IAEA
International Atomic Energy Agency

Multi-physics modeling and simulation of nuclear reactors using OpenFOAM

30 Aug 2022 – 6 October 2022 (every Tuesday & Thursday)

Contact: ONCORE@iaea.org

Lecture 2, part B:
A practical introduction to OpenFOAM – Geometry preparation and meshing

6 September 2022

Ezequiel Fogliatto

Course Enrollment : Multi-physics modelling and simulation of nuclear reactors using OpenFOAM

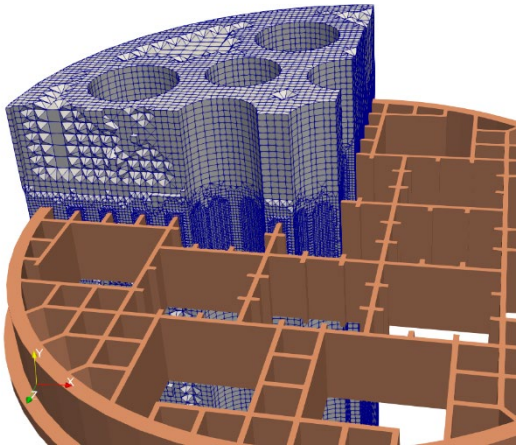
ONCORE: Open-source Nuclear Codes for Reactor Analysis

Before we start

- Recommended learning
  - J. Guerero: Introduction to OpenFOAM, training materials, http://www.wolfdynamics.com/tutorials.html
  - F. Moukalled, L. Mangani, M. Darwish: The Finite Volume Method in Computational Fluid Dynamics, Springer, 2016 https://link.springer.com/book/10.1007/978-3-319-16874-6
- Q&A
  - Brief intermediate break
  - At the end of this lecture
  - Forum: https://foam-for-nuclear.org/phpBB/

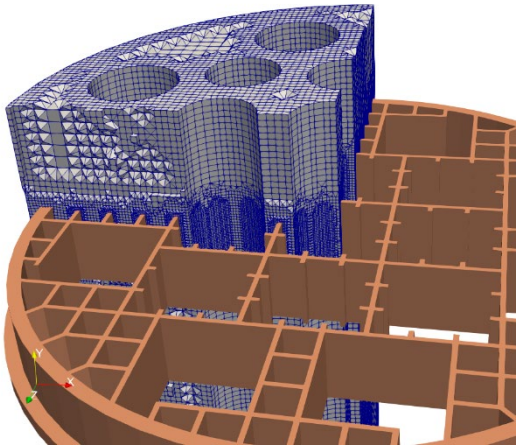
What will you learn?

- Main workflow before performing any simulation (pre-processsing)
- Identification of the critical steps and most important concepts
- Importance of the pre-processing stage
- Overview of some open-source tools
  - Geometry preparation
  - Meshing



What won't you learn?

- Geometry preparation and meshing are extensive and important parts of any simulation but, unfortunately, the time for this lecture is limited
- We will only have a general overview of the tools
- Mastering geometry preparation and meshing is not only about the tools: you need time, practice and expertise



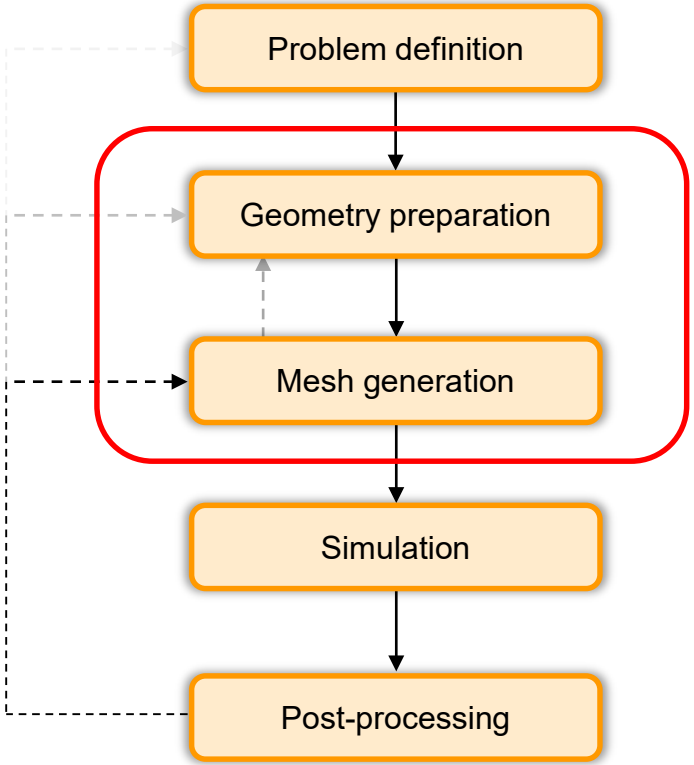
Lecture overview

- Geometry preparation
- Finite volume meshing:
  - What is a mesh?
  - Mesh generation
  - Aspects of mesh resolution and quality
- Mesh manipulation in OpenFOAM
  - Grouping regions into zones
  - Dealing with multiple meshes
- Open-source tools for geometry preparation and meshing:
  - SALOME
  - blockMesh
  - snappyHexMesh

Lecture overview

- Geometry preparation
- Finite volume meshing:
  - What is a mesh?
  - Mesh generation
  - Aspects of mesh resolution and quality
- Mesh manipulation in OpenFOAM
  - Grouping regions into zones
  - Dealing with multiple meshes
- Open-source tools for geometry preparation and meshing:
  - SALOME
  - blockMesh
  - snappyHexMesh

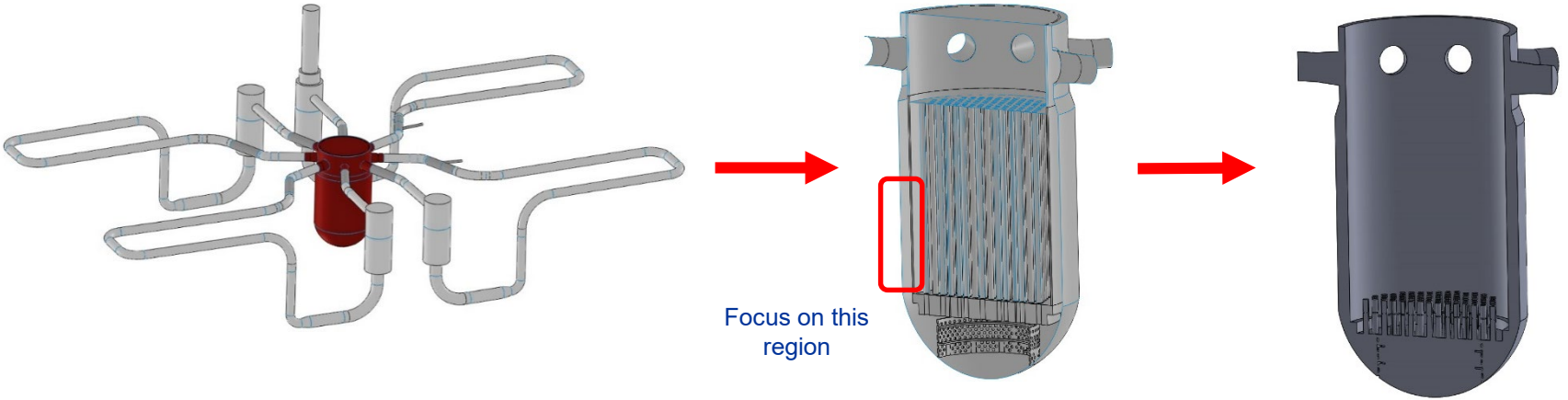
Basics



- The starting point of the workflow is the definition on the problem: relevant physics, geometrical domain, steady-state or transient, etc.
- When the geometry is ready, proceed with the mesh generation and the assignment of boundary conditions.
- The quality and convergence rate of the solution highly depend on the mesh: **do your best!**
- Geometry generation and meshing requires a significant amount of user's time.

Geometry definition

- Geometry selection depends on the problem and the goal.
- Reduce the complexity of the problem with appropriate boundary conditions
  - Warning: boundary effects

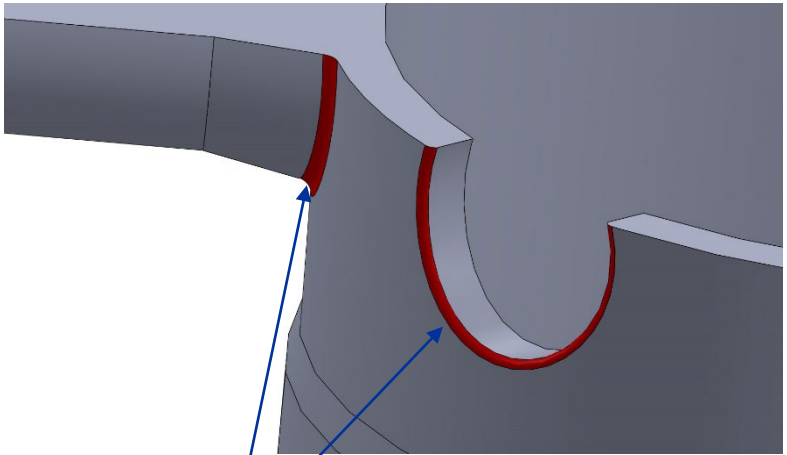


Geometry preparation

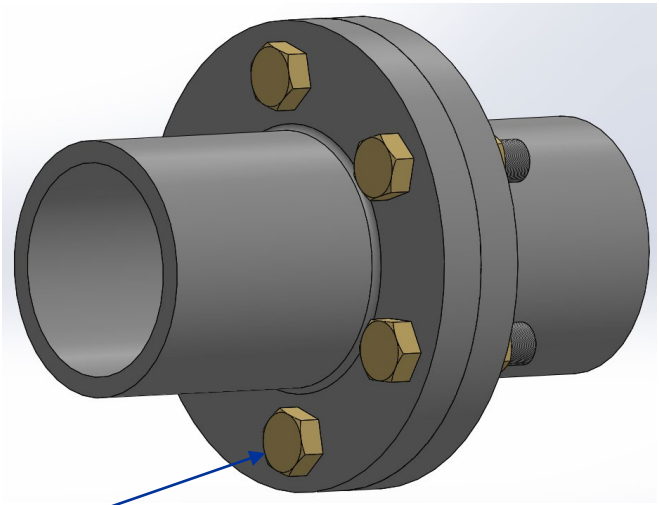
- Choose a proper CAD (Computer Aided Design) tool to create the geometry.
- A few open-source applications:
  - Salome
  - Free-CAD
  - Openscad
- The process for creating the geometry is not unique, but you should always get a complete, sufficiently clean and watertight geometry.
- The quality of the mesh, and hence the solution, greatly depends on the geometry. Always do your best when creating the geometry.
- Cleaning a CAD geometry before meshing is critical. Remove unnecessary and problematic details from the geometry.

Defeaturing

- Many times, it is not necessary to model all the details of the geometry. In these cases, you should consider simplifying the geometry: **geometry defeaturing.**



Do you really need those fillets in detail?



The bolts are not needed to model internal flow

CAD formats

After the solid modeling stage, you will need to export your design to the meshing tool (or other design tool)

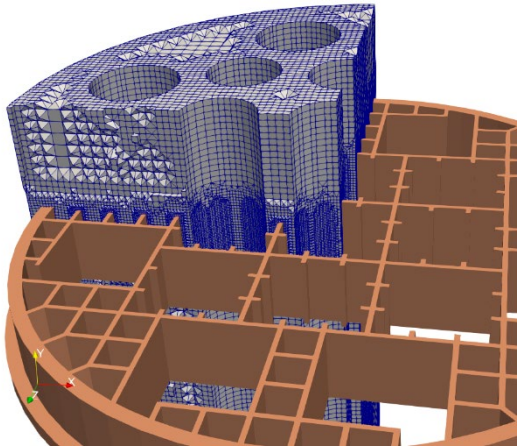
- STEP is one of the most reliable formats
- SAT is a good alternative but is less commonly supported
- IGES can be problematic (due to rounding errors)
- Avoid using STL format
  - Rapid prototyping industry's standard
  - Supported by most meshing tools
  - Doesn't preserve original geometry
  - No guarantee of watertight closed body
  - Resolution often not customizable
  - Only ever use this format in the final step of the geometry preparation process

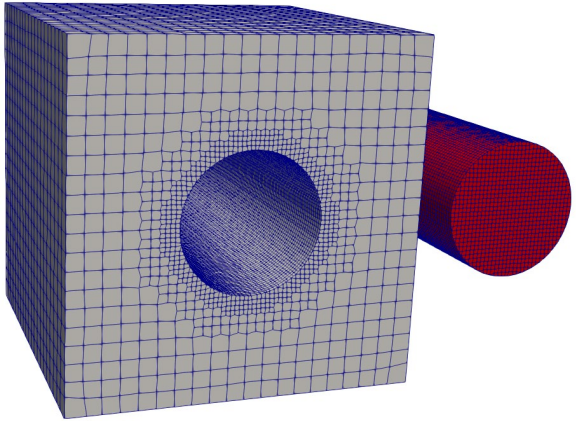
Lecture overview

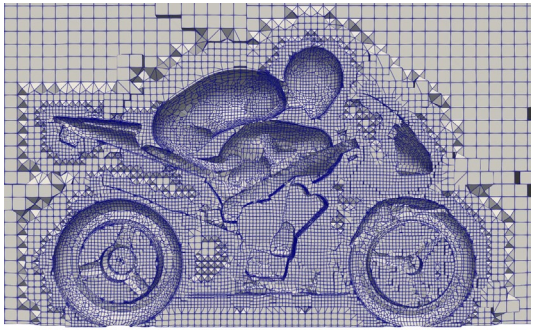
- Geometry preparation
- Finite volume meshing:
  - What is a mesh?
  - Mesh generation
  - Aspects of mesh resolution and quality
- Mesh manipulation in OpenFOAM
  - Grouping regions into zones
  - Dealing with multiple meshes
- Open-source tools for geometry preparation and meshing:
  - SALOME
  - blockMesh
  - snappyHexMesh

What is a mesh?

