POLITECNICO DI MILANO

SnappyHexMesh: scalable & automatic mesh generation for OpenFOAM

A. Montorfano, F. Piscaglia

[1] Dipartimento di Energia, POLITECNICO DI MILANO

June 19, 2015

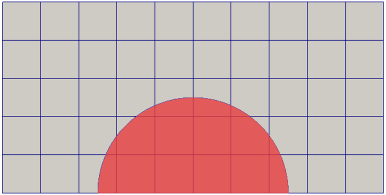
Overview

- Introduction
  - What is SHM
  - Why you could need SHM
  - Prerequisites
- Input geometry
- SnappyHexMesh workflow
  1. Generate base mesh
  2. Refine & select
  3. Snap
  4. Add layers
- Using SHM in parallel
- Summary

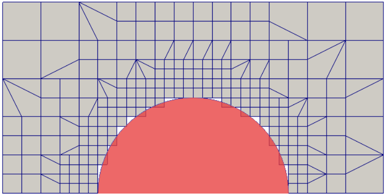
What is SnappyHexMesh

- SHM is a fully automatic, parallel, octree-refinement-based mesh generation app for Open-FOAM.
- Mesh generation is based on four steps:

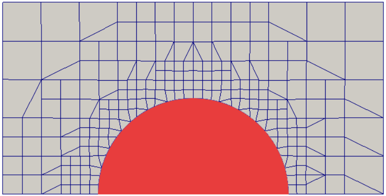
1) Background mesh



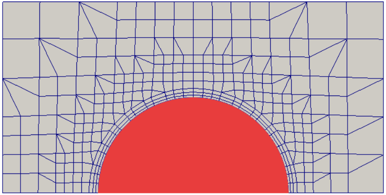
2) Castellated mesh



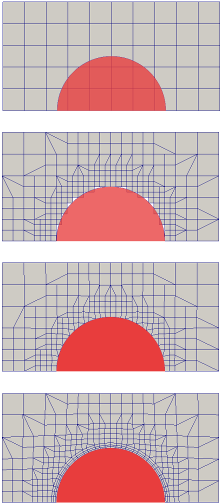
3) Snapped mesh

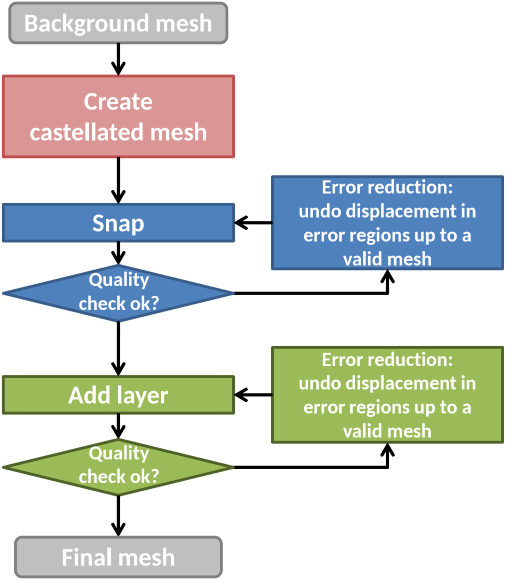


4) Layered mesh



SnappyHexMesh workflow





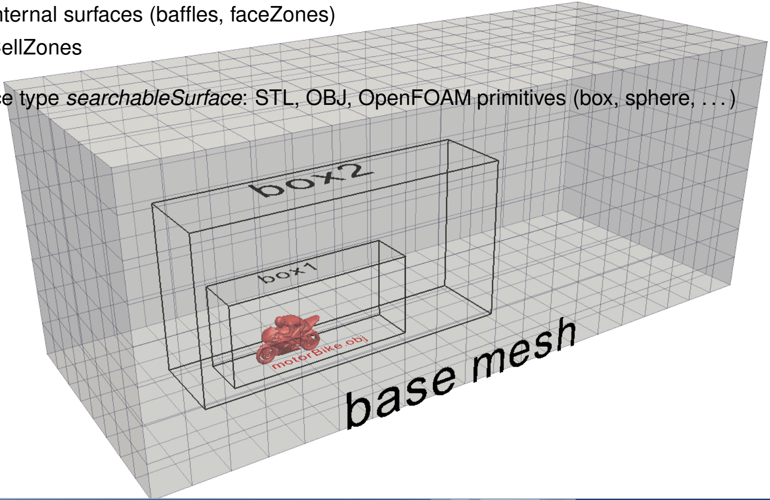
Prerequisites

,

- **OpenFOAM** – this presentation is based on version 2.4.x
- **A good amount of memory**: SHM uses a lot of RAM during its operation. A rough guide is 4 GB per million cells. Usually computing nodes have little memory per core. Use dedicated nodes.
- **MPI environment** if you want to run SHM in parallel
