Quantum is a mesh deformation module based on inverse distance weighting interpolation. To preserve the orthogonality of the mesh in the boundary layers when large displacements occur, the surfaces displacements are splitted in a rotation and a translation terms. These terms are then interpolated in the volume mesh. The Inverse distance weighting interpolation is speed up by a Fast Multipole like Algorithm.
Data used in mesh deformation module can either be set in the CGNS File or using set up class methods. The figure below show the data structure that mesh deformation module can handle. All mesh deformation parameters are located in the BC_t nodes. The mesh deformation boundary condition is set inside the .Solver#BC Nodes using Data_t node type named mesh_def_type.

Possible values for this field are:

- **“imposed”** or **“prescribed”**: displacements are imposed on this boundary conditions meaning that Quantum will search a .BCDisplacement#i node containing mesh displacement fields: DisplacementX, DisplacementY, DisplacementZ.
- **“free”**: displacements are free on this boundary.
- **“null”** or **“zero”**: displacements are null on this boundary.

**Warning:** be aware that this data structure is not fixed. it might change in the next Quantum version.
### CGNS parser

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>solver#BC</td>
<td>FamilyName_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>mesh_def_type</td>
<td>UserDefinedData_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>InwardNormal_t</td>
<td>Data_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>PointRange</td>
<td>IndexRange_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>upper_wing#1</td>
<td>BC_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>AdditionalData0</td>
<td>UserDefinedData_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>BCDisplacement0</td>
<td>DataArray_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>DisplacementT</td>
<td>DataArray_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>InwardNormal_t</td>
<td>IndexRange_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>PointRange</td>
<td>UserDefinedData_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>ZoneGridConnectivity</td>
<td>ZoneGridConnectivity_t</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>symmetry plane</td>
<td>BCFieldId</td>
<td>Z100, Z101, Z102</td>
</tr>
<tr>
<td>symmetry plane</td>
<td>symmetry plane</td>
<td>Z100, Z101, Z102</td>
</tr>
</tbody>
</table>

### Chapter 1. CGNS parser

- **Solver**
  - **FamilyName**
  - **Mesh Definition Type**
  - **Inward Normal Direction**
  - **Point Range**
  - **Upper Wing**
  - **Additional Data**
  - **BC Displacement**
  - **Displacement**
  - **Inward Normal Direction**
  - **Point Range**
  - **Zone Grid Connectivity**

- **Solver#BC**
  - **Family Name**
  - **User Defined Data**
  - **Data**
  - **Index Range**
  - **User Defined Data**
  - **Data Array**
  - **Index Range**
  - **User Defined Data**
  - **Zone Grid Connectivity**

- **BC Field ID**
- **Symmetry Plane**
- **Upper Wing**
KeDefGrid(pyTREE, **kwargs) Initialize the mesh deformation module.

KeDefGrid.computeMeshDisplacement() Run Fast Multipole method to solve interpolation problem.

KeDefGrid.getFinalTree() Returns Final Tree.

KeDefGrid.getSurfTree() Returns Source Tree.

KeDefGrid.makeSources([shape]) Build the Fast Multipole problem:

KeDefGrid.printFinalTree(Name) Print the Final Tree.

KeDefGrid.printSurfTree(Name) Print the IDW data Tree.

KeDefGrid.setBndSurfTo(Name, typeOfBnd[, ...]) Displacement field <-> pyTree boundary Association.

KeDefGrid.set_Amplitude(Amplitude) set Deformation Amplitude.

class Ael.Quantum.KeDefGrid(pyTREE, **kwargs)
Initialize the mesh deformation module.

Note: The initialization will build a deformation tree by splitting surfaces points and uinterior points.

Parameters

- **pyTREE** (python_CGNS) – Cassiopee PyTree containing the CFD geometry. Surface mesh displacements may be attached.

- **Approach** (string) – Choose the interpolation methods. Both are taking into account rotations.

  - Possible values:
    
    * ””**Matrix**””: interpolation using rotation matrices. **Default value**
    
    * ”**Quaternions**” : interpolation of the quaternions.
• "Vector": interpolation no rotation.

• **ComputeVelocity** (bool) – Compute the grid velocity through the interpolation using rotation matrices. Only for Matrix approach
  
  – Possible values:
    * "False": not taken into account. **Default value**
    * "True"

• **Mode** (string) – If set to True perform the adjoint mesh deformation.
  
  – Possible values:
    * "Direct": perform Mesh deformation. **Default value**
    * "Adjoint": perform Adjoint Mesh computation.

• **OneOrTwoSideFMM** (string) – Set interpolation speed up approach.
  
  – Possible values:
    * Two: two sides FMM. **Default value**
    * One: one side FMM.

• **Epsilon** (float) – Error for FMM method.
  