- Mesh generation or domain discretization consists in dividing the physical domain into a finite number of discrete regions, called control volumes or cells







Mesh generation

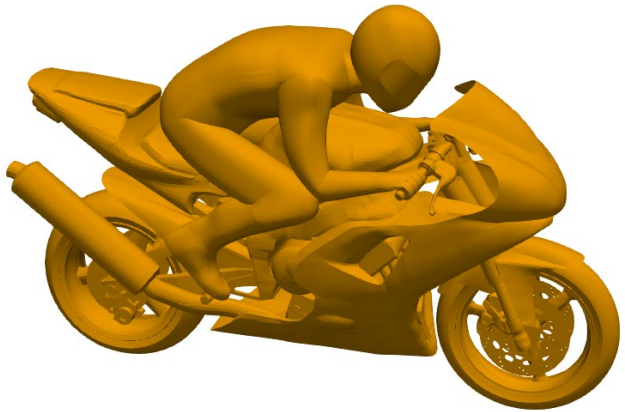
Three common steps

1. Geometry preparation: define and prepare the geometry that is going to be used by the meshing tool
2. Mesh generation: splitting of the domain in cells. Internal or external. Boundary layers
3. Definition of boundary surfaces: definition of physical surfaces where the boundary conditions are going to be applied

Mesh generation

Three common steps

1. Geometry preparation: define and prepare the geometry that is going to be used by the meshing tool
2. Mesh generation: splitting of the domain in cells. Internal or external. Boundary layers
3. Definition of boundary surfaces: definition of physical surfaces where the boundary conditions are going to be applied

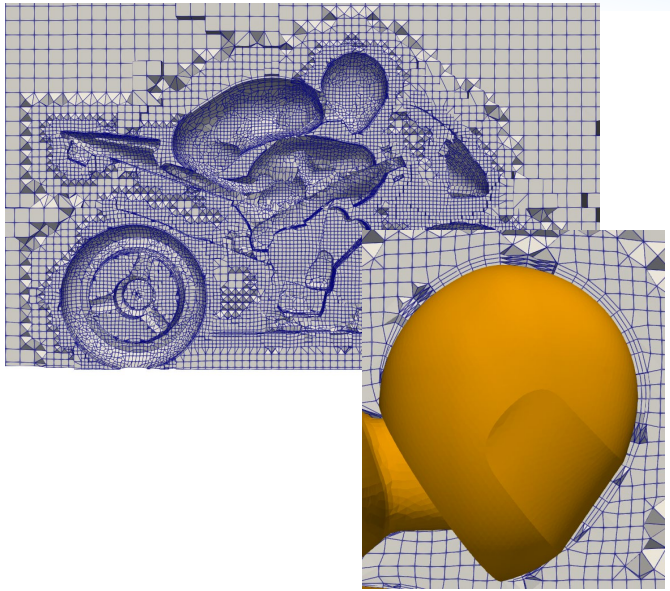


- Watertight
- The mesh quality depends on the geometry quality

Mesh generation

Three common steps

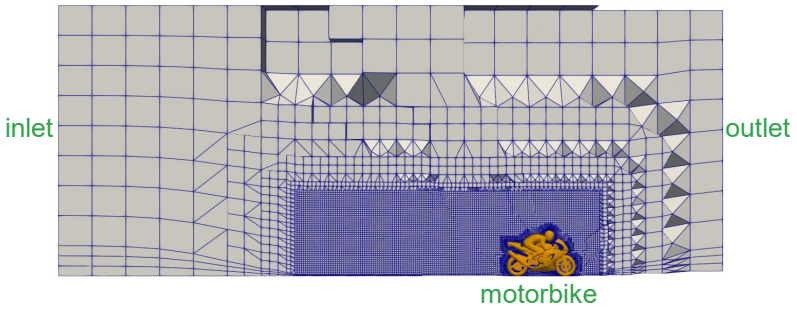
1. Geometry preparation: define and prepare the geometry that is going to be used by the meshing tool
2. Mesh generation: splitting of the domain in cells. Internal or external. Boundary layers
3. Definition of boundary surfaces: definition of physical surfaces where the boundary conditions are going to be applied



Mesh generation

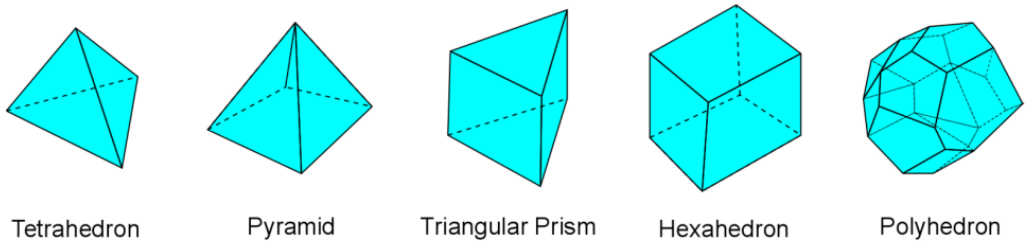
Three common steps

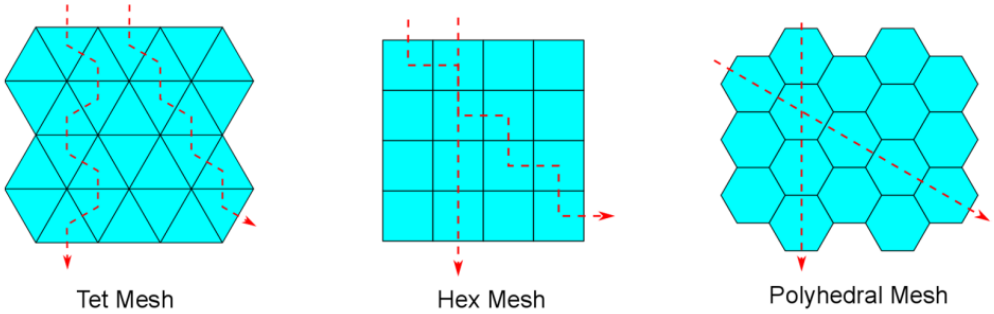
1. Geometry preparation: define and prepare the geometry that is going to be used by the meshing tool
2. Mesh generation: splitting of the domain in cells. Internal or external. Boundary layers
3. Definition of boundary surfaces: definition of physical surfaces where the boundary conditions are going to be applied



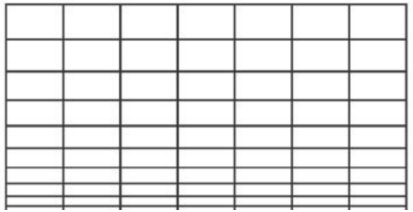
What cell type should I use?

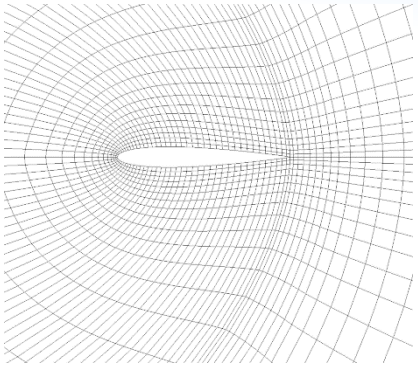
- OpenFOAM meshes are composed of polyhedral cells

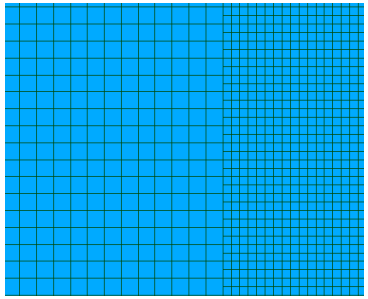


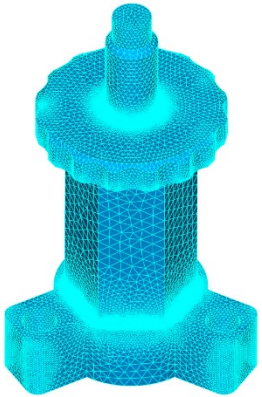


What cell type should I use?









- It depends on the geometry that you are trying to discretize
- A valid mesh for OpenFOAM can contain different types of cells (hybrid)
- Be careful!: Each cell type has its very own properties when it comes to approximating the gradients and fluxes
- In general, hexahedral meshes improve better: accuracy and stability.
- It is up to you. The overall quality of the final mesh should be acceptable, and your mesh should resolve the physics

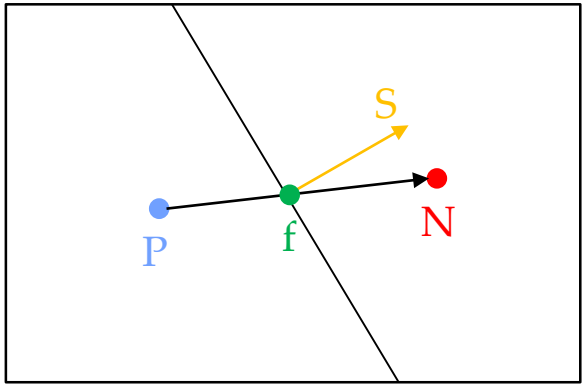
What is a “good” mesh?

- The meshing process heavily depends on the user’s experience and trial-and-error
- The user can rely on mesh quality metrics and suggested best practices
- Some of the most common metrics:
  - Orthogonality
  - Skewness
  - Aspect ratio
  - Smoothness
- After the mesh is generated, the quality metrics are measured and used to assess the goodness of the mesh.
- Always check your mesh using the checkMesh utility. If it reports errors, you should fix them.
- It is not only about a good looking mesh. Always do convergence checks!

Mesh quality metrics

Orthogonality

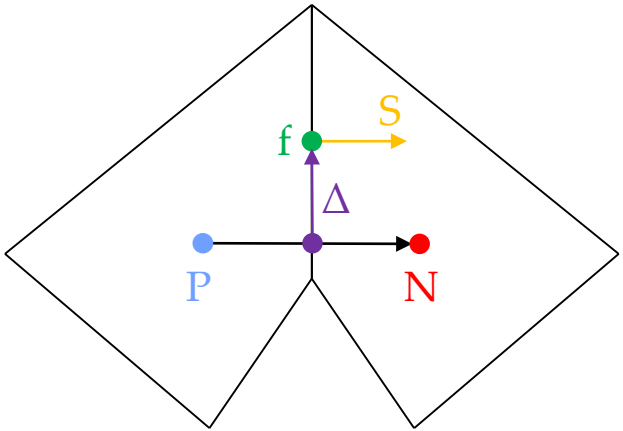
- Mesh orthogonality is the angular deviation of the vector S (located at the face center f) from the vector d connecting the two cell centers P and N
- It mainly affects the Laplacian (diffusive) terms and gradient terms at the face center f
- It adds numerical diffusion to the solution
- OpenFOAM applies explicit corrections



Mesh quality metrics

Skewness

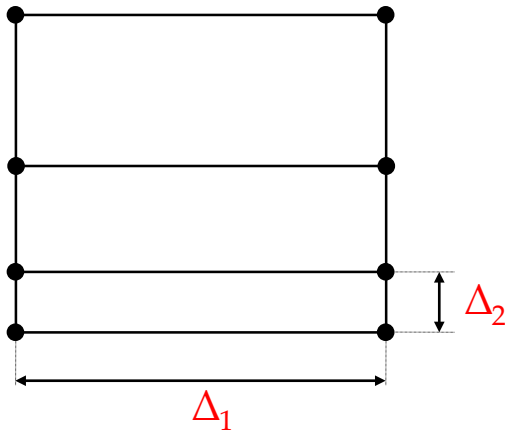
- Is the relative distance between the face center f, and the point where the line connecting the two cell centres P and N intersects the face
- It affects the interpolation of the cell centered quantities at the face center f
- It affects the convective and diffusive terms It adds numerical diffusion and oscillations to the solution



Mesh quality metrics

Aspect ratio

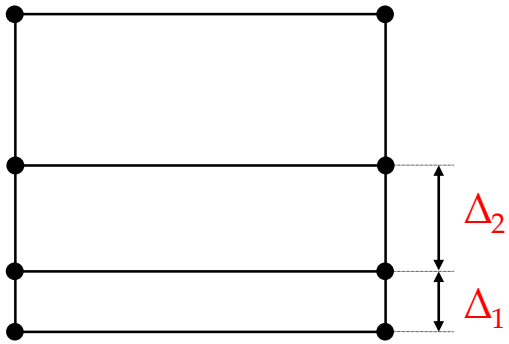
- Mesh aspect ratio AR is the ratio between the longest side and the shortest side
- Large AR are ok if gradients in the largest direction are small
- Large AR add numerical diffusion to the solution. Smear gradients.
- In RANS/URANS simulation large AR are acceptable



Mesh quality metrics

Smoothness

- Smoothness, also known as expansion rate, growth factor or uniformity, defines the transition in size between contiguous cells.
- Large transition ratios between cells add diffusion to the solution.



