Introduction to OpenFOAM meshing

Timofey Mukha

KTH Engineering Mechanics

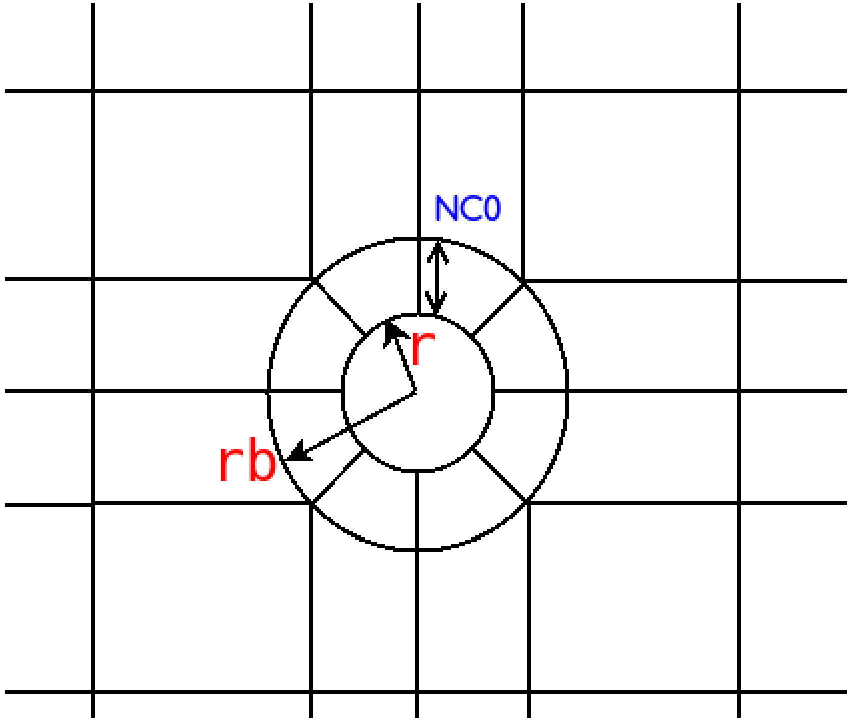
Available meshers

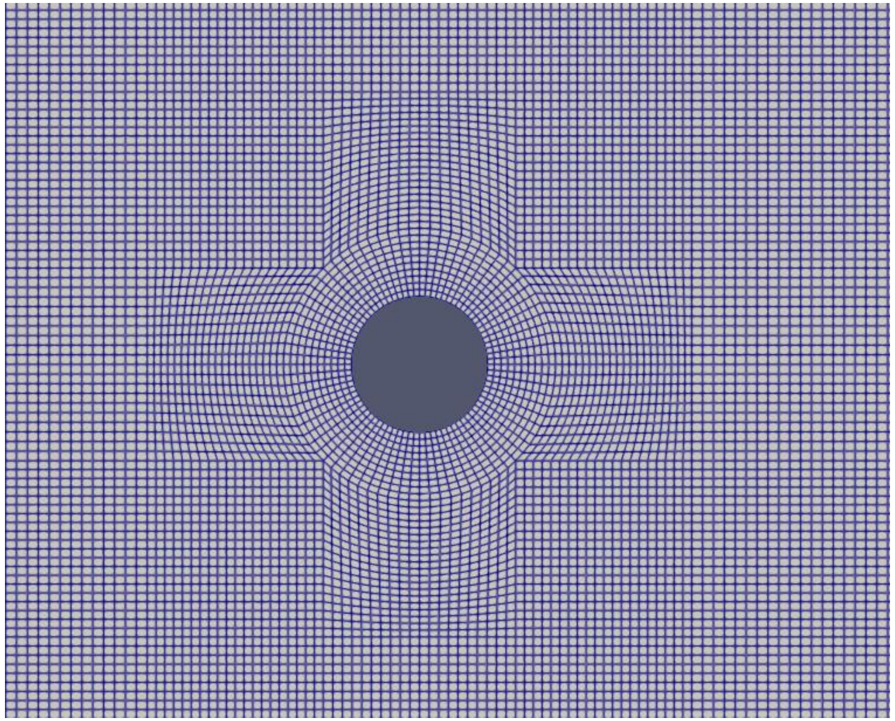
- `blockMesh` – Block-structured hexahedral mesher.
- `snappyHexMesh` – Unstructured hexa-dominated mesher.
- `cfMesh` – Unstructured mesher with different available meshing strategies.

We will briefly look at the first two.