- **ptscotch** support in OpenFOAM (enabled by default).

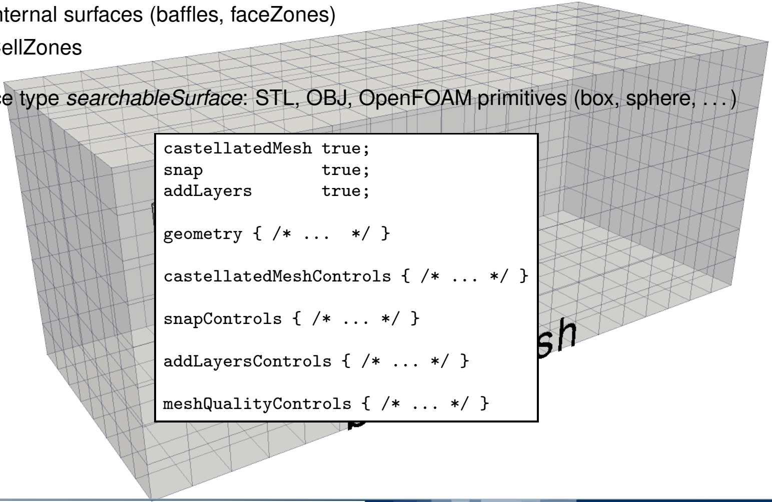
Input geometry

- Input surfaces are used to specify:
  - Solid walls
  - Refinement regions
  - Internal surfaces (baffles, faceZones)
  - CellZones
- Surface type *searchableSurface*: STL, OBJ, OpenFOAM primitives (box, sphere, . . .)



Input geometry

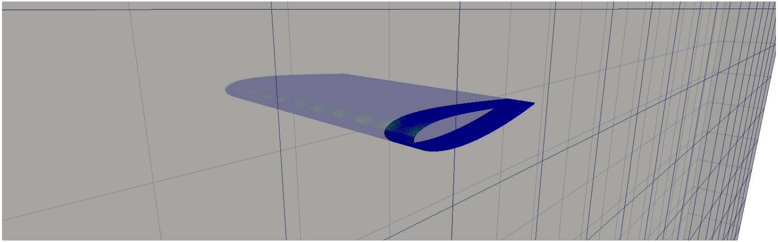
- Input surfaces are used to specify:
  - Solid walls
  - Refinement regions
  - Internal surfaces (baffles, faceZones)
  - CellZones
- Surface type *searchableSurface*: STL, OBJ, OpenFOAM primitives (box, sphere, . . .)



```
castellatedMesh true;
snap            true;
addLayers       true;

geometry { /* ...  */ }

castellatedMeshControls { /* ... */ }

snapControls { /* ... */ }

addLayersControls { /* ... */ }

meshQualityControls { /* ... */ }
```

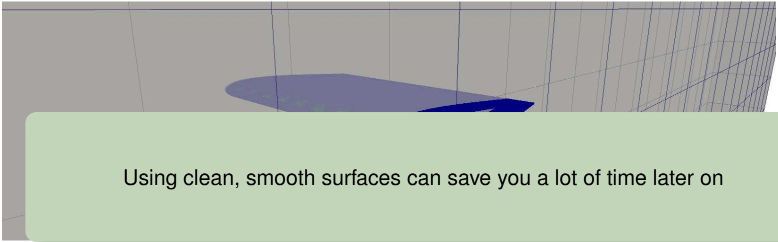
Input geometry

- Surface must be **closed**
  - Surface may extend its open ends beyond base mesh boundaries
  - There cannot be any gap in the surface
- Surface must have **no degenerate triangles**: use surfaceCheck
- Face normals must be consistent: a sudden change in surface normal is seen as a feature edge.
- STL solids will become final mesh patches
- Open surfaces can be used to specify faceZones, extra refinement regions.



