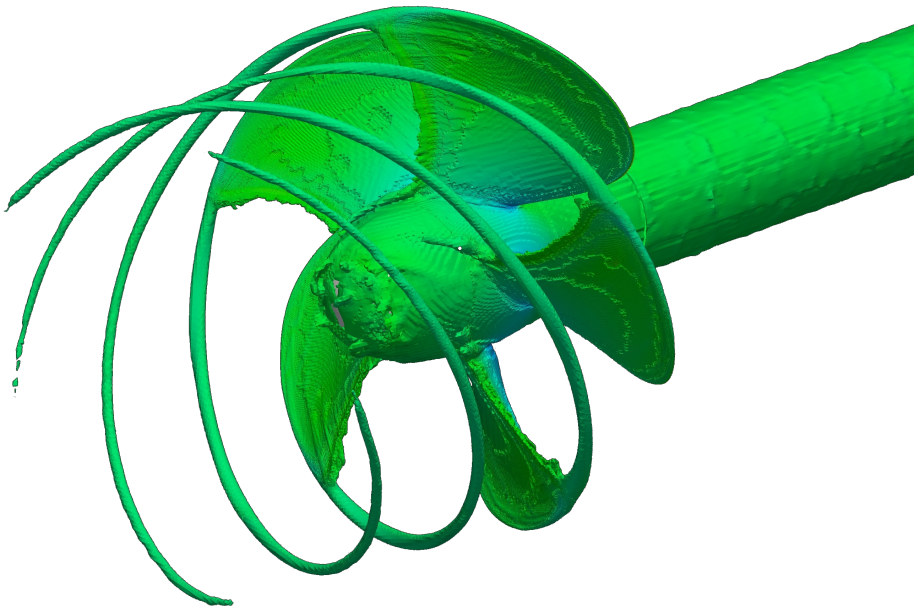
Meshing in OpenFOAM

Artur Lidtke

University of Southampton
akl1g09@soton.ac.uk

November 4, 2014

Content

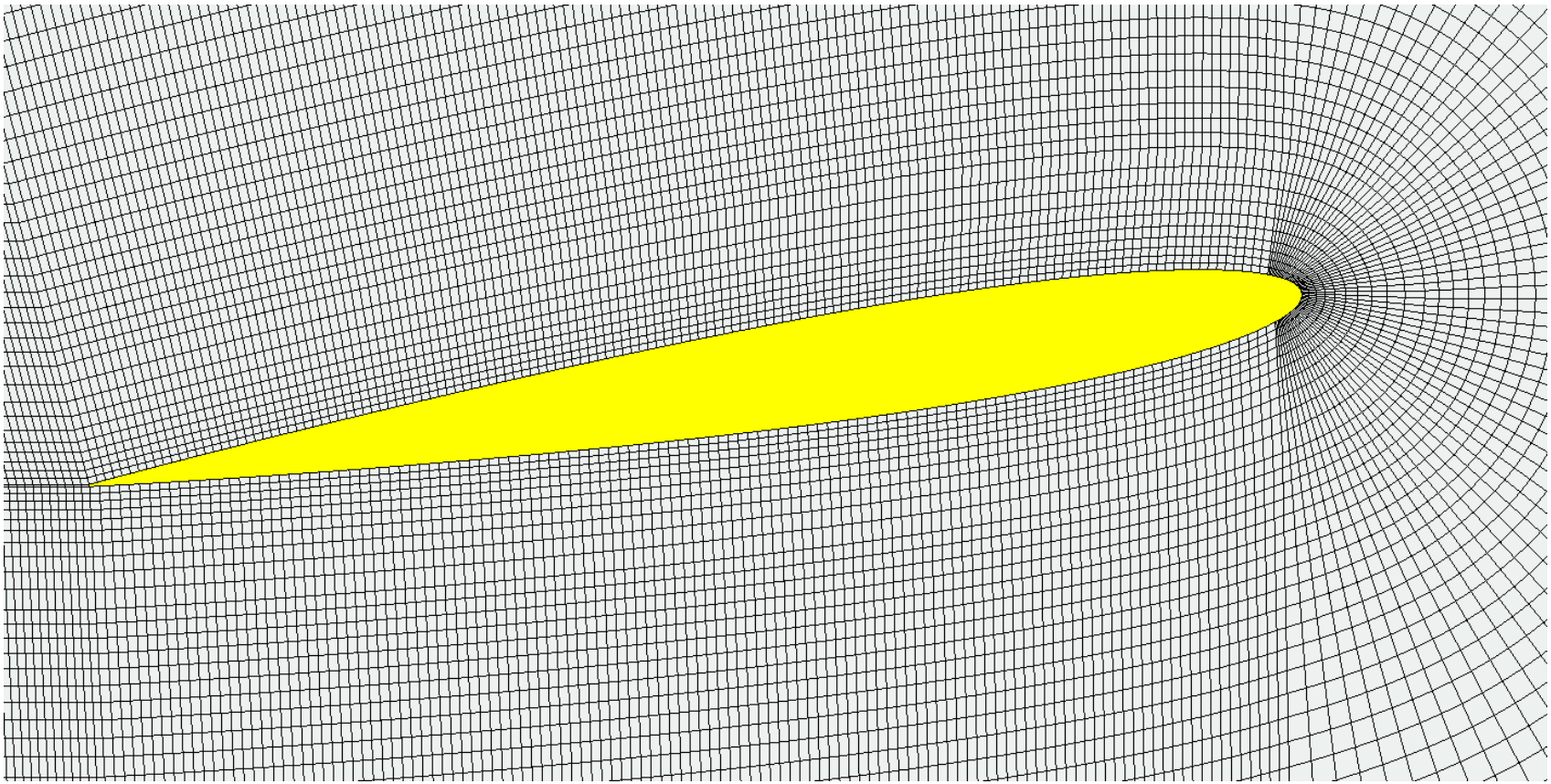


Structured meshing background

Basics

- Need to divide the computational domain into small finite volume elements
- "Structured" means that each element connects to as many elements as it has faces and the mesh is arranged in an ordely, topological fashion
- Typically use hexahedral elements (cubes) but may also use other polyhedra

NACA0009 - structured mesh example



Block topology

- Mesh is composed of one or more topological blocks
- Vertices of all cells of two neighbouring blocks must match
- The user may control the distribution of cell vertices along each edge of the block, thus having a lot of flexibility in terms of mesh design
- Many tools exist that allow this to be done; OpenFOAM provides the blockMesh utility

Test case: cavity tutorial

- Well covered in the official OpenFOAM documentation - useful to go through
- Simple, lid-driven cavity flow
- Now, let us examine how the mesh is generated

Test case: cavity tutorial

- Well covered in the official OpenFOAM documentation - useful to go through
- Simple, lid-driven cavity flow
- Now, let us examine how the mesh is generated

Go to

```
$FOAM_RUN/tutorials/incompressible/icoFoam/cavity
```

Test case: cavity tutorial

- Well covered in the official OpenFOAM documentation - useful to go through
- Simple, lid-driven cavity flow
- Now, let us examine how the mesh is generated

Edit

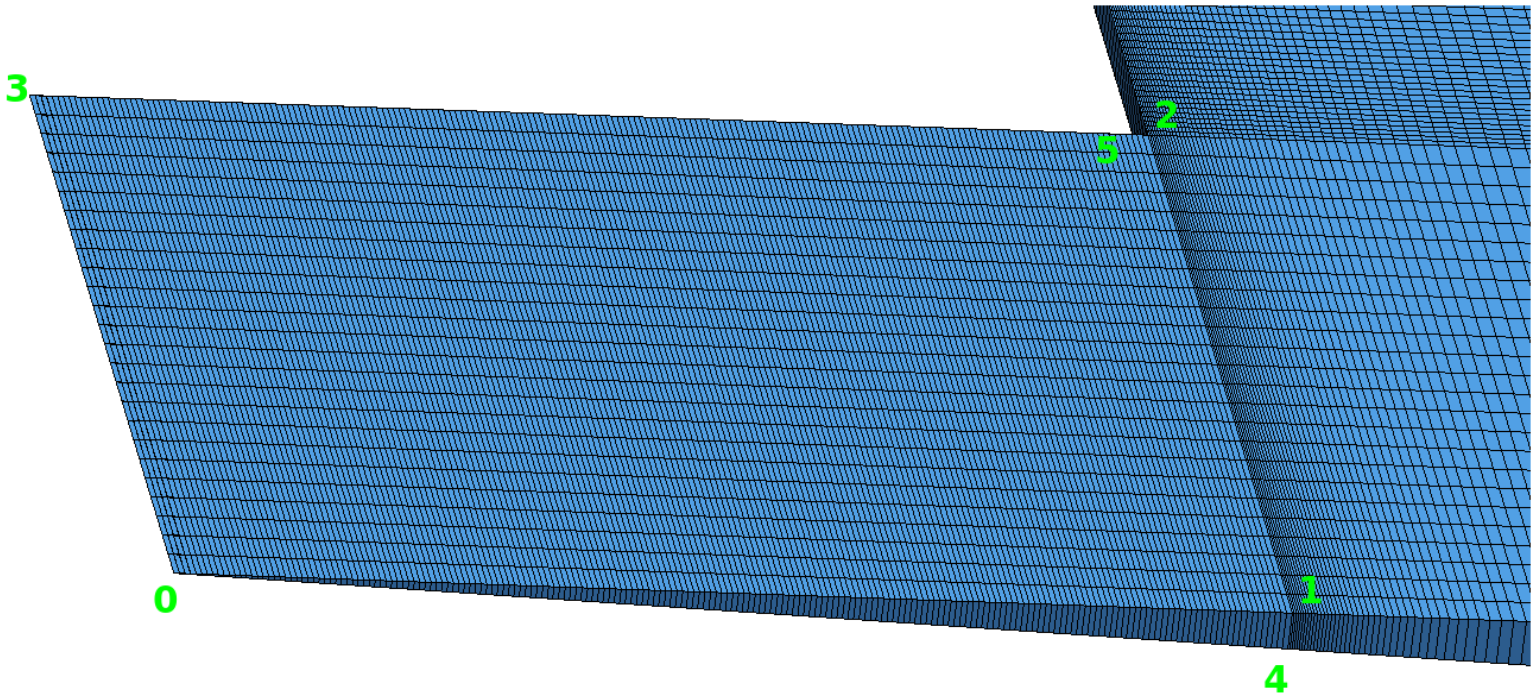
```
gedit ./constant/polyMesh/blockMeshDict
```