Mesh quality metrics

- Always check your mesh with the checkMesh utility
- If it reports errors, you should fix them

```
Mesh stats
    points:           1648202
    faces:            4717378
    internal faces:   4492640
    cells:            1535003
    faces per cell:   6
    boundary patches: 4
    point zones:      0
    face zones:       1
    cell zones:       1

Overall number of cells of each type:
    hexahedra:     1535003
    prisms:        0
    wedges:        0
    pyramids:      0
    tet wedges:    0
    tetrahedra:    0
    polyhedra:     0
```

```
Checking topology...
    Boundary definition OK.
    Cell to face addressing OK.
    Point usage OK.
    Upper triangular ordering OK.
    Face vertices OK.
    Number of regions: 1 (OK).

Checking patch topology for multiply connected surfaces...
    Patch          Faces    Points   Surface topology
    Inlet          724      775      ok (non-closed singly connected)
    Outlet         8700     9280     ok (non-closed singly connected)
    Top            11734    12374    ok (non-closed singly connected)
    Walls          203580   204810   ok (non-closed singly connected)
```

Mesh quality metrics

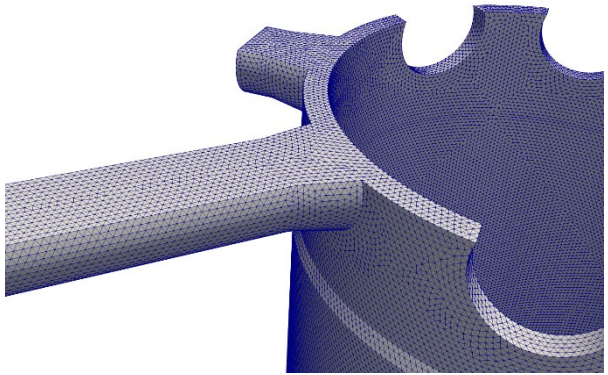
- Always check your mesh with the checkMesh utility
- If it reports errors, you should fix them

```
Checking geometry...
    Overall domain bounding box (-6.153343 -7.11 -2.43506) (3.012152 0.169 2.835811)
    Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
    Mesh has 3 solution (non-empty) directions (1 1 1)
    Boundary openness (-3.523128e-15 5.811175e-17 -1.12787e-17) OK.
    Max cell openness = 3.078141e-16 OK.
    Max aspect ratio = 3.659591 OK.
    Minimum face area = 0.0001747741. Maximum face area = 0.002554625.  Face area magnitudes OK.
    Min volume = 2.944914e-06. Max volume = 3.814499e-05.  Total volume = 27.39824.  Cell volumes OK.
    Mesh non-orthogonality Max: 34.31389 average: 4.070957
    Non-orthogonality check OK.
    Face pyramids OK.
    Max skewness = 0.5269406 OK.
    Coupled point location match (average 0) OK.

Mesh OK.
```

What cell type should I use? (rev.)

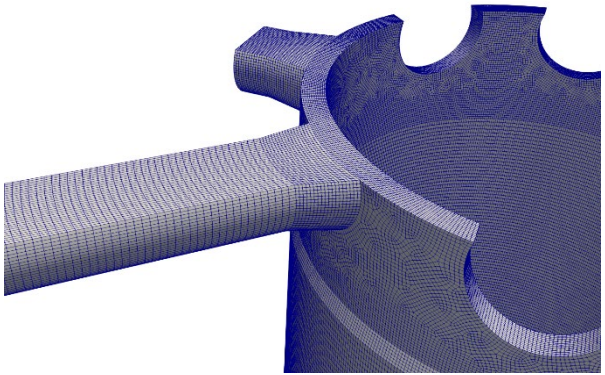
Tri/tet

- Can easily be adapted to any kind of geometry
- The mesh generation process is almost automatic
- Normally need more computing resources during the solution stage

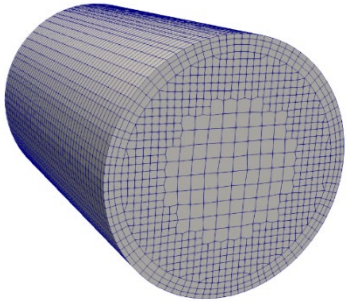


Poly (not tri/tet)

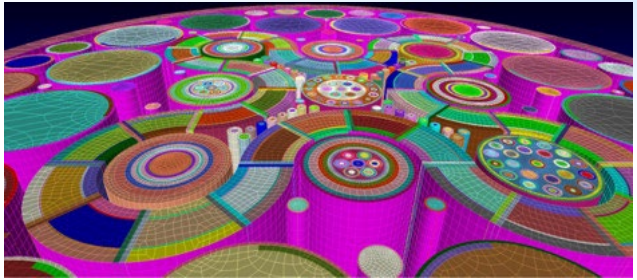
- Can be easily aligned with the flow
- Less numerical diffusion
- Can be stretched to resolve boundary layers without losing much quality
- Difficult to build in general geometries



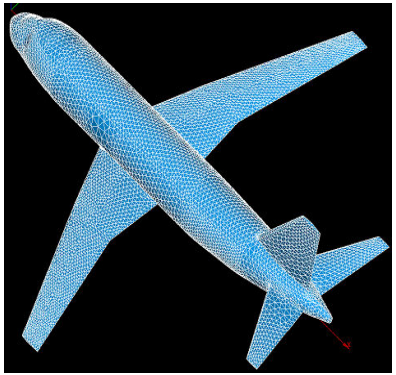
A few examples



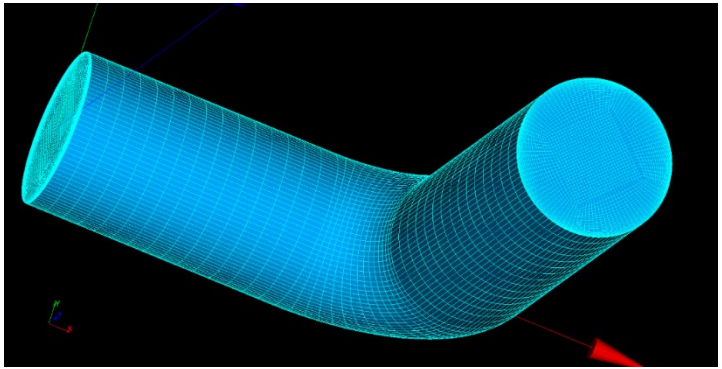
Cylinder meshed with snappyHexMesh. Hexa-dominant



INL's Advanced Test Reactor meshed using CUBIT (not open-source)



SALOME Tet-mesh



SALOME mesh with quadrangle mapping and extrusion

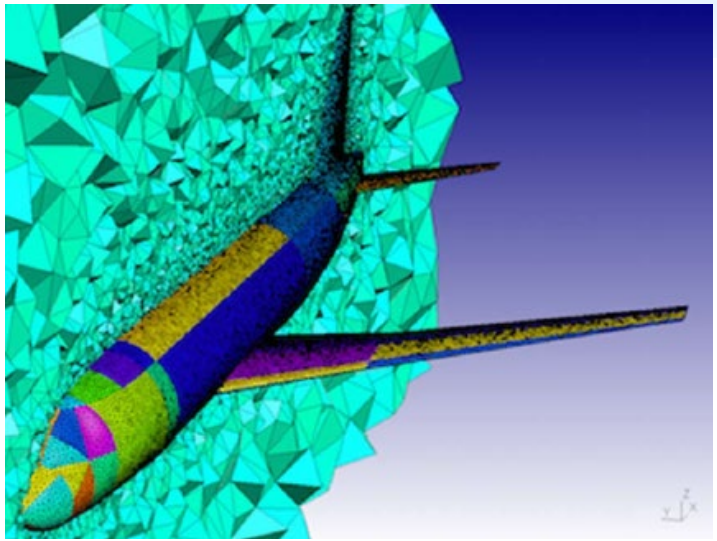
Looking for quality

- Avoid the GIGO syndrome (garbage in, garbage out)

- If the number of cells is increased, the solution accuracy will probably increase. But also the computational time. Order N^2, Nlog(N)

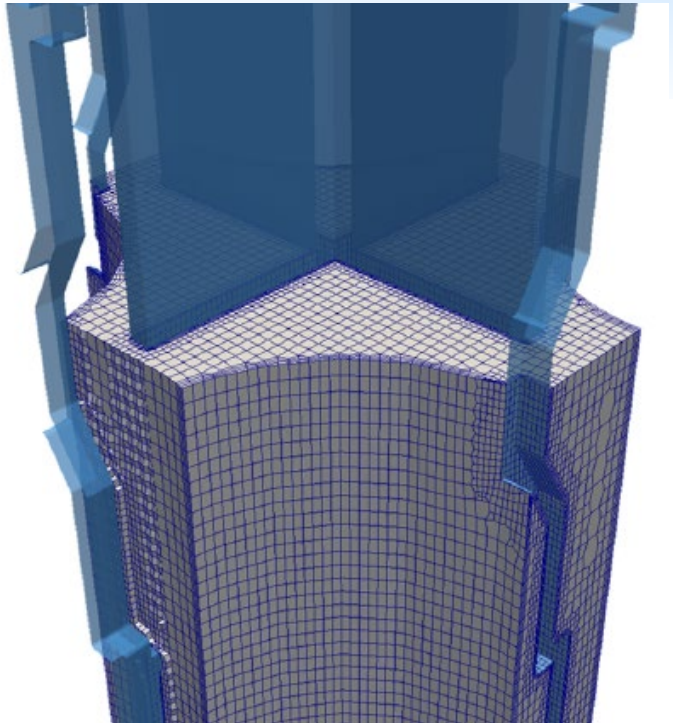
- Use non-uniform meshes and local refinement to cluster cells only where they are needed. Refine only on selected areas

- In turbulence modeling, the mesh next to the walls should be fine enough. This will increase the cell count and the computing time



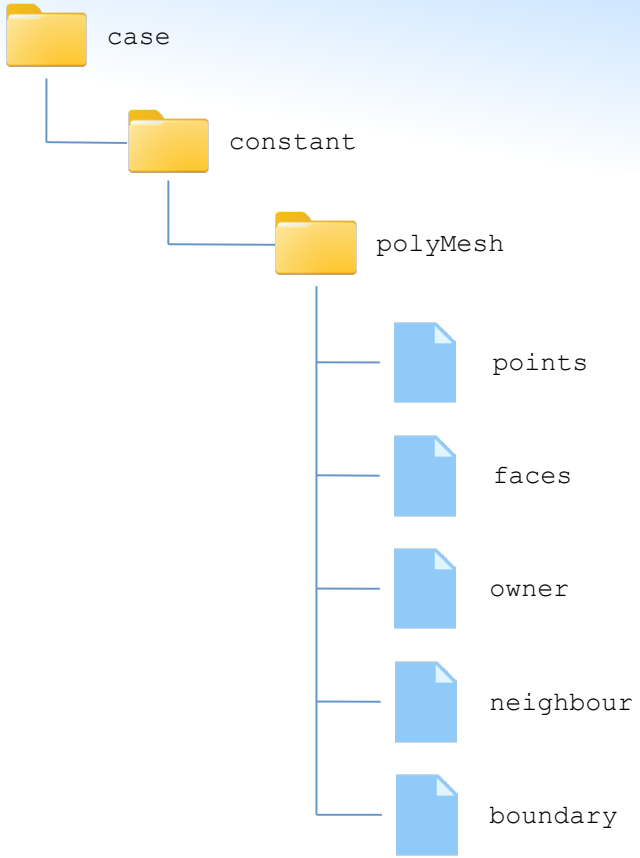
Looking for quality

- Use hexahedral meshes whenever is possible
- Keep orthogonality, skewness, and aspect ratio to a minimum
- Change in cell size should be smooth
- Always check the mesh quality. One single cell can cause divergence or give you inaccurate results
- A good mesh is a mesh that serves your project objectives
- Know your physics and generate a mesh able to resolve the physics involved
- Don’t forget: always perform a convergency test



