Dist2Walls 2.5

Stephanie Peron, Christophe Benoit, Pascal Raud, Sam Landier
- Onera -

1 Dist2Walls: wall distance computation

1.1 Preamble

Dist2Walls gathers efficient algorithms for computing the distance fields for arrays (as defined in Converter documentation) or for CGNS/python tree (pyTrees).

This module is part of Cassiopee, a free open-source pre- and post-processor for CFD simulations.

When using the Converter array interface, a (or b) denotes an array, and A (or B) denotes a list of arrays. Then, Dist2Walls module must be imported:

```
import Dist2Walls as DTW
```

When using the pyTree interface, import the module:

```
import Dist2Walls.PyTree as DTW
```

In that case, a is a zone node and A is a list of zone nodes or a pyTree.

1.2 Module functions

DTW.distance2Walls : computes the distance field from a set of bodies.
compute the distance field located at nodes or centers of zone a (or zones in A), provided a list of surfaces defining the bodies to which the distance is computed.
Two algorithms are available:
- type='ortho' means a distance computed by an orthogonal projection to the surface faces defined by bodies.
- type='mininterf' returns the minimum distance of the point to the vertices of bodies.

If loc='nodes', returns a distance computed at nodes of a (A), else if loc='centers, distance is computed at cell centers of a (A).
Parameter 'signed'=1 enables to compute a signed distance (negative inside bodies).

When using signed distances, each body in bodies list must be a closed and watertight surface.
In array version, cellnbodies provides the 'cellN' field for any vertex in bodies. Default value is 1.
The algorithm 'ortho' does not take into account a body face if cellN=0 for all the vertices of that face.
The algorithm 'mininterf' does not compute the distance to a vertex of cellN=0.

b = DTW.distance2Walls(a, bodies, cellnbodies=[], type='ortho', loc='centers', signed=0, dim=3) *.or.* B = DTW.distance2Walls(A, bodies, cellnbodies=[], type='ortho', loc='centers', signed=0, dim=3)

In the pyTree version, 'cellN' variable must be stored in bodies directly.
If loc='nodes', the distance field is stored as a 'TurbulentDistance' field located at nodes, and if loc='centers', it is stored in nodes located at centers:

b = DTW.distance2Walls(a, bodies, type='ortho', loc='centers', signed=0, dim=3) *.or.* B = DTW.distance2Walls(A, bodies, type='ortho', loc='centers', signed=0, dim=3)

(See: distance2Walls.py) (See: distance2WallsPT.py) (See: distance2FilePT.py)

1.3 Example files

Example file: distance2Walls.py

```
# - distance2Walls (array) -
import Dist2Walls
import Generator as G
import Converter as C
import Geom as D

# Bloc dont on cherche la distance a la paroi
a = G.cart((0.,0.,0.),(0.1,0.1,0.1),(10,10,10))

# Paroi
sphere = D.sphere((1.2,0.,0.), 0.2, 30)
cellN = C.initVars(sphere,'cellN',1.)
# Calcul de la distance a la paroi
dist = Dist2Walls.distance2Walls(a, [sphere], cellnbodies=[cellN],
                                 loc='centers',type='ortho')
ac = C.node2Center(a)
ac = C.addVars([ac, dist])
C.convertArrays2File([ac], 'out.plt')
```

Example file: distance2WallsPT.py

```
# - distance2Walls (pyTree) -
import Dist2Walls.PyTree as Dist2Walls
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D

a = G.cart((0.,0.,0.),(0.1,0.1,0.1),(10,10,10))
sphere = D.sphere((1.2,0.,0.),0.2,100)
t = C.newPyTree(['Base',a])
```

```
t = Dist2Walls.distance2Walls(t, sphere)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: distance2FilePT.py

```
# - distance2Walls (pyTree) -
# - Dump TurbulentDistance node to a file -
import Dist2Walls.PyTree as DW
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D
import Converter.Internal as Internal
import Converter
import numpy

a = G.cart((0.,0.,0.),(0.1,0.1,0.1),(10,10,10))
a = C.initVars(a,'centers:cellnf',1.)
sphere = D.sphere((1.2,0.,0.),0.2,100)
sphere = C.initVars(sphere,'centers:cellnf',1.)

t = C.newPyTree(['Base',a])
bodies = C.newPyTree(['Bodies',sphere])
t = DW.distance2Walls(t, bodies)
nodes = Internal.getNodesFromName(t, 'TurbulentDistance')
c = 0
for n in nodes:
    ni = n[1].shape[0]; nj = n[1].shape[1]; nk = n[1].shape[2]
    a = numpy.reshape(n[1], (ni*nj*nk), order='Fortran')
    a = numpy.reshape(a, (1,ni*nj*nk))
    array = ['walldistance', a, ni, nj, nk]
    array = Converter.initVars(array, 'wallglobalindex', 1)
    Converter.convertArrays2File([array], 'dist'+str(c)+'.v3d',
                                 'bin_v3d')
    c += 1
```