Vertex definition

```
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
```

- Order of the vertices is not important but they are then refered to by their index in the list
- Need 8 vertices for a single block of mesh
- May use fewer vertices in order to "collapse" a block, for instance to create a mesh wedge
- Neighbouring blocks must use the same vertices - avoid redefinition

Vertex definition



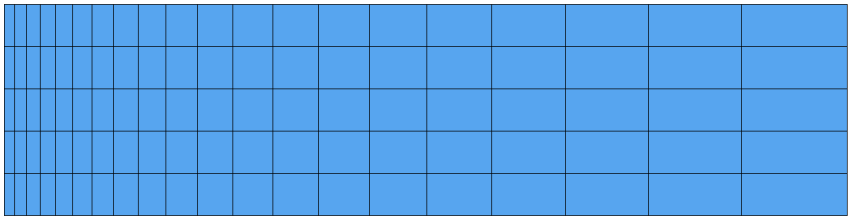
Mesh block definition

```
hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
```

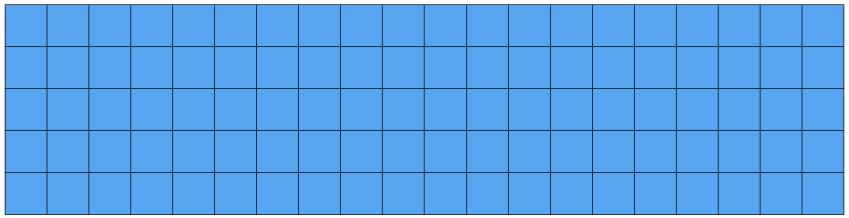
- `hex` stands for a hexahedral mesh block
- Followed by a list of vertices defining the block - order is *vital*
- First two vertices define the first principle direction of the block, next pair defines the second; the third principle direction is defined by moving between vertex 0 away from the plane defined by vertices 0-3
- The next integer list is the number of cells in each of the principle directions
- You may define separate grading for each edge separately or make it constant for each of the principle directions - keyword `simpleGrading`
- The last list is the expansion ratio for each of the directions - use 1 for constant spacing

Mesh block definition

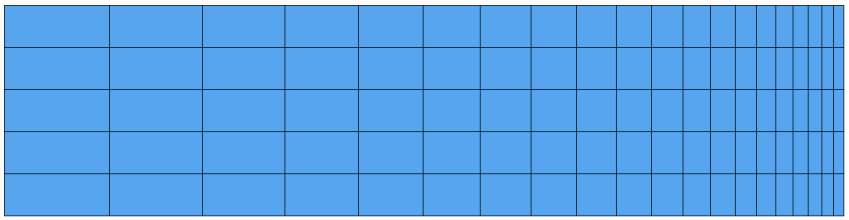
```
hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (10 1 1)
```



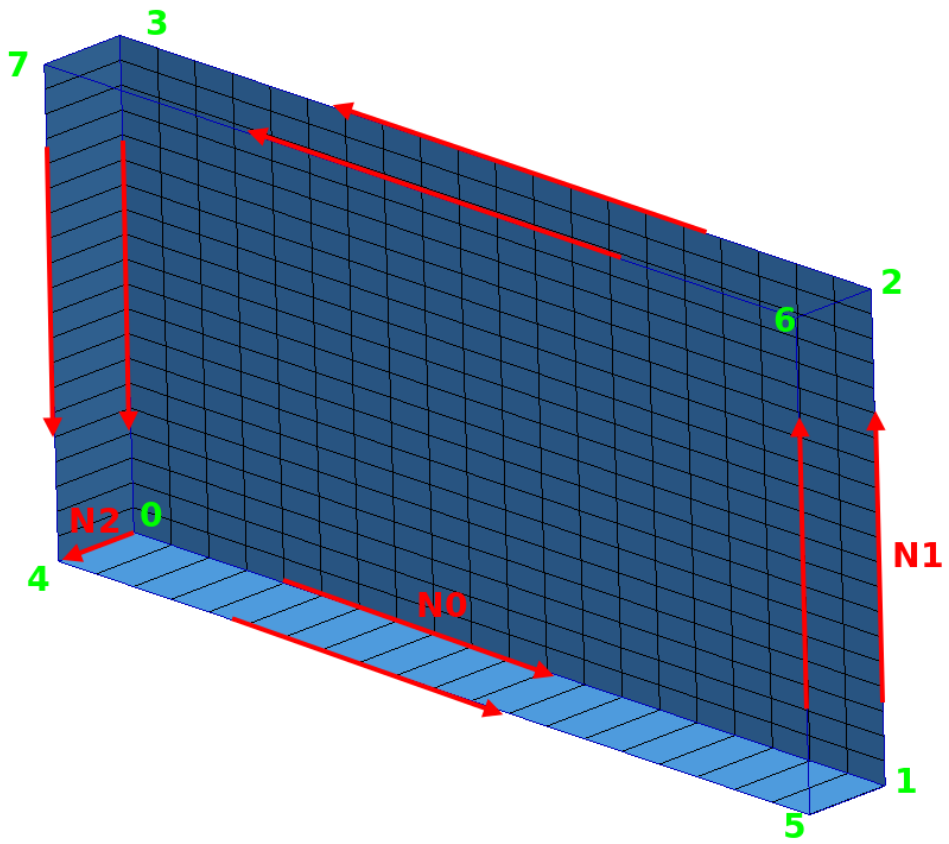
```
hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
```



```
hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (0.1 1 1)
```



Cavity case mesh and vertices



Boundary definition

```
movingWall
{
    type wall;
    faces
    (
        (3 7 6 2)
    );
}
```

- The first line defines the name of the new boundary - `movingWall`
- The type distinguishes between the physical type of the boundary - these may be `wall`, `empty`, `cyclic`, generic `patch` and more
- This is followed by a list of vertices defining the face of a mesh block; these should point *outside* of the mesh block according to the right-hand rule
- Several faces might be put together into one boundary

Cavity case mesh and vertices - viewing the grid

Use

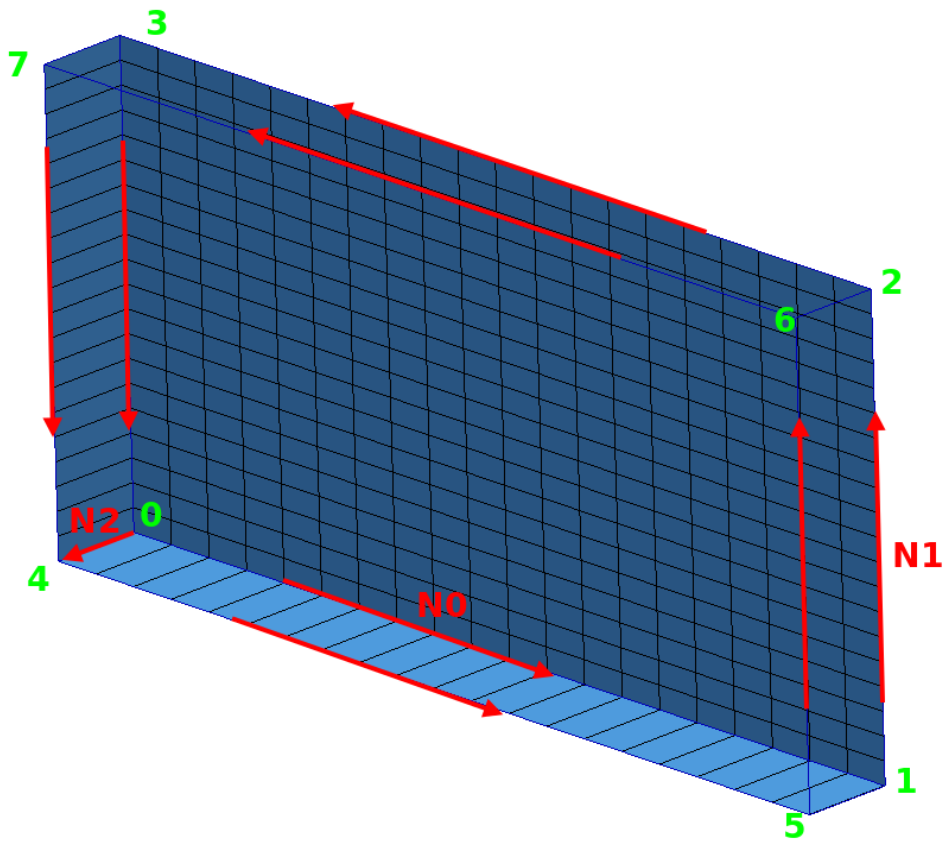
```
blockMesh
```

Cavity case mesh and vertices - viewing the grid

Use

- `paraFoam`
  OR
  `foamToVTK -latestTime && paraview`
- Select "Surface with edges" appearance