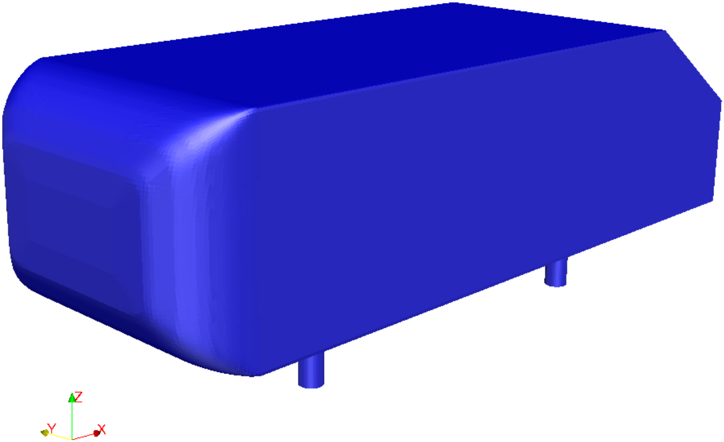
Input geometry

- Surface must be **closed**
  - Surface may extend its open ends beyond base mesh boundaries
  - There cannot be any gap in the surface
- Surface must have **no degenerate triangles**: use surfaceCheck
- Face normals must be consistent: a sudden change in surface normal is seen as a feature edge.
- STL solids will become final mesh patches
- Open surfaces can be used to specify faceZones, extra refinement regions.



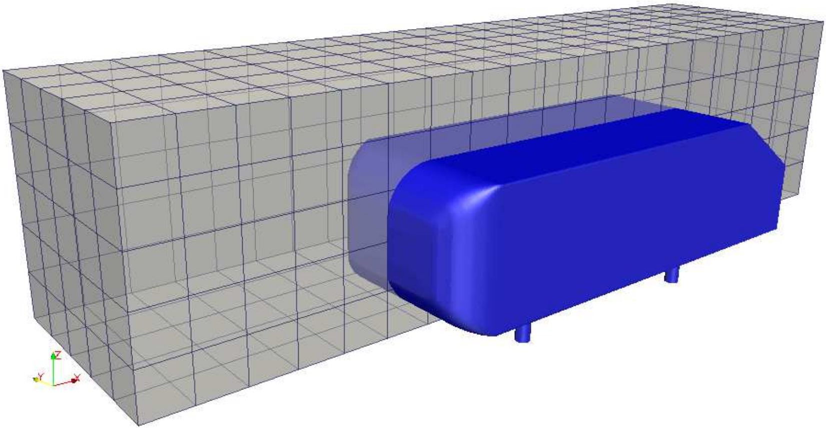
Using clean, smooth surfaces can save you a lot of time later on

Example of application: the Ahmed body



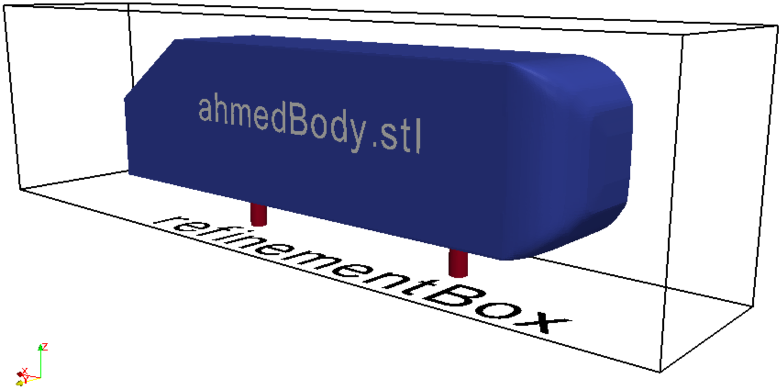
1. Generate the base mesh

- Cell aspect ratio must be ≈ 1
- Base mesh must be generated with other tool than SHM (blockMesh, ICEM, . . .)
- **tip:** Orient base mesh to adapt it to geometry/flow



