- Cassiopée -

CFD Advanced Set of Services In an Open Python EnvironmEnt

S. Péron, C. Benoit, S. Landier, P. Raud
ONERA – CFD & Aeroacoustics Dept

12th Symposium on Overset Composite Grids and Solution Technology
October 6-9, 2014, Georgia Institute of Technology, Atlanta

ONERA
THE FRENCH AEROSPACE LAB

return on innovation

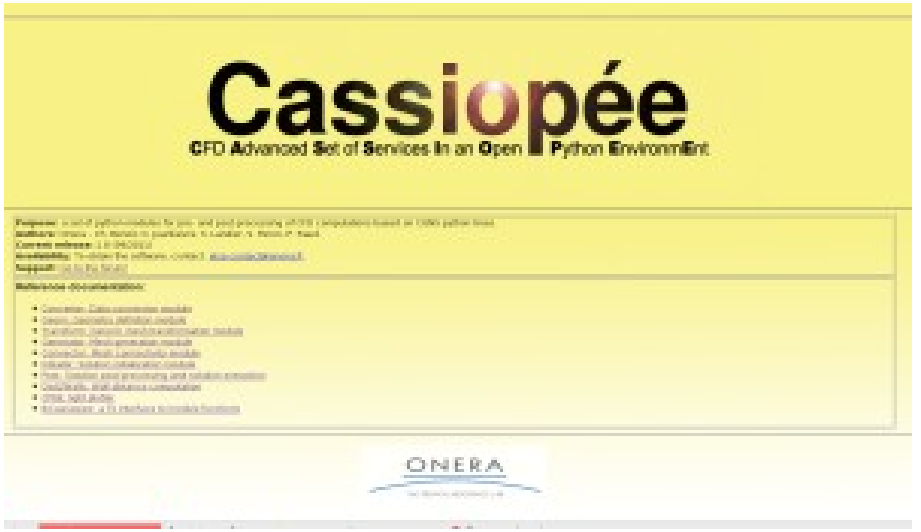
- Cassiopée: a set of open python modules:
  - Based on python/CGNS standard
    - http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/python/sidstopython.pdf
  - Each module compiles and can be installed independently
  - Capitalization of pre- and post-processing functions

- Developed by ONERA (2008-today):
  - Used for CFD, CAA, ...
    - Minor mesh modifications
    - Preparation of computations
    - Code coupling
    - Solution post-processing
  - Used by ONERA, Safran, AIRBUS, EDF

On-line documentation:

http://elsa.onera.fr/Cassiopee/Userguide.html

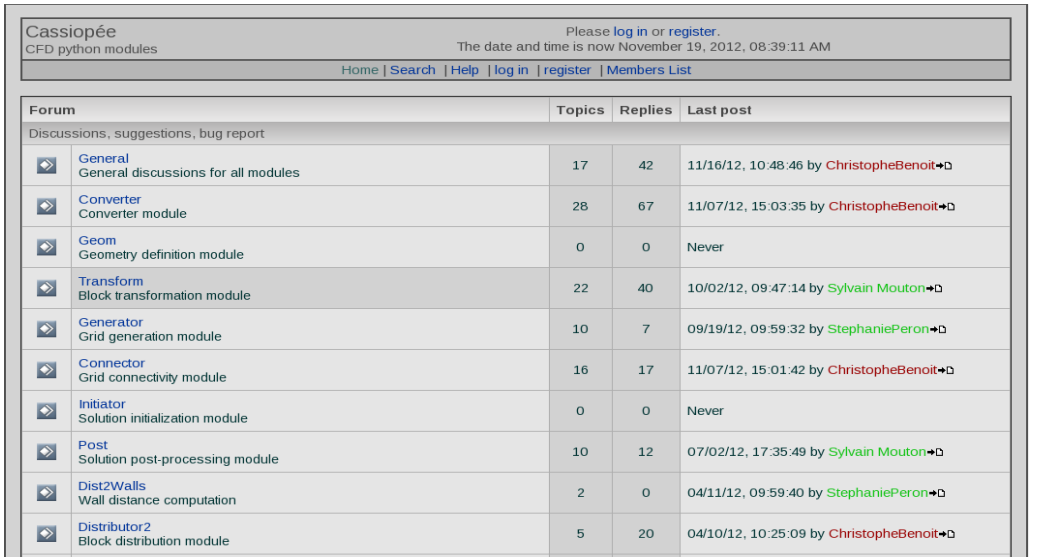
- Reference documentation
- Tutorials
- Gallery



On-line discussion forum:

http://elsa.onera.fr/Cassiopee/Forum/index.php

- News
- Bug reports
- Scripts
- Improvements, suggestions



Cassiopée
CFD python modules

Please log in or register.
The date and time is now November 19, 2012, 08:39:11 AM

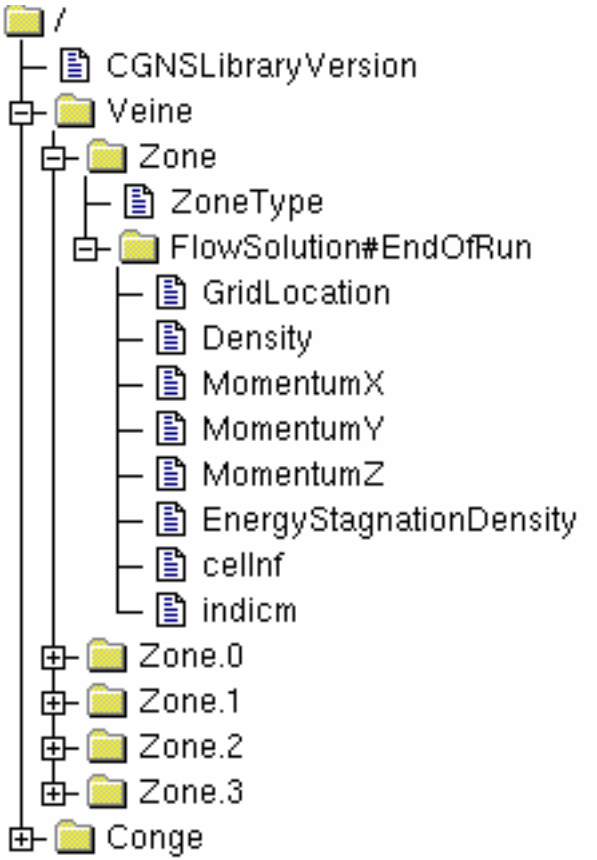
Home | Search | Help | log in | register | Members List

Forum	Topics	Replies	Last post
Discussions, suggestions, bug report			
General General discussions for all modules	17	42	11/16/12, 10:48:46 by ChristopheBenoit
Converter Converter module	28	67	11/07/12, 15:03:35 by ChristopheBenoit
Geom Geometry definition module	0	0	Never
Transform Block transformation module	22	40	10/02/12, 09:47:14 by Sylvain Mouton
Generator Grid generation module	10	7	09/19/12, 09:59:32 by StephaniePeron
Connector Grid connectivity module	16	17	11/07/12, 15:01:42 by ChristopheBenoit
Initiator Solution initialization module	0	0	Never
Post Solution post-processing module	10	12	07/02/12, 17:35:49 by Sylvain Mouton
Dist2Walls Wall distance computation	2	0	04/11/12, 09:59:40 by StephaniePeron
Distributor2 Block distribution module	5	20	04/10/12, 10:25:09 by ChristopheBenoit

- Diffusion:
  - Full version is delivered with ONERA elsA software
  - 95% as Open-source (since dec. 2013)

pyTree

- Full computation data stored in a tree : the **pyTree**
  - Mesh, BCs, fields...
- A **pyTree** is an imbricated python list

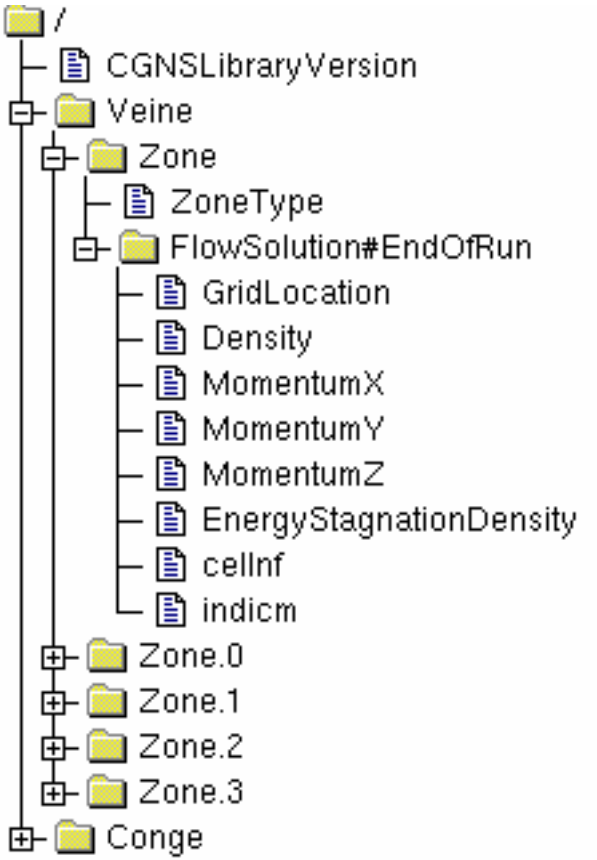


pyTree

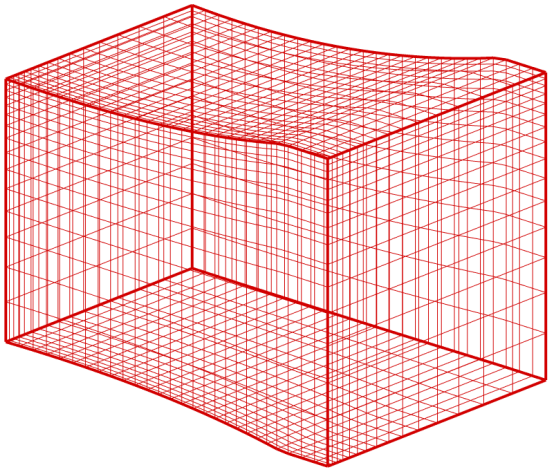
- Cassiopée: a set of functions:

  t' = **f**(t), where t is a **pyTree**

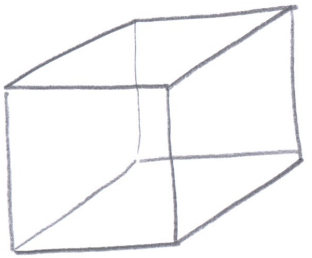
- Each function **f** acts on:
  - Mesh coordinates
  - Connectivity (if relevant)
  - FlowSolutionNodes (if any)
  - FlowSolutionCenters (if any)
  - BCs (if relevant)



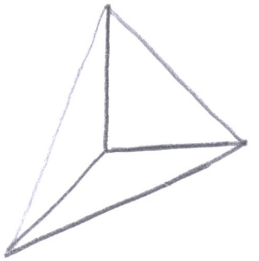
Mesh types



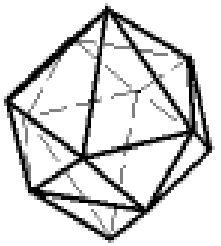
STRUCT



HEXA

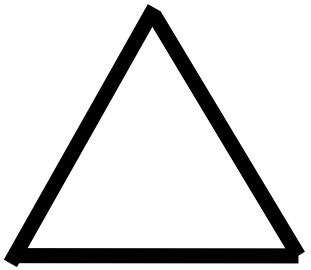


TETRA

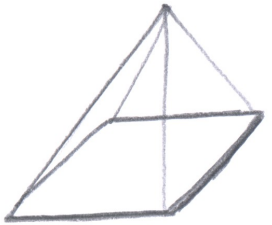


NGON

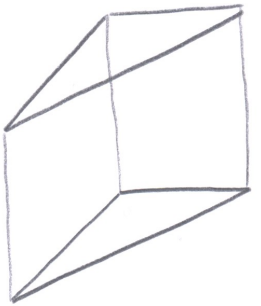
BAR



TRI



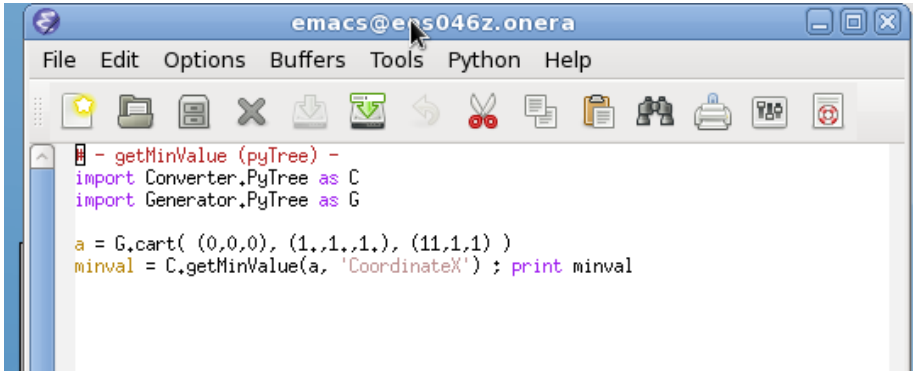
PYRA

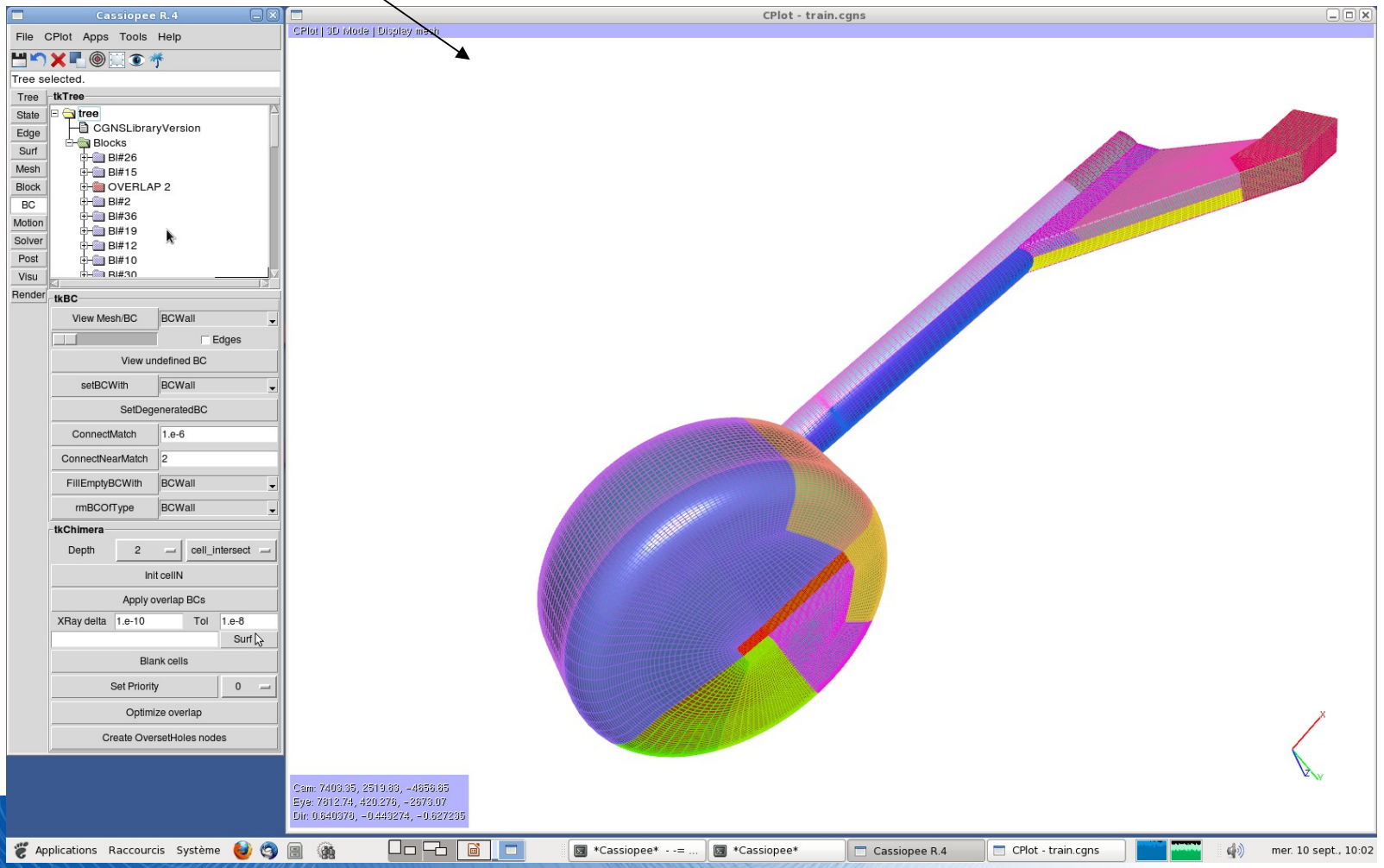


PENTA

QUAD

- Usage:
  - Python scripts
  - GUI (tkCassiopee)