Cavity case mesh and vertices - viewing the grid



Practical session 1 - blockMesh

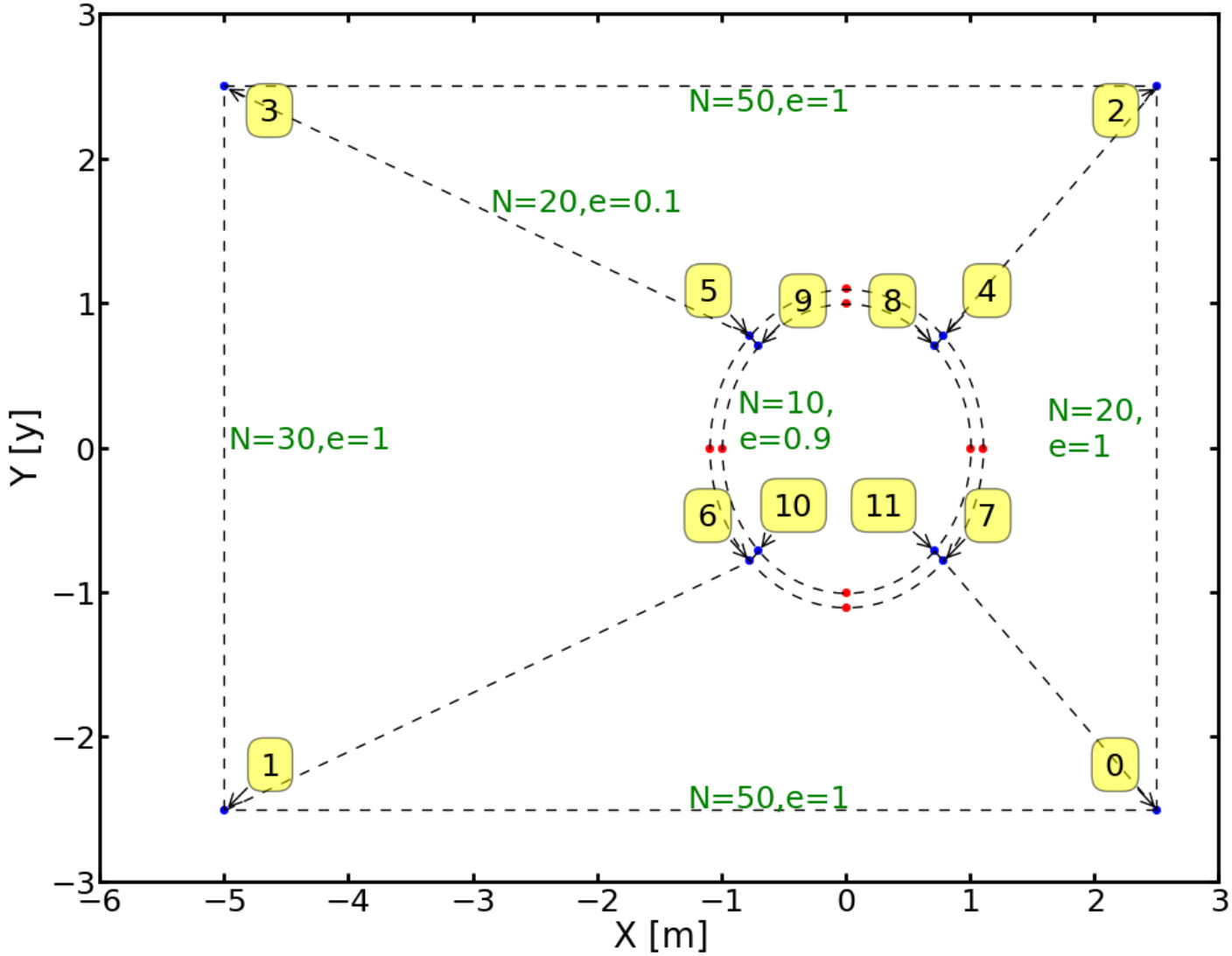
Aims and Objectives

- To show more advanced structured meshing concepts
- Introduce the idea of controlling the `blockMesh` utility using Python or other programming language
- Develop a basic understanding of what differentiates a good and a bad mesh

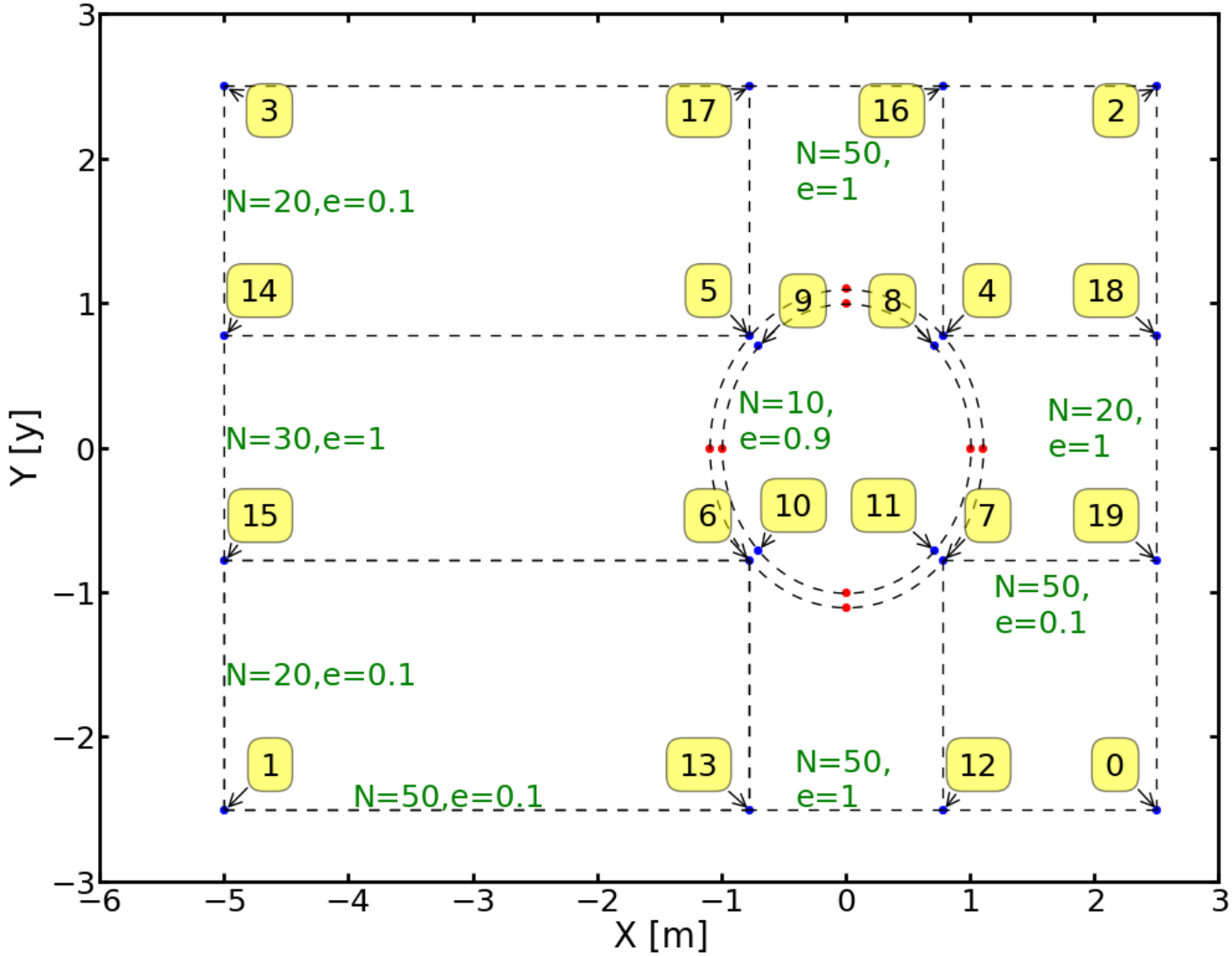
Test case

- Cylinder in uniform flow - canonical problem, relevant to most engineering applications
- For now just a 2D test case - easier to focus on mesh design and experiment a bit with short simulation times
- Will look a two conceptually different meshing approaches and discuss their potential advantages and disadvantages
- Both meshes set up using a Python scripts - makes it easier to define the block vertices and interpolation edges

Cylinder - Approach 1 - mesh layout



Cylinder - Approach 2 - mesh layout



Hands-on task - generate the first mesh

- Let us take a look at how the more basic of the two Python scripts works first - just a glance is enough

Hands-on task - generate the first mesh

- Let us take a look at how the more basic of the two Python scripts works first - just a glance is enough

Go to

```
cd $FOAM_RUN/cylinder2D_meshSimple
```

Hands-on task - generate the first mesh

- Let us take a look at how the more basic of the two Python scripts works first - just a glance is enough

Edit

```
gedit structuredMeshGenerator_simple_WHOI_2014.py
```

Hands-on task - generate the first mesh

- Let us take a look at how the more basic of the two Python scripts works first - just a glance is enough
- Execute the script to create the blockMeshDict file

Hands-on task - generate the first mesh

- Let us take a look at how the more basic of the two Python scripts works first - just a glance is enough
- Execute the script to create the blockMeshDict file

Use

`python structuredMeshGenerator_simple_WHOI_2014.py`

OR

edit in Spyder/Idle and press F5

Hands-on task - generate the first mesh

- Let us take a look at how the more basic of the two Python scripts works first - just a glance is enough
- Execute the script to create the blockMeshDict file
- Now, in terminal execute `blockMesh` on the newly created dictionary

Hands-on task - generate the first mesh

- Let us take a look at how the more basic of the two Python scripts works first - just a glance is enough
- Execute the script to create the blockMeshDict file
- Now, in terminal execute `blockMesh` on the newly created dictionary

Use

`blockMesh`

Hands-on task - generate the first mesh

- Let us take a look at how the more basic of the two Python scripts works first - just a glance is enough
- Execute the script to create the blockMeshDict file
- Now, in terminal execute `blockMesh` on the newly created dictionary
- Load the case to ParaView

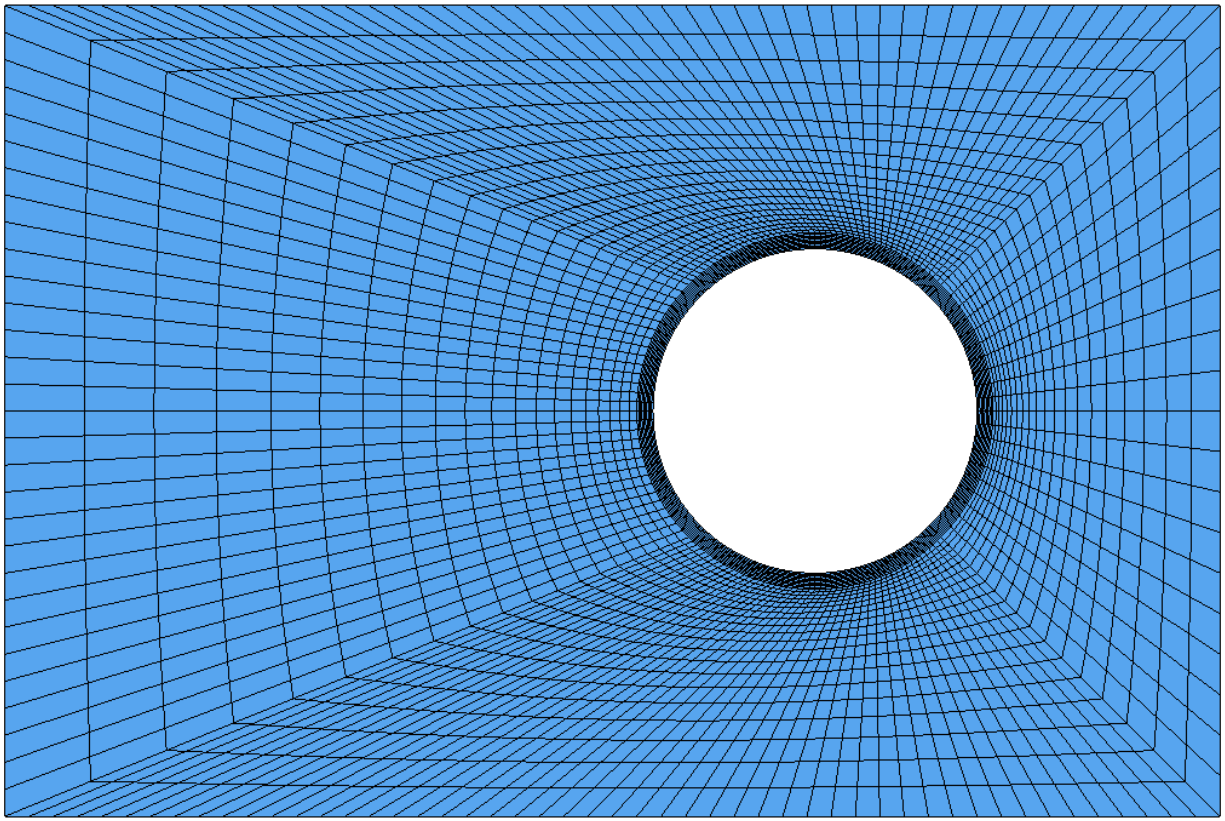
Hands-on task - generate the first mesh

- Let us take a look at how the more basic of the two Python scripts works first - just a glance is enough
- Execute the script to create the blockMeshDict file
- Now, in terminal execute `blockMesh` on the newly created dictionary
- Load the case to ParaView