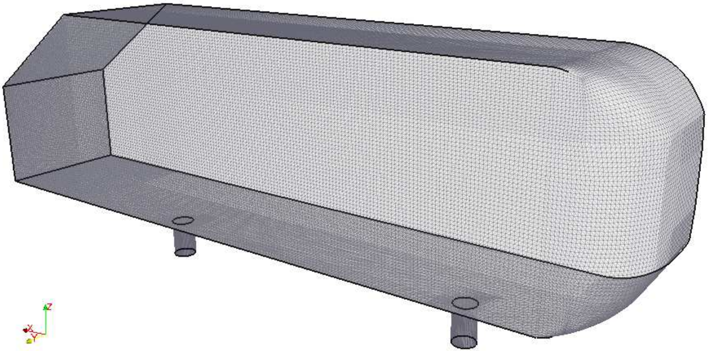
2. Define geometry

- Each STL solid will be a patch in the final mesh (ASCII format required!)
- Solid name in the file can be redefined
- Define also refinement boxes, etc. . . ICEM, . . .)



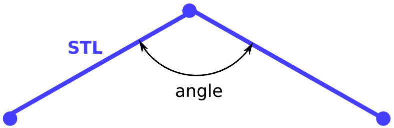
3. Explicit feature extraction (optional)

- Use **surfaceFeatureExtract**
- Can use ParaView to select angle (filter: **Feature Edges**)
- Can use ParaView to load edgeMesh.obj



tool: surfaceFeatureExtract

```
ahmedBody.stl
{
    extractionMethod    extractFromSurface;

    extractFromSurfaceCoeffs
    {
        includedAngle   150;
        geometricTestOnly       yes;
    }
    // for post-processing
    writeObj                    yes;
}
```



4. Select refinement levels

- **refinementSurfaces:** cells are refined if intersected

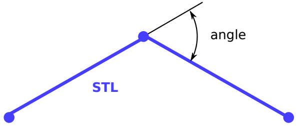
```
refinementSurfaces
{
    ahmedBody
    {
        level (0 0);

        regions
        {
            body { level (3 4); }
            legs { level (4 5); }
        }
    }
}
```

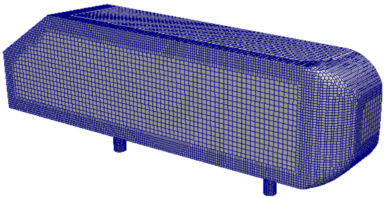
- **refinementRegions:** cells are refined if inside/outside/within distance from surface:

```
refinementRegions
{
  refinementBox
  {
    mode inside; //outside, distance
    levels ((1e15 3 ));
  }
}
```

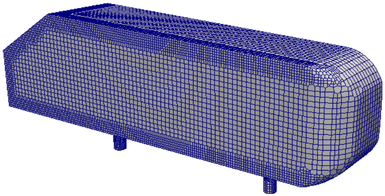
Select feature angle

- if angle $>$ featureAngle, cells are refined up to the maximum level





- angle = 10°

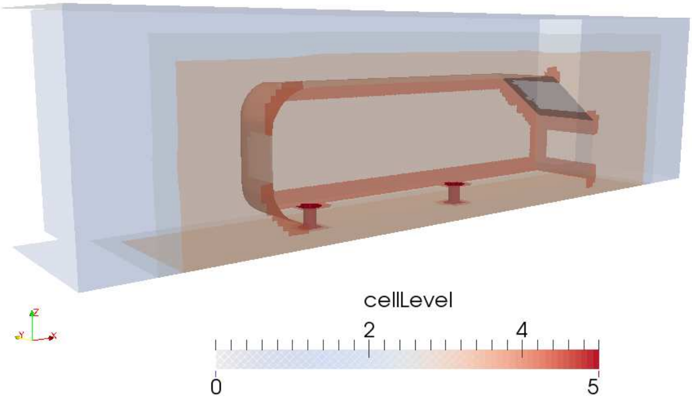


- angle = 50°

Check refinement levels

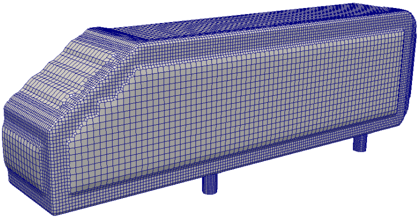
- You can save scalarField **cellLevels** for later post-processing

```
writeFlags
(
    scalarLevels
);
```