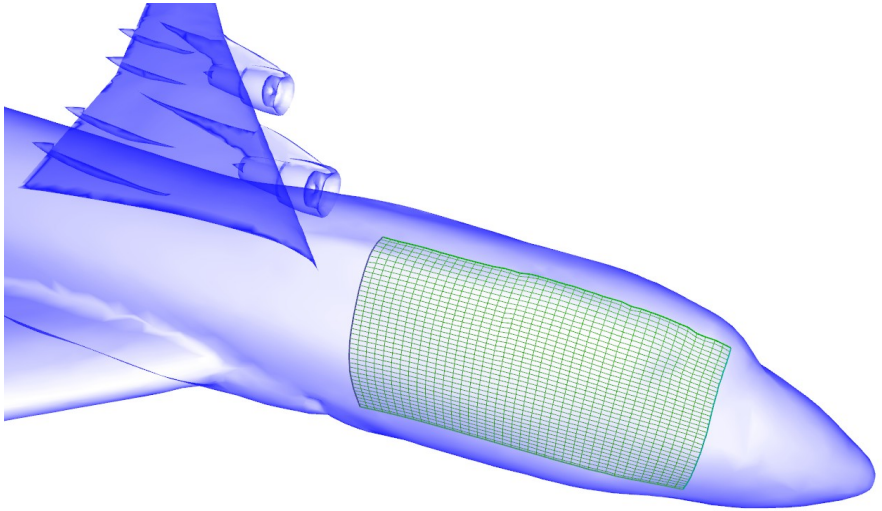


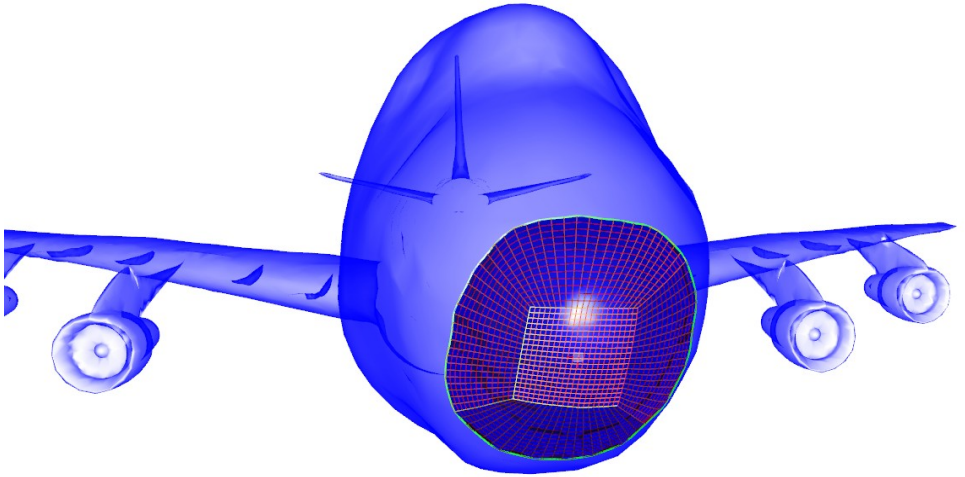


List of Cassiopée modules

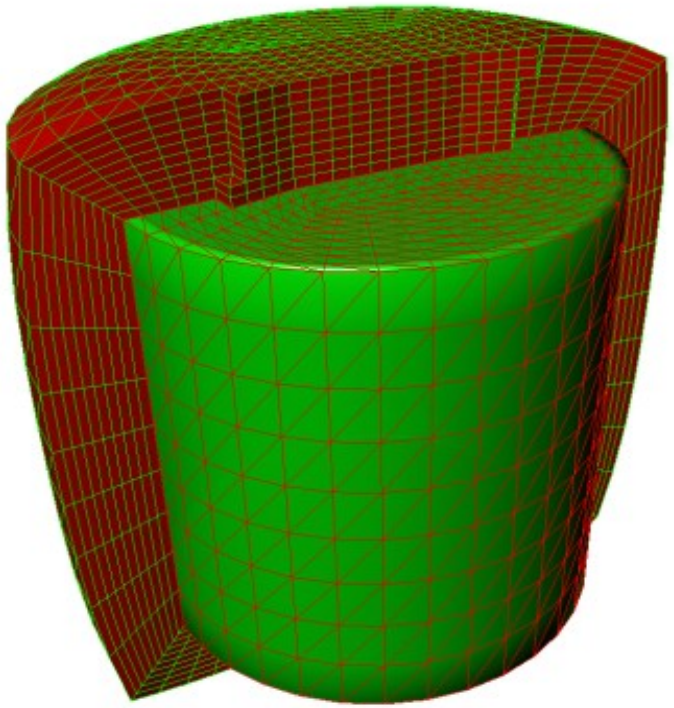
- Converter/Internal [C]: conversion / handling of arrays / pyTrees
- Geom [D]: geometry/surface definition functions
- Generator [G]: mesh generation functions
- Transform [T]: mesh transformation functions
- Post [P]: CFD solution post-processing functions
- Initiator [I]: solution initialization functions
- Connector [X]: connectivity computation
- Dist2Walls [DTW]: distance to walls computation
- Distributor2 [D2]: load balance functions
- RigidMotion [R]: rigid motion definition
- CPlot [CPlot]: graphic display of pyTrees

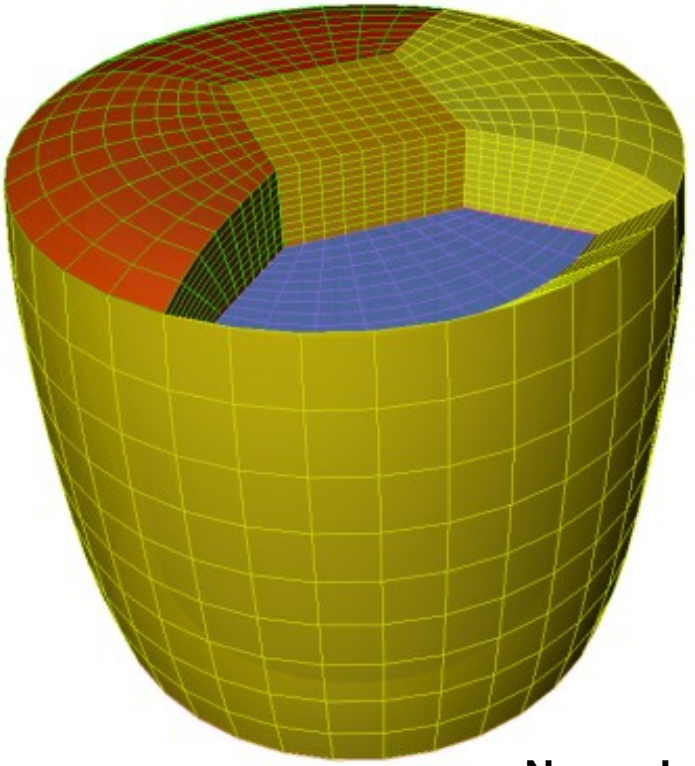
Generator module [G]





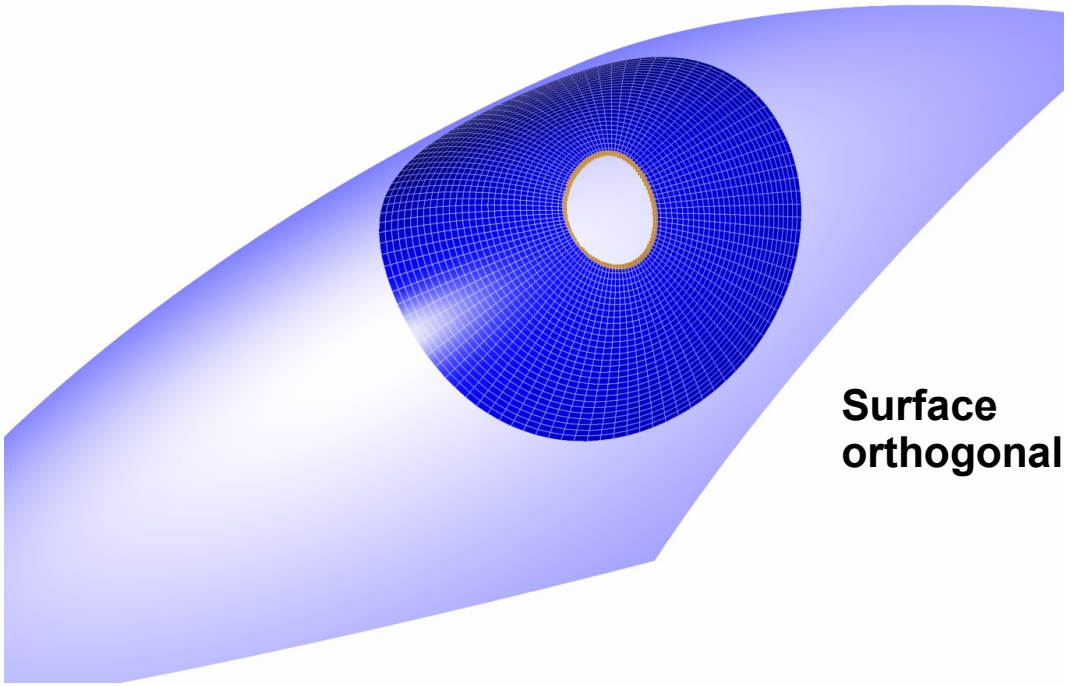
TFIs



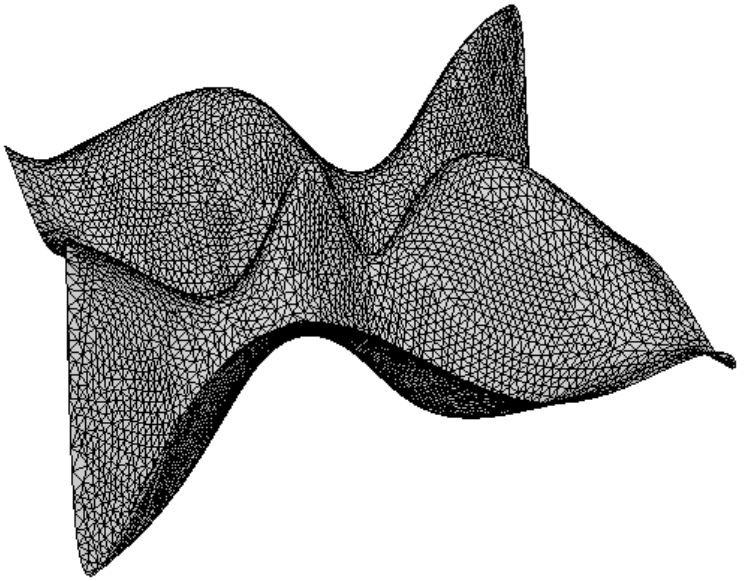


Normal extrusion

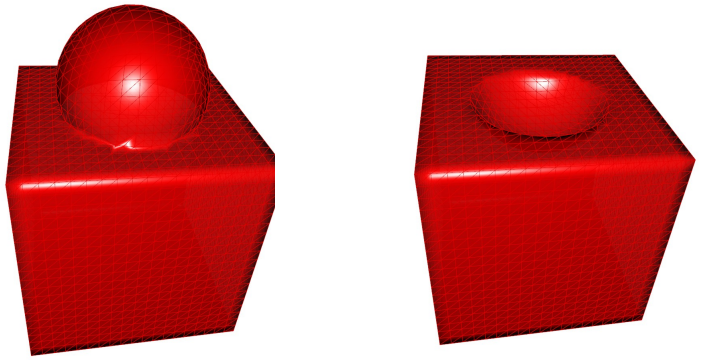
Generator module [G]



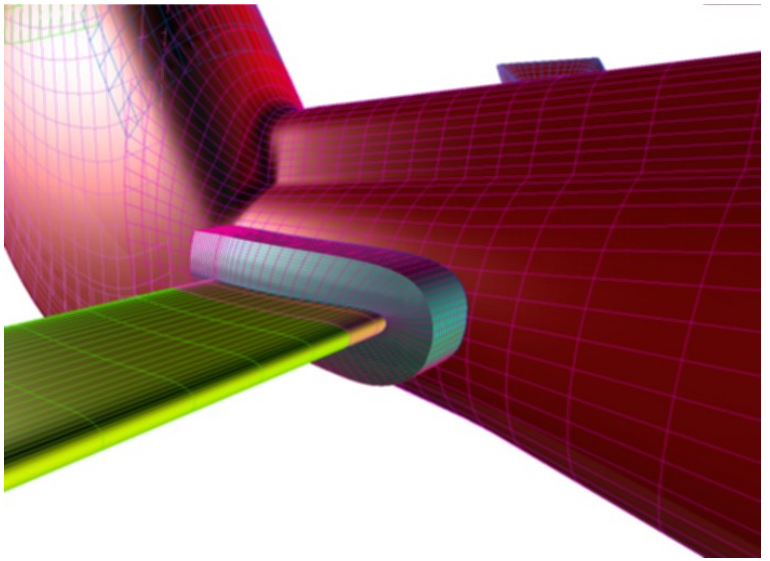
Surface
orthogonal walk



2D Delaunay

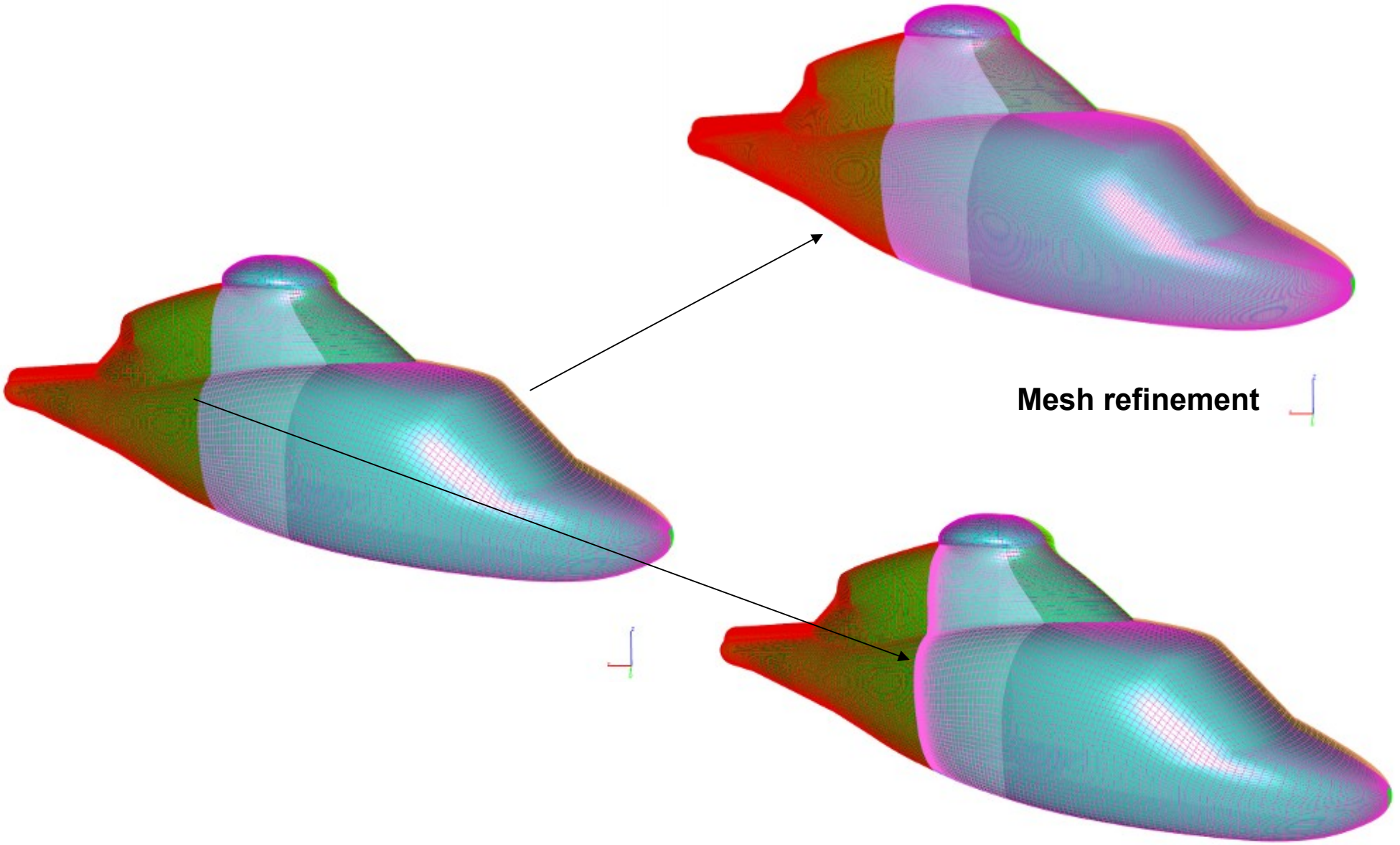


Boolean operators on surfaces

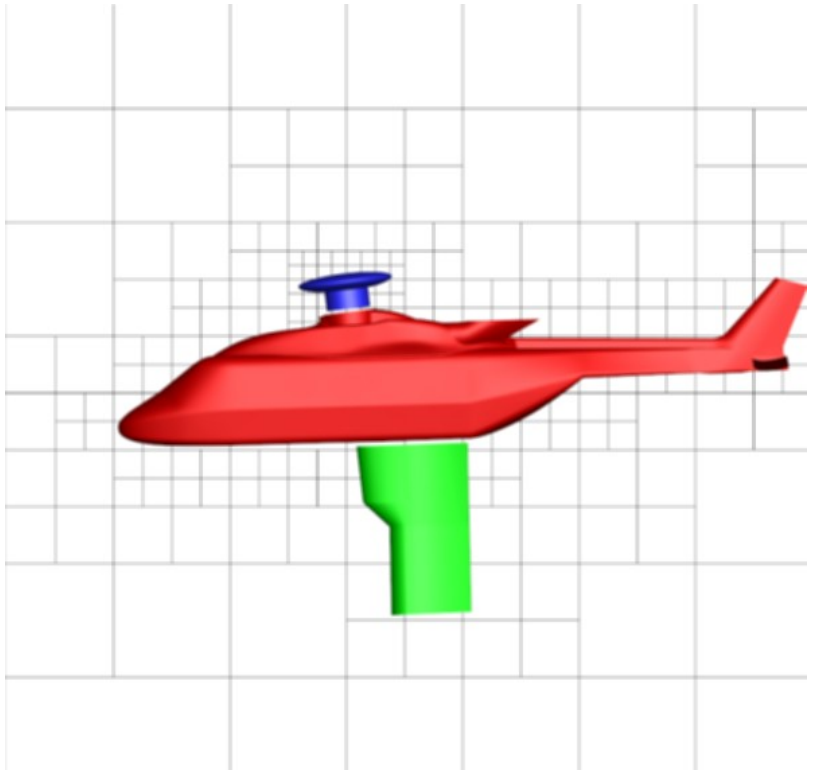


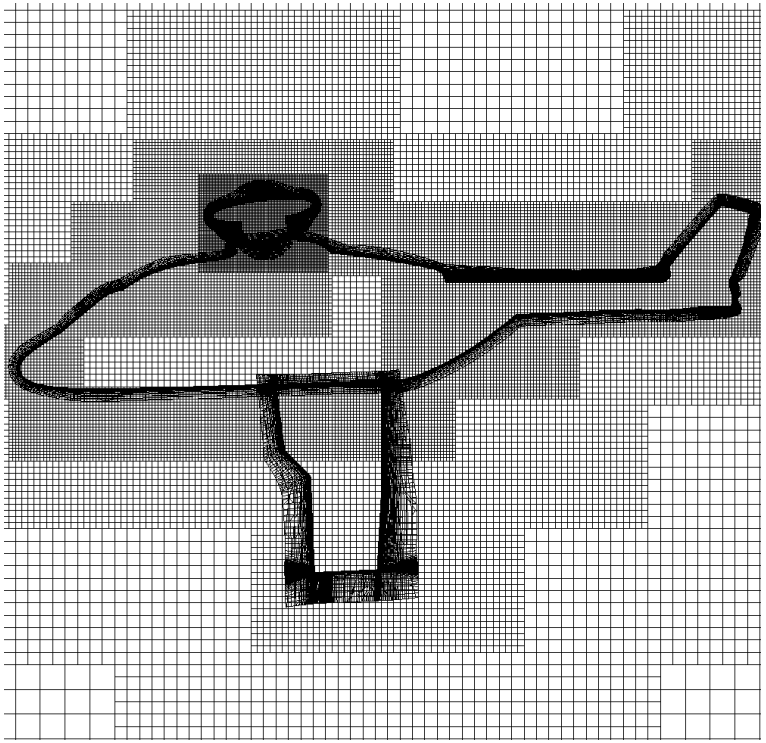
Collar grids

Generator module [G]