OpenFOAM mesh format

- In each case, the mesh is stored in `constant/polyMesh`
- `constant/<region>/polyMesh` is used when dealing with multiple regions/meshes
- Several utilities to convert an external mesh format
  - `fluentMeshToFoam`
  - `starToFoam`
  - `gambitToFoam`
  - `ideasToFoam`
  - `ideasUNVToFoam`
  - `cfx4ToFoam`



Some useful tips

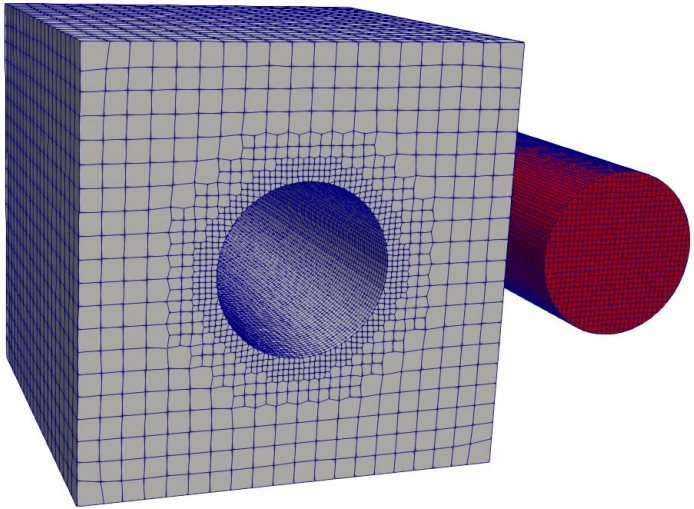
- Always preview your mesh using `ParaView/paraFoam` to ensure everything looks good (boundaries correctly positioned, etc.)
  - If visualisation is slow (large mesh) then set up a meshing workflow using a coarser mesh, check this in `ParaView`, and then increase the refinement
  - An alternative approach is to mesh larger models in separate regions and then join the regions using non-conformal interfaces (AMI). This workflow is good for mesh convergence studies.
  - If your mesh contains polyhedral elements (`snappyHexMesh`, `cfMesh`, `polyDualMesh`, `foamyHexMesh`), then make sure you disable "Decompose Polyhedra" in `ParaView` or enable "Use vtkPolyhedron" in `paraFoam` to properly view your mesh. This option might be missing in newer versions of `ParaView`.
- Optional: run `potentialFoam` before starting your simulation and carefully look at the resulting velocity field. If the solution is unbounded (extreme values in single cells) then throw away the mesh and try again.

Lecture overview

- Geometry preparation
- Finite volume meshing:
  - What is a mesh?
  - Mesh generation
  - Aspects of mesh resolution and quality
- Mesh manipulation in OpenFOAM
  - Grouping regions into zones
  - Dealing with multiple meshes
- Open-source tools for geometry preparation and meshing:
  - SALOME
  - blockMesh
  - snappyHexMesh

Zones in OpenFOAM

- OpenFOAM can group mesh entities under sets and zones
  - sets: simple group of entities, e.g. pointSet
  - zones: group of entities with topological info, like orientation. E.g. cellZone
- Used for assigning different behavior
  - Physical properties (e.g. density)
  - Physical models in the same simulation (e.g. solid and liquid in CHT)
  - Mesh displacement
  - Assignment of boundary conditions
- Identification marks to flag points, faces or cells for further processing or checking, e.g.
  - highly skewed faces after checkMesh



Zones in OpenFOAM

- Zones can be generated in the meshing stage or assigned later
- If you already have the mesh, sets and zones can be created with the `topoSet` utility
- `topoSet` group entities that match the criteria specified in the `topoSetDict` dictionary, located in `system`

- Create a new cellSet with the cells that are inside the boxes. The points correspond to opposed vertices
- Create a new faceZone with the faces that are lying on the specified plane
- More examples in `$FOAM_TUTORIALS/incompressible/pisoFoam/RAS/cavity/`

```
actions
(
    {
        name    box1a;
        type    cellSet;
        action  new;
        source  boxToCell;
        boxes
        (
            (0 0 0) (0.05 0.05 0.01)
            (0.1 0.1 0) (0.02 0.02 0.01)
        );
    }
    {
        name      planeToFaceZone;
        type      faceZone;
        action    new;
        source    planeToFaceZone;
        point     (0.05 0 0);
        normal    (1 0 0);
        option    closest;
    }
);
```

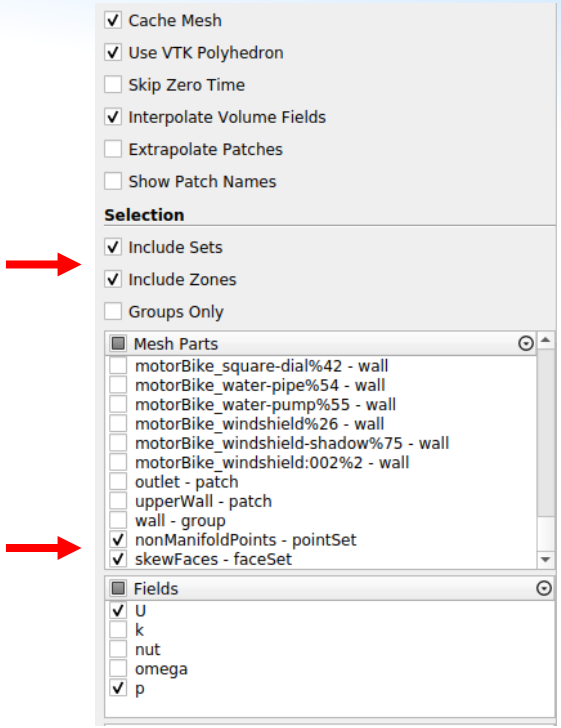
Visualizing sets

- You can load the sets/zones directly within paraFoam.
- In paraFoam, simply select the option Include Sets and then select the sets you want to visualize.
- If you are using ParaView, convert the sets to VTK format or an alternative format.
- To convert the faces/cells/points to VTK format

```
$> foamToVTK -set_type name_of_sets
```

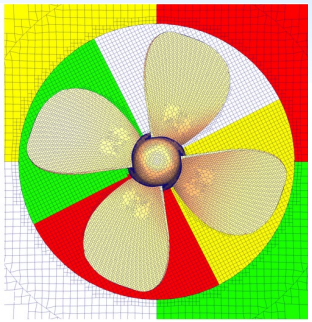
  where set_type is the type of sets (faceSet, cellSet, pointSet, surfaceFields) and name_of_sets is the name of the set located in the directory `constant/polyMesh/sets`

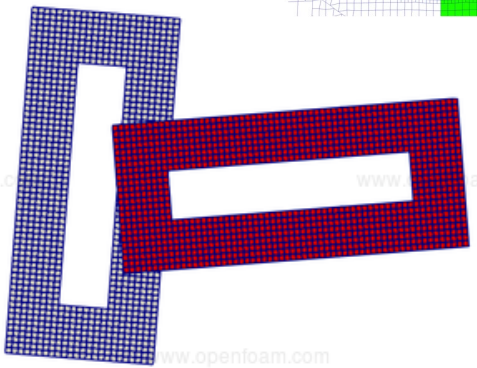
- `foamToVTK` will create a directory named VTK, where you will find the failed faces/cells/points in VTK format



Multiple meshes

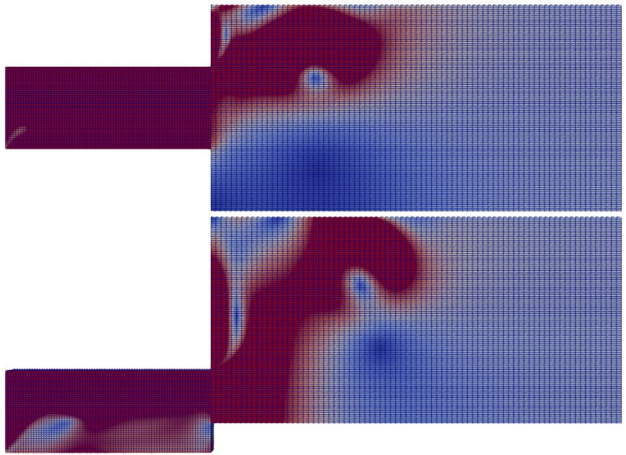
- Sometimes, different regions require different mesh resolution
- Typical scenarios:
  - Conjugated heat transfer
  - Multiphysics solvers
  - Moving domains
- If it is possible to implement, it can simplify the meshing process and reduce the simulation time
- Several approaches
  - Coupled meshes with AMI
  - Non-conformal coupling (OpenFOAM 10)
  - Overset (chimera) meshes (ESI version)
- Based on interpolation: numerical errors



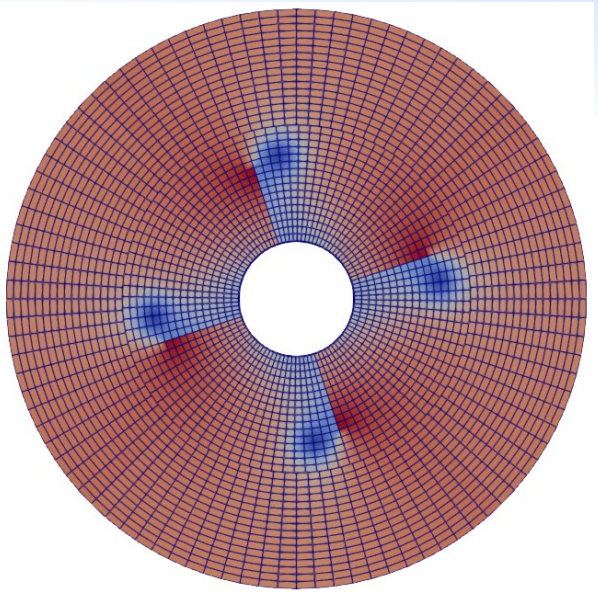


Coupling with AMI

- The Arbitrary Mesh Interface (AMI) is a boundary condition that couples non overlapping meshes
- Performs a weighted interpolation
- Single mesh can be created first, then split
- Separate meshes can be merged



`oscillatingInletACMI2D`



`mixerVesselAMI2D`

Lecture overview

- Geometry preparation
- Finite volume meshing:
  - What is a mesh?
  - Mesh generation
  - Aspects of mesh resolution and quality
- Mesh manipulation in OpenFOAM
  - Grouping regions into zones
  - Dealing with multiple meshes
- Open-source tools for geometry preparation and meshing:
  - SALOME
  - blockMesh
  - snappyHexMesh

Some open-source/free tools

Geometry

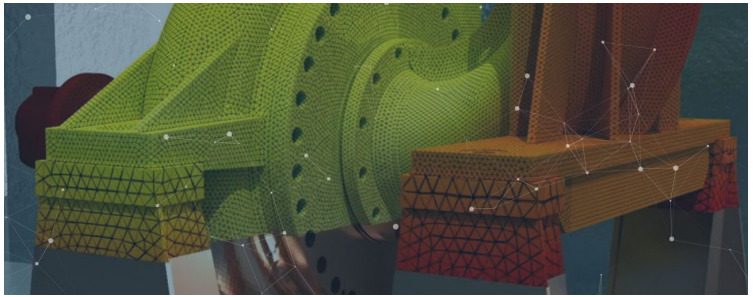
- SALOME: An open-source platform for numerical simulation. www.salome-platform.org
- Free-cad: light tool for simple designs. http://sourceforge.net/apps/mediawiki/free-cad/
- OpenSCAD: 3D solid modeling from script files. http://openscad.org/
- Onshape: cloud (web) service. Not open-source, but you can create a free account. https://www.onshape.com/

Meshing

- OpenFOAM toolkit: blockMesh, snappyHexMesh, foamyHexMesh
- SALOME
- cfMesh: open-source library for volume mesh generation based on OpenFOAM. https://sourceforge.net/projects/cfmesh/
- Gmsh: open source 3D finite element mesh generator. https://gmsh.info/

SALOME

- SALOME is an extensible platform that can integrate users´s computational codes or modules.
- The entire project is distributed under the open source LGPL license
- Aims to support pre and post-processing needs for numerical simulations.
- The platform includes many features:
  - CAD modelling
  - Mesh generation
  - Scientific visualization
  - Computational code supervision
  - Field manipulation
  - PDEs solver