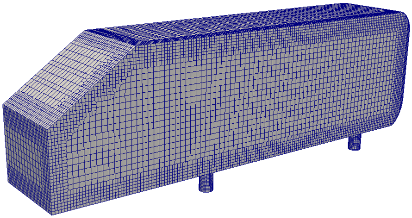


5. Snap the mesh

```
// Settings for the snapping.
snapControls
{
  nSmoothPatch 3;
  tolerance 2.0;
  nSolveIter 30;
  nRelaxIter 5;
  nFeatureSnapIter 10;

  implicitFeatureSnap false;
  explicitFeatureSnap true;
  multiRegionFeatureSnap false;
}
```



tolerance = 0.1



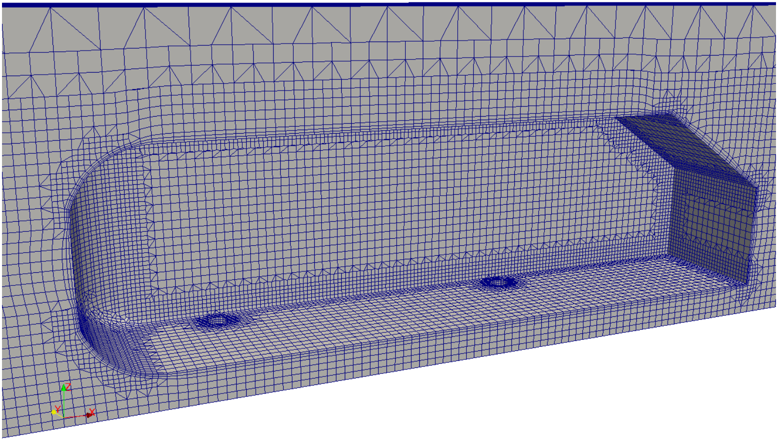
tolerance = 2.0

Snap controls: tips

- Default values are OK for most situations
- Increasing the number of iterations can increase a lot the generation time...
- ...But may produce a mesh with less errors
- Explicit feature recognition produce always best results...
- ...but requires additional steps

```
// Settings for the snapping.
snapControls
{
  nSmoothPatch 3;
  tolerance 2.0;
  nSolveIter 30;
  nRelaxIter 5;
  nFeatureSnapIter 10;

  implicitFeatureSnap false;
  explicitFeatureSnap true;
  multiRegionFeatureSnap false;
}
```

6. Adding wall layers



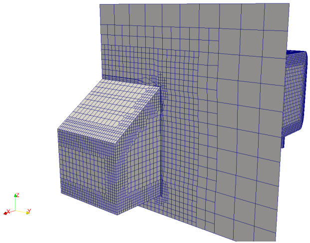
6. Adding wall layers

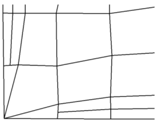
```
relativeSizes true;

// Per final patch
layers
{
    body
    {
        nSurfaceLayers 3;
    }
    legs
    {
        nSurfaceLayers 3;
    }
}

expansionRatio 1.3;
finalLayerThickness 0.4;

minThickness 0.1;
```

- Thickness in absolute units or relative to cell size
- Different methods to specify thickness:
  - expansionRatio and finalLayerThickness (cell nearest internal mesh)
  - expansionRatio and firstLayerThickness (cell on surface)
  - overall thickness and firstLayerThickness
  - overall thickness and finalLayerThickness
  - overall thickness and expansionRatio