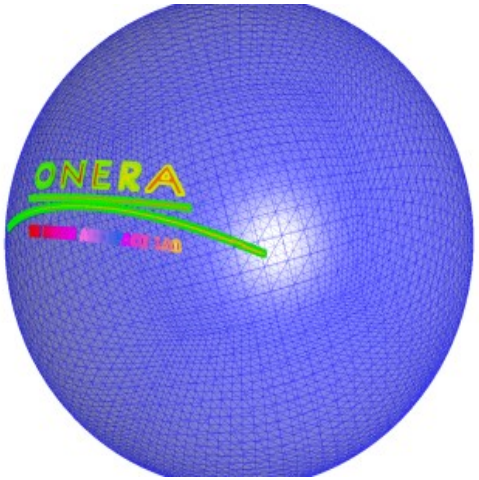
Generator module [G]



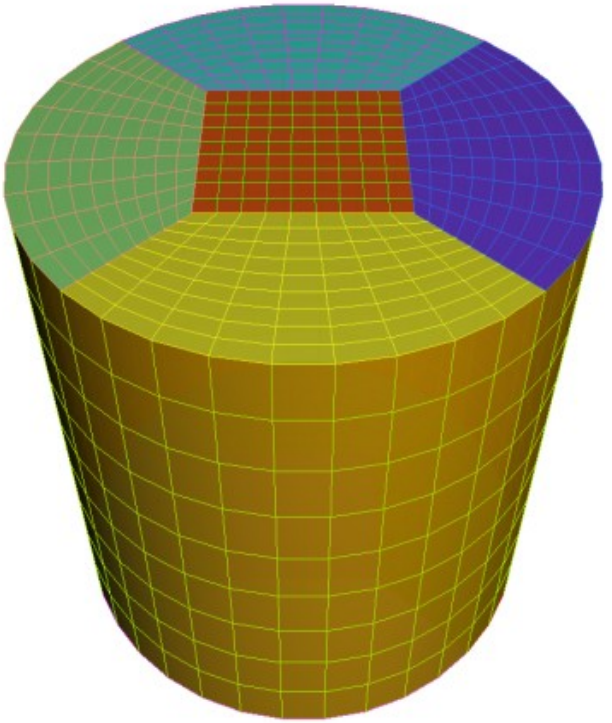


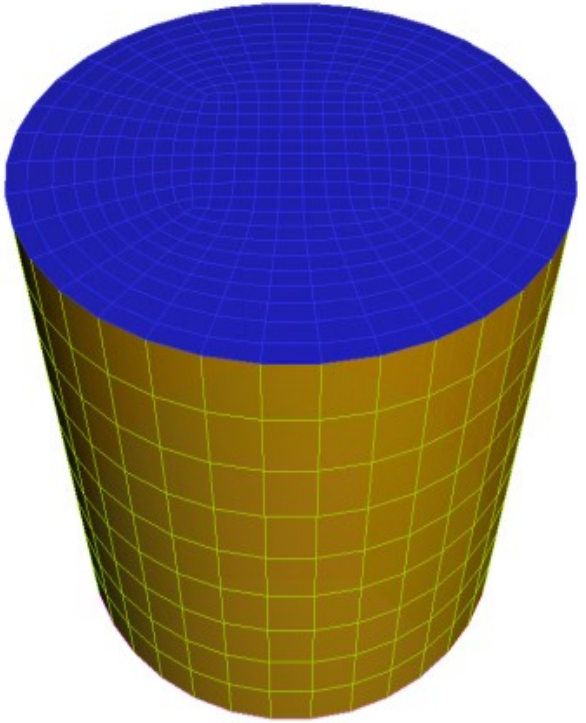
Octrees (generation/adaptation) → **Set of structured Cartesian grids**

Transform module [T]

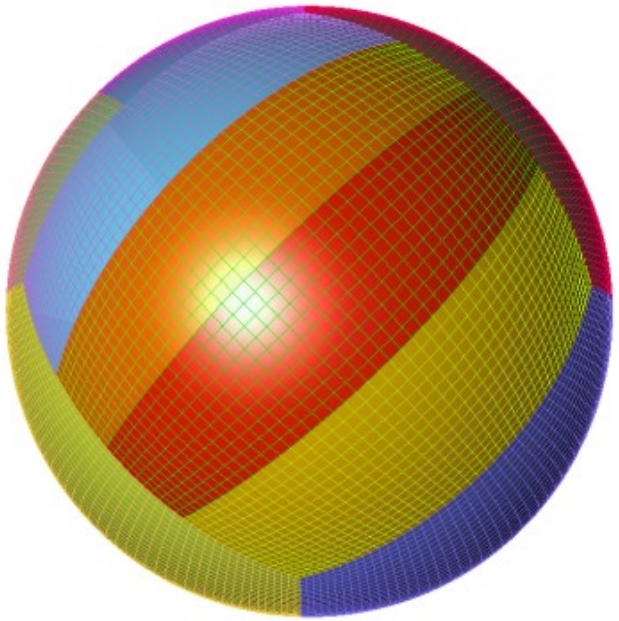


Projections

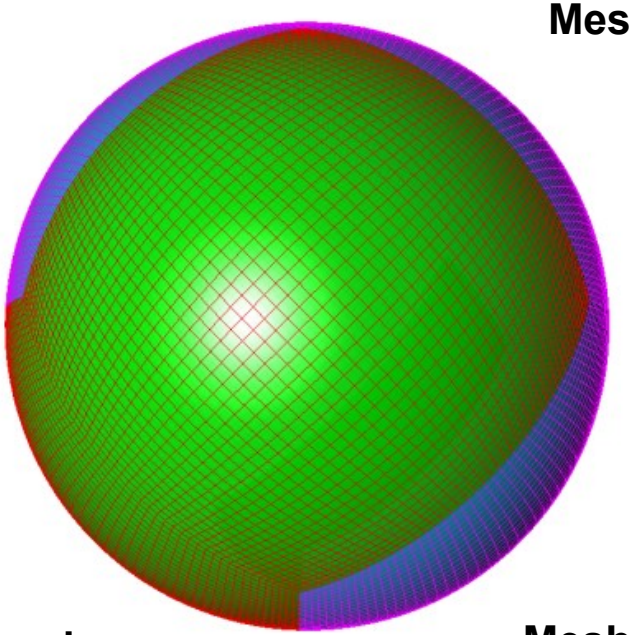


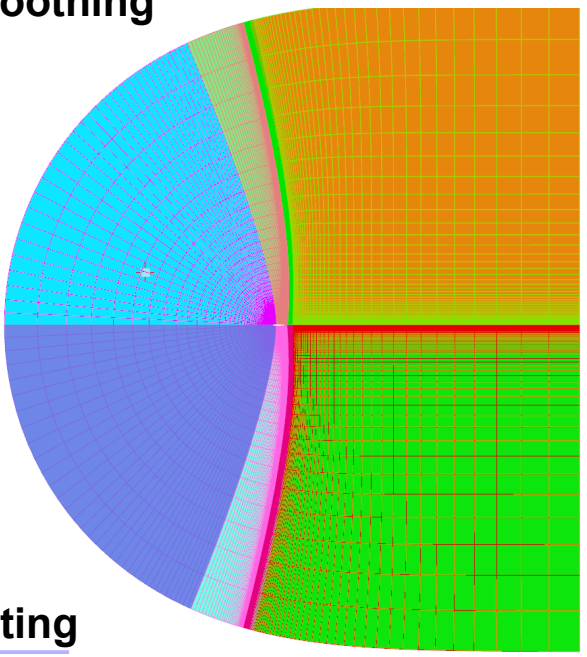


Mesh smoothing



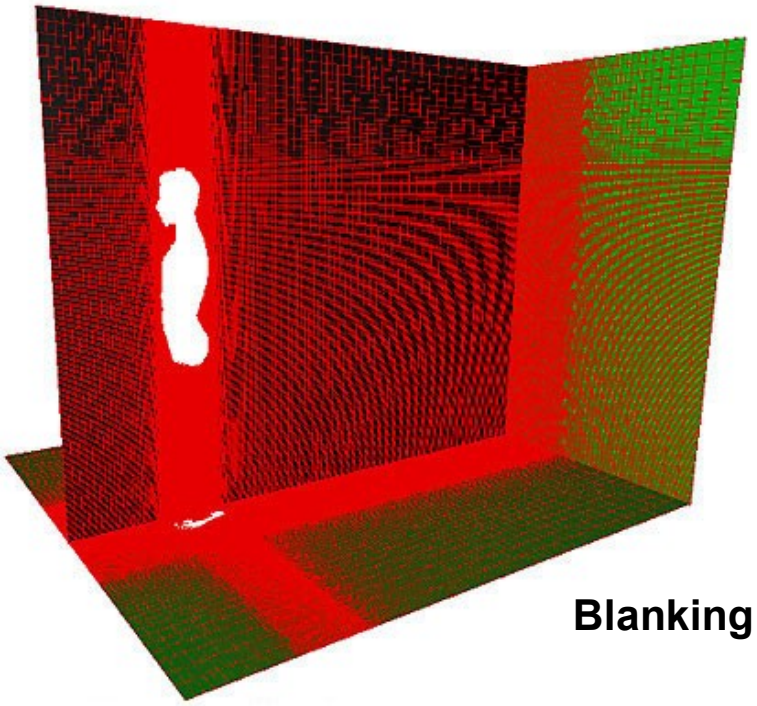
Mesh merging



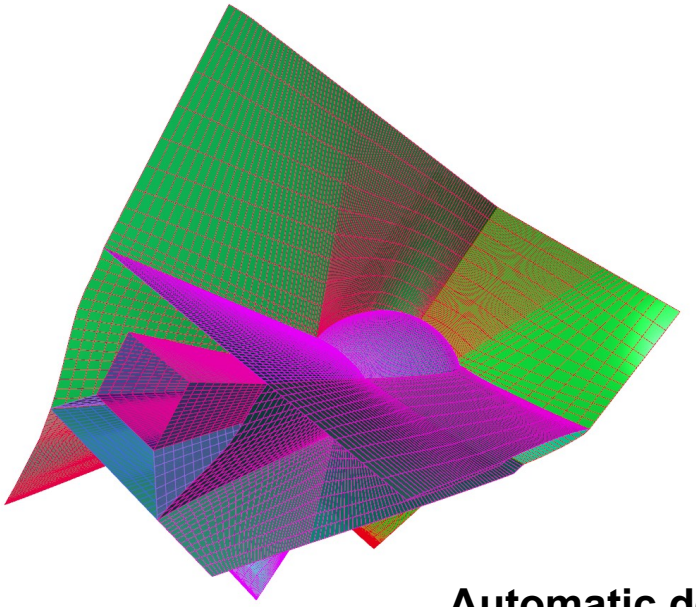


Mesh splitting

Connector module [X]

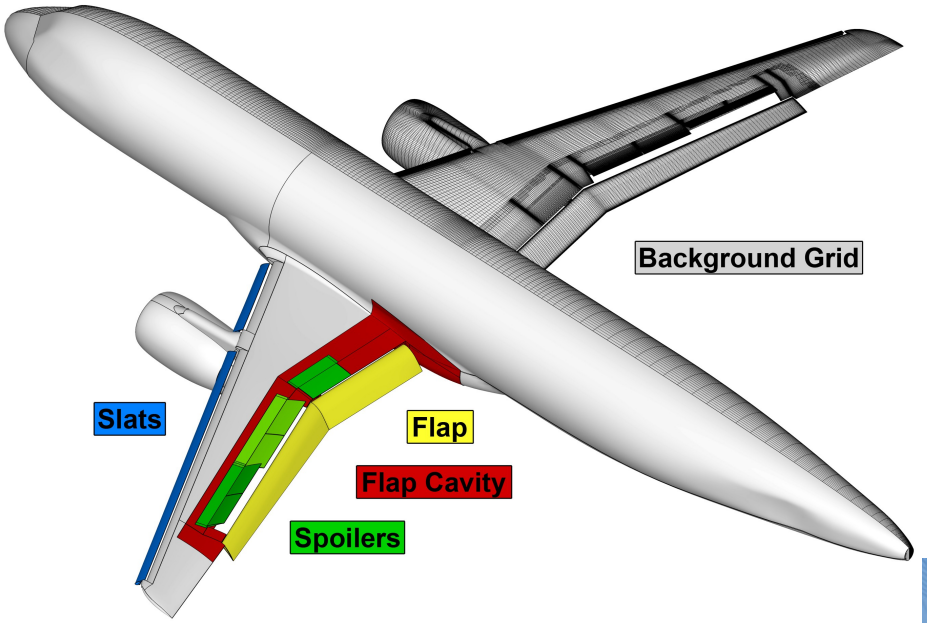


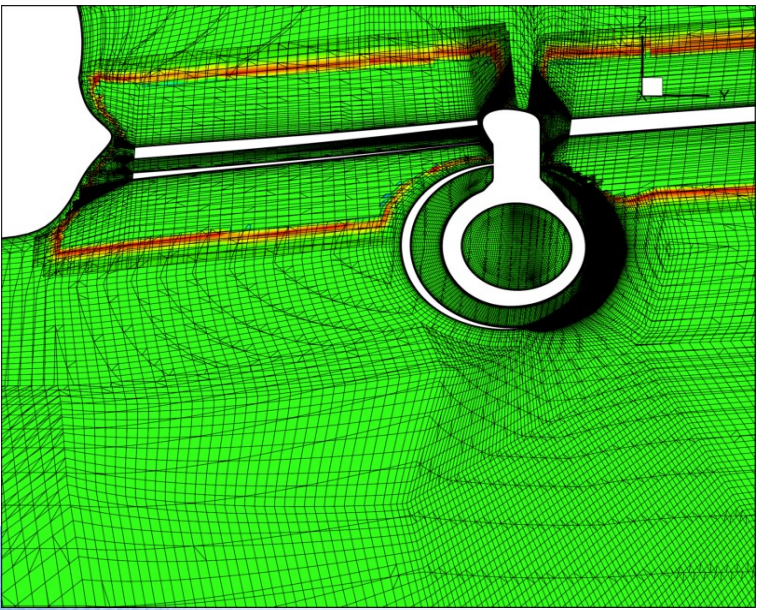
Blanking



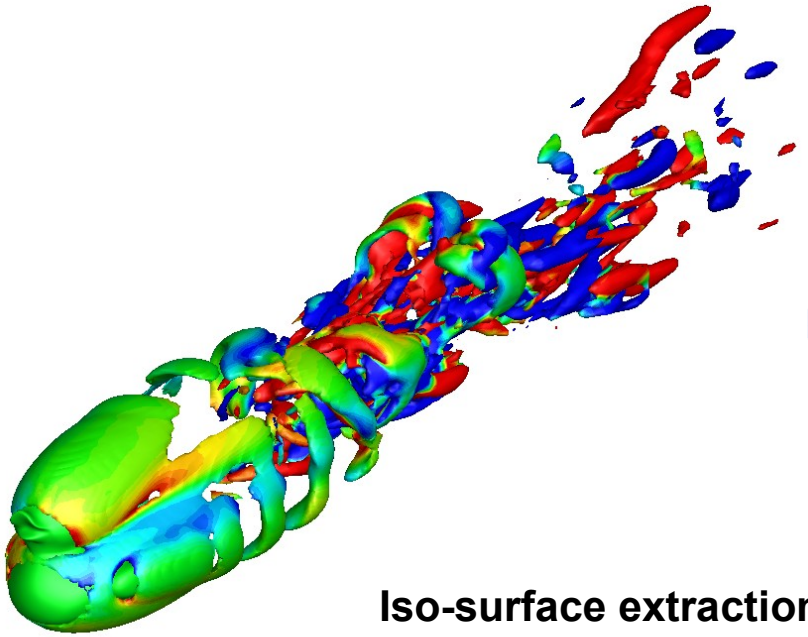
Automatic detection of matching boundaries