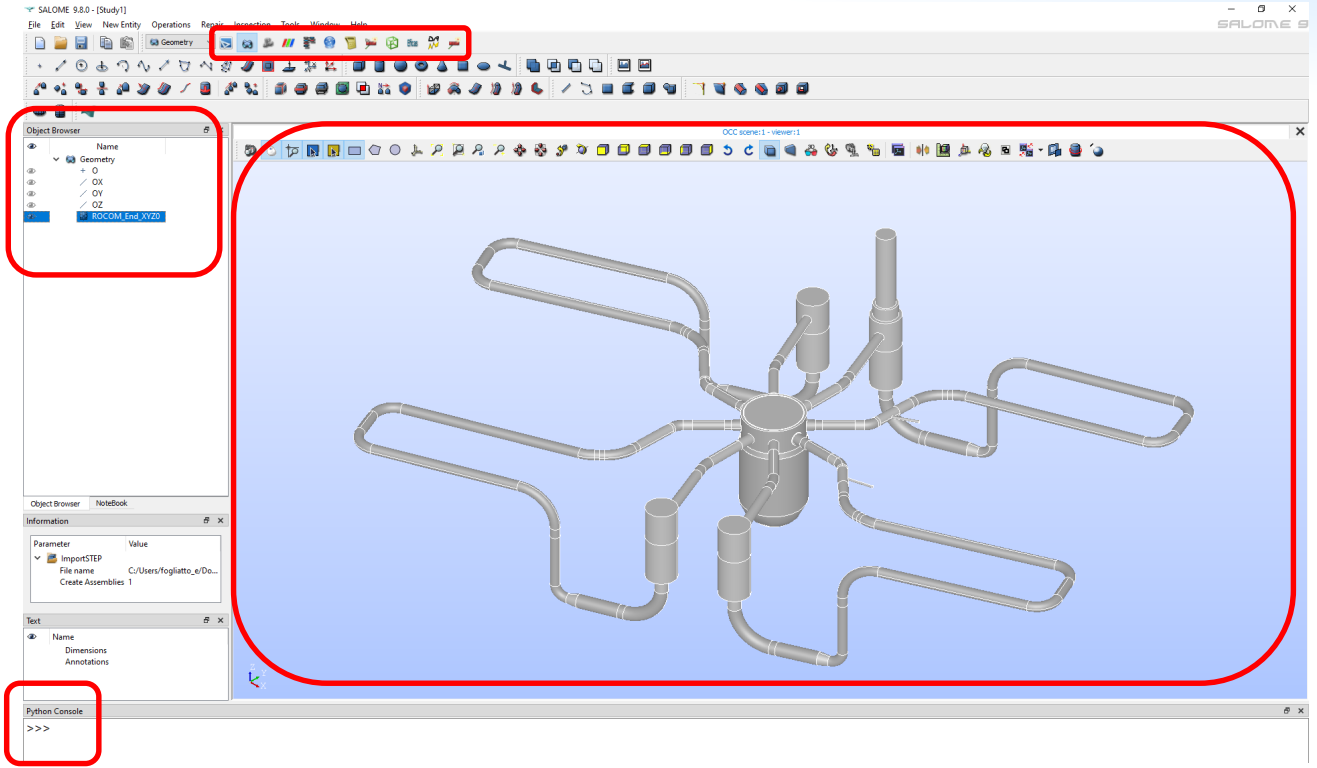
What can you do with SALOME?

- Define geometrical models (create/modify geometrical items), import and export them using the BREP, IGES and STEP formats
- Define meshing of these geometrical items, import and export them
- Handle physical properties and quantities attached to geometrical items, import and export them to a reusable format
- Perform computations using a solver (optionally provided): read input data, configure the solver, and write calculation results
- Visualize result fields in 3D, 2D and export images of their visualization to an appropriate format
- Manage study schemes: definition, save/restore
- Manage computation schemes: definition, execution
- Available for Windows and Linux

What can you do with SALOME?

Modules

Object inspector



Scene viewer

Python console

Geometry module

CAD modeling can be performed with two modules: Shaper and Geometry

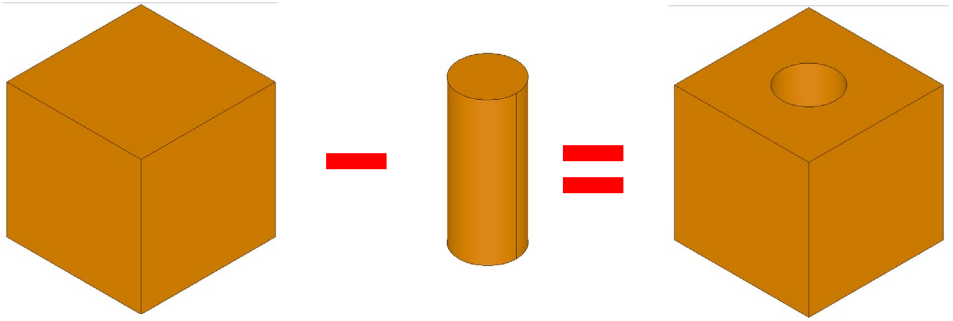
The model is built using simple geometrical objects

- Basic objects: points, lines, circles
- Primitives: cubes, spheres, cones

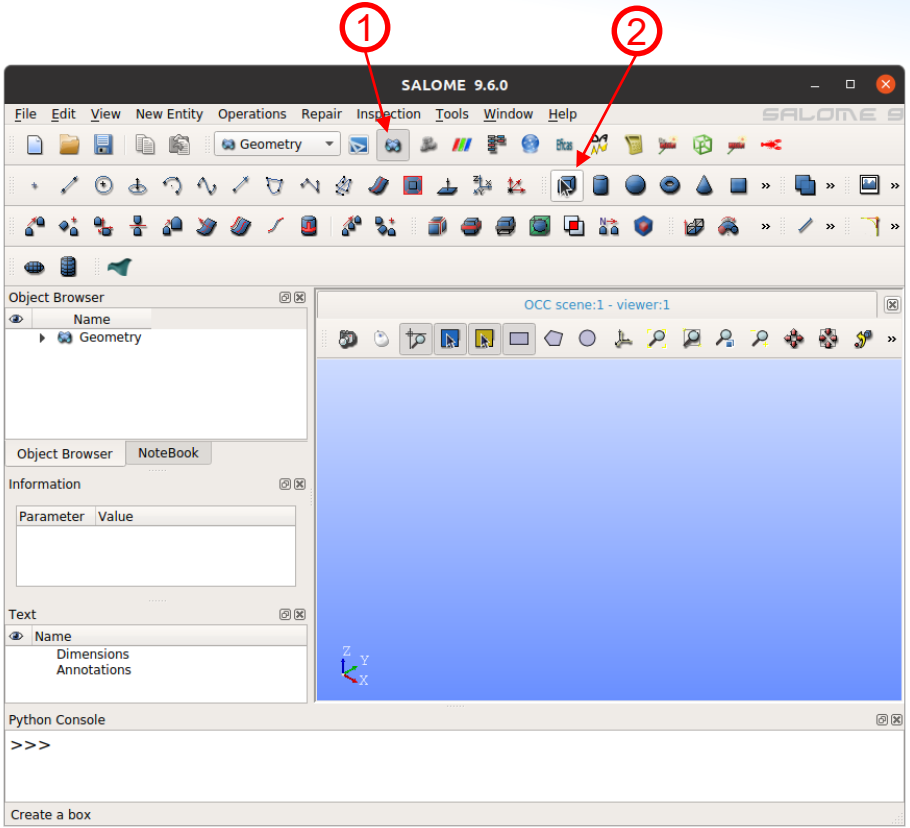
Complex objects can be created by:

- Extrusion
- Revolution
- Interpolation

Modelling approach: create your final geometry as a combination of simpler objects

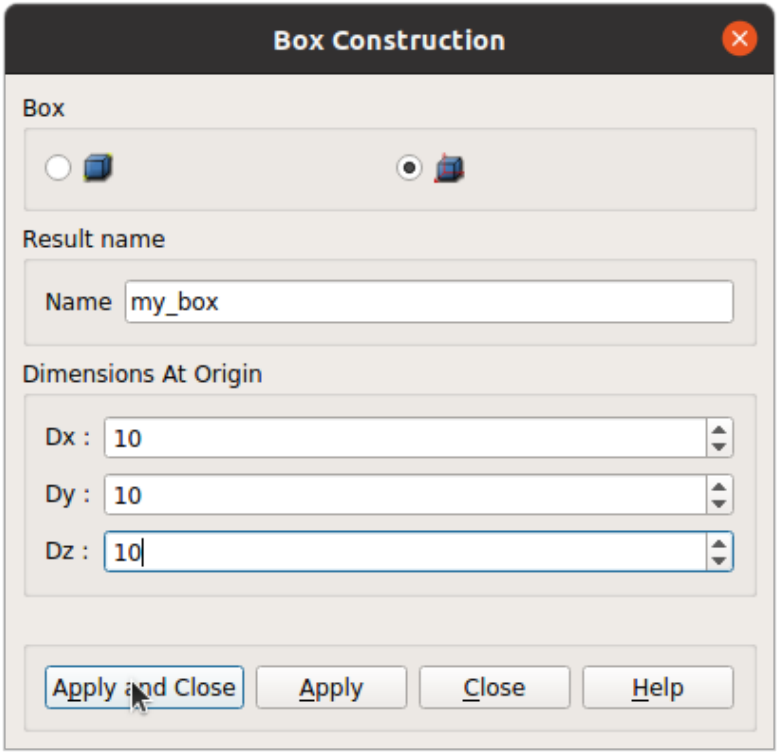


Example: building a box



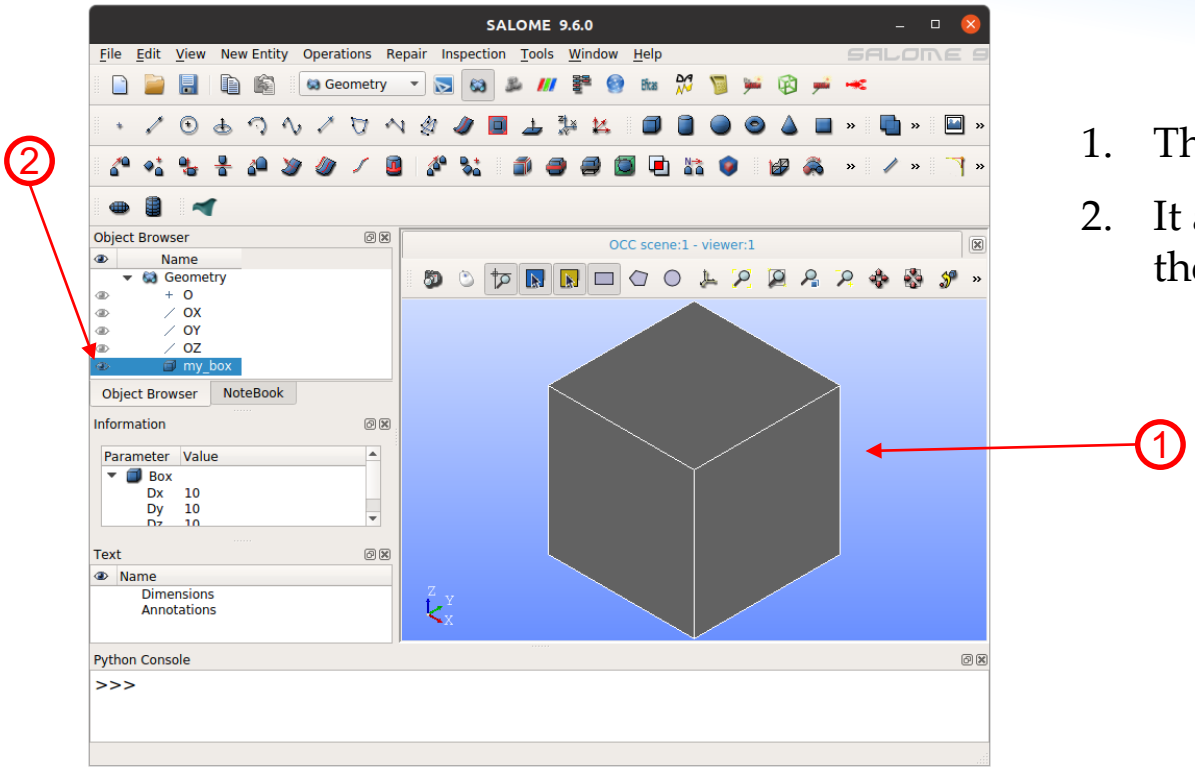
1. In a new session, select the geometry module
2. Click on Create a Box