Use

- Load the case to ParaView
- Select "Surface with edges" appearance

Cylinder - Approach 1 - mesh



Running the solver

- The test case has already been set up to run - we will cover more on solution techniques another time

Running the solver

- The test case has already been set up to run - we will cover more on solution techniques another time
- Execute the potential flow solver

Running the solver

- The test case has already been set up to run - we will cover more on solution techniques another time
- Execute the potential flow solver

Use

```
potentialFoam
```

Running the solver

- The test case has already been set up to run - we will cover more on solution techniques another time
- Execute the potential flow solver
- The case should converge quickly, let us examine the results now by importing them to ParaView

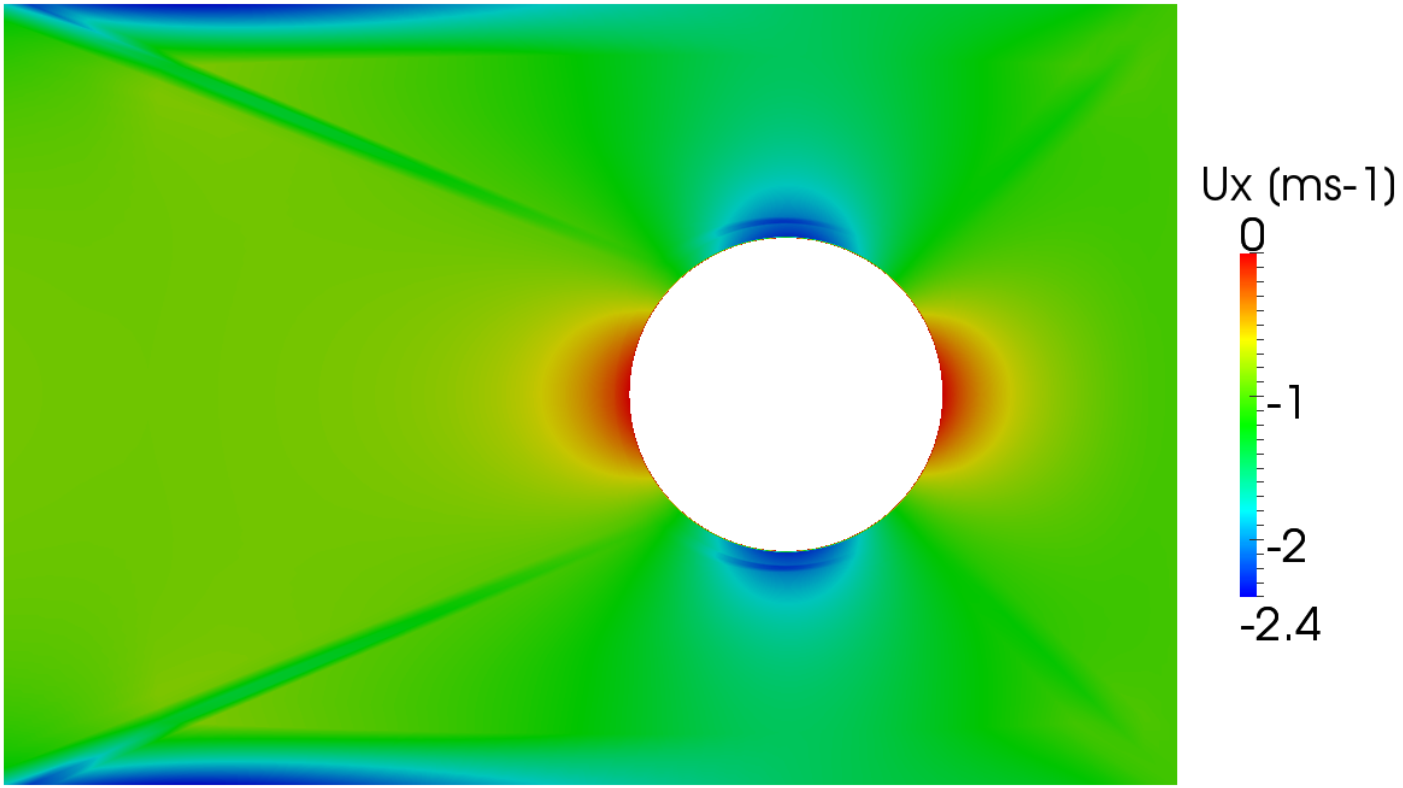
Running the solver

- The test case has already been set up to run - we will cover more on solution techniques another time
- Execute the potential flow solver
- The case should converge quickly, let us examine the results now by importing them to ParaView

Use

- Load the case to ParaView
- Select "Surface" appearance
- Colour by Ux

Cylinder - Approach 1 - Ux (potential flow)



Cylinder - Approach 1 - What we have learned

- In regions where mesh is not aligned with the flow unphysical jumps in the velocity field are visible
- At the horizontal walls, near the outlet more of problems occur
- Less prominent features seen in the boundary layer mesh - the matching of cell sizes and orientations is not perfect and may be seen in the solution
- Let us now examine the second approach

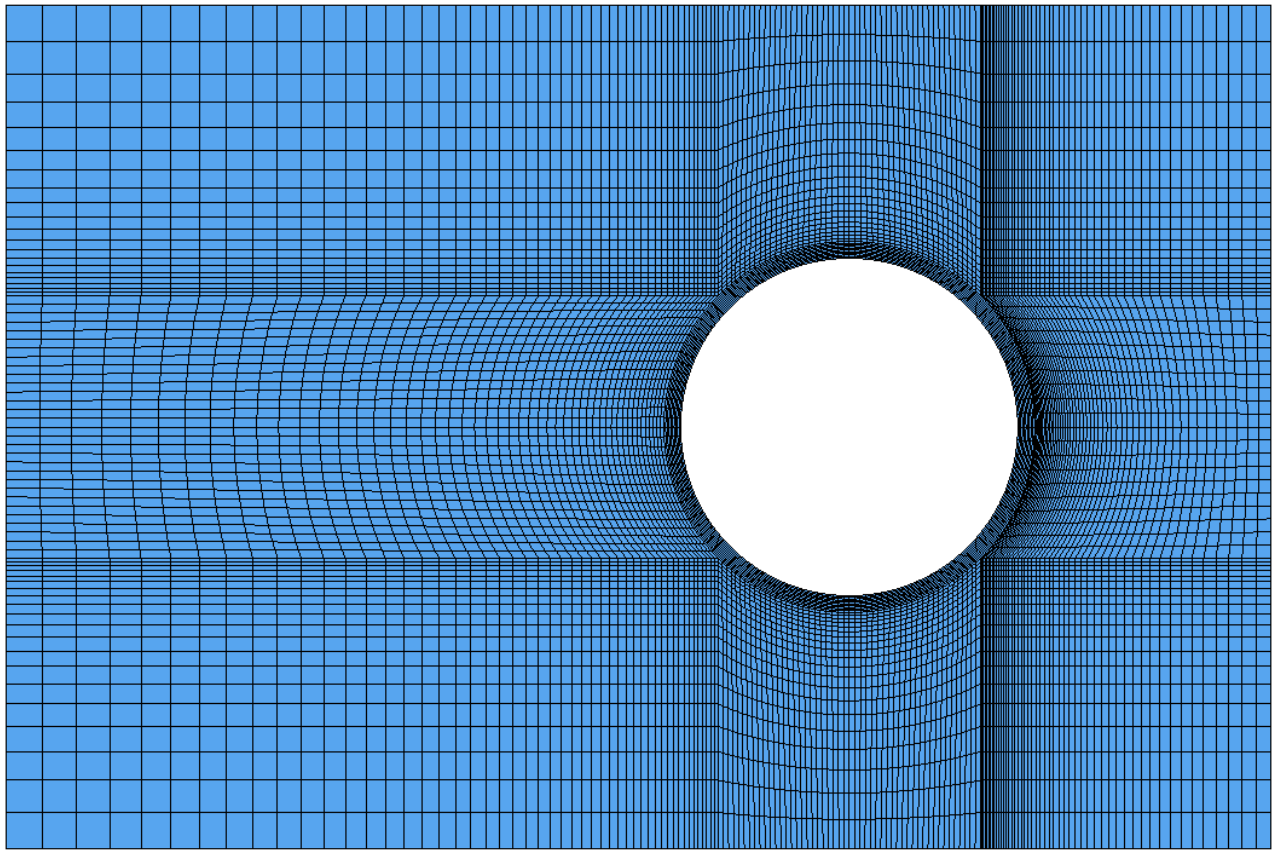
Cylinder - Approach 2 - mesh

Do

Create and visualise the mesh as in the first case

Use the `cylinder2D_meshComplex` case

Cylinder - Approach 2 - mesh



Cylinder - Approach 2 - Ux (potential flow)