Overset grid connectivity

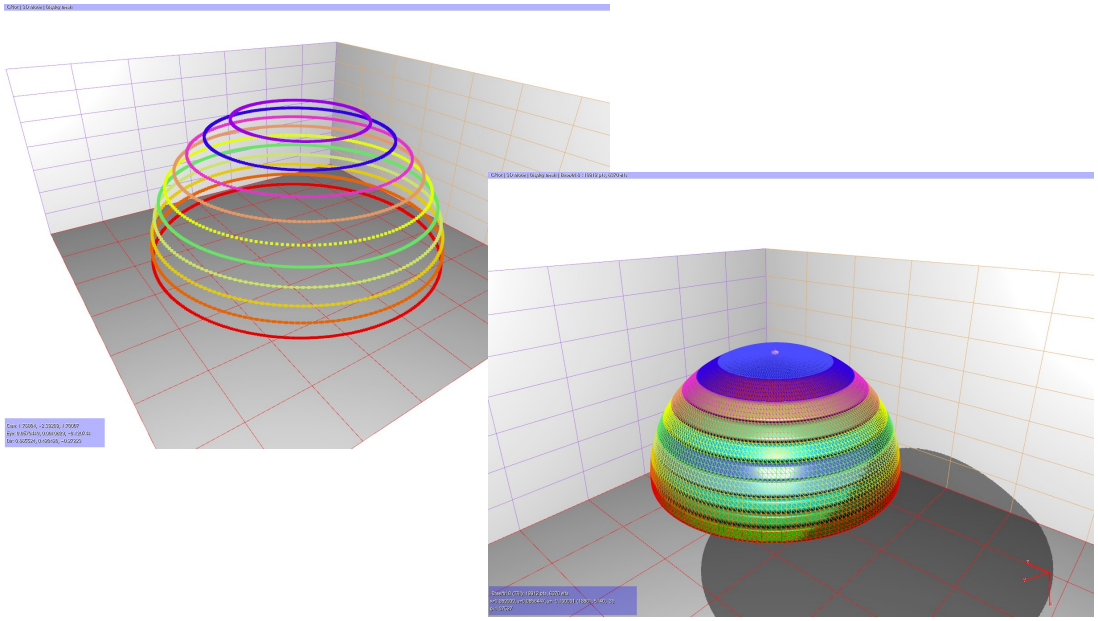




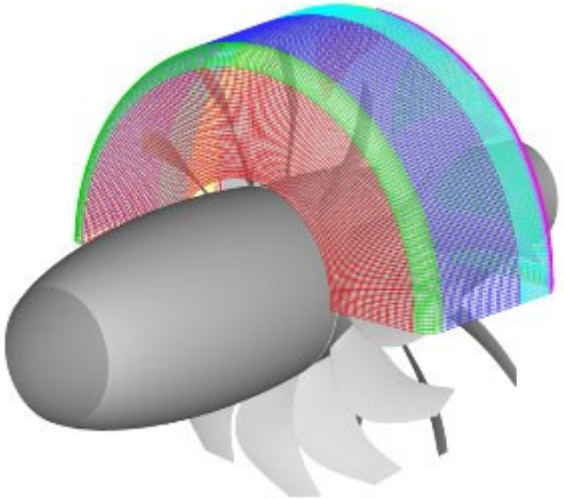
Post module [P]

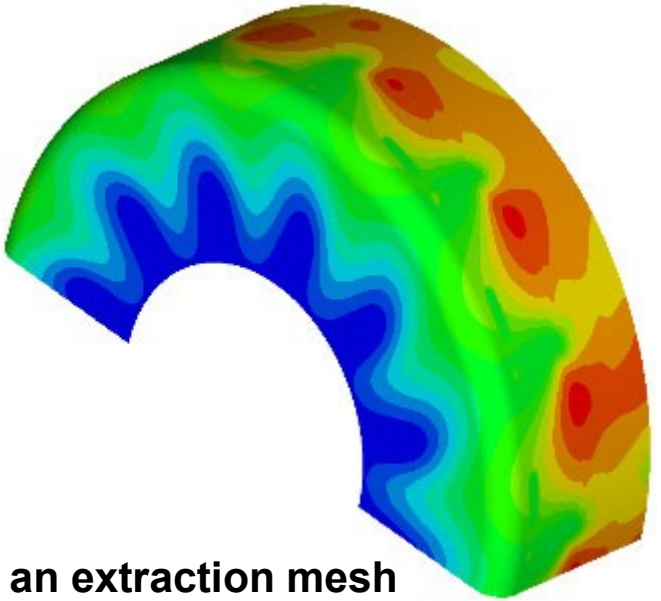


Iso-surface extraction



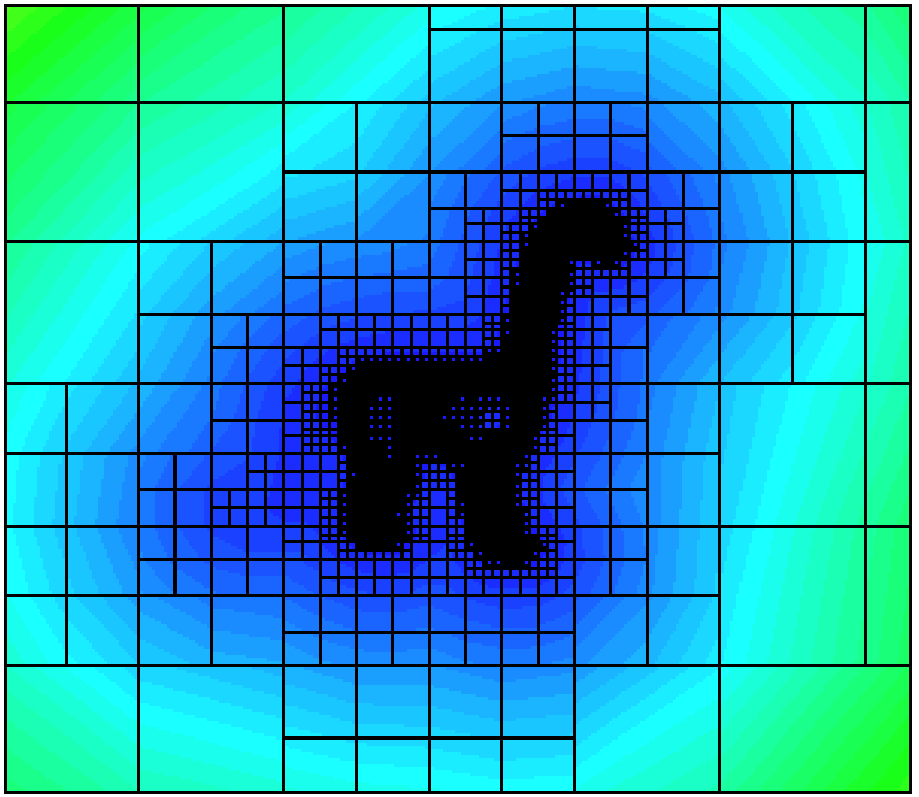
Field integration on curves/slices/surfaces





Interpolation on an extraction mesh

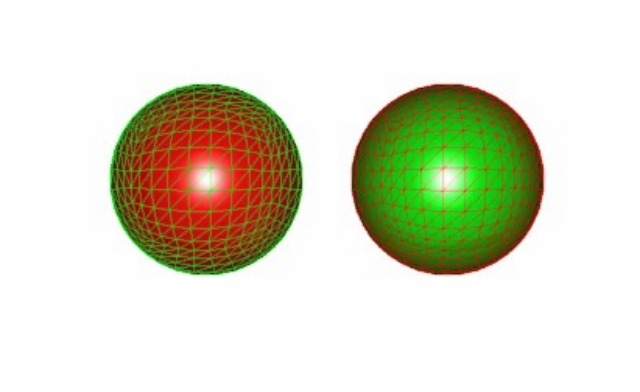
Dist2Walls module [DTW]



Signed distance field
(turbulent distance, level set)

