriorFaces(NGON3Dmesh)
```

(See: triangulateExteriorFaces.py) (See: triangulateExteriorFacesPT.py)

XOR.convexifyFaces: Makes a convex decomposition of any concave polygon (NGON format)

```
b = XOR.convexifyFaces(NGON3Dmesh)
```

(See: convexifyFaces.py) (See: convexifyFacesPT.py)

1.4 Example files

Example file: conformUnstr.py

```
# - conformUnstr (array) -
# Conforming 1 or 2 TRI/BAR together (same type for both operands
import Generator as G
import Intersector as XOR
import Converter as C
import Geom as D
from Geom.Parametrics import base
```

```
import Transform as T

s1 = D.sphere((0,0,0), 1, N=20)

s2 = D.surface(base['plane'], N=30)
s2 = T.translate(s2, (0.2,0.2,0.2))

s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)

x = XOR.conformUnstr(s1, s2, 0., 2)
C.convertArrays2File([x], 'out.plt')

c1 = D.circle((0,0,0), 1, N=100)
c2 = D.circle((0.2,0,0), 1, N=50)

c1 = C.convertArray2Tetra(c1); c1 = G.close(c1)
c2 = C.convertArray2Tetra(c2); c2 = G.close(c2)

x = XOR.conformUnstr(c1, c2, tol=0.)
C.convertArrays2File([x], 'out1.plt')
```

Example file: conformUnstrPT.py

```
# - conformUnstr (pyTree) -
# Conforming 1 or 2 TRI/BAR together (same type for both operands)
import Generator.PyTree as G
import Intersector.PyTree as XOR
import Converter.PyTree as C
import Geom.PyTree as D
from Geom.Parametrics import base
import Transform.PyTree as T

s1 = D.sphere((0,0,0), 1, N=20)

s2 = D.surface(base['plane'], N=30)
s2 = T.translate(s2, (0.2,0.2,0.2))

s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)

x = XOR.conformUnstr(s1, s2, tol=0.)
C.convertPyTree2File(x, 'out.plt')

c1 = D.circle((0,0,0), 1, N=100)
c2 = D.circle((0.2,0,0), 1, N=50)

c1 = C.convertArray2Tetra(c1); c1 = G.close(c1)
c2 = C.convertArray2Tetra(c2); c2 = G.close(c2)

x = XOR.conformUnstr(c1, c2, tol=0.)
C.convertPyTree2File(x, 'out1.plt')
```

Example file: booleanUnion.py

```
# - booleanUnion (array) -
import Intersector as XOR
import Generator as G
import Converter as C
import Geom as D
```

```
s1 = D.sphere((0,0,0), 1, N=20)
s2 = D.sphere((0.,1.,0.), 1, N=30)

s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)

x = XOR.booleanUnion(s1, s2, tol=0.)
C.convertArrays2File([x], 'out.plt')
```

Example file: booleanUnionPT.py

```
# - booleanUnion (pyTree) -
import Intersector.PyTree as XOR
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D

s1 = D.sphere((0,0,0), 1, N=20)
s2 = D.sphere((0.,1.,0.), 1, N=30)

s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)

x = XOR.booleanUnion(s1, s2, tol=0.)
C.convertPyTree2File(x, 'out.cgns')
```

Example file: booleanMinus.py

```
# - booleanMinus (array) -
import Intersector as XOR
import Generator as G
import Converter as C
import Geom as D

s1 = D.sphere((0,0,0), 1, N=20)
s2 = D.sphere((0.,1.,0.), 1, N=30)

s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)

x = XOR.booleanMinus(s1, s2, tol=0.)
C.convertArrays2File([x], 'out.plt')
```

Example file: booleanMinusPT.py

```
# - booleanMinus (pyTree) -
import Intersector.PyTree as XOR
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D

s1 = D.sphere((0,0,0), 1, N=20)
s2 = D.sphere((0.,1.,0.), 1, N=30)

s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)

x = XOR.booleanMinus(s1, s2, tol=0.)
C.convertPyTree2File(x, 'out.cgns')
```

Example file: booleanIntersection.py

```
# - boolean intersection (array) -
import Intersector as XOR
import Generator as G
import Converter as C
import Geom as D

s1 = D.sphere((0,0,0), 1, N=20)
s2 = D.sphere((0.,1.,0.), 1, N=30)

s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)

x = XOR.booleanIntersection(s1, s2, tol=0.)
C.convertArrays2File([x], 'out.plt')
```

Example file: booleanIntersectionPT.py

