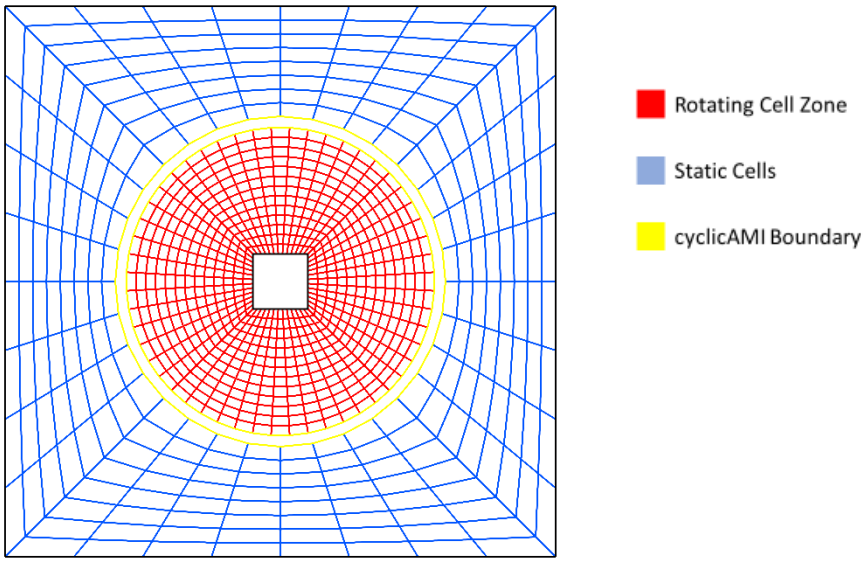
A simple rotating dynamic mesh in Openfoam is added to a static mesh with the following steps

1. The Mesh
   i. The Cyclic Arbitrary Mesh Interface
   ii. A Cell Zone to specify which cells move
2. The Simulation
   i. The Dynamic Mesh Dictionary in the folder /constant
   ii. 0 second initial conditions

The simplest case we can think of is a 2D box rotating with a flow across it. In fig we show the geometry of this.

1. The Mesh
   i. The cyclicAMI boundaries are created directly in blockmesh along with every other boundary.
   ii. The cellZone “rotor” is defined when creating blocks in blockmesh.
2. The Simulation
   i. The Dynamic Mesh Dictionary in the folder /constant
   ii. 0 second initial conditions



This can be created with purely blockmesh, since it is a simple 2D geometry. The gap shown in TUT16 and TUT17 is there for visualization purposes and would not necessarily be there in a real mesh.

The important things about the blockMeshDict file are found in defining the boundary between the rotating cell zone and the static cells.

The cyclicAMI boundary is defined in blockMesh as

```
AMI1
{
    type            cyclicAMI;
    neighbourPatch  AMI2;
    transform       noOrdering;
    faces
```

```
        (
            (12 4 20 28)
            (4 3 19 20)
            (3 11 27 19)
            (11 12 28 27)
        );
    }

    AMI2
    {
        type            cyclicAMI;
        neighbourPatch  AMI1;
        transform       noOrdering;
        faces
        (
            (13 5 21 29)
            (5 2 18 21)
            (2 10 26 18)
            (10 13 29 26)
        );
    }
```

where the points defining the faces are defined just like anything else. The two boundaries do not have to be called “AMI1” and “AMI2” but will be done so in tutorials for clarity.

The initial conditions defined in the 0 second folder are always

```
type            cyclicAMI;
value           $internalField;
```

if there is no change across the interface. In full form this is

```
    AMI1
    {
        type            cyclicAMI;
```

```
    value           $internalField;
}

AMI2
{
    type            cyclicAMI;
    value           $internalField;
}
```

cellZones are created in blockMesh with the after the block is defined. In the below block, the cellZone rotor is being defined.

```
// block 4
hex (12 4 7 9 28 20 23 25)
rotor
(13 13 1)
simpleGrading (1 1 1)
```