  – Recommended values:
    * 0.5: for two sides multipole expansion. **Default value**
    * 0.8: for one side multipole expansion.

• **Ndivision** (int) – number of implicit step in the Ndivision process.

• **OmpAllInOne** (bool) – If set to True group all the blocks on a processor. **Default value**: True

• **isIntegrale** (bool) – If set to False surface area are not taken into account in the weighting functions. **Default value**: True

• **Leafsize** (int) – Minimum number of point in the surfaces and volumes ADT Trees leaves. In 3D 9 is the minimum. **Default value**: 8

• **NullDisplacements** (string) – if set to 1 null displacements are weighting the IDWs else damping function is used.
  
  – Possible values:
    * Weighted: IDW. **Default value**
    * Damped: Damping function is used.
• **Smoothing** (bool) – if set to True, Smoothing is applied. False is the Default value

**Warning:** not very stable

• **MpiComm** (MPI_communicator) – Set to Comm_World by default, can be set to local MPI communicator values. Global MPI communicator is the Default value

• **DEBUG** (bool) – set verbosity and checking to maximum.

**computeMeshDisplacement()**
Run Fast Multipole method to solve interpolation problem.

CGNS Tree is filled with the volume displacements

**Note:** If more than one surface Tree is moving post processing will include smoothing iteration.

**getFinalTree()**
Returns Final Tree.

**Return type** CGNS/Python

**getSurfTree()**
Returns Source Tree.

**Return type** CGNS/Python

**makeSources**(shape=1)
Build the Fast Multipole problem:

• Compute displacement of sliding surfaces.

• Build surfaces and volume ADT-Trees.

. **Note:** If MPI is set to True aggregate surfaces points on all processors.

**Parameters** shape – surface shape number that user want to use for mesh deformation

**printFinalTree**(Name)
Print the Final Tree.

**Parameters** Name – FileName.

**printSurfTree**(Name)
Print the IDW data Tree. Surfaces and displacements and interior points.
**Parameters**

- **Name** – FileName.

**setBndSurfTo***(Name, typeOfBnd, FileName=None, Modenumber=0, path='h**')***

Displacement field <-> pyTree boundary Association. it Will fill the Deformation PyTree

**Parameters**

- **FileName** – Path to the CGNS boundary. must be given as: “ZoneName#BoundaryName”.

- **typeOfBnd** – Set the mesh deformation boundary type.
  - Possible values :
    - ”imposed” : in this case FileName must be filled.
    - ”null” :
    - ”slidingonsurface” :

- **FileName** – Displacements Filename or list of numpy array.

- **Modenumber** – If more than one displacement field is in the file, set the Field number to take into account.

- **path** – Path to displacement variable Names (use wildcard).

**set_Amplitude**(Amplitude)

set Deformation Amplitude.

**Parameters**

- **Name** – FileName.
3.1 Pure openMP set up

First set the number of Thread you want to use.

```bash
export OMP_NUM_THREADS=4
```

**Warning:** when the number of threads is superior to 6 lower performances are observed.

Then run normally your python script. Here is an example of a python script

```python
import numpy as num
import sys
import Converter.PyTree as C
import Converter.Internal as I
import Ael.Quantum as KDG
import Generator.PyTree as G

# #Read Cgns File and return CGNS/Python Tree
#

Tree=C.convertFile2PyTree("aileM6_all.cgns")

# #Parametres de la deformation de maillage
#
DeformationArgs={"Approach" : "Quaternions",
                 "Epsilon" : 0.5,
                 "Leafsize" : 8,
                 "Ndivision" : 250,
                 # (continues on next page)
```
"OneOrTwoSideFMM" : "Two",
"NullDisplacements" : "Weighted",
"Smoothing" : False}

# Declare mesh deformation object deftree

defTree = KDG.KeDefGrid(Tree,**DeformationArgs)

# Set amplitude deformation
#
defTree.set_Amplitude(5000.)

# Impose some Boundary conditions (by default boundaries are set to null_displacement)
#
defTree.setBndSurfTo("Zone1#BCWall.4","imposed","mode0001_fm.tp",Modenumber=2)
defTree.setBndSurfTo("Zone5#BCWall","imposed","mode0002_fm.tp",Modenumber=2)
defTree.setBndSurfTo("Zone5#BCFarfield.63","slidingonsurface")

# Eventually print the deformation TREE
#
defTree.printSurfTree("windows.cgns")

# Build FMM problem
#
defTree.makeSources()

# Run mesh displacement
#
defTree.computeMeshDisplacement()

I.__FlowSolutionNodes__ = 'Displacement#0'
Tree=C.initVars(Tree, '{CoordinateX} = {CoordinateX}+{DisplacementX}')
Tree=C.initVars(Tree, '{CoordinateY} = {CoordinateY}+{DisplacementY}')
Tree=C.initVars(Tree, '{CoordinateZ} = {CoordinateZ}+{DisplacementZ}')
Tree = G.getVolumeMap(Tree)
vol=I.getNodesFromName(Tree,"vol")
concatenated=[]