```
# - booleanIntersection (pyTree) -
import Intersector.PyTree as XOR
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D

s1 = D.sphere((0,0,0), 1, N=20)
s2 = D.sphere((0.,1.,0.), 1, N=30)

s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)

x = XOR.booleanIntersection(s1, s2, tol=0.)
C.convertPyTree2File(x, 'out.cgns')
```

Example file: intersection.py

```
# - intersection (array) -
import Intersector as XOR
import Generator as G
import Converter as C
import Geom as D

s1 = D.sphere((0,0,0), 1, N=20)
s2 = D.sphere((0.,1.,0.), 1, N=30)
s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)
x = XOR.intersection(s1, s2, tol=0.)
C.convertArrays2File([x], 'out.plt')
```

Example file: intersectionPT.py

```
# - intersection (pyTree) -
import Intersector.PyTree as XOR
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D

s1 = D.sphere((0,0,0), 1, N=20)
s2 = D.sphere((0.,1.,0.), 1, N=30)
s1 = C.convertArray2Tetra(s1); s1 = G.close(s1)
s2 = C.convertArray2Tetra(s2); s2 = G.close(s2)
x = XOR.intersection(s1, s2, tol=0.)
C.convertPyTree2File(x, 'out.cgns')
```

Example file: XcellNPT.py

```
# - XcellN (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Intersector.PyTree as XOR
import Geom.PyTree as D

# Test 1
# Mask
masking = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
masking = C.convertArray2NGon(masking)
# Mesh to blank
bgm = G.cart((-3.,-3.,-3.), (0.5,0.5,0.5), (20,20,20))
t = C.newPyTree(['Cart', bgm])
t = C.convertArray2NGon(t)

# celln init
t = C.initVars(t, 'centers:cellN', 1.)
# Blanking with floating cellN computation
t = XOR.XcellN(t, [[masking]], [])
C.convertPyTree2File(t, 'out1.cgns')

# Test 2
# Tet mask
masking = D.sphere((0,0,0), 15., 30)
masking = C.convertArray2Tetra(masking)
masking = G.close(masking)
masking = G.tetraMesher(masking, algo=1)
#C.convertPyTree2File(masking, 'sph.cgns')
# Mesh to blank
bgm = G.cart((-5.,-5.,-5.), (0.8,0.8,0.8), (40,40,40))
t = C.newPyTree(['Cart', bgm])
t = C.convertArray2NGon(t)
# celln init
t = C.initVars(t, 'centers:cellN', 1.)
# Blanking
t = XOR.XcellN(t, [[masking]], [])
C.convertPyTree2File(t, 'out2.cgns')
```

Example file: triangulateExteriorFaces.py

```
# - triangulateExteriorFaces (array) -
import Intersector as XOR
import Converter as C

m = C.convertFile2Arrays('boolNG_M1.tp')
m = C.convertArray2NGon(m[0])

m = XOR.triangulateExteriorFaces(m)
C.convertArrays2File([m], 'out.plt')
```

Example file: triangulateExteriorFacesPT.py

```
# - triangulateExteriorFaces (PyTree) -
import Intersector.PyTree as XOR
import Converter.PyTree as C

t = C.convertFile2PyTree('boolNG_M1.tp')
t = C.convertArray2NGon(t)
```

```
t = XOR.triangulateExteriorFaces(t)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: convexifyFaces.py

```
# - convexifyFaces (array) -
# convexify any concave polygon in the mesh
import Intersector as XOR
import Converter as C

M1 = C.convertFile2Arrays('boolNG_M1.tp')
M1 = C.convertArray2NGon(M1[0])

M2 = C.convertFile2Arrays('boolNG_M2.tp')
M2 = C.convertArray2NGon(M2[0])

tol = -0.5e-3

m = XOR.booleanMinus(M1, M2, tol, preserve_right=1, solid_right=1, agg_mode=2) #full agg to convexify afterward
#C.convertArrays2File([m], 'i.plt')
m = XOR.convexifyFaces(m)

C.convertArrays2File([m], 'out.plt')
```

Example file: convexifyFacesPT.py

```
# - convexifyFaces (pyTree) -
# convexify any concave polygon in the mesh
import Intersector.PyTree as XOR
import Converter.PyTree as C

M1 = C.convertFile2PyTree('boolNG_M1.tp')
M1 = C.convertArray2NGon(M1)

M2 = C.convertFile2PyTree('boolNG_M2.tp')
M2 = C.convertArray2NGon(M2)

tol = -0.5e-3

m = XOR.booleanMinus(M1, M2, tol, preserve_right=1, solid_right=1, agg_mode=2) #full agg to convexify afterward
m = XOR.convexifyFaces(m)

C.convertPyTree2File(m, 'out.cgns')
```