Example: building a box



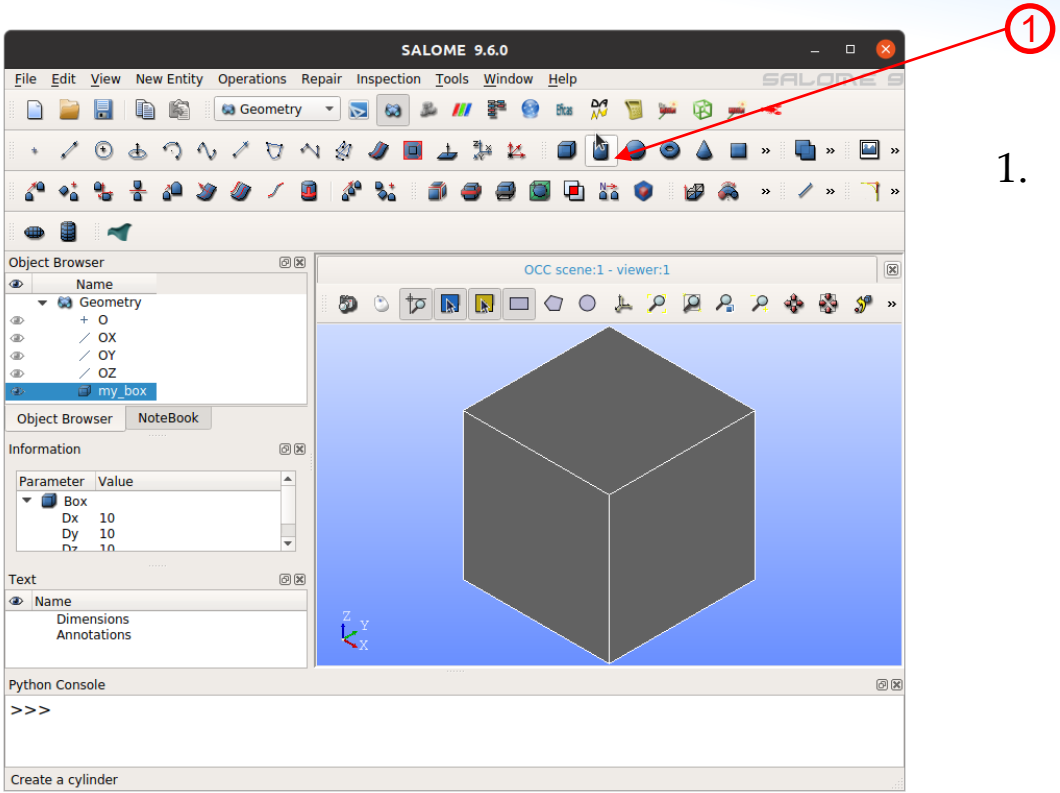
1. Add a name (optional) and dimensions
2. Click Apply and Close

Example: building a box



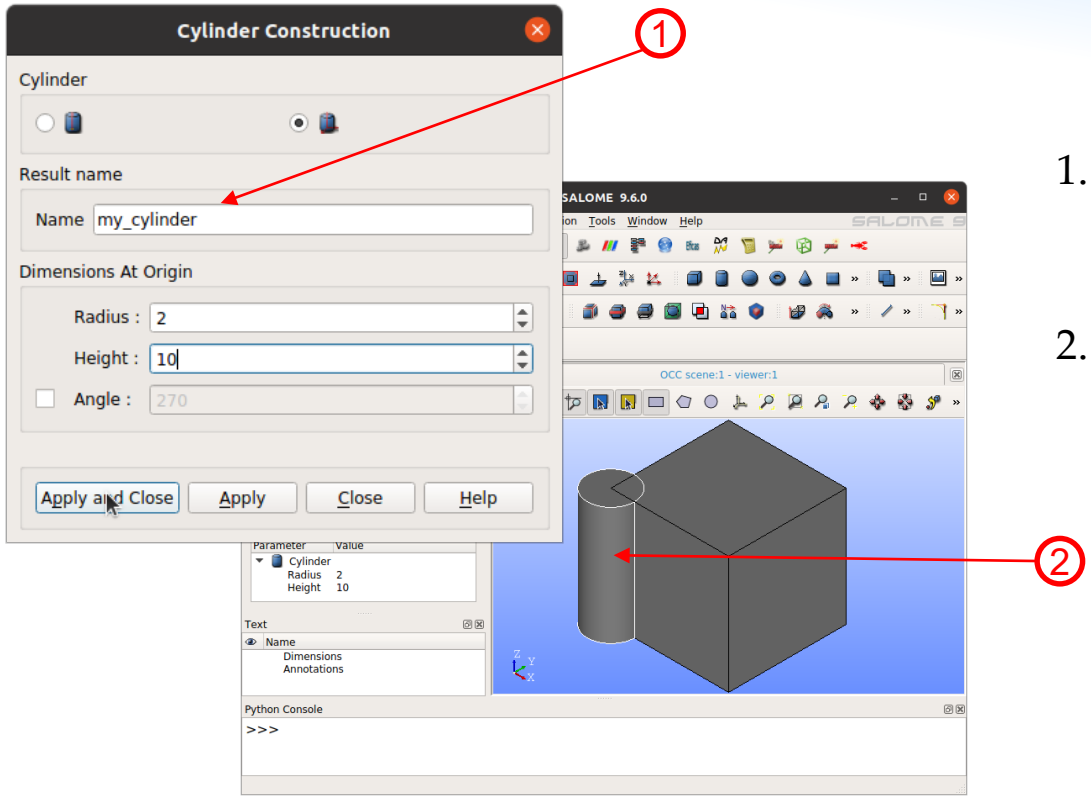
1. The box can be seen in the viewer
2. It also appears in the browser with the chosen name

Example: building a cylinder



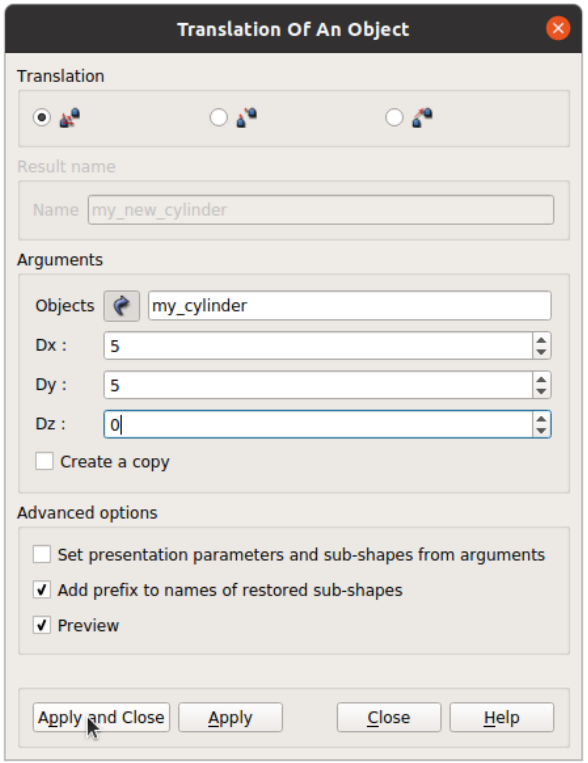
1. Click on Create a Cylinder

Example: building a cylinder



1. Add a name (optional), radius and height. The cylinder will be built at the origin
2. Click Apply and Close, and the cylinder will appear in the viewer

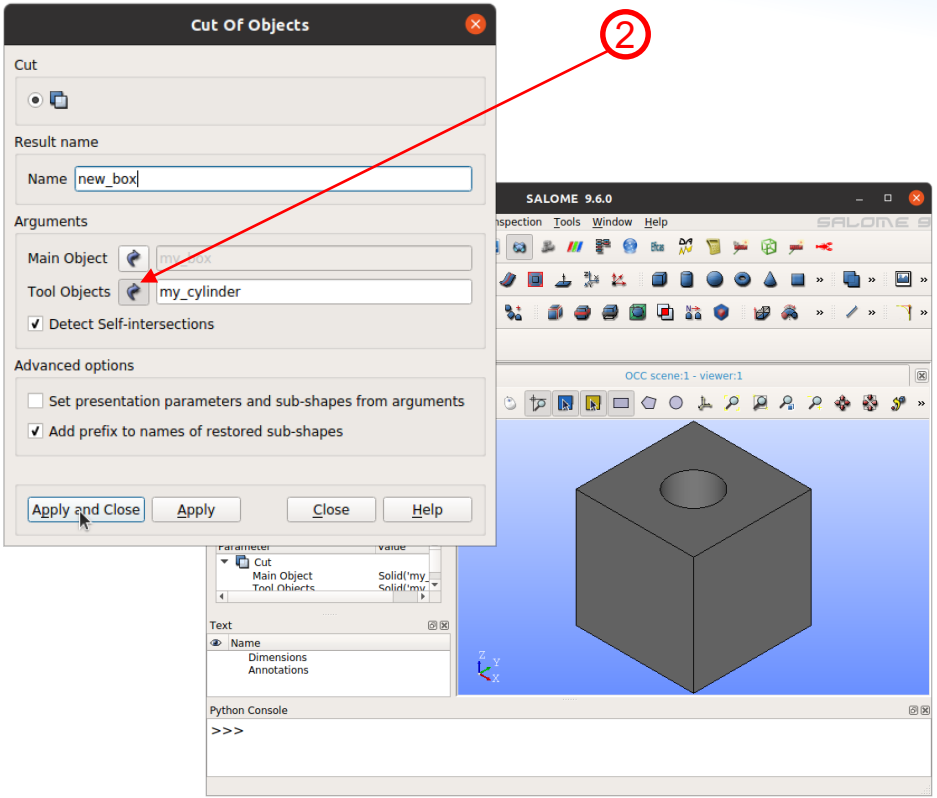
Example: building a box with a hole



1. We need to use the cylinder as a cutting tool
2. To move the cylinder, apply a transformation.

Operations>Transformation>Translation

Example: building a box with a hole



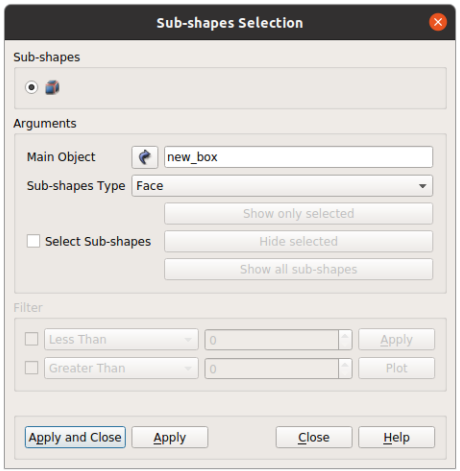
1. Perform the cutting using Boolean operations

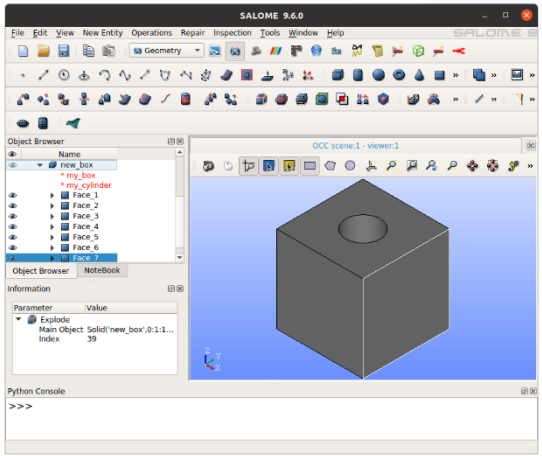
   Operations>Boolean>Cut

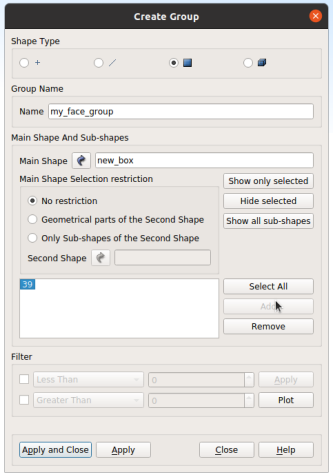
2. Add a name (optional), select the main object and the tool from the browser. Press the arrow first

3. Click Apply and Close, and the new object will appear in the viewer

Working with features





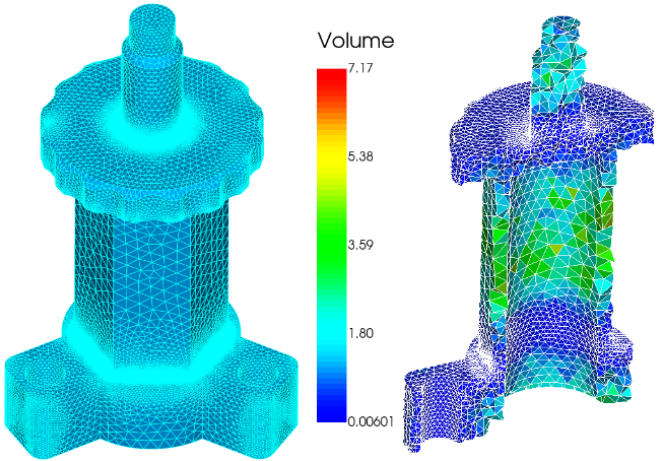


② ③ ④

1. The user can create groups to handle objects of similar types
2. Get the desired sub-entities with *New Entity>Explode*
3. The sub entities are now shown in the browser
4. Create groups with *New Entity>Group>Create Group*. Sub-entities can be selected from the browser or viewer
5. Faces, or groups of faces, can be exported to other meshing tools (e.g. in STL format)