Do

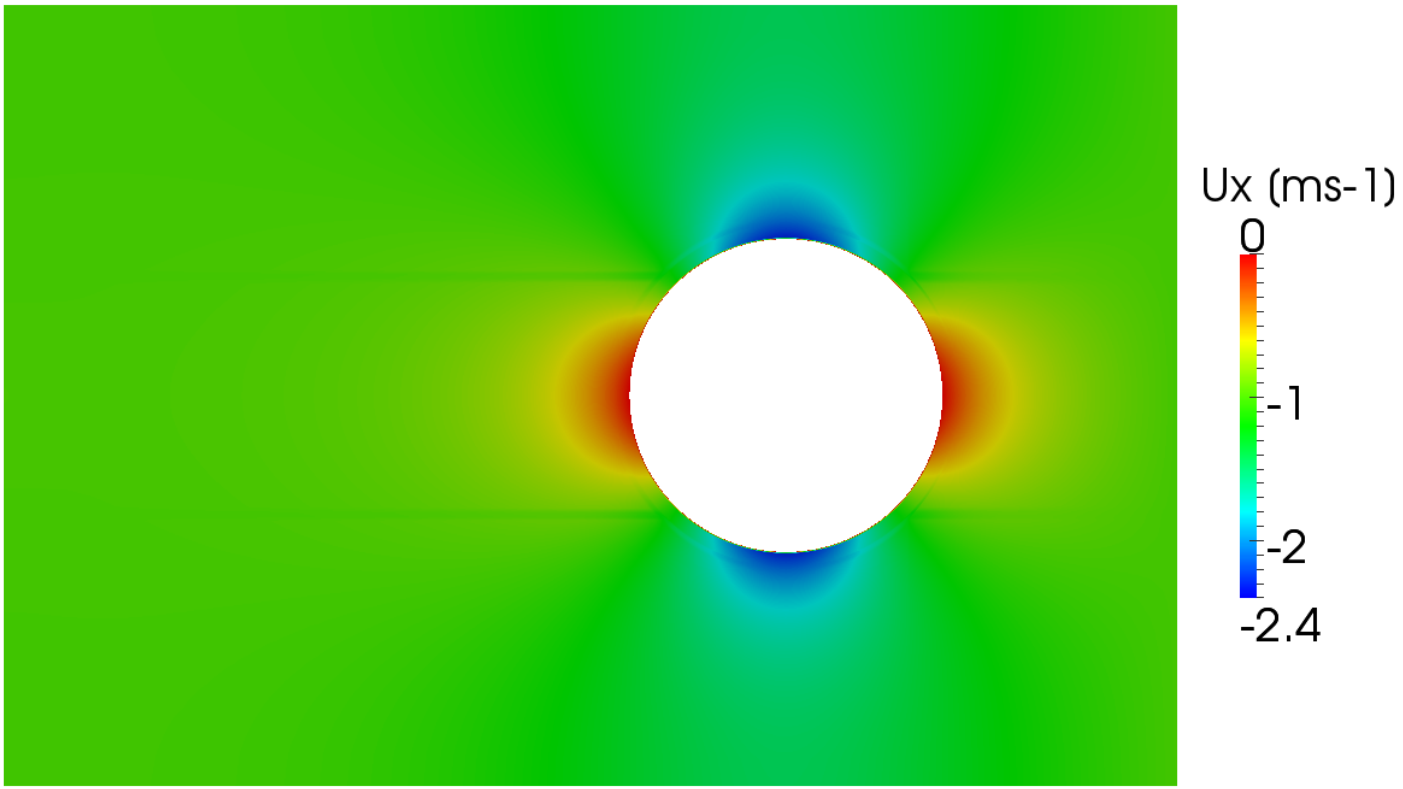
Run the potential flow solver

Cylinder - Approach 2 - Ux (potential flow)

Do

Load the solution to ParaView

Cylinder - Approach 2 - Ux (potential flow)



Cylinder - Approach 2 - What we have learned

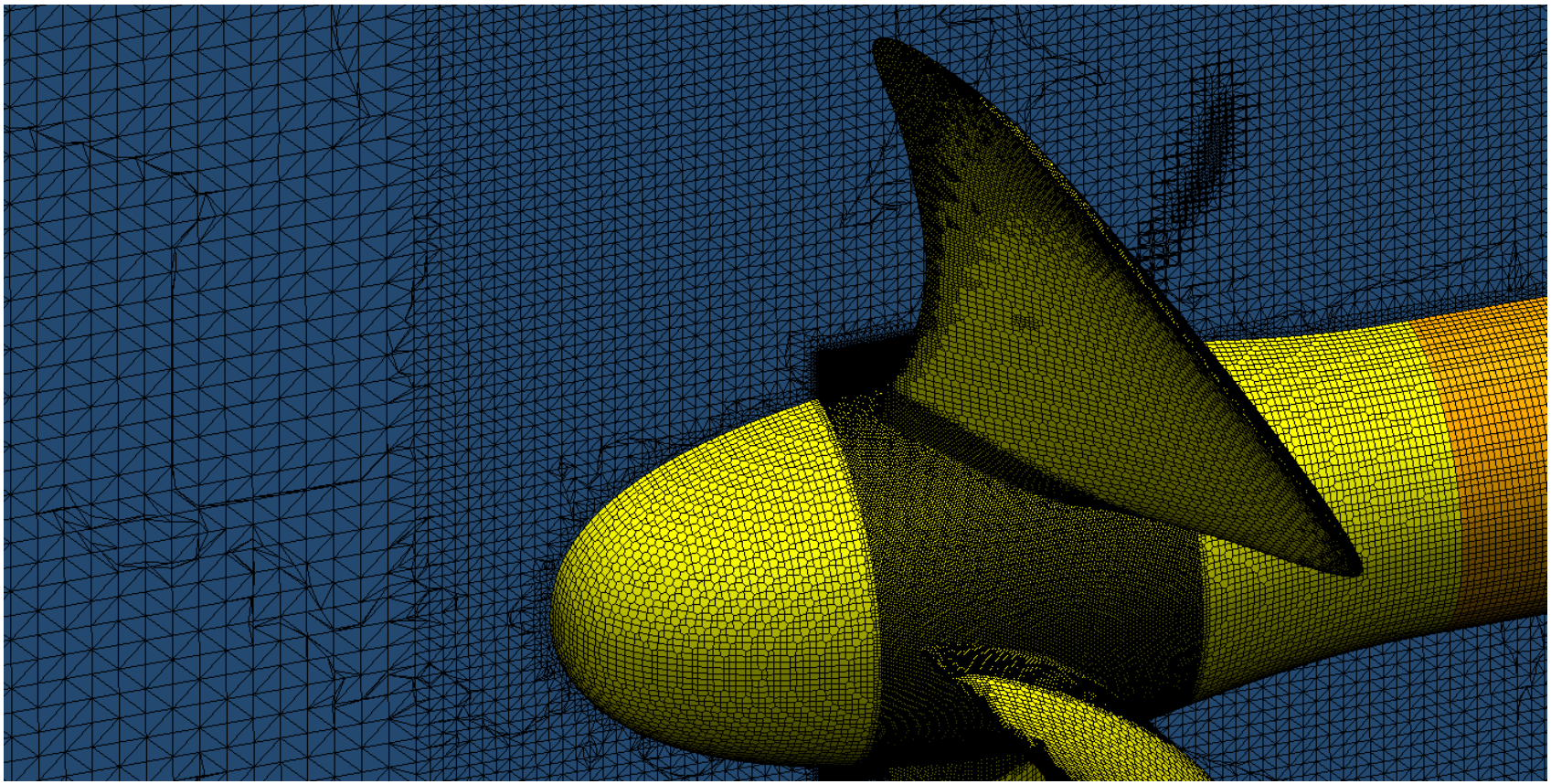
- Still can see some problems where mesh regions intersect but overall the solution is far better
- Aligning the mesh with the flow has had a profound effect on the predicted flow
- This indicates that a high amount of care needs to be taken when the mesh is being designed, for the sake of both stability and accuracy

Unstructured meshing background

Basics

- Instead of regularly spaced finite volumes like in a structured mesh in this case the domain is discretised with elements of arbitrary shapes and orientation
- More than one neighbouring face may be connected to a single face of any cell
- Most common practices use tetra- or hexahedral cells, but in principle any arbitrary shape is possible
- More flexbible than structured meshes, easier to prepare (in theory!) but often pose more difficulties in obtaining a stable and accurate solution; keeping the cell count managable may also prove a challenge

Propeller - unstructured mesh example



The process

- Create a background structured mesh - want cell aspect ratio as close to 1 as possible for good effect
- Three basic steps:
  1. Castellated mesh step:
     - Refine the grid where better flow resolution is needed - more on this later
     - Remove the cells bounded by the geometry of the body given as an .stl file
  2. Snapping phase - morph the mesh so that it adopts the shape of the geometry
  3. Layer addition phase - move the mesh away from the body surface in order to add the boundary layer cells

More on snappyHexMesh

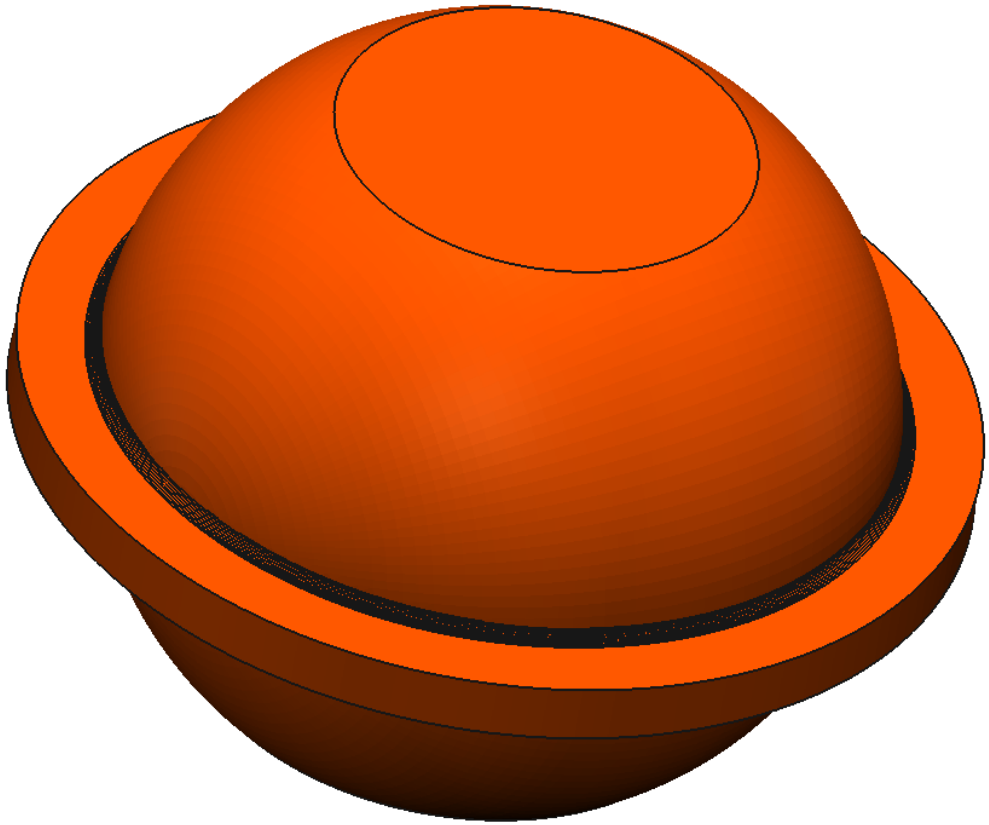
- Many control parameters - takes a substantial amount of time to learn all of them and truly understand their effect on the mesh generation process
- Difficult to experiment with all parameters when under time pressure - important to look at tutorials and develop a few example test cases for yourself to re-use
- Worth doing each of the 3 steps independently and only proceeding forward when you are happy with what you have so far
- Mesh quality controls play an important role but are not very obvious - spend some time looking at them
- All tutorials describe the meaning of each parameters - good source of information