Application to surface offset

Surface offset

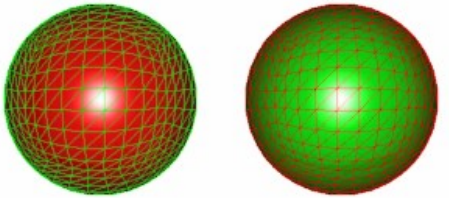


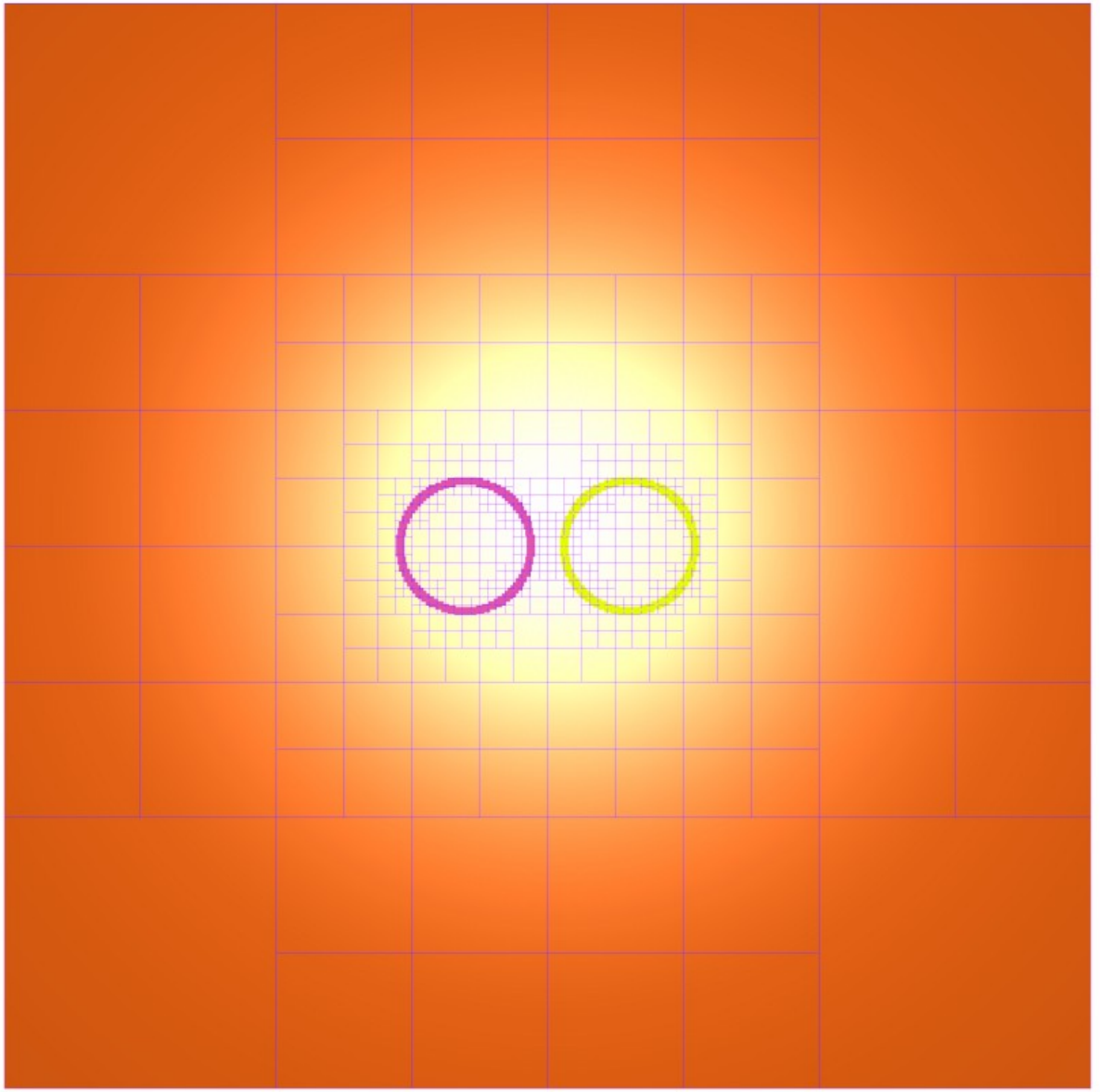
=> Useful for blanking

Surface s

↓

Unstructured octree
`o=G.octree(s)`

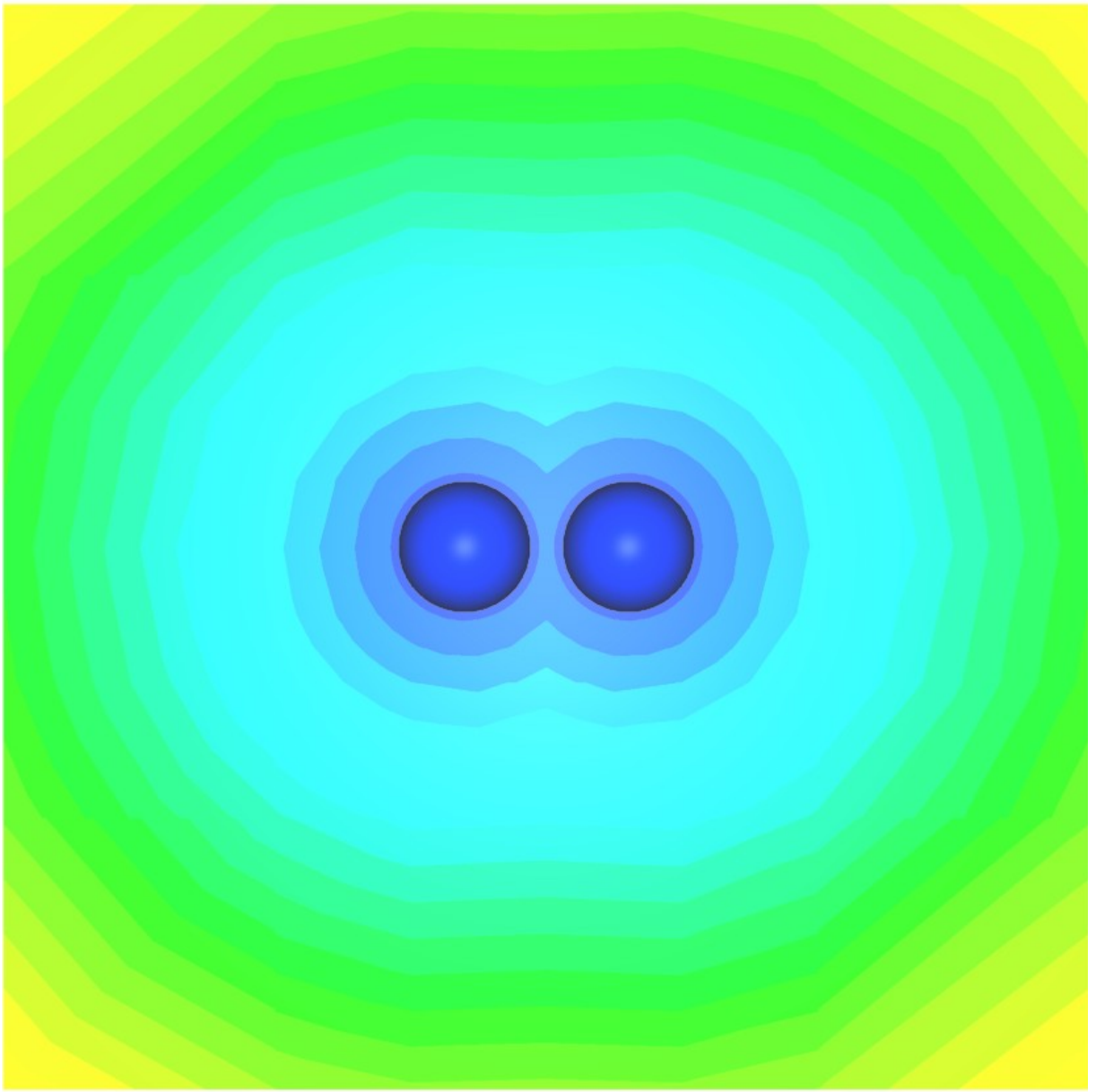




Surface s

Unstructured octree
`o=G.octree(s)`

Wall distance
`o=DTW.distance2Wall(o,s)`



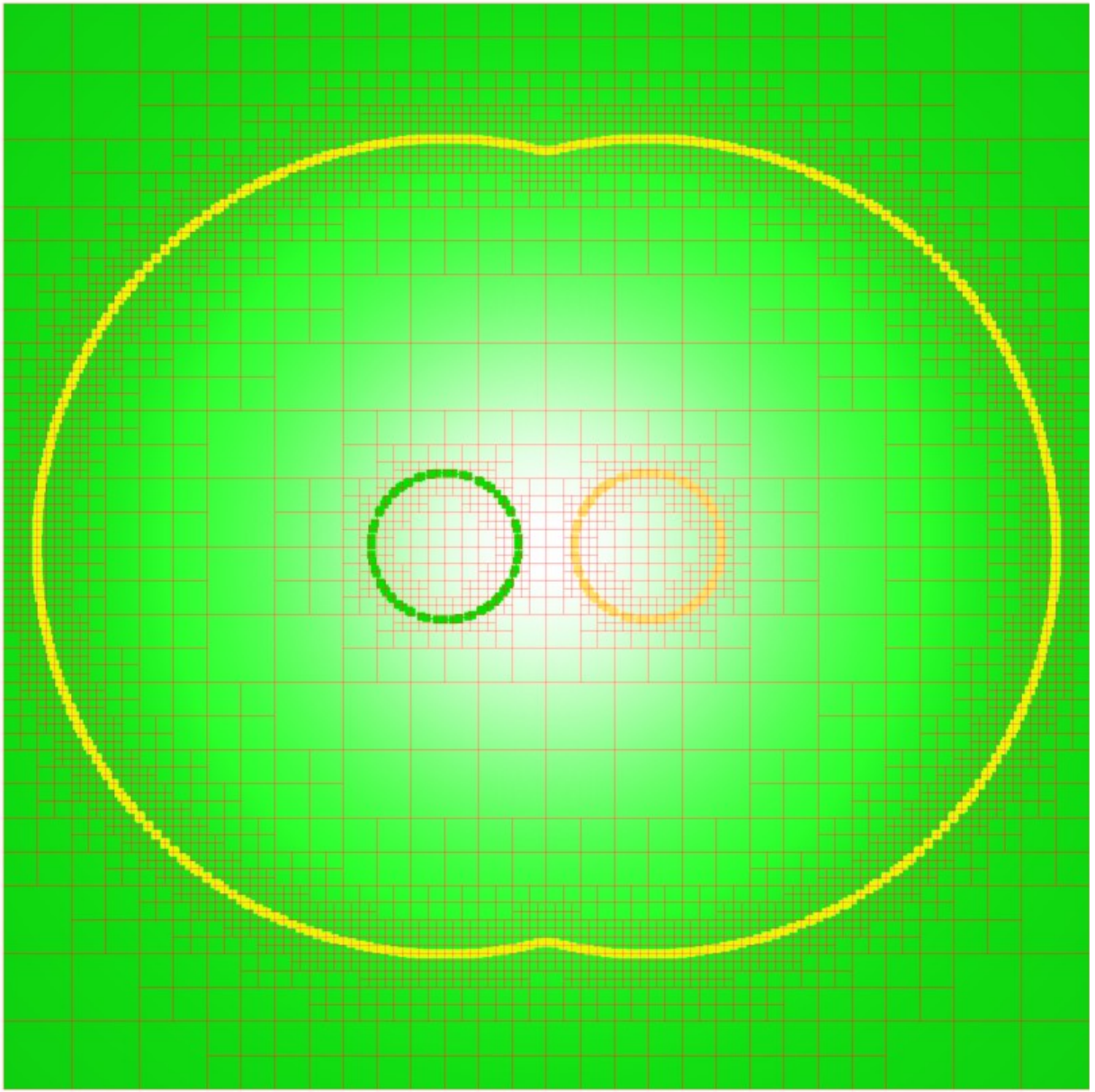
Surface s

Unstructured octree
`o=G.octree(s)`

Wall distance
`o=DTW.distance2Wall(o,s)`

Indicator field
`o=C.initVars(o,formula)`

Octree adaptation
`o=G.adaptOctree(o)`



Surface s

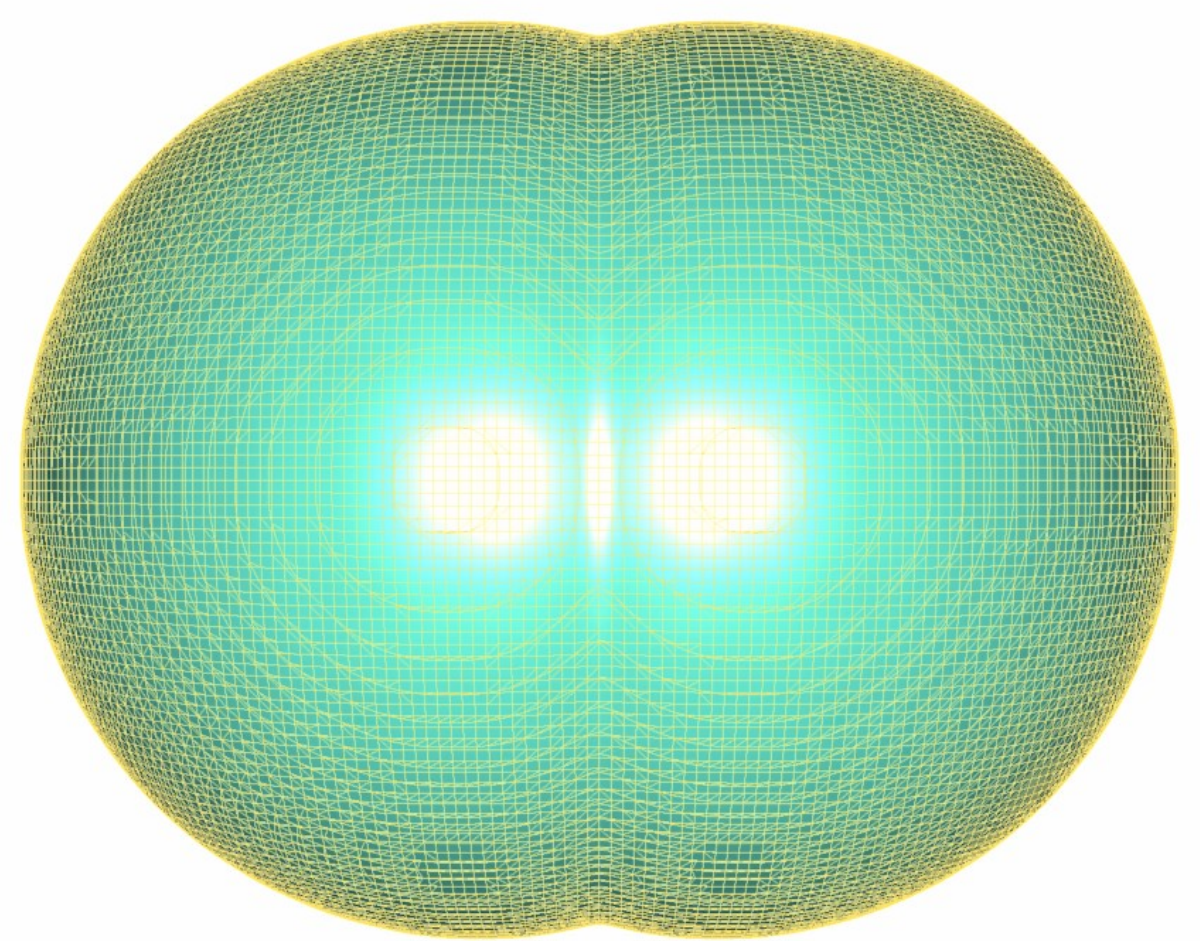
Unstructured octree
`o=G.octree(s)`

Wall distance
`o=DTW.distance2Wall(o,s)`

Indicator field
`o=C.initVars(o,formula)`

Octree adaptation
`o=G.adaptOctree(o)`

Isosurface d=offset
`s2=P.isoSurfMC(o,dist=d)`



Surface s

Unstructured octree
`o=G.octree(s)`

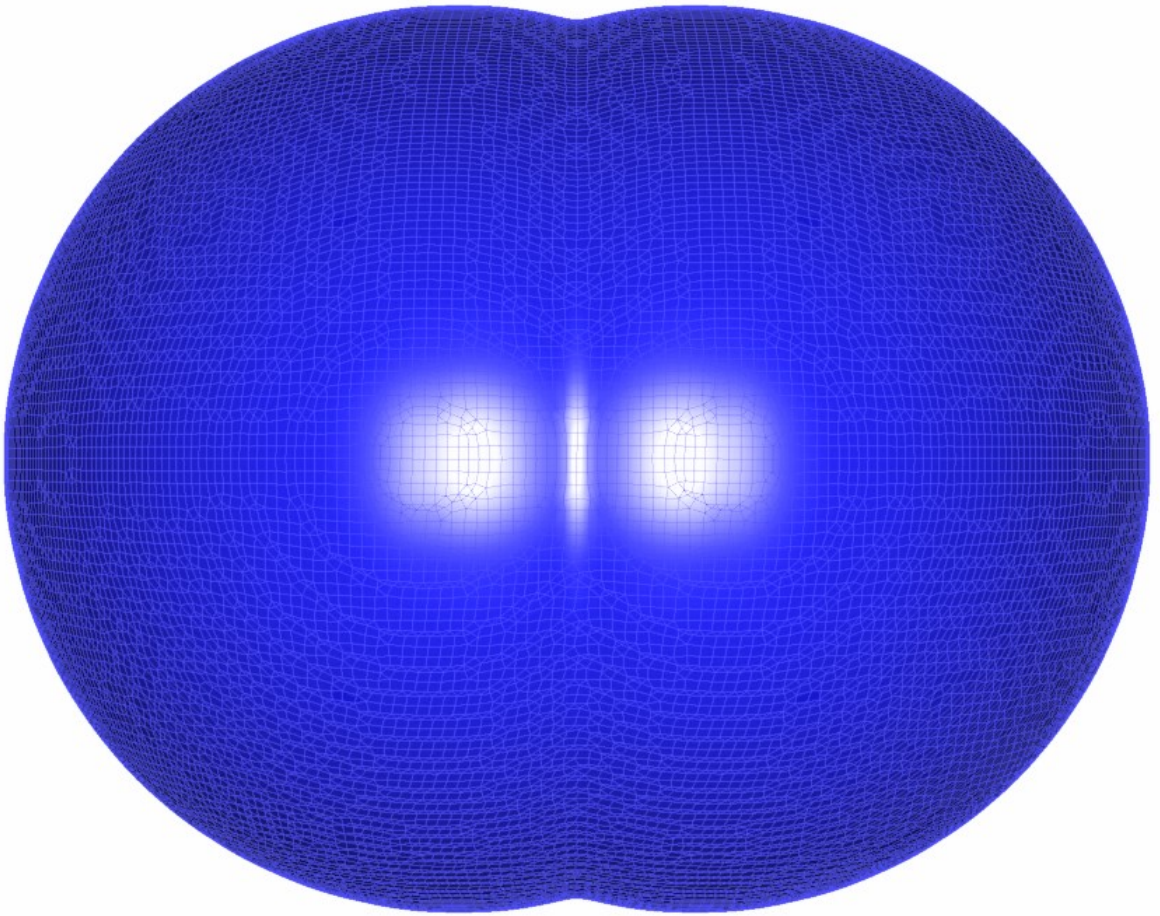
Wall distance
`o=DTW.distance2Wall(o,s)`

Indicator field
`o=C.initVars(o,formula)`

Octree adaptation
`o=G.adaptOctree(o)`

Isosurface d=offset
`s2=P.isoSurfMC(o,dist=d)`

Mesh smoothing
`s2=T.smooth(s2)`



Overset connectivity

- Overset grid assembly can be performed using « simple » separated functions :
  - Blanking
  - Overlap optimization
  - Interpolation coefficients and donor search
  - Transfer of the solution

Overset connectivity

- A field cellN (located at nodes or centers) is used to mark points as:
  - computed (cellN=1)
  - interpolated (cellN=2)
  - blanked (cellN=0)

- Finally, overset connectivity data is stored in the pyTree (donors,receivers, interpolation method, order, coefficients)

Overset connectivity

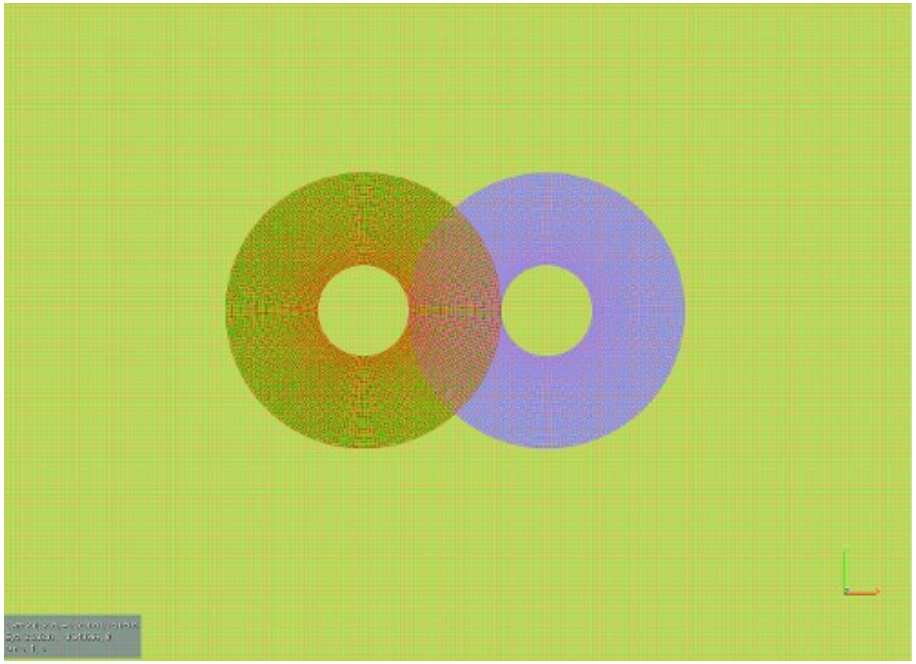
Implemented techniques:

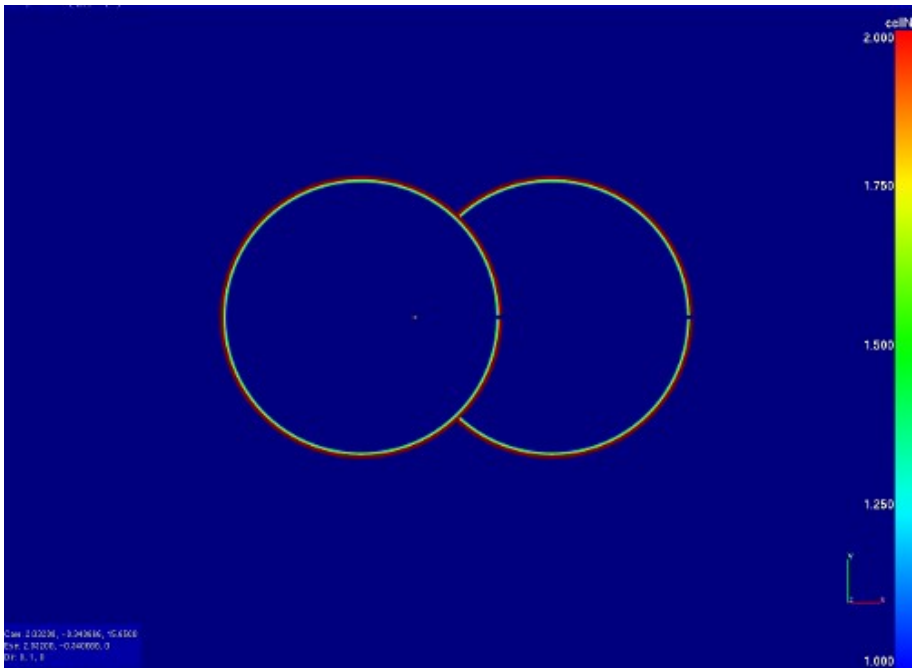
- Blanking with Object X-Rays (Meakin)
- Blanking with TETRA volumes
- Overlap optimization: PEGASUS algorithm
- Projection for interpolated wall points on grid surfaces
- Interpolations :
  - 2nd order, 3rd and 5th orders (Lagrangian) for structured grids
  - Moving-Least Squares (3rd order currently) for all grid types

Mark cellN=2 for **d** points at overlap borders

t=X.applyBCOverlaps(t,d)

2 near-body grids around cylinders, off-body Cartesian grid





Interpolated (cellN=2)
Computed (cellN=1)

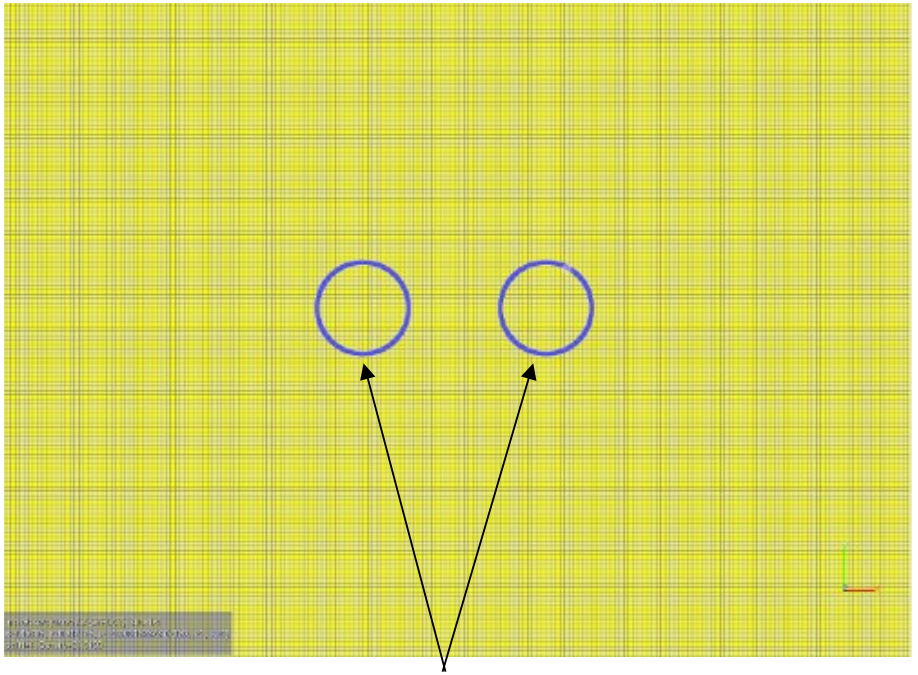
Mark cellN=2 for **d** points at overlap borders