NB: you can also use your favourite commercial mesher!

blockMesh





By S. Rezaeiravesh

blockMesh

- Mesh fully defined in one dictionary: `blockMeshDict`. Lives in `system`.
- Manually define everything: vertices, blocks, curved edges, boundaries.
- Official guide: `https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.3-mesh-generation-with-the-blockmesh-utility`

blockMeshDict

Global scaling of all

Vertex locations

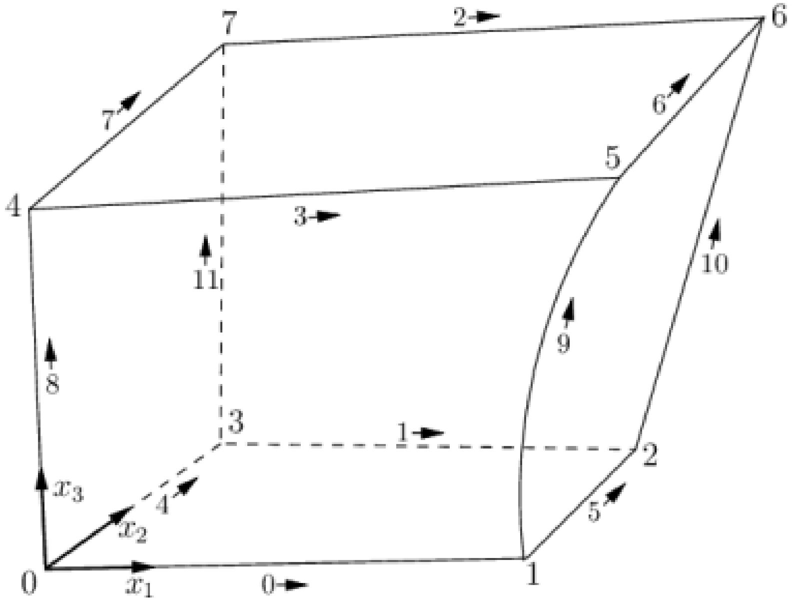
Vertices that will be use to define blocks.

Numbering from 0.

```
scale   0.1;

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)
);

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

edges
(
);
```

blockMeshDict

Vertex and edge order



From openfoam.com

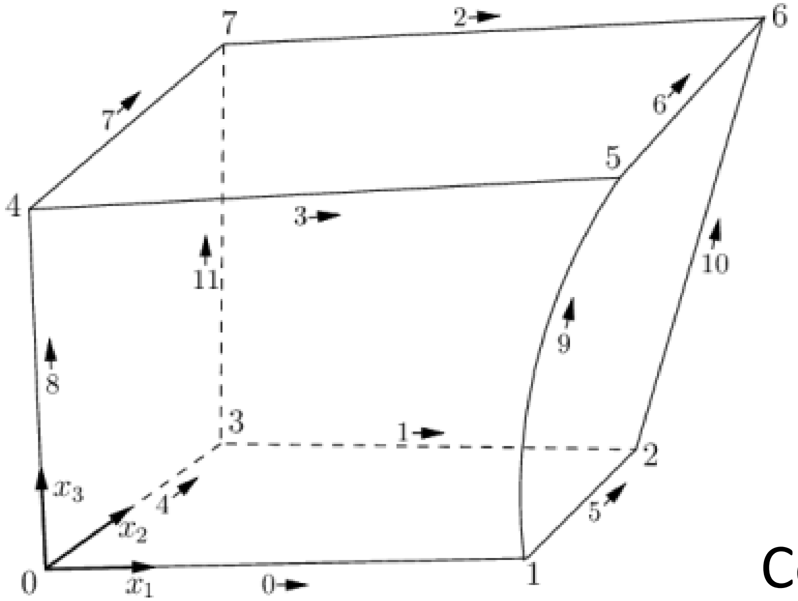
List of blocks

Vertex indices defining the block

```
scale   0.1;

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)
);

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

edges
(
);
```