Practical session 2 - snappyHexMesh

Aims and Objectives

- (Briefly) Explain how changing individual snappyHexMesh parameters affects the final mesh
- Indicate the important steps needed to create a decent grid for an engineering problem
- Compare and contrast the fundamental differences between the structured and unstructured meshing techniques
- Now, let's look at an example geometry of a ocean bottom "seismometer"

Sample geometry - "seismometer"



Hands-on task - creating an unstructured mesh

- Go step-by-step (castellated - snap - layers) to create a final mesh
- Decide on final set-up and create the grid
- Check mesh quality
- Identify the problematic areas and think about how to improve them

Including the geometry

Go to

```
cd $FOAM_RUN/seismometer
```

Including the geometry

Edit

```
gedit system/snappyHexMeshDict
```

Including the geometry

```
seismometer.stl // name of the file
{
    type triSurfaceMesh;
    // name of the solid inside the .stl file
    name seismometer;
    // w/o this patch will be named bullet_patch
    regions
    {
        // solid name inside the .stl file
        patch0 // different for all CAD programs
        {
            name seismometer; // new name of the patch
        }
    }
}
```

Prescribed shape definition

```
// this allows one to define constants in an external
// file and use them here
#include "runConditions"

...

// specify points on the long axis and radius
refinementCyl
{
    type searchableCylinder;
    // note the syntax for including values from
    // and external file using the $ sign
    point1 ($xuCyl 0.0 $zCyl);
    point2 ($xlCyl 0.0 $zCyl);
    radius $rCyl;
}
```

Using external control files - sidenote

- Makes adapting the test case easier - no need to edit dozens of files if all crucial parameters are in one place
- Makes using bash scripts a bit easier (arguably)
- May link core OpenFOAM files, for instance to use topology manipulation based on snappyHexMesh refinement regions
- Example:

```
xuCyl  0.5;
xlCyl  -3.5;
```

Castellated mesh step - general settings

```
// how many intermediate cells to put between regions
// whose refinement levels differs by more than 1
nCellsBetweenLevels $nCellBetLvl;

// the bigger this is the higher the incentive to
// refine surfaces to the maximum level
resolveFeatureAngle $featAngleRef;

// define a point (inside a cell) to mark a region
// which is kept (every cell that cannot be reached
// from this point without crossing a geometry will
// be discarded)
locationInMesh $locInMesh;

// controls whether to keep faces just inside the .stl
allowFreeStandingZoneFaces false;
```

Castellated mesh step - refinement surfaces

```
// this will use the imported .stl, refine it and turn
// it into a physical boundary
seismometer
{
    // minimum and maximum refinement level
    // final refinement also depends on the
    // resolveFeatureAngle and how
    // the geometry is aligned with the background mesh
    level           ($seisLvlMin $seisLvlMax);
}
```

Castellated mesh step - refinement regions

```
// define minimum and maximum levels as for a surface
// -> might be useful to keep it constant to be sure
// of what the final level will be exactly
refinementCyl
{
    mode inside;
    levels ((1e15 $refCylLvl));
}

refinementBox
{
    mode inside;
    levels ((1e15 $refBoxLvl));
}
```

Castellated mesh step - refinement regions

```
// may also refine at a distance from a body - order
// by incresing distance & decreasing level
seismometer
{
    mode distance;
    levels
    (
        ($seisMaxLvlRefDist $seisLvlMax)
        ($seisMinLvlRefDist $seisLvlMin)
    );
}
```

Castellated mesh step - edges

- Created using `surfaceFeatureExtract`, dictionary placed in `$CASE/system`
- The higher the extraction angle the more features will be included
- Very important feature for good edge quality

```
{
    file        "seismometer.eMesh";
    level       $seisLvlMax;
}
```

Castellated mesh step - effect

Edit

Set "snap" and "addLayers" parameters to false in the file preamble

Castellated mesh step - effect

Edit

```
gedit prepareCase
```

Castellated mesh step - effect

Use

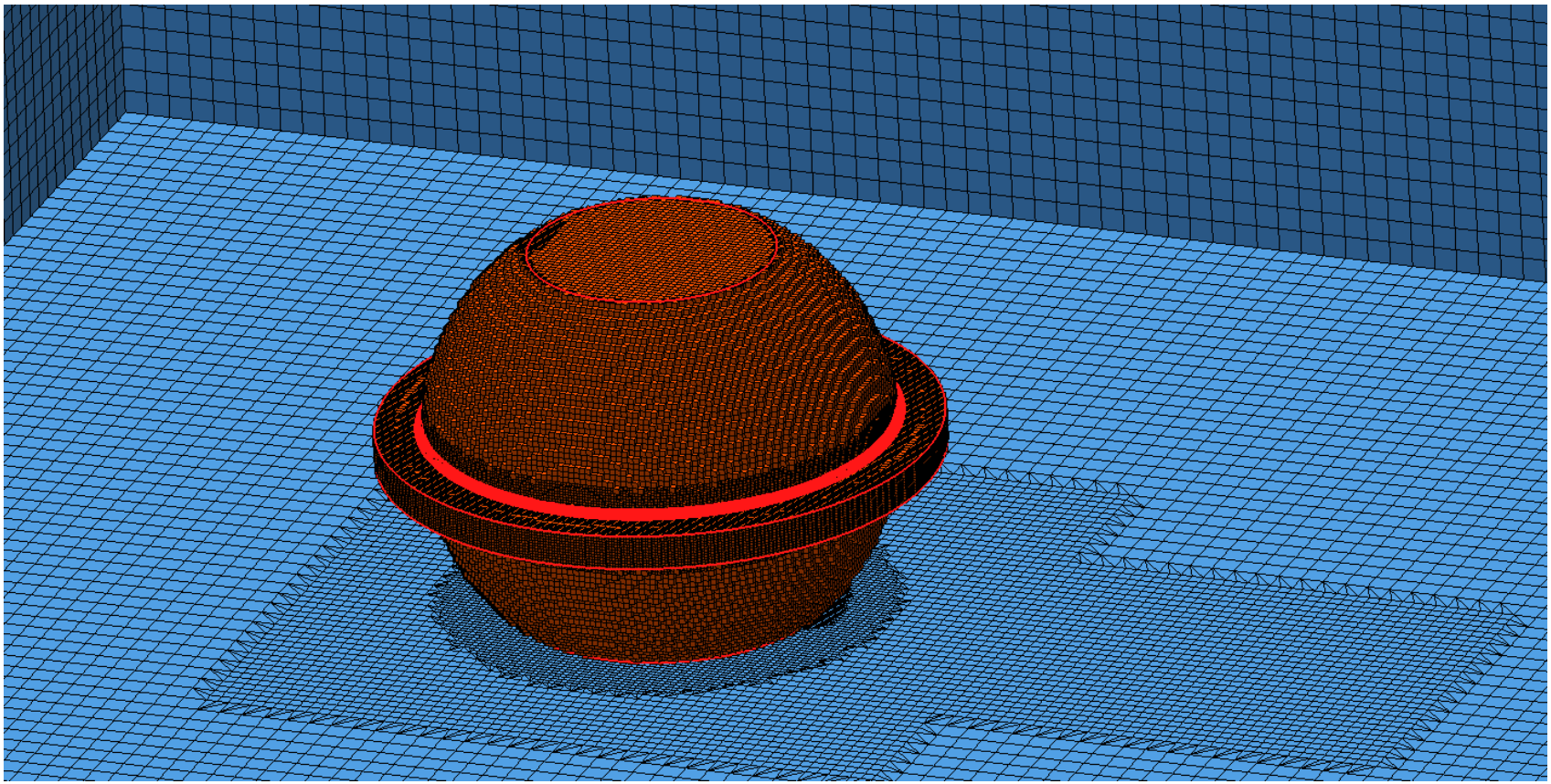
```
./prepareCase
```

Castellated mesh step - effect

Use

- Load the case to ParaView
- Load the "seismometer" and "bottom" patches
- Set appearance to "Surface with edges"

Castellated mesh step - effect



Snapping step - settings

```
// how many times to smoothen patch points
nSmoothPatch 2;

// how far to look for vertices to snap
// (distance relative to cell size)
tolerance 1.1;

// no. iterations of the solver for the point
// displacement field; reasonably cheap, can
// keep it high for good results
nSolveIter 300;

// no. iterations for undoing the displacement
// in order to reduce the errors
nRelaxIter 5;
```

Snapping step - settings

```
// how many times to perform the morphing phase
// most expensive part of mesh generation but
// keeping it high tends to give better quality meshes
nFeatureSnapIter $nSnapIter;

// whether to detect feature edges automatically
implicitFeatureSnap false;

// whether to use edges specified in castellated
// mesh settings
explicitFeatureSnap true;

// share explicit feature edges between different
// surfaces; useful for complex, compund geometries
// (e.g. fully appended ship)
multiRegionFeatureSnap false;
```

Snapping step - effect

Do

- Reset the case - `./resetCase`
- Create the mesh witht the "snap" phase enabled

Snapping step - effect

Note

In the log watch out for:

`Did not succesfully snap mesh. Giving up`

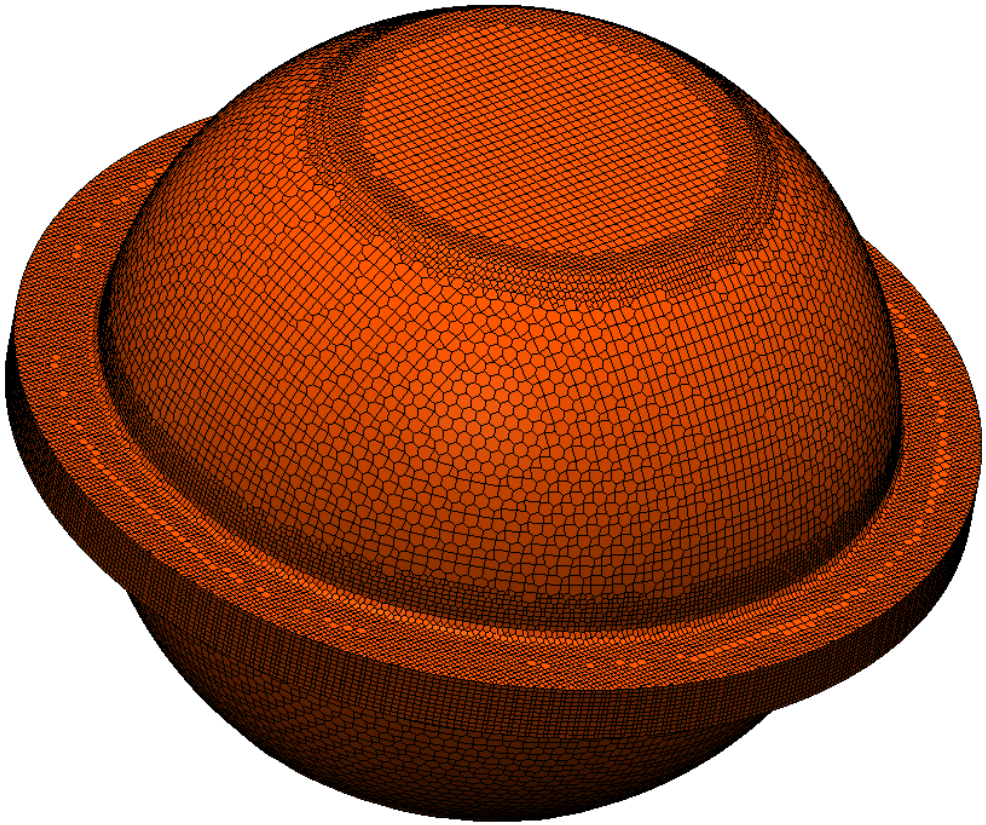
If this occurs revisit your quality controls

Snapping step - effect

Do

Visualise the mesh for the "seismometer" patch

Snapping step - effect



Layer addition step - settings

```
// size relative to background mesh?
relativeSizes true;
// growth expansion ratio
expansionRatio 1.3;
// thickness of outer-most layer
finalLayerThickness 0.7;
// minimum thickness below which no layers are added
minThickness 0.025;
// angle below which not to extrude layers
// 180 everywhere, 0 flat surface only
featureAngle 180;
// total no. layer addition iterations
nLayerIter 50;
```

Layer addition step - settings

```
// setting for each patch
seismometer
{
    // no. layers - mandatory
    nSurfaceLayers $nSeisLayers;
    // optional redefinition of layer
    // parameters for this patch
    expansionRatio      $expRat;
    finalLayerThickness $finalThick;
    minThickness        0.1;
}
```

Layer addition step - settings

```
// If points get not extruded do nGrow layers of
// connected faces that are also not grown. This
// helps convergence of the layer addition process
// close to features.
nGrow 0;
// Maximum number of snapping relaxation iterations
nRelaxIter 2;
// no. surface normal smoothing iterations
nSmoothSurfaceNormals 1;
// no. smoothin iterations of interior mesh movement
nSmoothNormals 3;
// no. thickness equalisation iterations
nSmoothThickness 10;
```

Layer addition step - settings

```
// avoid extrusion for highly warped cells
// (the larger the value the more layers added
// but may create invalid cells)
maxFaceThicknessRatio 0.7;
// similar criterion but with a median angle
maxThicknessToMedialRatio 0.4;
// medial axis choice
minMedianAxisAngle 90;
// add a buffer region for layer termination
nBufferCellsNoExtrude 0;
// Max number of iterations after which relaxed
// meshQuality controls get used. Up to nRelaxIter
// it uses the settings in meshQualityControls,
// after nRelaxIter it uses the values in
// meshQualityControls::relaxed.
nRelaxedIter 20;
```

Layer addition step - effect

Do

- Reset the case - `./resetCase`
- Create the mesh witht the "addLayers" phase enabled

Layer addition step - effect

Note

In the log file always check - want as close to 100% as possible:

```
Extruding 75576 out of 75576 faces (100%).
Removed extrusion at 0 faces.
Added 150848 out of 151152 cells (99.8%).
```

Layer addition step - effect

Do

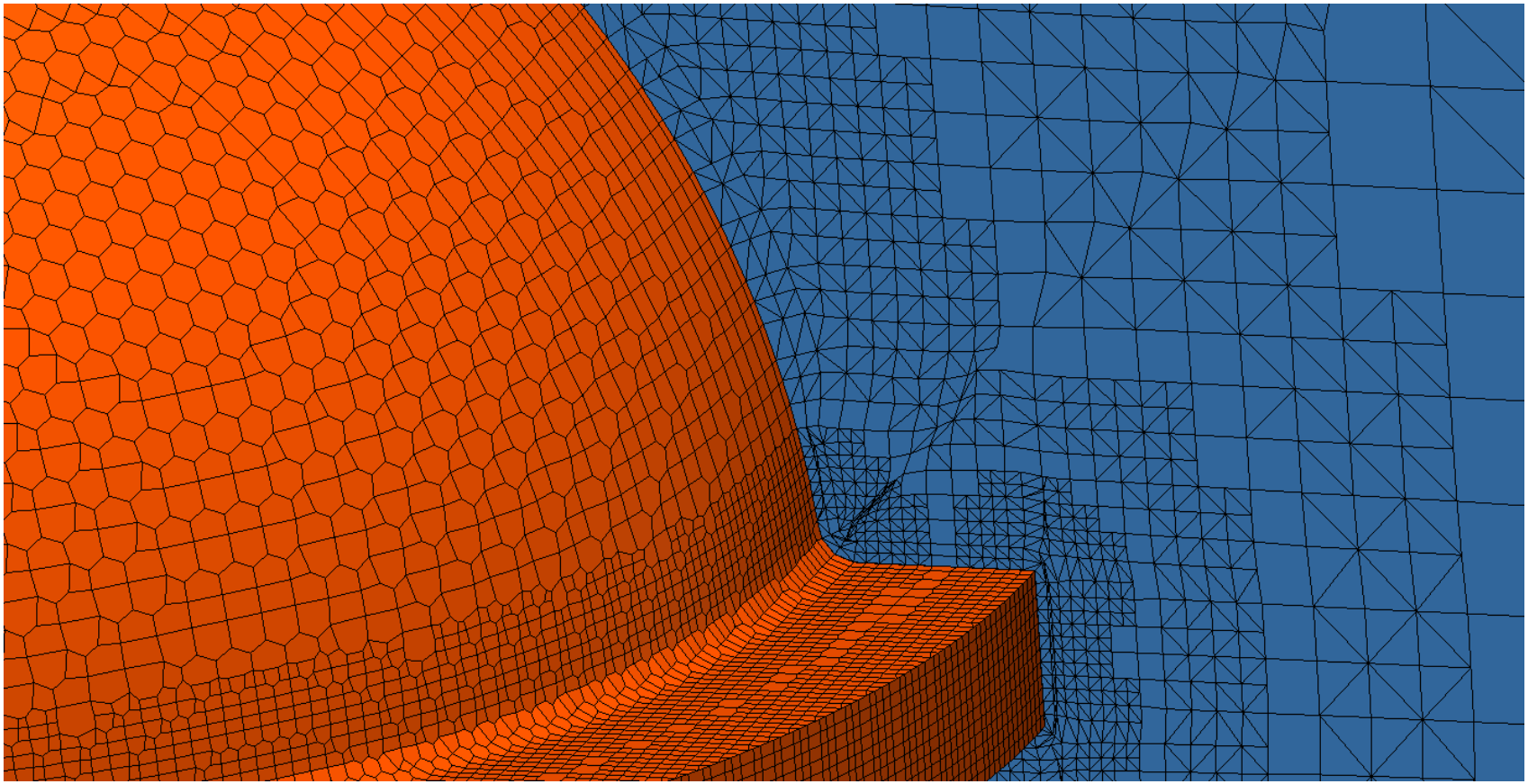
Visualise the mesh for the "seismometer" patch