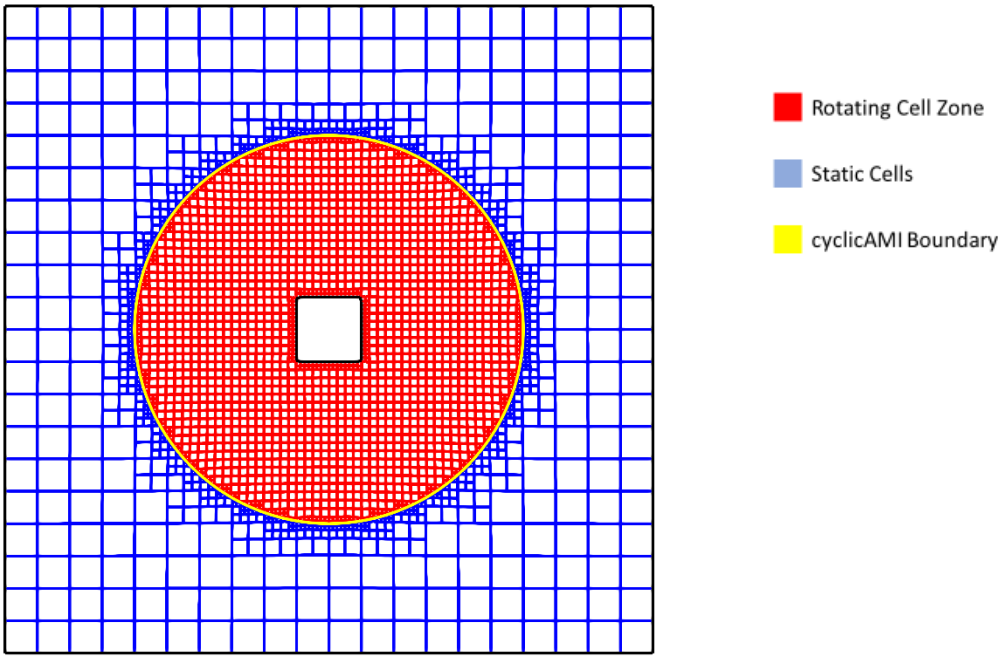
The 0 second folder has its boundary conditions changed by the cyclicAMI.

These are all the same.

Please examine the *blockMeshDict.m4* file until you understand it. Try and recreate it from scratch to truly understand what it is.

The next level of complexity is using snappyHexMesh to create the mesh.

1. The Mesh
   i. The cyclicAMI faceset is created by specifying the facezone in the refinementSurfaces section of snappyHexMeshDict. It has a name *name* and *name_slave*.
   ii. A Cell Zone is created in snappyHexMeshDict in the refinementSurfaces part of snappyHexMeshDict. The cellZone line is used to name it and the cellZoneInside is used to state where the cell zone is relative to the geometry.
2. The Simulation
   i. The Dynamic Mesh Dictionary in the folder /constant
   ii. Use of the DyM solver



Snappy is changed in the following ways. First, there has to be a .STL file defined on the boundary between the rotor sand stator zones, secondly that refinementSurface entry must look like the following

```
refinementSurfaces
{
    innerCylinder
    {
        // Surface-wise min and max refinement level
        level (3 3);
```

```
        faceType    boundary;
        faceZone    innerCylinder;
        cellZone    rotor;
        cellZoneInside  inside;
    }
  }
```

CreatePatch defines the cyclicAMI from already existing patches. It does what we would normally do in blockMesh but have to delay because we are using snappyHexMesh.

```
patches
(
    {
        name            AMI1;
        patchInfo
        {
            type            cyclicAMI;
            neighbourPatch  AMI2;
            transform       noOrdering;
        }
        constructFrom patches;
        patches (innerCylinder);
    }

    {
        name            AMI2;
        patchInfo
        {
            type            cyclicAMI;
            neighbourPatch  AMI1;
            transform       noOrdering;
```

```
        }
        constructFrom patches;
        patches (innerCylinder_slave);
    }
)
```

The 0 second folder has to be changed the same way as it’s done with blockMesh.

Further Considerations:

Alejandro Roger Ull has made a fine report for a much more complicated problem, his report and case represents an excellent description of point deformation.