blockMeshDict

Vertex and edge order



From openfoam.com

Number of cells

Along x_1, x_2, x_3

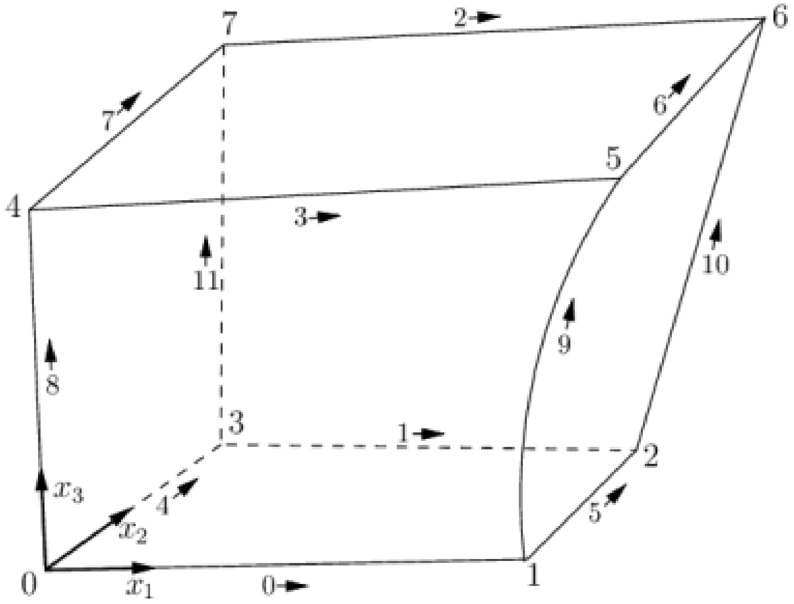
Cell size expansion

ratio along x_1, x_2, x_3

```
scale   0.1;

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)
);

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

edges
(
);
```

blockMeshDict

Vertex and edge order



From openfoam.com

Non-linear edges

```
scale   0.1;

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)
);

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

edges
(
);
```

blockMeshDict

The physical boundaries of the case

Boundary type

List of faces, defined by vertex ids, forming the boundary

```
boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

Get written to
`constant/polyMesh/boundary`

Boundary types

- “patch” – generic type used for most boundary boundaries.
- “wall” – for walls.
- “empty” – for a 2D case, defining the side boundaries parallel to the 2D plane in which the solution is obtained.
- “cyclic”, “cyclicAMI” – for periodic boundary conditions. Come in pairs.
- “symmetry” – symmetry boundary.

Boundary types vs boundary conditions

- Boundary conditions for fields are set in the 0 folder!
- But some boundary types essentially define the condition, e.g. cyclic.
- In this case, the conditions in the 0 folder must match the boundary type. E.g. cyclic boundary condition for the cyclic boundary type. Same with “empty”

Boundary types vs boundary conditions

0/U

```
boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            noSlip;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

blockMeshDict

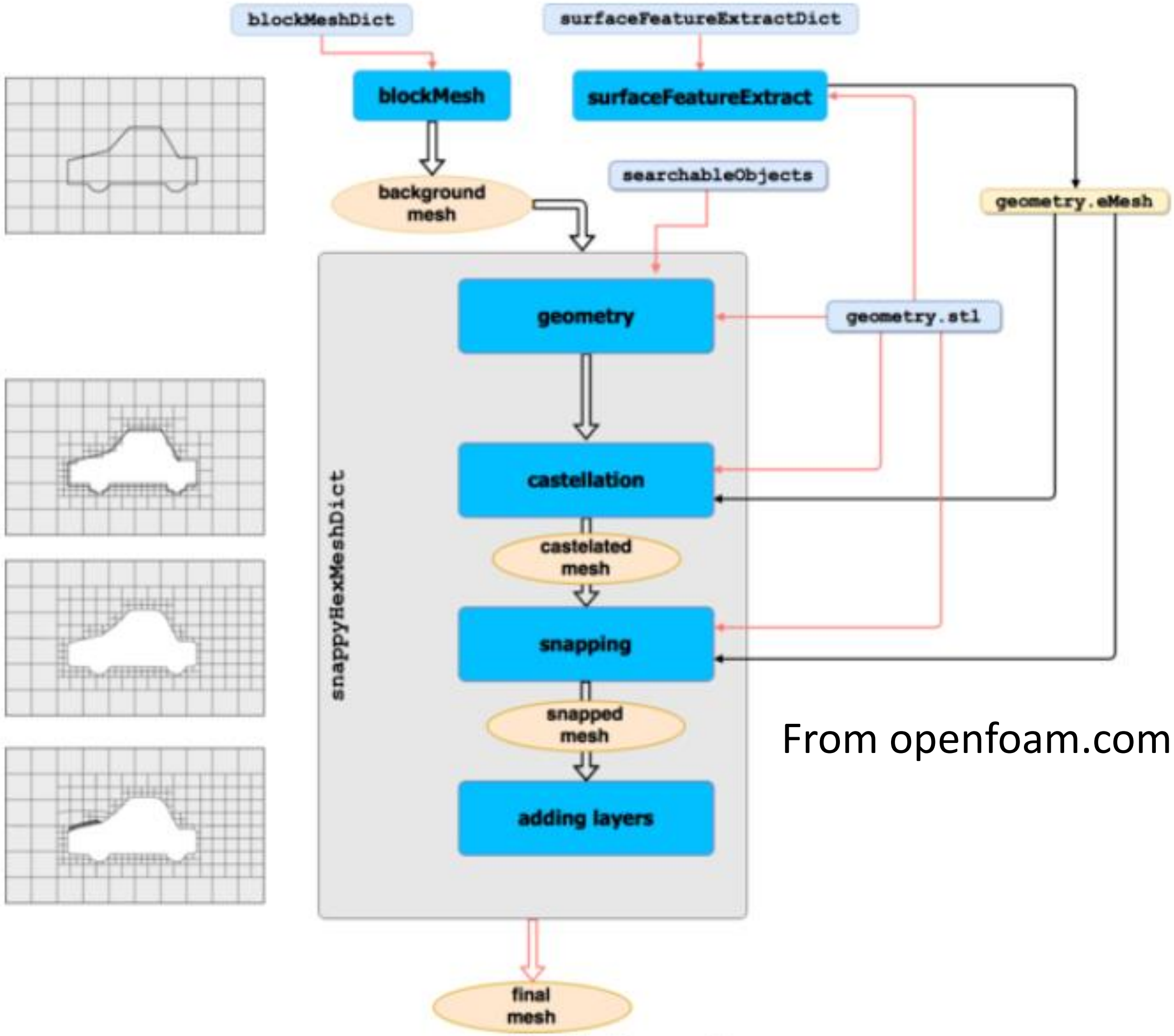
```
boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

Some stuff we skipped

- Various curved edges.
- More complicated mesh grading.
- `mergePatchPairs` – block-unstructured hex meshes.
- Possibility to define variables and then use then in geometry definitions.

snappyHexMesh

- For snappy we will only be able to give a rough overview.
- Creating the mesh is a 3-step process.
    1. Castellation
    2. Snapping
    3. Adding layers



snappyHexMesh overview

From openfoam.com

snappyHexMesh, preparation

- Put the stl of the geometry to `constant/triSruface`
- Create a blockMeshDict with one block of cubic cells. This will define the largest cell size.
- Create a surfaceFeatureExtractDict in system
- Run `surfaceFeaterExtract`
- Run `blockMesh`