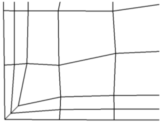
7. Wall layers: advanced settings

```
nGrow 0;

featureAngle 60;
slipFeatureAngle 30;

nRelaxIter 3;
nSmoothSurfaceNormals 1;
nSmoothNormals 3;
nSmoothThickness 10;

maxFaceThicknessRatio 0.5;
maxThicknessToMedialRatio 0.3;
minMedianAxisAngle 90;

nBufferCellsNoExtrude 0;
nLayerIter 50;
nRelaxedIter 20;
```





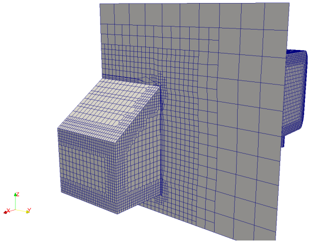
featureAngle 45

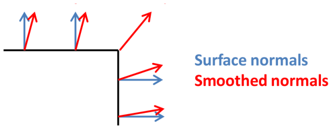


featureAngle 180

7. Wall layers: advanced settings

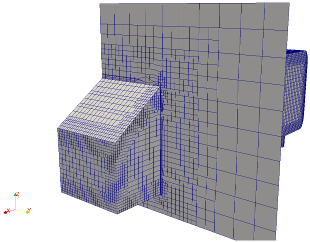
```
nGrow 0;

featureAngle 60;
slipFeatureAngle 30;

nRelaxIter 3;
nSmoothSurfaceNormals 1;
nSmoothNormals 3;
nSmoothThickness 10;

maxFaceThicknessRatio 0.5;
maxThicknessToMedialRatio 0.3;
minMedianAxisAngle 90;

nBufferCellsNoExtrude 0;
nLayerIter 50;
nRelaxedIter 20;
```

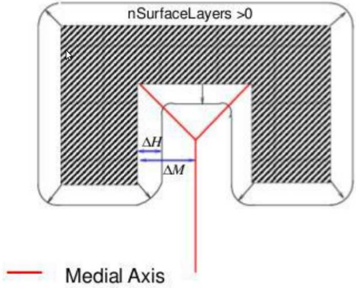




7. Wall layers: advanced settings

```
nGrow 0;

featureAngle 60;
slipFeatureAngle 30;

nRelaxIter 3;
nSmoothSurfaceNormals 1;
nSmoothNormals 3;
nSmoothThickness 10;

maxFaceThicknessRatio 0.5;
maxThicknessToMedialRatio 0.3;
minMedianAxisAngle 90;

nBufferCellsNoExtrude 0;
nLayerIter 50;
nRelaxedIter 20;
```

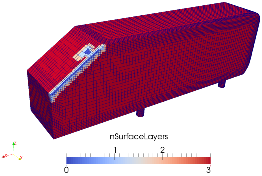


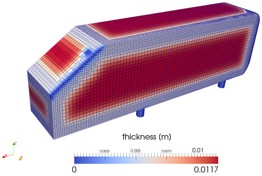


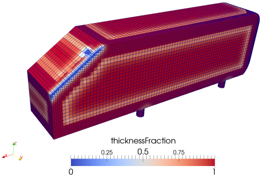
8. Wall layers: tips

- **layerFields** can help you to find what is wrong

```
writeFlags
(
    scalarLevels
    layerFields
);
```