`t=X.applyBCOverlaps(t,d)`

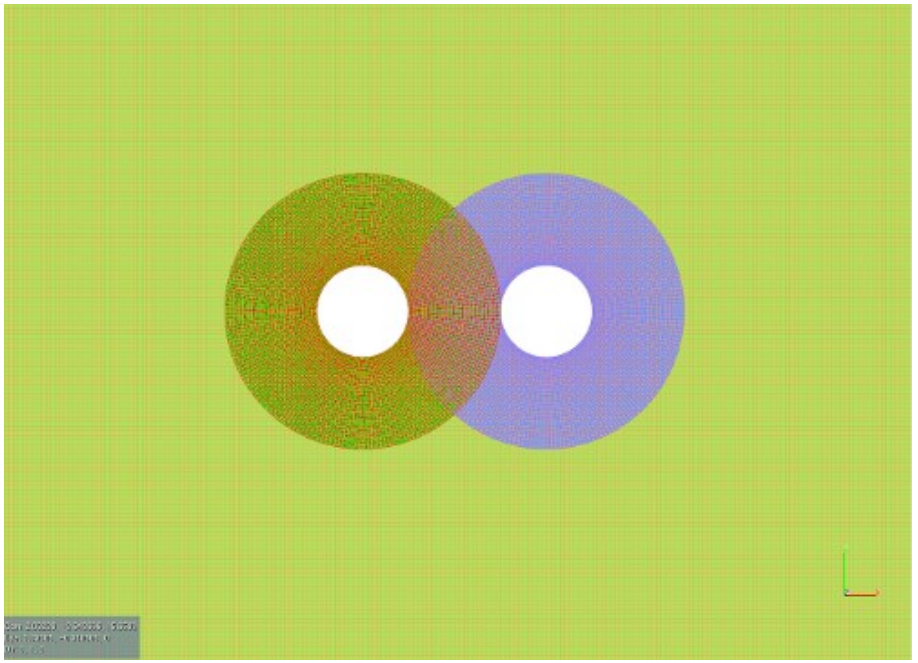
↓

Blank mesh w.r.t bodies and assembly rules

`t=X.blankCells(t,bodies,AM)`



Bodies



Mark cellN=2 for **d** points at overlap borders

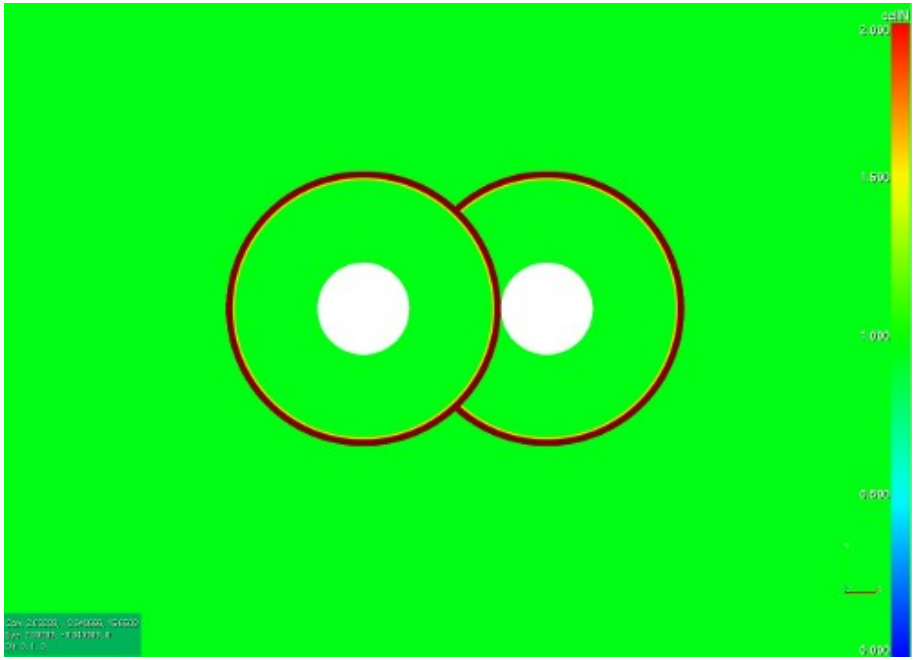
`t=X.applyBCOverlaps(t,d)`

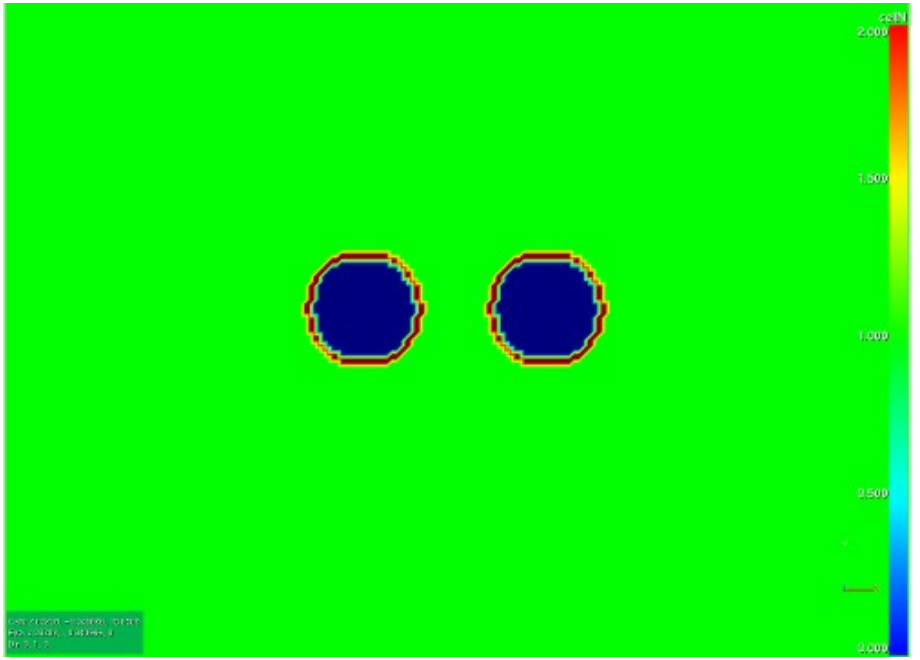
Blank mesh w.r.t bodies and assembly rules

`t=X.blankCells(t,bodies,AM)`

Mark **d** fringe points around blanked points

`t=X.setHoleInterpolatedPts(t,d)`





Blanked (cellN=0)
Interpolated (cellN=2)
Computed (cellN=1)

Mark cellN=2 for **d** points at overlap borders

`t=X.applyBCOverlaps(t,d)`

Blank mesh w.r.t bodies and assembly rules

`t=X.blankCells(t,bodies,AM)`

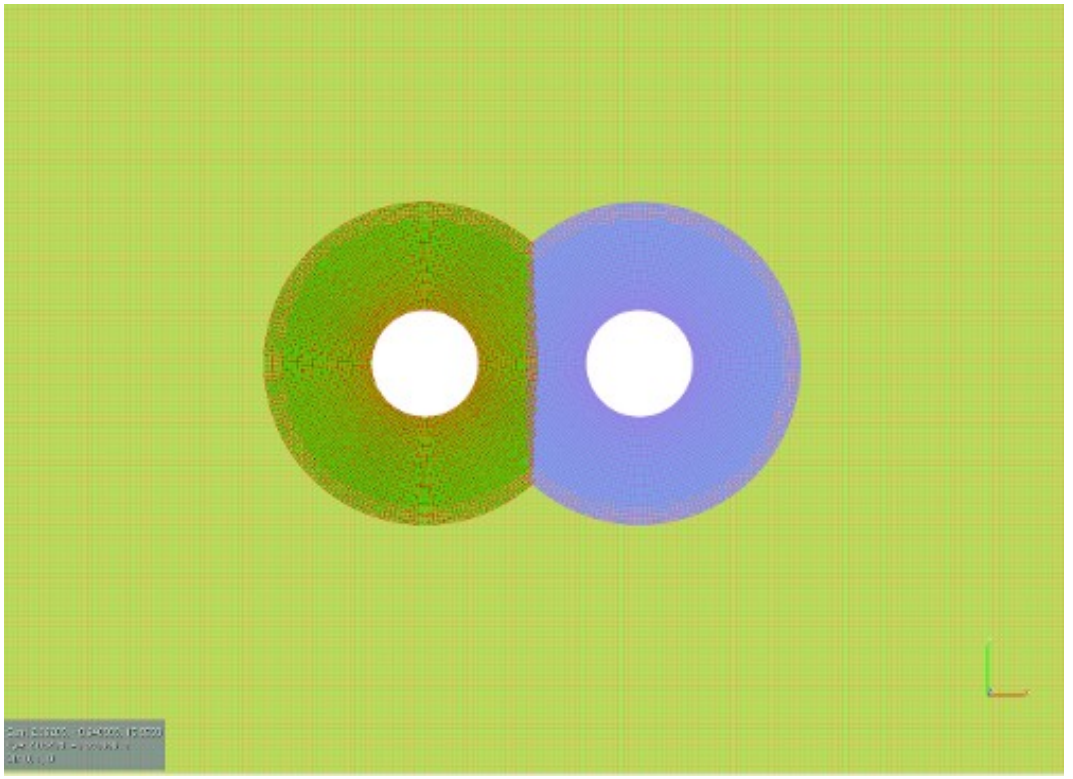
Mark **d** fringe points around blanked points

`t=X.setHoleInterpolatedPts(t,d)`

Overlap optimization (d layers of interpolated points)

`t=X.optimizeOverlap(t)`

`t=X.maximizeBlankedCells(t,d)`



Mark cellN=2 for **d** points at overlap borders

`t=X.applyBCOverlaps(t,d)`

Blank mesh w.r.t bodies and assembly rules

`t=X.blankCells(t,bodies,AM)`

Mark **d** fringe points around blanked points

`t=X.setHoleInterpolatedPts(t,d)`

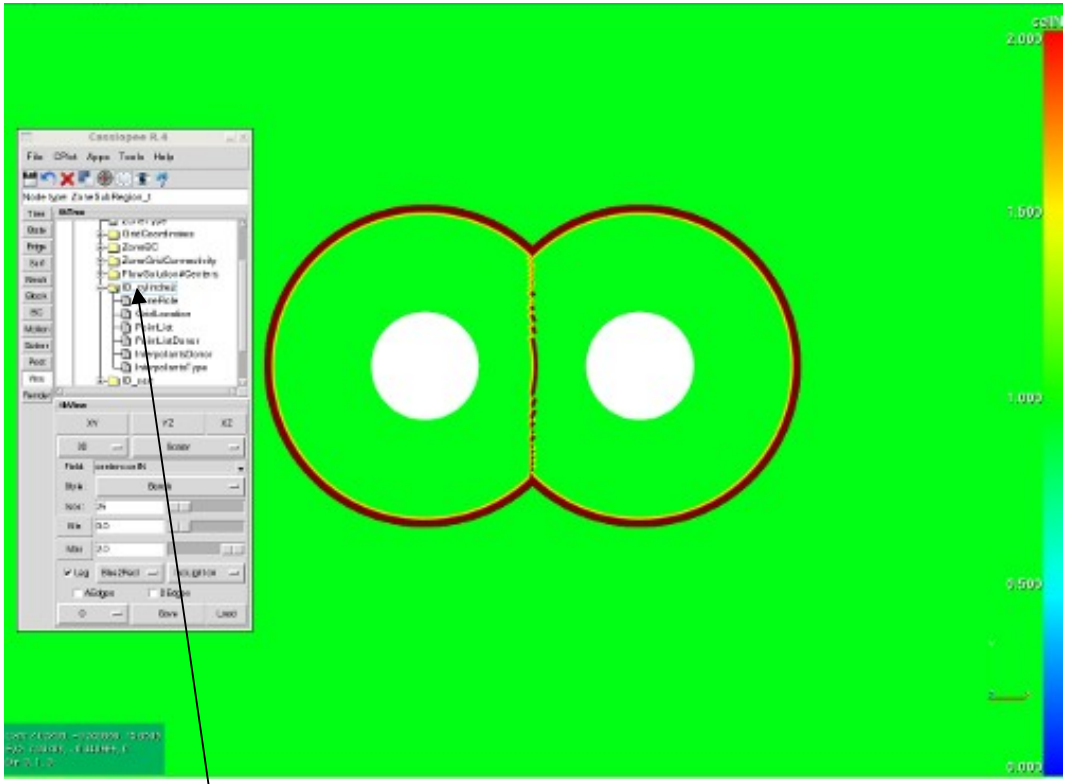
Overlap optimization (d layers of interpolated points)

`t=X.optimizeOverlap(t)`

`t=X.maximizeBlankedCells(t,d)`

Computes overset connectivity

`t=X.setInterpData(t,...)`



Interpolation data stored in the pyTree as ZoneSubRegion_t nodes

Mark cellN=2 for **d** points at overlap borders
`t=X.applyBCOverlaps(t,d)`

Blank mesh w.r.t bodies and assembly rules
`t=X.blankCells(t,bodies,AM)`

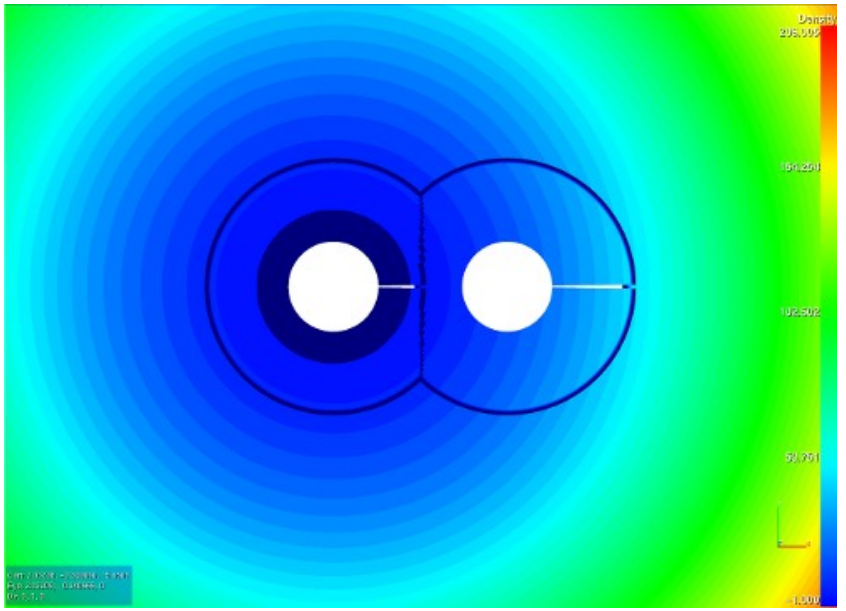
Mark **d** fringe points around blanked points
`t=X.setHoleInterpolatedPts(t,d)`

Overlap optimization (d layers of interpolated points)
`t=X.optimizeOverlap(t)`
`t=X.maximizeBlankedCells(t,d)`

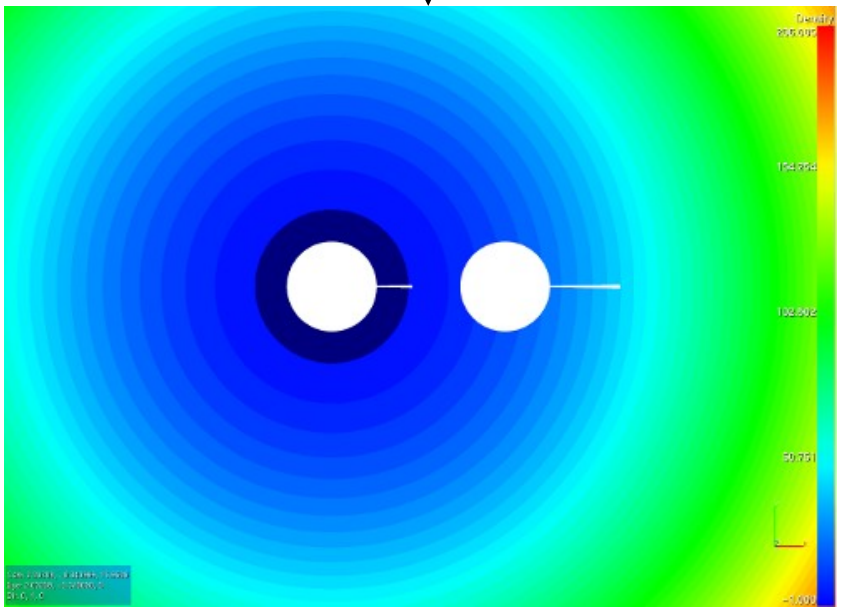
Computes overset connectivity
`t=X.setInterpData(t,...)`

Transfers
`t=X.setInterpTransfer(t,...)`

Test field: F=x**2+y**2 if cellN=1, else 0



Transfer of F



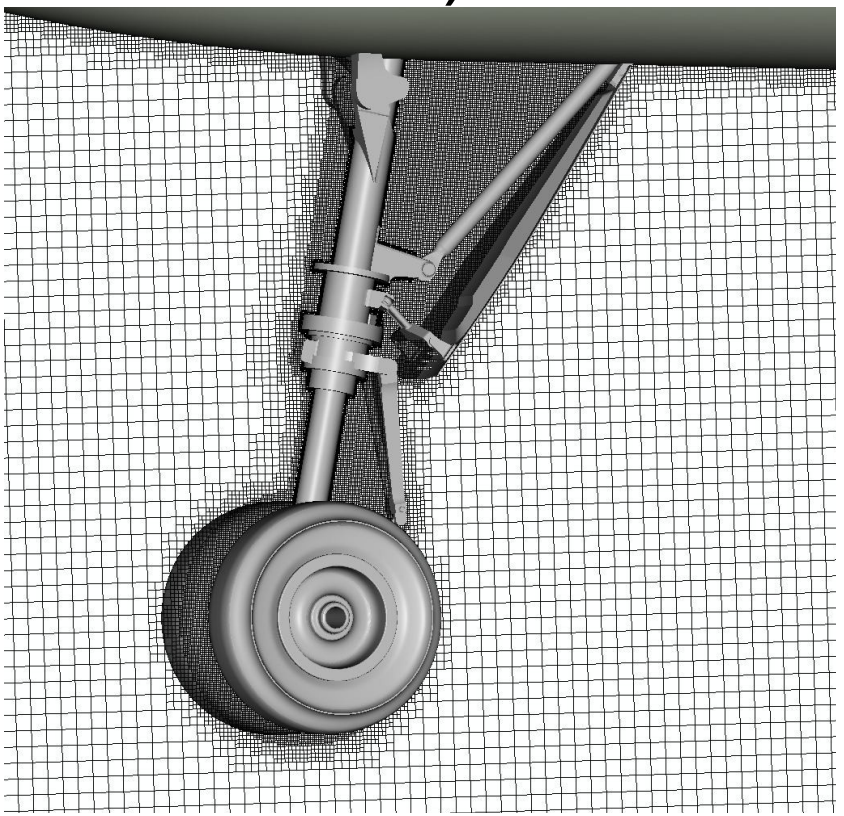
Overset connectivity

Remarks :

- Choice of location for receivers (nodes,centers)
- Works for structured and unstructured zones
- Donors are explicitly given by the user (mesh defined as nodes, centers, with or without n ghost cells,...)