```
flange.stl
{
    // How to obtain raw features (extractFromFile || extractFromSurface)
    extractionMethod    extractFromSurface;

    // Mark edges whose adjacent surface normals are at an angle less
    // than includedAngle as features
    // - 0  : selects no edges
    // - 180: selects all edges
    includedAngle       150;

    // Write options

    // Write features to obj format for postprocessing
    writeObj            yes;
}
```

snappyHexMeshDict

- Lives in `system`
- Which of the 3 steps to run
- Define the geometry. Can add geometricl primitives for e.g. defining refinement regions.

```
// Which of the steps to run
castellatedMesh true;
snap            true;
addLayers       false;
```

```
geometry
{
    flange.stl
    {
        type triSurfaceMesh;
        name flange;
    }

    //- Refine a bit extra around the small centre hole
    refineHole
    {
        type    sphere;
        origin  (0 0 -0.012);
        radius  0.003;
    }
}
```

snappyHexMeshDict

- A dictionary with MANY options for each step.
- We will now go through one example dictionary and highlight a few important ones.

```
castellatedMeshControls
{

    // Refinement parameters
    // ~~~~~~~~~~~~~~~~~~~~~~
```

```
// Settings for the snapping.
snapControls
{
    //- Number of patch smoothing iterations before finding correspondence
    //  to surface
    nSmoothPatch 3;
```

```
// Settings for the layer addition.
addLayersControls
{
    // Are the thickness parameters below relative to the undistorted
    // size of the refined cell outside layer (true) or absolute sizes (false).
    relativeSizes true;

    // Per final patch (so not geometry!) the layer information
    layers
    {
        "flange_.*"
        {
            nSurfaceLayers 3;
```

snappyHexMeshDict

- Running `snappyHexMesh` will produce a separate directory for each step of the meshing process. The mesh in constant will be intact.
- Run `snappyHexMesh -overwrite` to write only the final mesh directly to constant.

Conclusions

- OpenFOAM has several meshing tools, suitable for both simple and complex geometries.
- It’s possible to do a lot with snappy, including industrial flows.
- That being said, it seems to take a lot of parameter tweeking and one has to know the tool well.
- I have heard from many that cfMesh is less painful to work with. Try that as well.
- Generally, speciallized commercial meshers are still quite a bit better in my opinion.