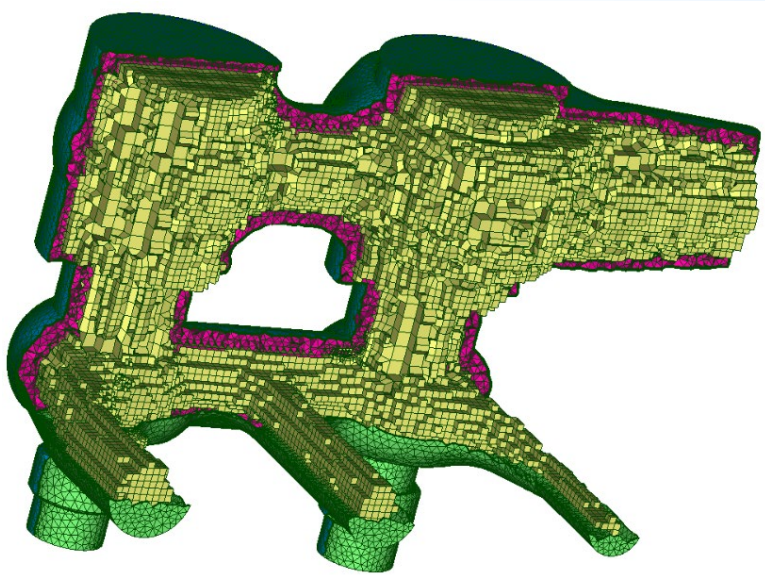
Meshing with SMESH

- SMESH — Salome MESH — is the mesh module of SALOME
- Provides a wide range of meshing algorithms particularly suited for finite element and finite volume methods
- A mesh can be enriched with groups/labels to distinguish different regions of the geometry. This allows to differentiate the properties of the meshes or to identify boundaries to apply boundary conditions
- Transformations can be applied to produce complex meshes or compounds: rotation, symmetry, change of scale, etc.
- Editing a mesh in order to delete, add, or transform a few or several of its mesh elements is also possible via the easy-to-use GUI

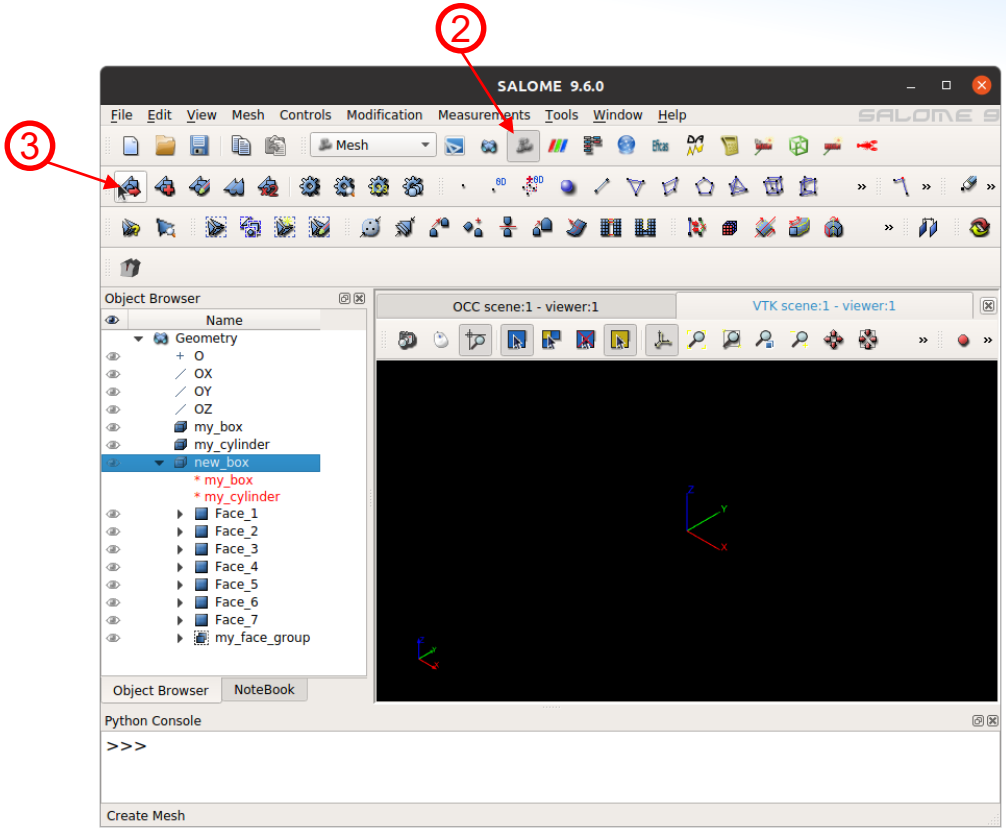


Meshing algorithms

- The SMESH module contains:
  - its own meshing tools for quadrangles, hexahedra, boundary-layer meshing, etc.;
  - open-source meshing tools: NETGEN and Gmsh;
  - commercial meshing tools from the MeshGems suite: MG-CADSurf, MG-Tetra, MG-Hybrid, MG-Hexa, published by 3DS Spatial and require a commercial license.
- These meshing tools are based on different algorithms and properties (local size, growth rate, forced vertices, etc.) that can be adjusted to obtain the best mesh quality for each specific numerical simulation.
- These different meshing algorithms can be combined in SALOME. For non-trivial meshing, a versatile feature of SMESH is that it allows to combine different meshing algorithms if needed

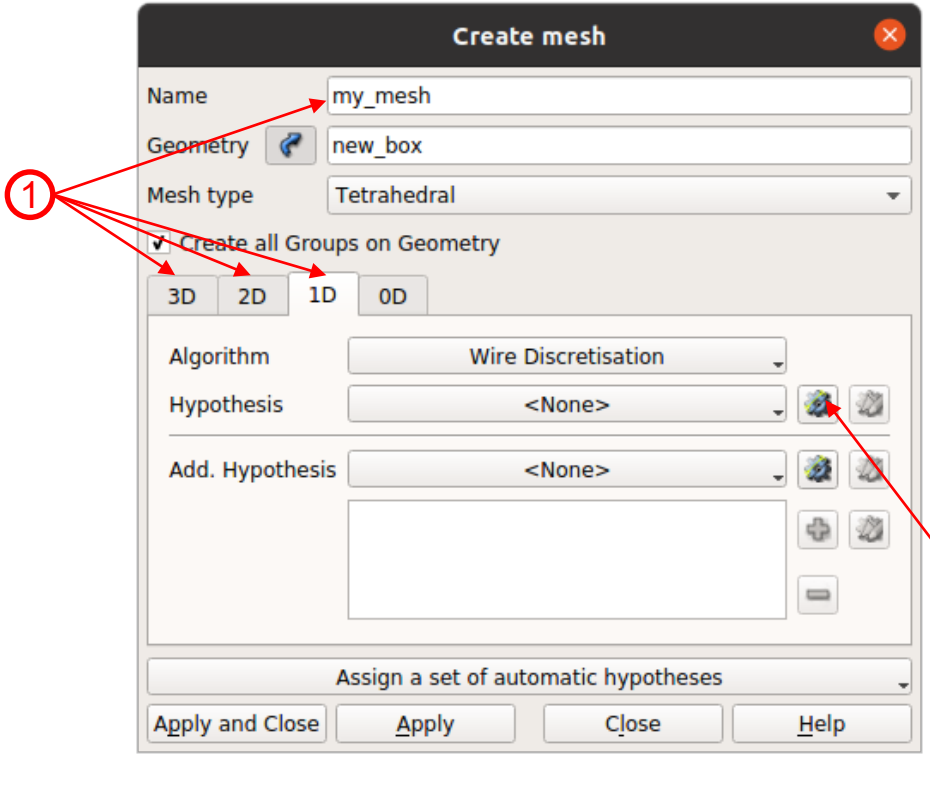


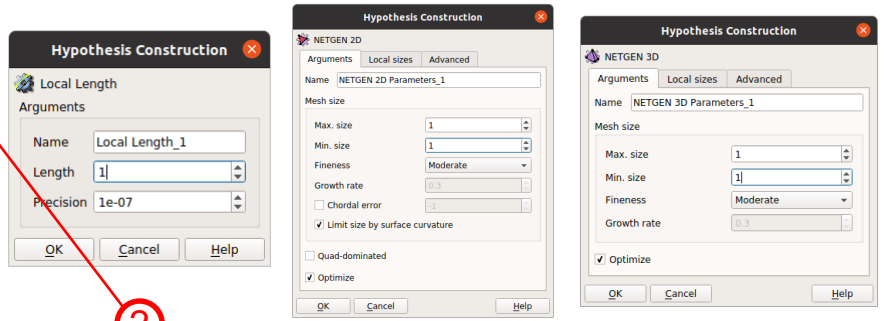
Example: meshing with SMESH



1. Let's continue with the previous example
2. Go to the mesh module
3. Select the desired object, and click on Create Mesh

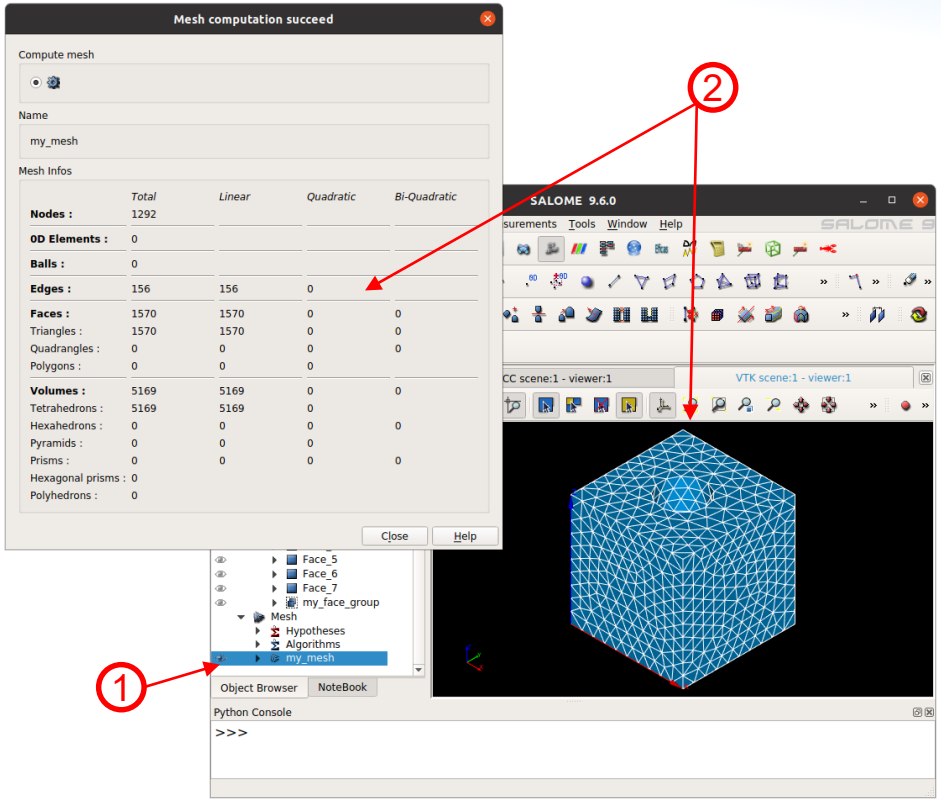
Example: meshing with SMESH





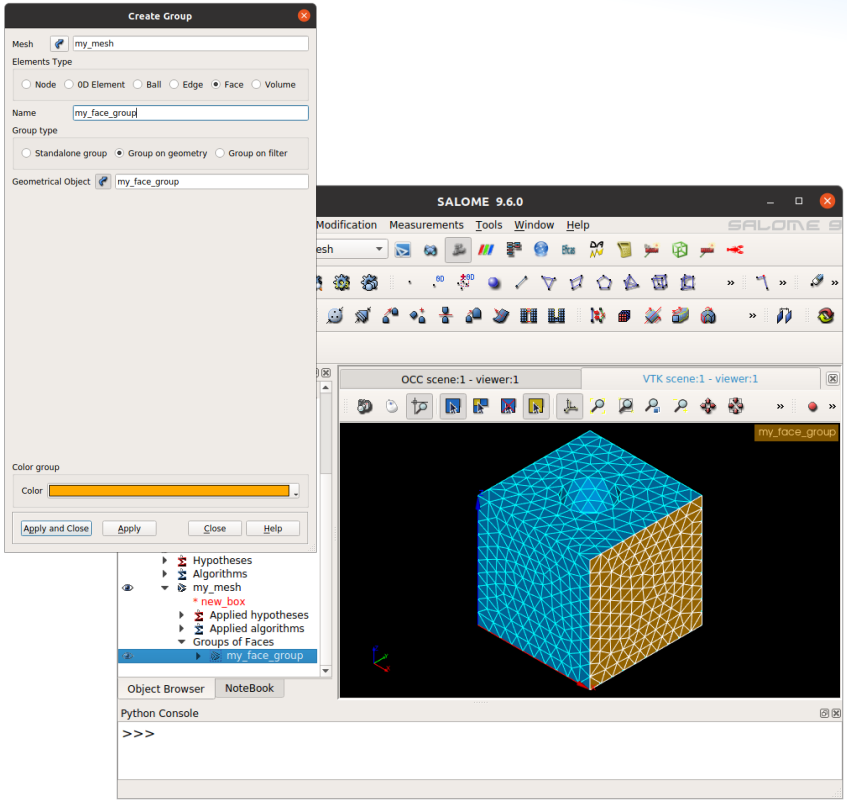
1. Add a name (optional) and select the desired meshing algorithms (meshing engine) for each entity
2. For each Algorithm, you need to define the Hypothesis