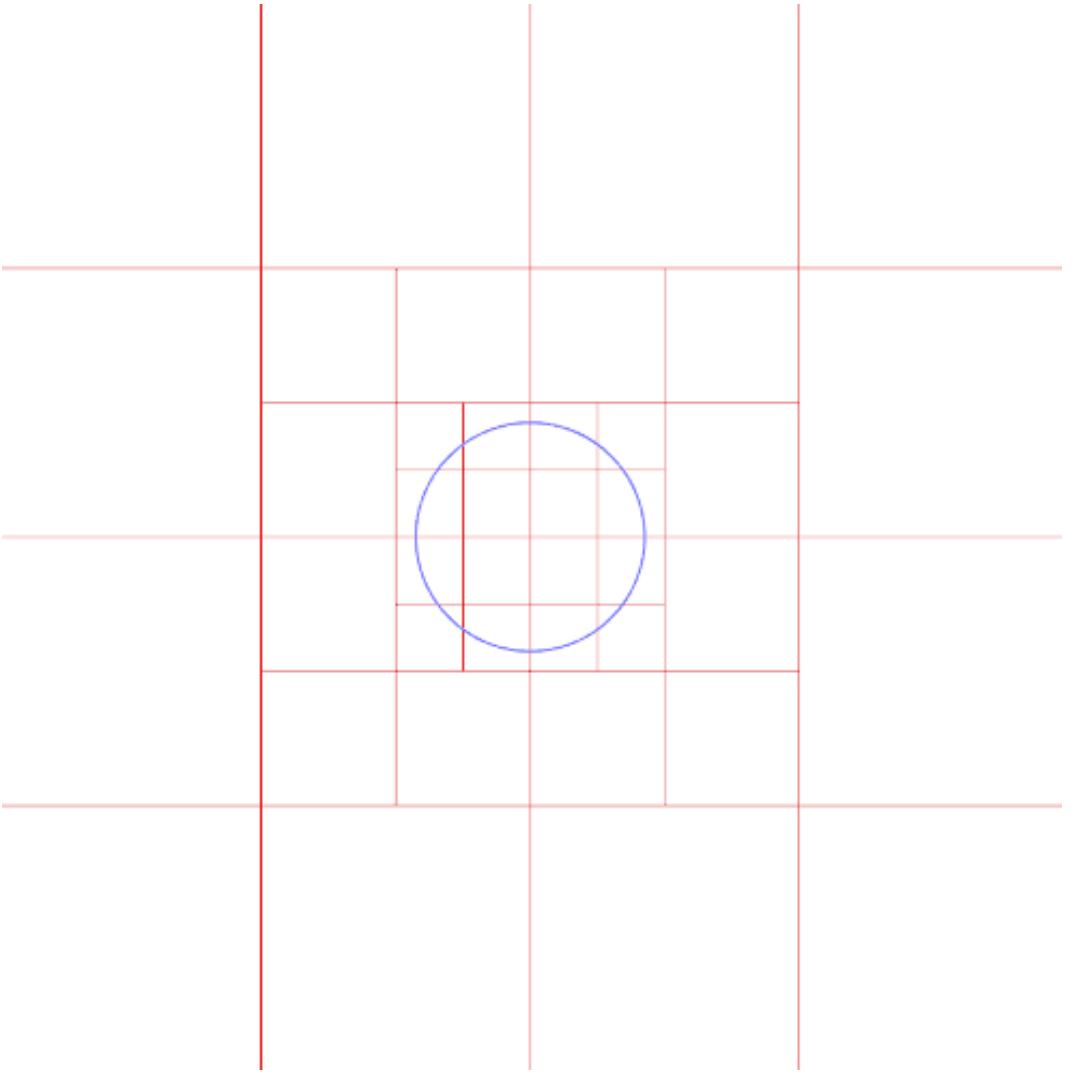
IBC workflow

- Cassiopée functions can be used to perform the geometrical preprocessing for Immersed Boundary Method (ghost fluid method)



Input: set of bodies defined by triangular meshes

Creation of the octree mesh
`o=G.octree(bodies)`

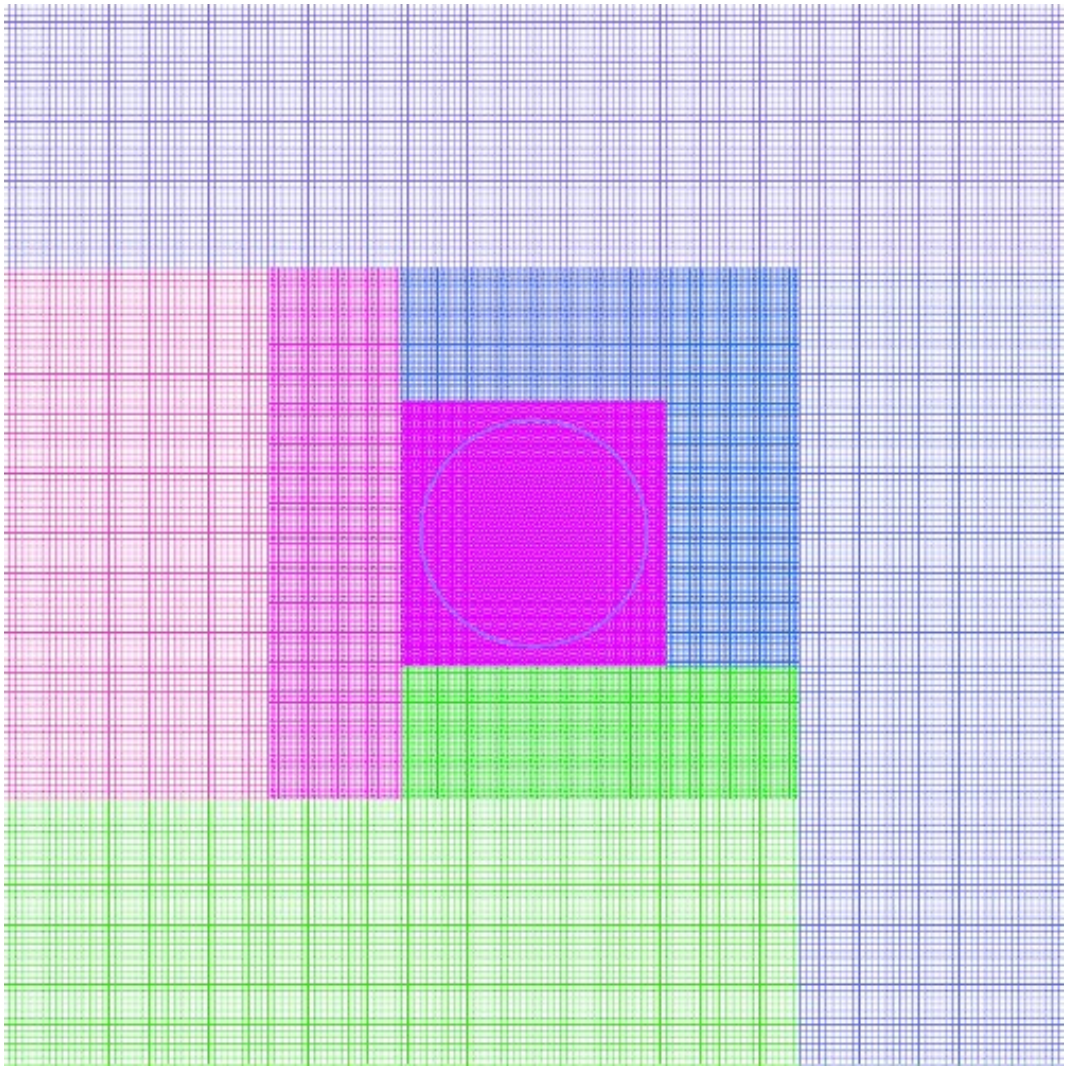


Input: set of bodies defined by triangular meshes

Creation of the octree mesh
`o=G.octree(bodies)`

↓

Generation of Cartesian grids
`t=G.octree2Struct(o)`



Input: set of bodies defined by triangular meshes

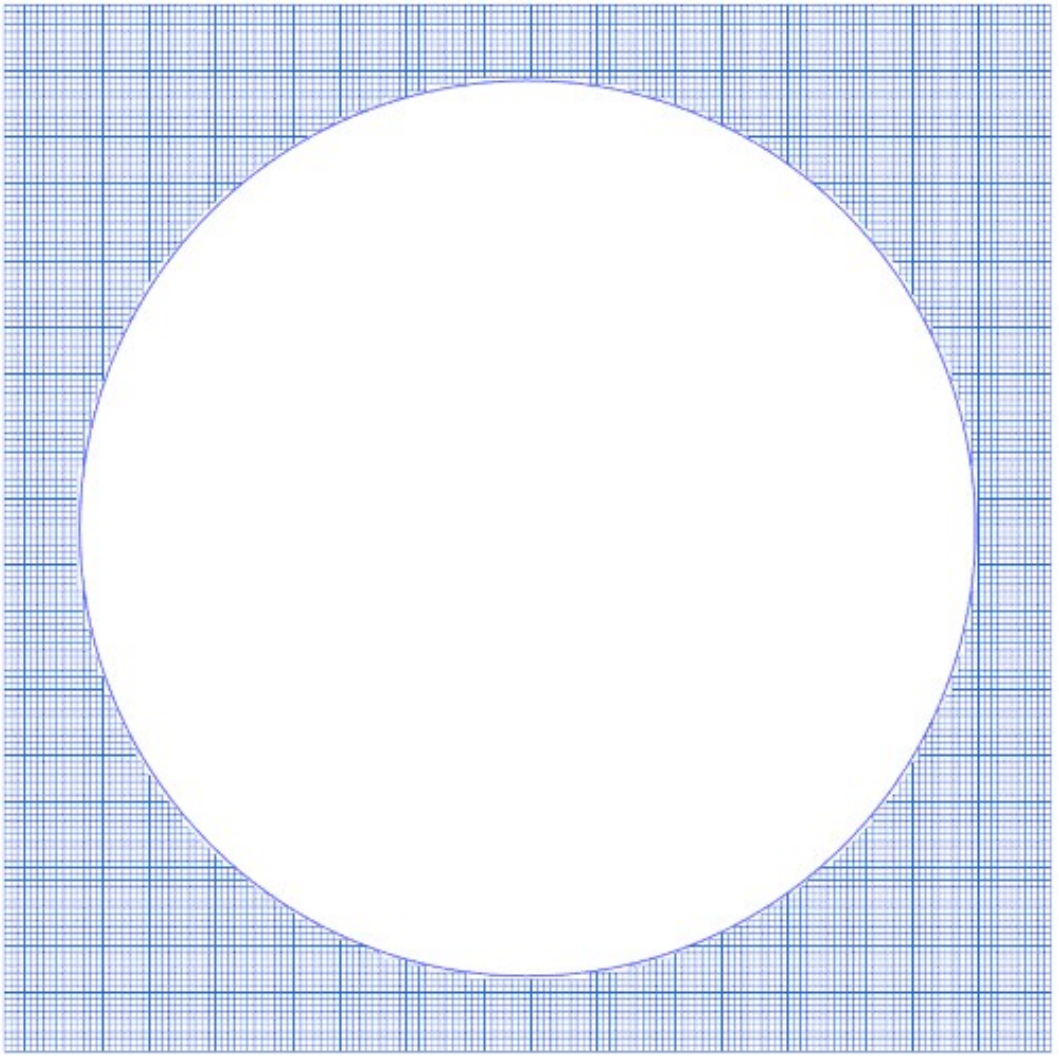
Creation of the octree mesh
`o=G.octree(bodies)`

↓

Generation of Cartesian grids
`t=G.octree2Struct(o)`

↓

Mark solid cells
`t=X.blankCells(t,bodies)`



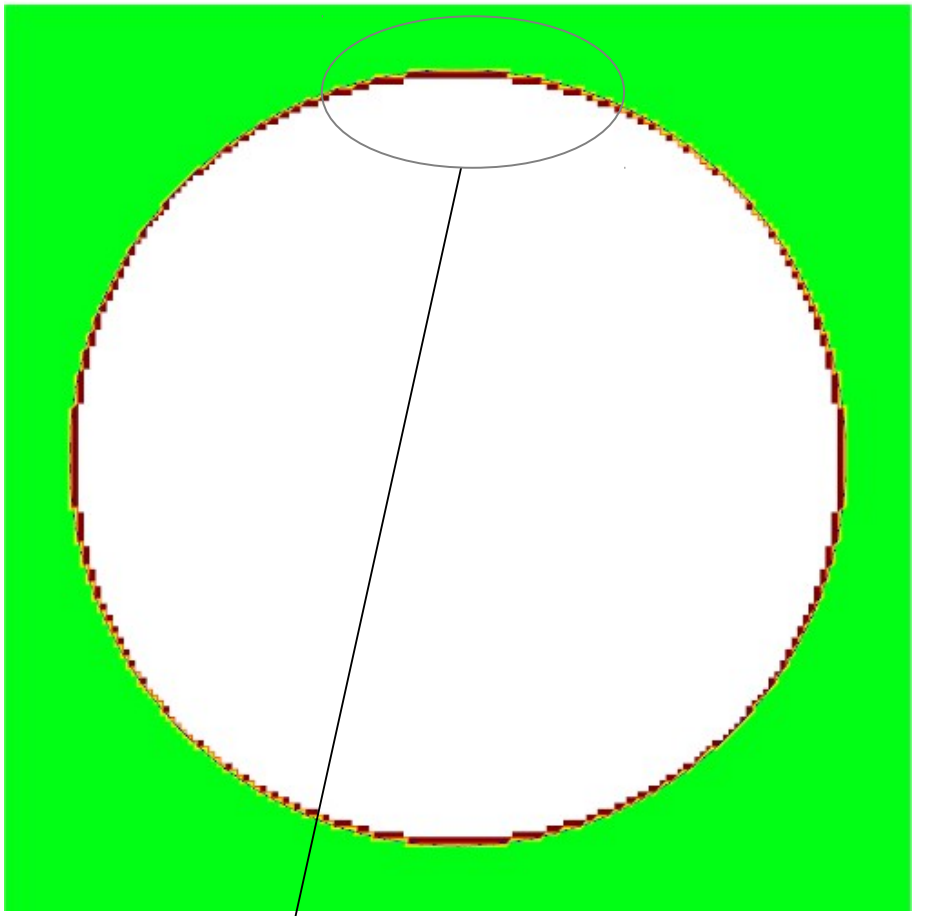
Input: set of bodies defined by triangular meshes

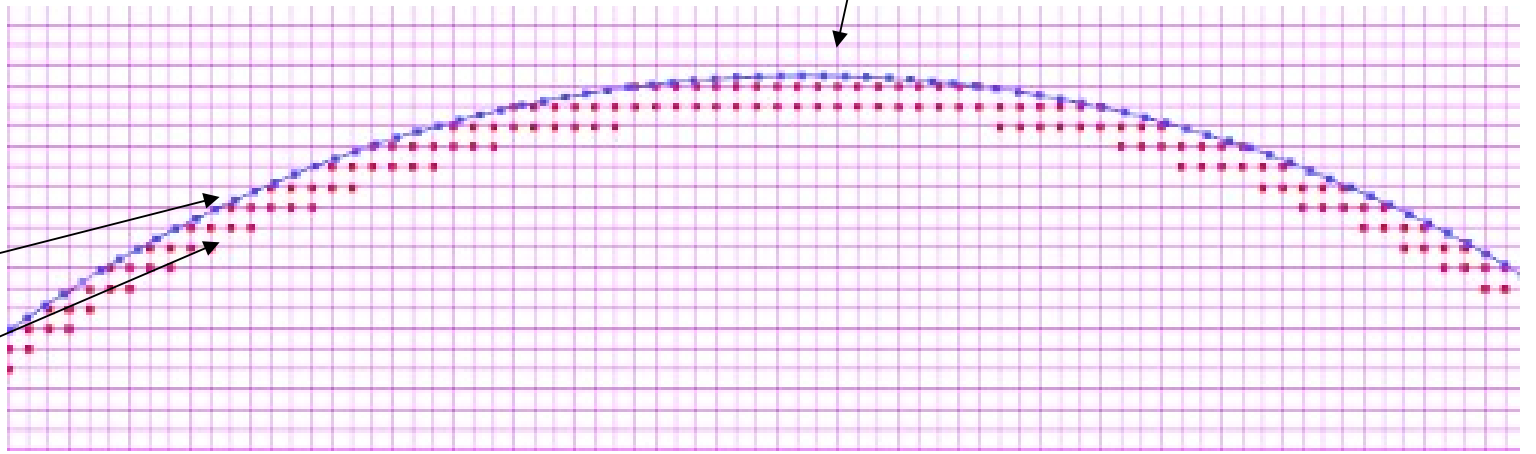
Creation of the octree mesh
`o=G.octree(bodies)`

Generation of Cartesian grids
`t=G.octree2Struct(o)`

Mark solid cells
`t=X.blankCells(t,bodies)`

Defined **d** layers of IBC cells (**d** < 0 : inside)
`t=X.setHoleInterpolatedPts(t,depth=d)`





Input: set of bodies defined by triangular meshes

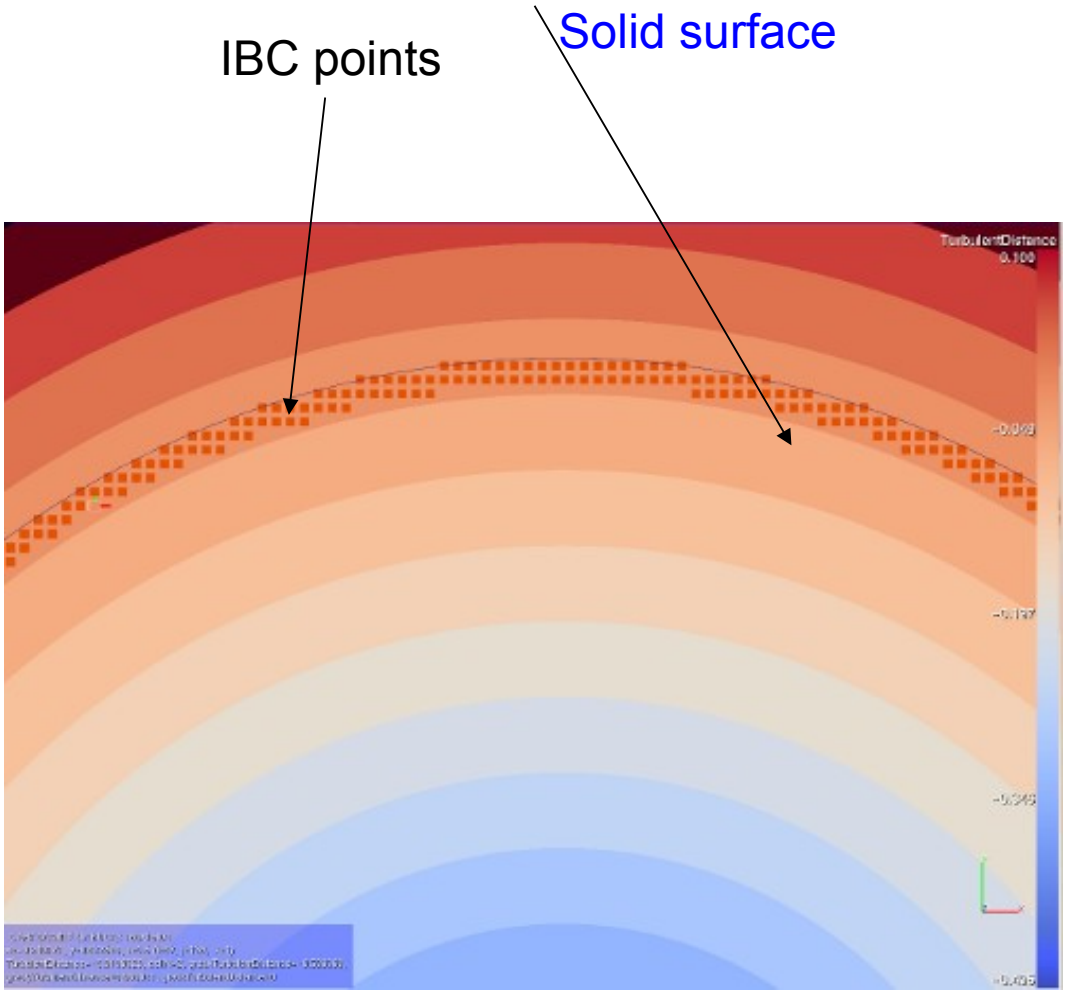
Creation of the octree mesh
`o=G.octree(bodies)`