Layer addition step - effect

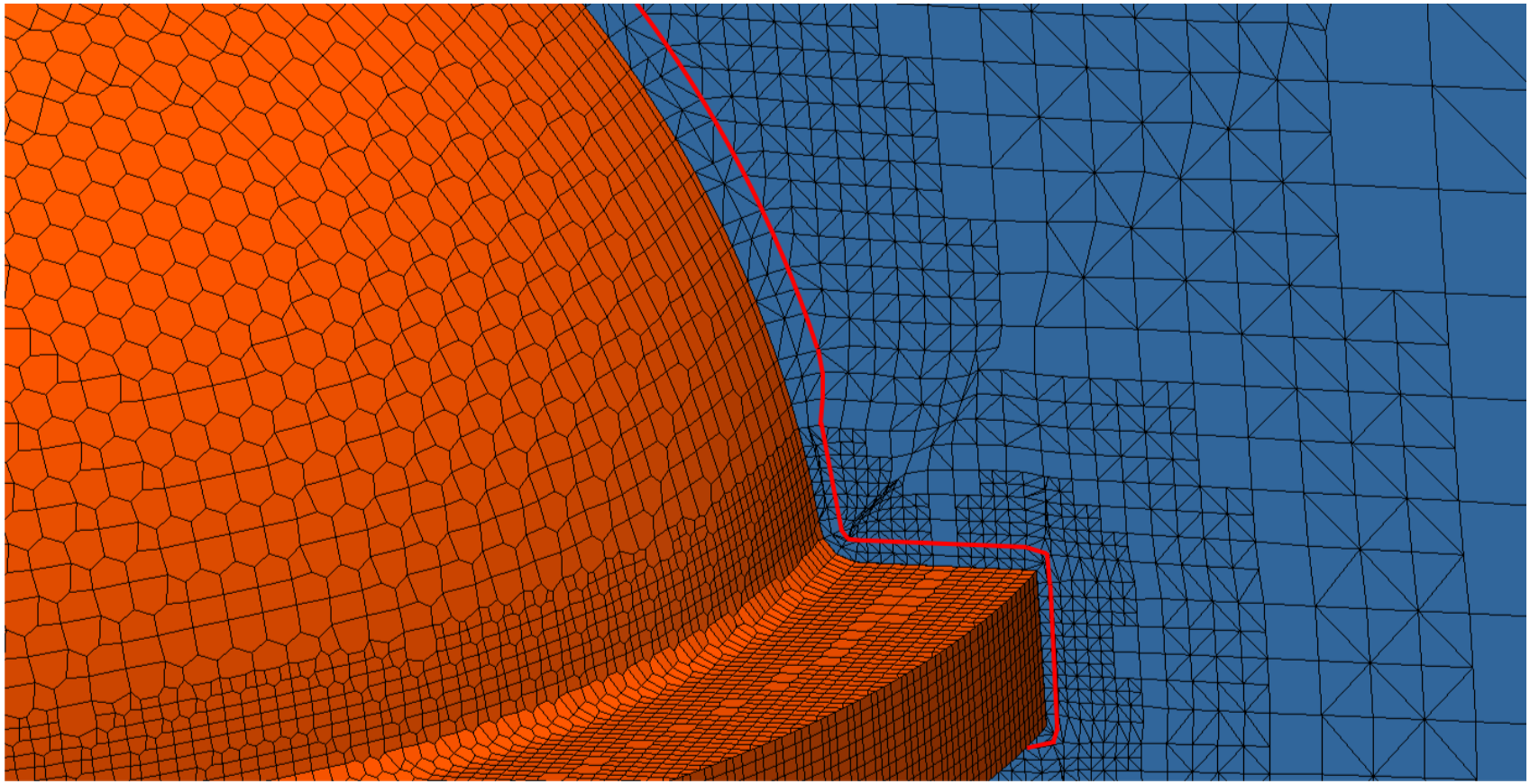
Use

- Apply the "Cut plane" filter normal to y-axis
- Set "Surface with edges" appearance
- Examine the mesh close to the "seismometer"

Layer addition step - effect



Layer addition step - effect



Next steps

- Check the mesh quality

Next steps

- Check the mesh quality

Use

`checkMesh -latestTime`

OR

`checkMesh -constant` - if using "-overwrite" option for snappyHexMesh

Next steps

- Check the mesh quality
- If problems occur new sets of cells/faces will be created

Next steps

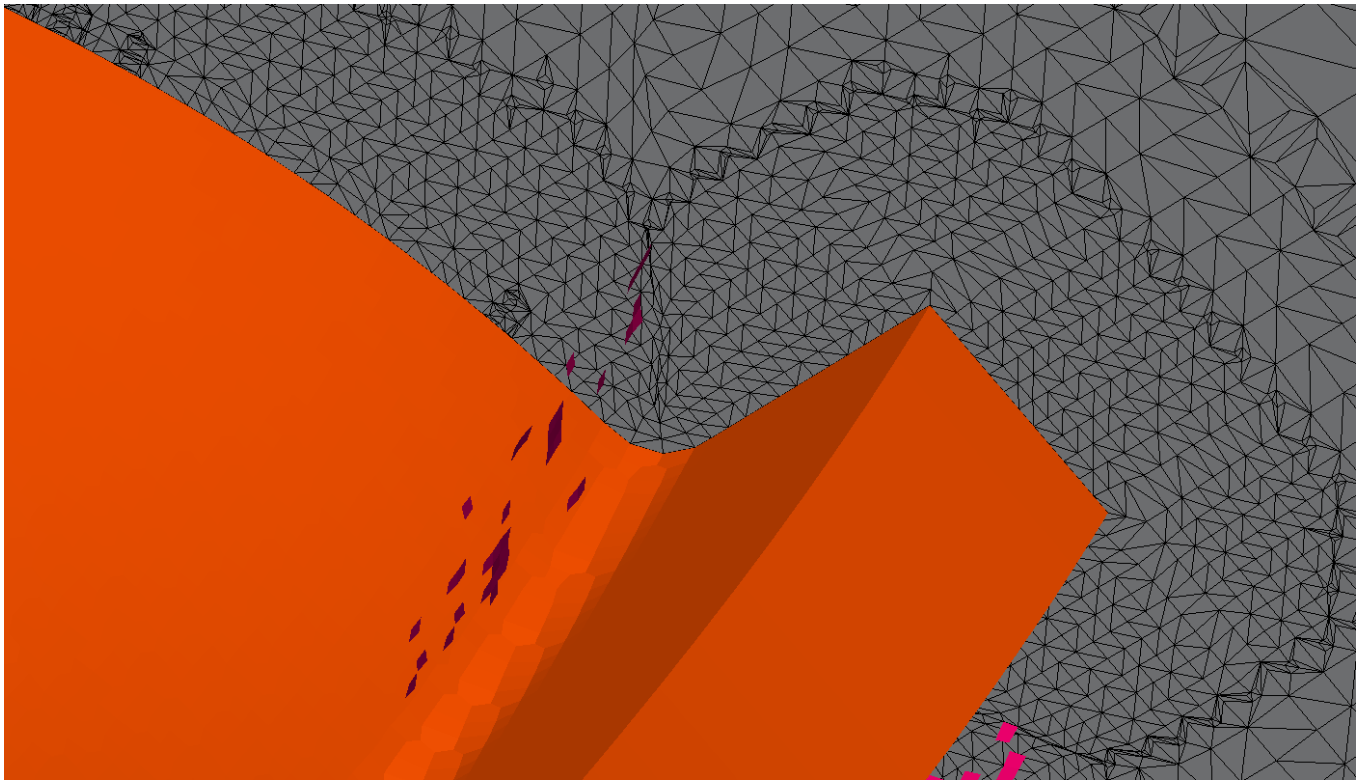
- Check the mesh quality
- If problems occur new sets of cells/faces will be created

Use

- `foamToVTK -constant -faceSet nonOrthoFaces`
- `mv ./VTK ./VTK_sets`
- `foamToVTK -constant`
- Load the new VTK of the face set and the entire case
- Assess if the problem is serious

Next steps

- Check the mesh quality
- If problems occur new sets of cells/faces will be created



Next steps

- Check the mesh quality
- If problems occur new sets of cells/faces will be created
- Any suggestions on where the problem is coming from and what can be done about it?

Mesh size & quality

Knowing the grid size - blockMesh

- Define non-dimensional spacings based on the number of cells, N, and expansion ratio, e
- Use them to compute the actual cell point locations along a parametric curve defining the edge of a mesh block (straight line, spline, etc.)
- Geometric expansion coefficient:

$$G = e^{1/(N-1)}$$

- Non-dimensional ordinate for the i^{th} mesh point, where $(i \in < 0, N >)$:

$$s_i = (1 - G^i)/(1 - G^N)$$

Knowing the grid size - blockMesh

- Depending on how the mesh is defined, it is either the spacing between 0^{th} and 1^{st} or N^{th} and $(N-1)^{th}$ points that corresponds to the first cell height away from the wall
- As the positions of mesh block vertices are known, say h_0 and h_1, the cell heigh may be obtained as:

$$y_i = s_i(h_1 - h_0)$$

Knowing the grid size - snappyHexMesh

- The grid generation process is far less analytical - need to watch out for outlying cells
- The principle parameters are the unrefined cell size, L, and the refinement level, r
- For every refinement level the cell size in one dimension becomes halved:

$$L_r = L/2^r$$

- Now, for the added layers, described by the relative size s, expansion ratio e and the number of layers N we may write:

$$y_1 = L_r s/e^{N-1}$$

Choosing the wall spacing

- Depending on the wall treatment (resolved boundary layer or wall functions, more on this later), we want non-dimensional cell height for the first cell away from the wall, y^+, to be < 1 or roughly between 30 and 150, respectively - varies for different turbulence models!
- The first cell height defined as:

$$y^+ = y_1 U_\tau / \nu$$

- Where

$$U_\tau = \sqrt{\tau_{wall}/\rho} = \sqrt{\frac{1}{2} Cf U^2}$$

- **Crucial point:** we need to know Cf in order to run the simulation properly!

Choosing the wall spacing

- Need some Cf estimates, such as:

$$\text{ITTC: } Cf = 0.075/(log10(Re) - 2)^2$$

$$\text{F. M. White: } Cf = 0.026/Re^{1/7}$$

- These are not very accurate in terms of local skin friction for complex geometries and hence may need to gradually change the mesh as more simulations are run and flow becomes better understood
- For LES we also need to mind $x^+ \in (100 - 600)$ and $z^+ \in (100 - 300)$ - read the literature to confirm these for you particular problem and turbulence model!
- The above formulae may easily be incorporated into a spreadsheet/program to make choosing the appropriate values easier

Verifying the wall spacing - NACA0009

- May use the `yPlusRAS` or `yPlusLES` utilities to create a y^+ field
- **Note:** These in fact yield y^* and not y^+ according to the overally accepted definitions (slightly different scaling); an unoficial utility giving the correct values may be found online: `http://www.cfd-online.com/Forums/openfoam-post-processing/80175-problems-yplusras-wallshearstress.html`
- Make sure you follow the changes to the code to keep track of whether this gets changed or not!

Verifying the wall spacing - NACA0009

Mesh quality

- Construction of finite volume grids is a science on its own
- No general answer as to "what grid type is the best"
- OpenFOAM has a built in tool for quality check - checkMesh (see snappyHexMesh practical session for an example of usage)
- A good way of examining the mesh is by eye - make a lot of cut planes!

Some things to look out for

- Want to align the mesh with the flow as much as possible
- Avoid skewed or warped cells
- Think carefully about how to orient your mesh blocks to avoid topological problems
- For LES we typically want hexahedral grids, avoid using tetrahedrals if possible
- Keep the expansion ratios in regions of interest close to 1.0 to preserve the turbulent structures

Grid convergence

- Because we use discretised equations an error will be introduced to the solution
- As grid becomes finer, this contribution of the error should decrease
- Always need to have an idea about the uncertainty in this respect
- Mesh convergence study - keep refining the baseline mesh until the significant flow parameters do not change (look at more than just forces: pressure coefficient, wake, boundary layer profiles, etc.)

Summary

Further reading recommendations

- Slides by Andrew Jackson from 7th OpenFOAM workshop
  http://openfoamwiki.net/images/f/f0/Final-AndrewJacksonSlidesOFW7.pdf
- Official OpenFOAM documentation on mesh generation
  http://www.openfoam.org/docs/user/mesh.php
- Pointwise & OpenFOAM tutorial by K. Martin and J. M. Cimbala
  http://www.mne.psu.edu/cimbala/Learning/CFD/Tutorial_Straight_Pipe_Pointwise_OpenFOAM.pdf
- enGrid, Blender & OpenFOAM tutorial by P. Bomke and A. Baars
  https://github.com/enGits/engrid/wiki/Unstructured-Grids-for-OpenFOAM-With-Blender-and-enGrid-1.2

The End