(continues on next page)
for v in vol:
    concatenated+=set(v[1].flatten())
print " Sort :: volmin=",num.amin(num.array(concatenated))
I._FlowSolutionNodes__ = 'FlowSolution'

# Tree has been changed inplace it just has to be write in file.
# C.convertPyTree2File(Tree,"zones")

### 3.2 MPI/openMP set up

Again set the number of Thread you want to use, then run the case using mpirun:

```
export OMP_NUM_THREADS=4
mpirun -np nb_procs python Yourcase.py
```

Parallel mpi mesh deformaton implementation use Converter.mpi. Few more lines are needed to run the previous case in parallel:

```
import numpy as num
import sys
import Converter.PyTree as C
import Converter.Internal as I
import Ael.Quantum as KDG

# Need for mpi
# import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G

# Read Cgns File but leave numpy arrays on all procs
# skeletonTree= Cmpi.convertFile2SkeletonTree('aileM6_all.cgns')

# Distribute over processors
# 
# print distribute on N procs
(skeletonTree, dic) = Distributor2.distribute(skeletonTree, NProc=Cmpi.size, _algorithm='fast')
```
# load numpy arrays on procs
#
skeletonTree = Cmpi.readZones(skeletonTree, 'aileM6_all.cgns', format="bin_hdf", rank=Cmpi.rank)
#
# remove empty zones
#
skeletonTree = Cmpi.convert2PartialTree(skeletonTree)
#
#Parametres de la deformation de maillage
#
DeformationArgs={"Approach" : "Quaternions",
                  "Epsilon" : 0.5,
                  "Leafsize" : 8,
                  "Ndivision" : 250,
                  "OneOrTwoSideFMM" : "Two",
                  "NullDisplacements" : "Weighted",
                  "Smoothing" : False}
#
defTree = KDG.KeDefGrid(skeletonTree,**DeformationArgs)
#
#Set amplitude deformation
#
defTree.set_Amplitude(5000.)
#
#Impose some Boundary conditions ( by default boundarys are set to null displacement)
#
defTree.setBndSurfTo("Zone1#BCWall.4", "imposed", "mode0001_fm.tp", Modenumber=2)
defTree.setBndSurfTo("Zone5#BCWall", "imposed", "mode0002_fm.tp", Modenumber=2)
defTree.setBndSurfTo("Zone5#BCFarfield.63", "slidingonsurface")
#
#Build FMM problem
#
defTree.makeSources()
#
# Run mesh displacement
#
defTree.computeMeshDisplacement()

I.__FlowSolutionNodes__ = 'Displacement#0'
skeletonTree=C.initVars(skeletonTree, '{CoordinateX} = {CoordinateX}+{DisplacementX}
→')
skeletonTree=C.initVars(skeletonTree, '{CoordinateY} = {CoordinateY}+{DisplacementY}
→')
skeletonTree=C.initVars(skeletonTree, '{CoordinateZ} = {CoordinateZ}+{DisplacementZ}
→')
skeletonTree = G.getVolumeMap(skeletonTree)
vol=I.getNodesFromName(skeletonTree,"vol")
concatenated=[]
for v in vol:
    concatenated+=set(v[1].flatten())
print " Sort :: volmin=",num.amin(num.array(concatenated))
I.__FlowSolutionNodes__ = 'FlowSolution'

#
# gather all zones
#
d = Cmpi.allgather(skeletonTree)
skeletonTree = I.merge(d)
Cmpi.convertPyTree2File(skeletonTree, 'out.cgns')
#
# Write tree on proc 0
#
if Cmpi.rank==0:
    C.convertPyTree2File(skeletonTree,"zones")
Table 1: 2D airfoil rotation

<table>
<thead>
<tr>
<th>Initial mesh</th>
<th>Final mesh</th>
</tr>
</thead>
</table>

[Images of initial and final mesh]
### Table 2: 2D icing

<table>
<thead>
<tr>
<th>Final mesh</th>
<th>Zoom</th>
</tr>
</thead>
</table>

[Image of final mesh and zoom]
Table 3: 3d Unstructured CRM

<table>
<thead>
<tr>
<th>Initial</th>
<th>Final</th>
</tr>
</thead>
</table>

- Initial: [Image of initial 3D Unstructured CRM]
- Final: [Image of final 3D Unstructured CRM]
Table 4: 3d Structured M6 Wing
INDICES AND TABLES

- genindex
- modindex
- search