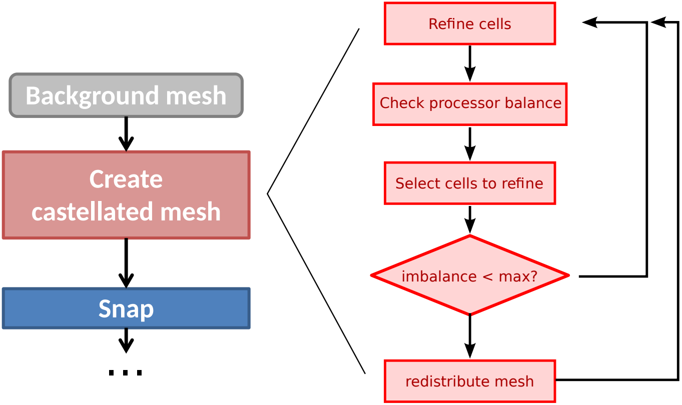


9. Mesh quality controls

```
meshQualityControls
{
    maxNonOrtho 65;
    maxBoundarySkewness 20;
    maxInternalSkewness 4;
    maxConcave 80;
    minVol 1e-13;
    minTetQuality 1e-15;
    minArea -1;
    minTwist 0.02;
    minDeterminant 0.001;
    minFaceWeight 0.05;
    minVolRatio 0.01;
    minTriangleTwist -1;
    //minVolCollapseRatio 0.1;

    nSmoothScale 4;

    errorReduction 0.75;

    relaxed
    {
        maxNonOrtho 75;
        maxBoundarySkewness 25;
        maxInternalSkewness 8;
    }
}
```

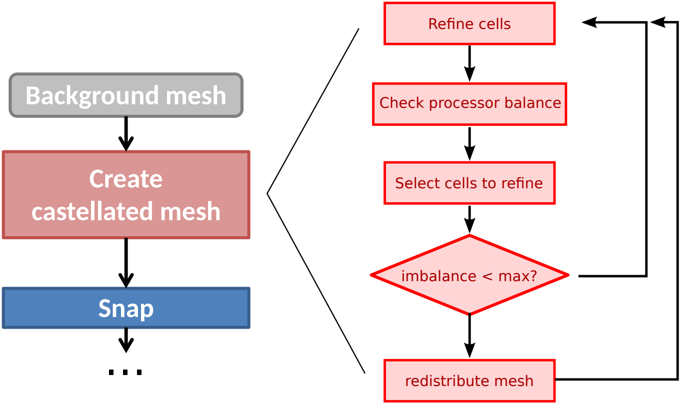
- If a constraint is not fulfilled, the action (snap, layer addition) is undone and another trial is made with a relaxed displacement
- Too loose constraints may result in a poor mesh
- Too tight constraint will result in a long process and, possibly, in no result (esp. layers)
- If the layer loop exits because **nLayerIter** has been reached, additional **nRelaxedIter** with looser quality constraints are performed

SnappyHexMesh on parallel architectures



- SnappyHexMesh can be run on any number of processors, thus achieving **parallel mesh generation**
- Exploits **ptscotch** parallel decomposition library
- Mesh needs to be redistributed so long as cells are refined, to maintain a balanced decomposition

SnappyHexMesh on parallel architectures



- Mesh redistribution is very time-consuming: a tradeoff must be sought
- Inter-processor communication is required also during snap and layer addition

Conclusions

- SnappyHexMesh is a tool for **automatic** mesh generation.
  - SHM is **not** a CAD tool: supplied geometry must be already 'clean'
  - Algorithm parameters must be supplied by the user (for some of them, default values are OK)
  - Perfect for parametric/optimization studies
- SnappyHexMesh is a tool for **parallel** mesh generation
  - Very convenient for large cases (up to 100M cells)
  - Works on any cluster with MPI architecture
  - Mesh redistribution is an actual bottleneck
  - Drawback: non-negligible memory consumption
- Mesh is hex-dominant: very good performance with OpenFOAM numerical solvers

SnappyHexMesh

- **Why should I need snappyHexMesh?** ,
  - You want an open-source tool capable of dealing with complex geometries
  - You want a cheap, hex-dominant mesh
  - You work with large cases, so scalability is important
  - You carry out parameter studies/optimization, so automatic operation is sought
- **When is it better not to use snappyHexMesh?** ,
  - you feel uncomfortable with dictionary interfaces (though some alternatives exist. . .)
  - you want a 101% control on mesh quality
  - you want a pure-hex mesh or an oriented mesh

Thank you for your attention!

Andrea Montorfano, Ph.D.
Post-doc Researcher

CONTACT INFORMATION

Address	Dipartimento di Energia, Politecnico di Milano via Lambruschini 4, 20156 Milano (ITALY)
E-Mail:	andrea.montorfano@polimi.it
Phone:	(+39) 02 2399 3804
Web page:	http://www.engines.polimi.it/

@ICEPoliMi