Generation of Cartesian grids
`t=G.octree2Struct(o)`

Mark solid cells
`t=X.blankCells(t,bodies)`

Defined **`d`** layers of IBC cells (**`d`** < 0 : inside)
`t=X.setHoleInterpolatedPts(t,depth=d)`

Signed distance field
`t=DTW.distance2Walls(t,bodies,signed=1)`



Input: set of bodies defined by triangular meshes

Creation of the octree mesh
`o=G.octree(bodies)`

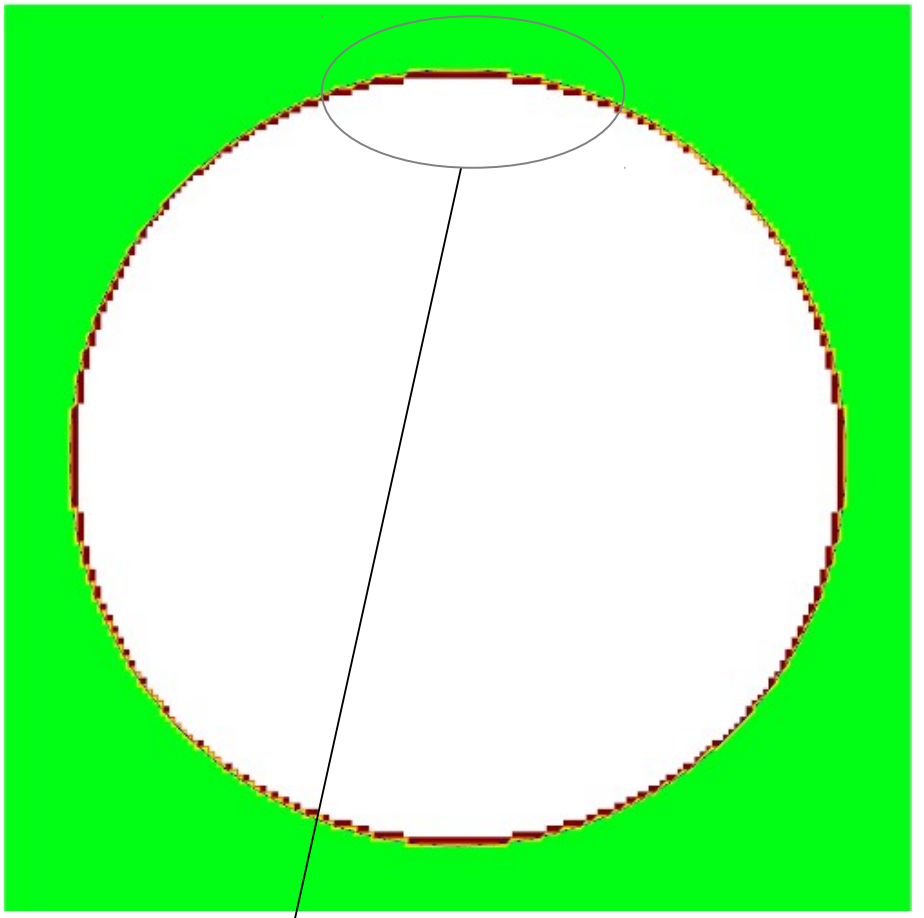
Generation of Cartesian grids
`t=G.octree2Struct(o)`

Mark solid cells
`t=X.blankCells(t,bodies)`

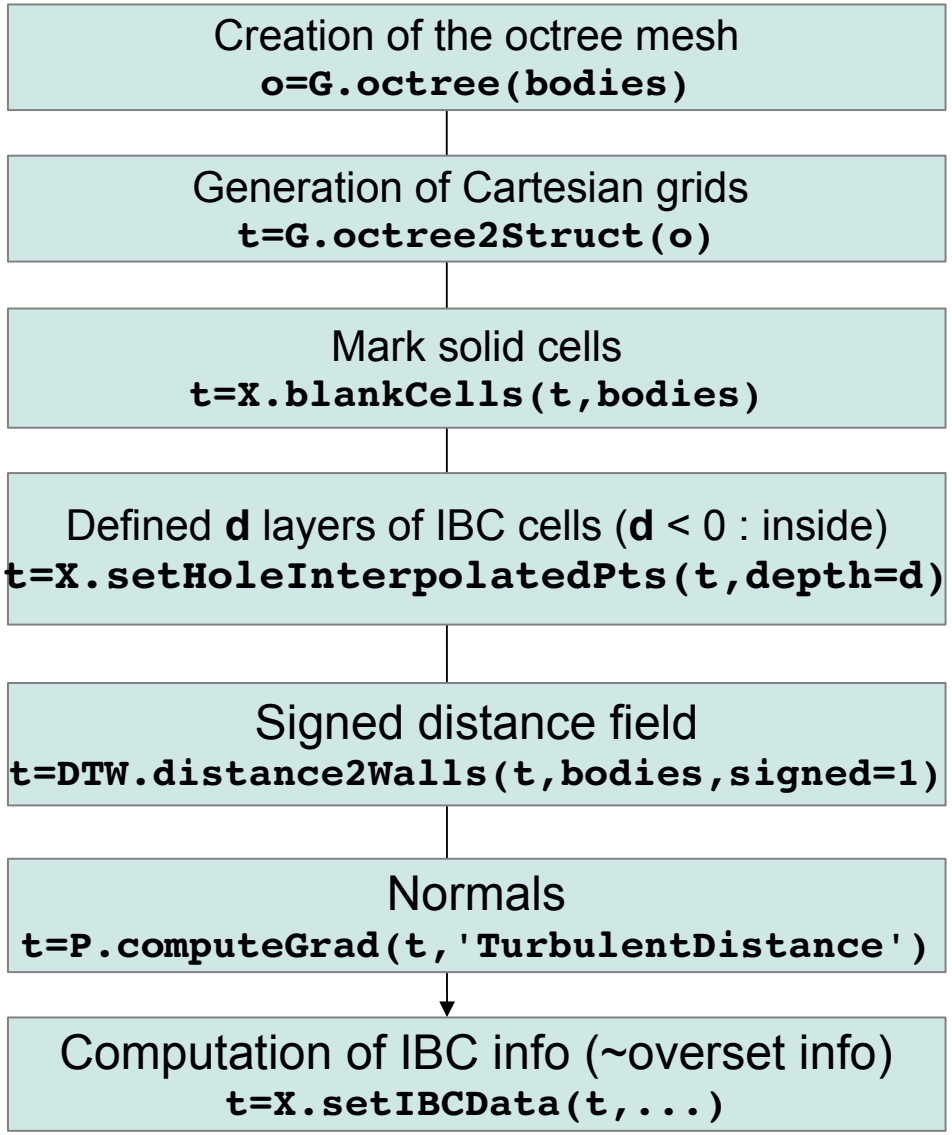
Defined **d** layers of IBC cells (**d** < 0 : inside)
`t=X.setHoleInterpolatedPts(t,depth=d)`

Signed distance field
`t=DTW.distance2Walls(t,bodies,signed=1)`

Normals
`t=P.computeGrad(t,'TurbulentDistance')`



Input: set of bodies defined by triangular meshes



Input: set of bodies defined by triangular meshes

Creation of the octree mesh
`o=G.octree(bodies)`

Generation of Cartesian grids
`t=G.octree2Struct(o)`

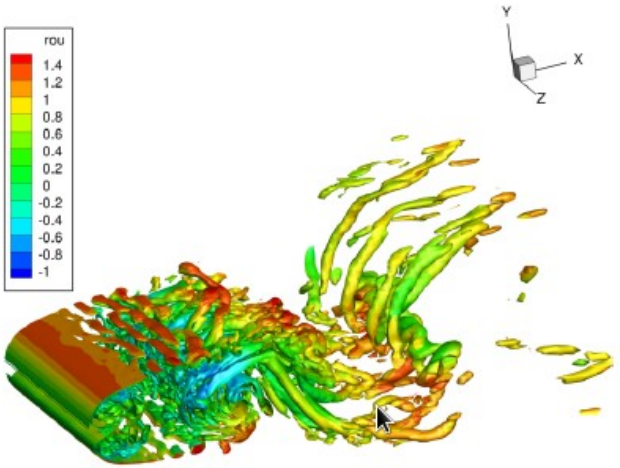
Mark solid cells
`t=X.blankCells(t,bodies)`

Defined **d** layers of IBC cells (**d** < 0 : inside)
`t=X.setHoleInterpolatedPts(t,depth=d)`

Signed distance field
`t=DTW.distance2Walls(t,bodies,signed=1)`

Normals
`t=P.computeGrad(t,'TurbulentDistance')`

Computation of IBC info (~overset info)
`t=X.setIBCData(t,...)`



LES simulation of the flow around a cylinder at Re=3900, computation by M. Terracol (ONERA/CFD & Aeroacoustics Dept), with FUNk solver

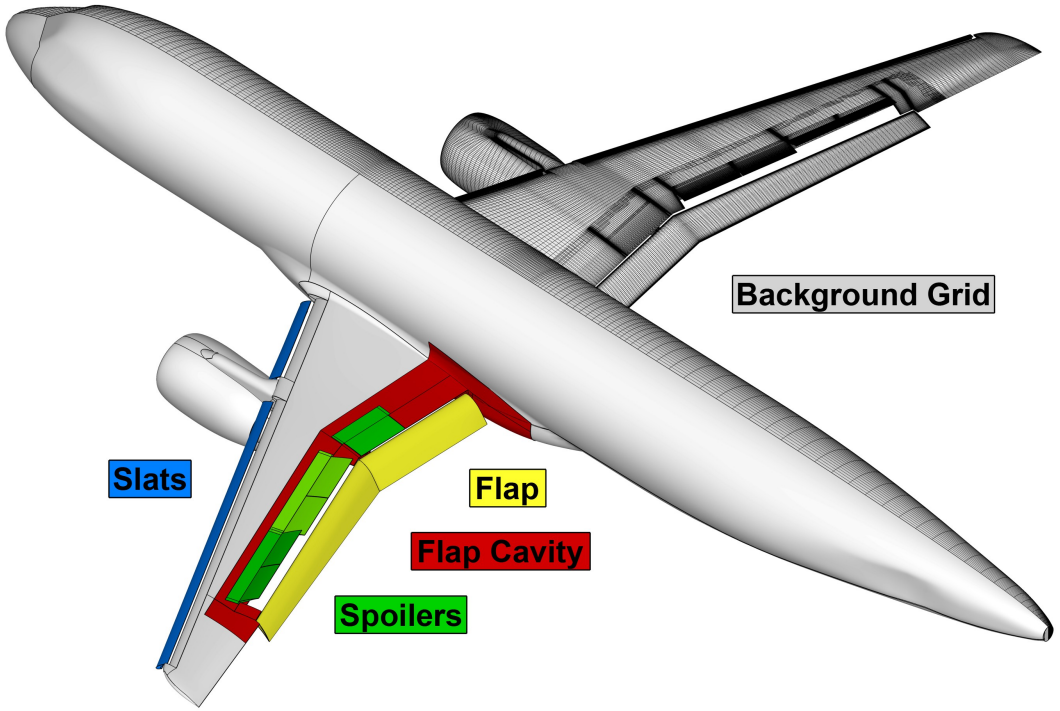
IBC solver (computes {u,v,w,p} at each iteration)

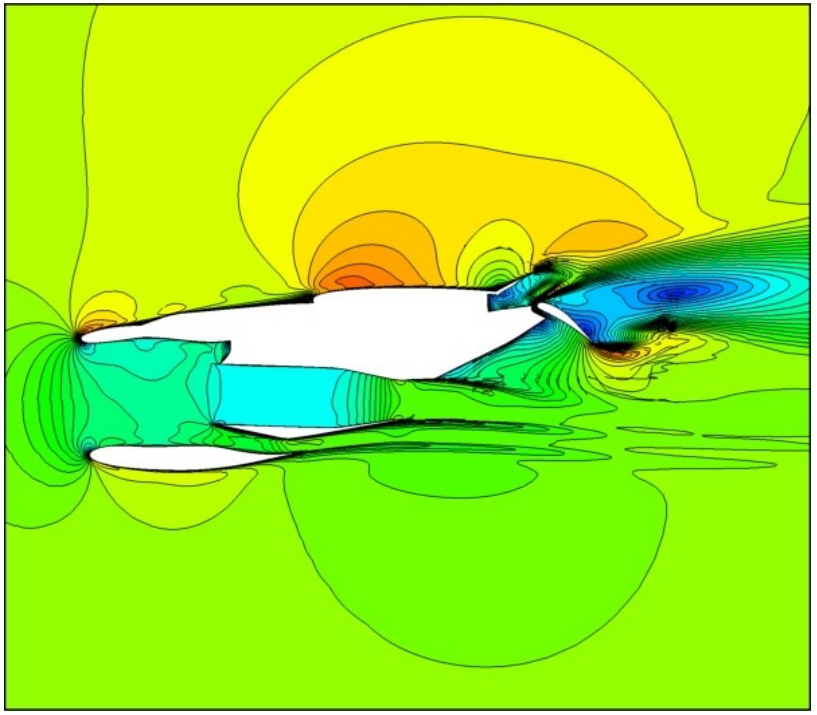
Computation of (u,v,w,p) at IBC points
`t=X.setIBCTransfer(t,...)`

If coupled to an IBC solver

Some applications

High-lift configuration of an aircraft

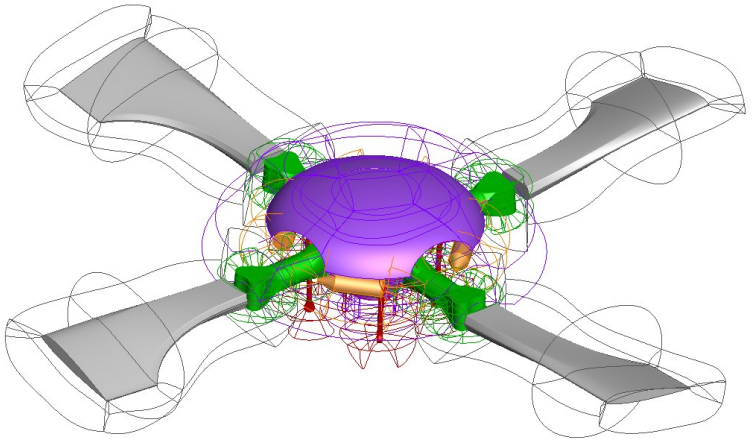


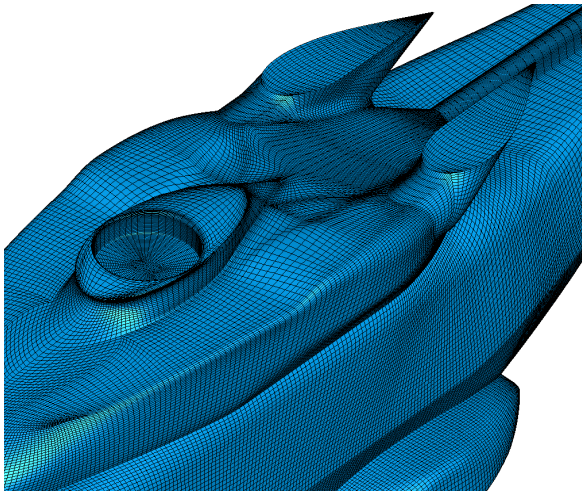


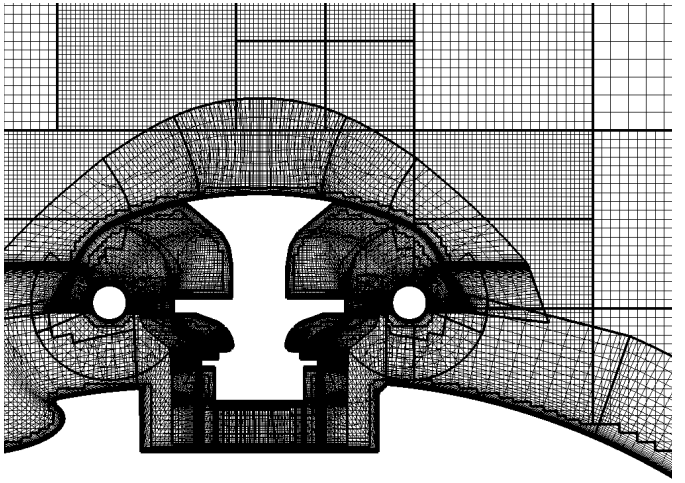
Mach number (slice in the spanwise direction)
RANS simulation using *elsA*

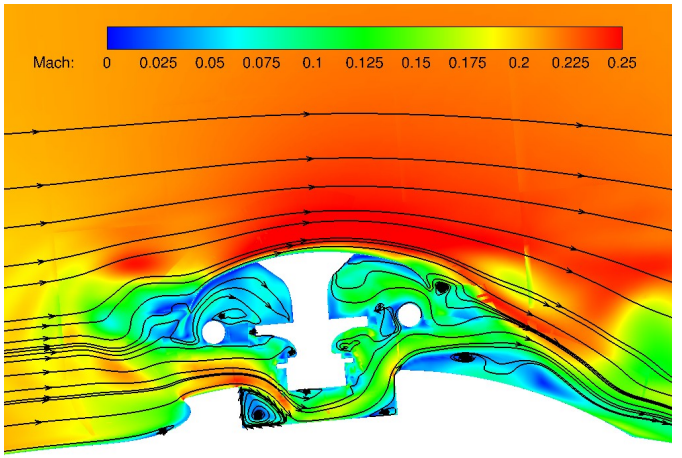
Application achieved by Christophe François and Mickaël Meunier
ONERA/Applied Aerodyn. Dept, Civil Aircrafts Unit

NH90 fuselage with rotor head









Mach number contours near the rotor head
Unsteady RANS simulation using *elsA*

Application achieved by Thomas Renaud
ONERA/Applied Aerodyn Dept, Helicopters, Propellers & Turbomachinery Unit

Conclusions

- Cassiopée contains a set of pre- and post-processing functions
- All the functions operate on the same data (Python/CGNS tree)
- This enables to quickly design solutions for mesh generation/adaptation/assembly and post-processing.