Example: meshing with SMESH



1. The mesh is in the browser. Select and click Compute
2. Information about the meshing process and the mesh are shown

Example: meshing with SMESH



1. Groups can be created with *Mesh>Create Group*
2. Typically faces for boundary conditions and volumes for cell zones
3. When the mesh is ready, select the mesh in the browser, and then *File>Export>UNV file*
4. The mesh can be converted with the utility `ideasUnvToFoam`

Python scripting (TUI)

- SALOME can run python scripts
- You can do all your tasks (CAD modeling, meshing, etc) without a GUI
- Extremely powerful if you need to do repetitive tasks, or perform extensive calculations/modifications on the geometrical features.
- Easy to start: do some operations using the GUI, and then save the corresponding Python commands with *File>Dump Study*

Python scripting (TUI)

```
…

geompy = geomBuilder.New()

O = geompy.MakeVertex(0, 0, 0)
OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)

my_box = geompy.MakeBoxDXDYDZ(10, 10, 10)
my_cylinder = geompy.MakeCylinderRH(2, 10)
geompy.TranslateDXDYDZ(my_cylinder, 5, 5, 0)
new_box = geompy.MakeCutList(my_box, [my_cylinder],True)

[Face_1,Face_2,Face_3,Face_4,Face_5,Face_6,Face_7] =
geompy.ExtractShapes(new_box,geompy.ShapeType["FACE"], True)

my_face_group =
geompy.CreateGroup(new_box,geompy.ShapeType["FACE"])

geompy.UnionIDs(my_face_group, [39])
```

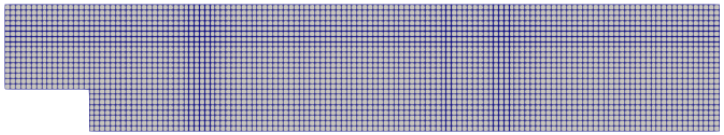
```
import  SMESH, SALOMEDS
from salome.smesh import smeshBuilder

smesh = smeshBuilder.New()

my_mesh = smesh.Mesh(new_box)
Regular_1D = my_mesh.Segment()
Local_Length_1 = Regular_1D.LocalLength(1,None,1e-07)
NETGEN_2D = my_mesh.Triangle(algo=smeshBuilder.NETGEN_2D)
NETGEN_2D_Parameters_1 = NETGEN_2D.Parameters()
NETGEN_2D_Parameters_1.SetMaxSize( 1 )
…
NETGEN_3D = my_mesh.Tetrahedron()
NETGEN_3D_Parameters_1 = NETGEN_3D.Parameters()
NETGEN_3D_Parameters_1.SetMaxSize( 1 )
…
isDone = my_mesh.Compute()
…
my_mesh.ExportUNV( r'my_mesh.unv' )
…
```

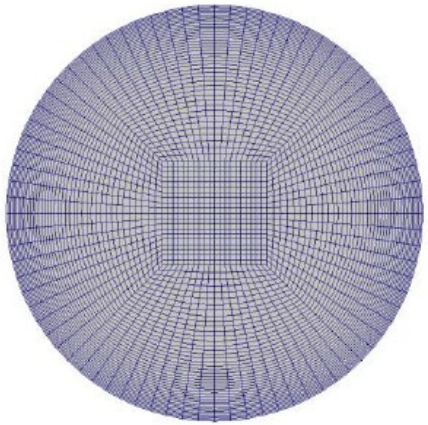
Lecture overview

- Geometry preparation
- Finite volume meshing:
  - What is a mesh?
  - Mesh generation
  - Aspects of mesh resolution and quality
- Mesh manipulation in OpenFOAM
  - Grouping regions into zones
  - Dealing with multiple meshes
- Open-source tools for geometry preparation and meshing:
  - SALOME
  - blockMesh
  - snappyHexMesh

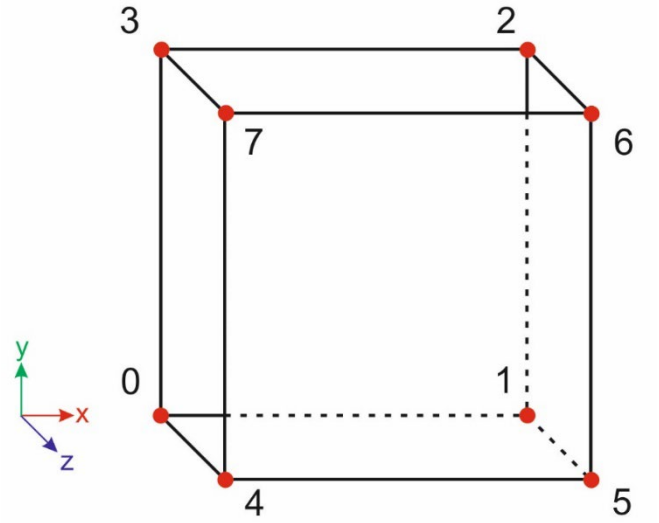
The `blockMesh` utility

- `blockMesh` is a multi-block mesh generator which can be used for simple geometries
- This meshing tool generates high quality meshes
- As the complexity of the geometry increases, the effort and time required to prepare a case increases exponentially
- The mesh is generated from a dictionary file named `blockMeshDict` located in the `system` directory
- Usually, the background mesh used with `snappyHexMesh` consist of a single rectangular block and `blockMesh` can be used.





The blockMesh utility



Example: 3D box

Vertices

0: (0 0 0)
1: (1 0 0)
2: (1 1 0)
3: (0 1 0)
4: (0 0 1)
5: (1 0 1)
6: (1 1 1)
7: (0 1 1)

Blocks

0 1 2 3 4 5 6 7

Faces

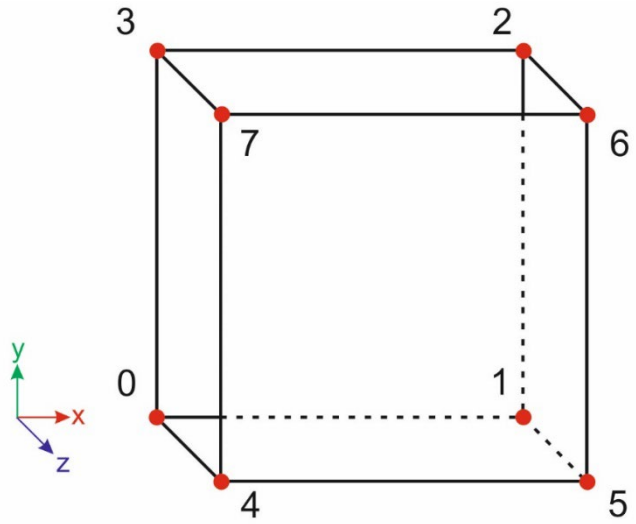
(0 4 7 3)
(0 1 5 4)
(1 2 6 5)
(3 7 6 2)
(0 3 2 1)
(4 5 6 7)

- To generate a mesh with blockMesh, you will need to define vertices for the blocks, block connectivity and number of cells in each direction.
- The boundary patches are defined using the faces conectivity

The `blockMeshDict` dictionary

```
convertToMeters 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
(
);
```

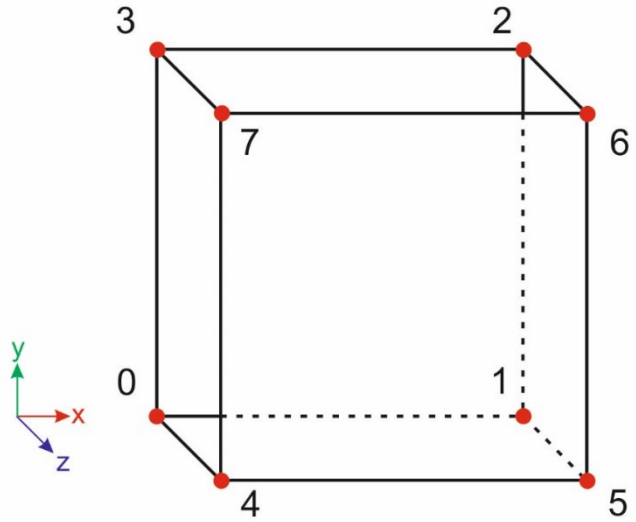
The keyword convertToMeters (or scale in some versions) is a scaling factor. In this case, the dimensions are not scaled



The `blockMeshDict` dictionary

```
convertToMeters 0.1;

vertices
(
    (0 0 0)       // 0
    (1 0 0)       // 1
    (1 1 0)       // 2
    (0 1 0)       // 3
    (0 0 0.1)     // 4
    (1 0 0.1)     // 5
    (1 1 0.1)     // 6
    (0 1 0.1)     // 7
);

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

edges
(
);
```

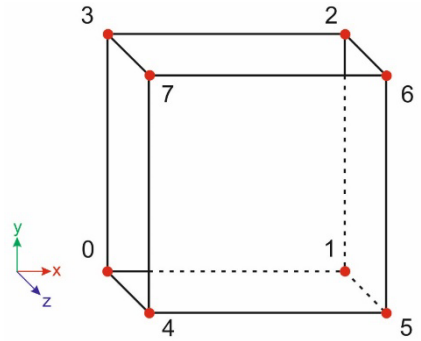
- In the `vertices` entry, include the points needed to construct each block. For a single 3D block, 8 vertices are needed
- Vertex numbering starts from 0



The `blockMeshDict` dictionary

```
convertToMeters 0.1;

vertices
(
    (0 0 0)      // 0
    (1 0 0)      // 1
    (1 1 0)      // 2
    (0 1 0)      // 3
    (0 0 0.1)    // 4
    (1 0 0.1)    // 5
    (1 1 0.1)    // 6
    (0 1 0.1)    // 7
);

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

edges
(
);
```

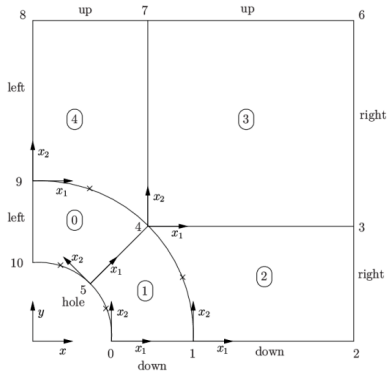
- In the `block` entry, **hex** means that it is a structured hexahedral block
- `(0 1 2 3 4 5 6 7)` are the vertices used to define the block. The order is important!. The first vertex in the list represents the origin of the coordinate system
- `(20 20 1)` is the number of mesh cells in each direction (X Y Z)
- `simpleGrading (1 1 1)` is the grading or mesh stretching in each direction (X Y Z)



The `blockMeshDict` dictionary

```
convertToMeters 0.1;

vertices
(
    (0 0 0)       // 0
    (1 0 0)       // 1
    (1 1 0)       // 2
    (0 1 0)       // 3
    (0 0 0.1)     // 4
    (1 0 0.1)     // 5
    (1 1 0.1)     // 6
    (0 1 0.1)     // 7
);

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

edges
(
);
```

- Edges are constructed from the vertices definition.
- Each edge joining two vertices is assumed to be straight by default. The user can specify any edge to be curved by entries in the section edges
- Possible options are Bspline, arc, line, polyline, project, projectCurve, spline.

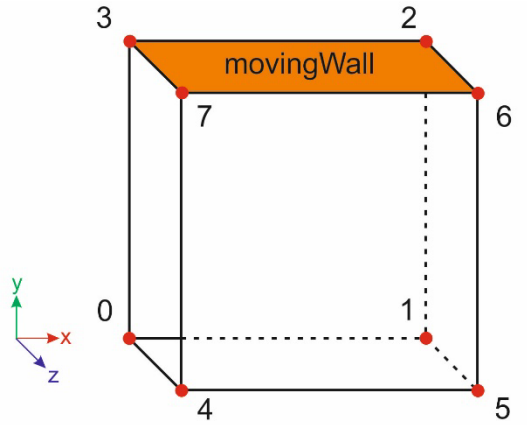


The `blockMeshDict` dictionary

```
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)
        );
    }
);
```

→ 2D simulation

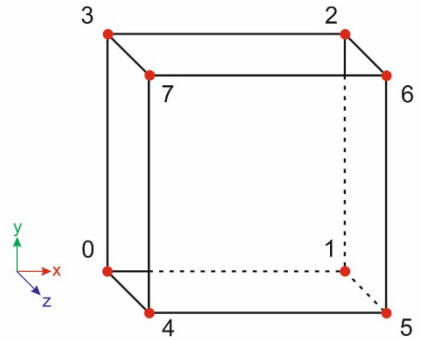
- Patch names are defined in the `boundary` entry.
- Used to apply physical boundary conditions. `type` can be `patch`, `wall`, `empty`, etc.
- The `faces` entry contains a connectivity list of the vertices that made up the surface patch or face, that is, (3 7 6 2). The order does not matter.
- If a face is not listed, it is assigned as `defaultFaces`.



The `blockMeshDict` dictionary

```
mergePatchPairs
(
);
```

- Blocks can be merged in `mergePatchPairs`. Used when contiguous blocks have different `spacing`.
- The block patches to be merged must be first defined in the boundary list, `blockMesh` then connect the two blocks.
- If there is one single block there is no need to merge patches.

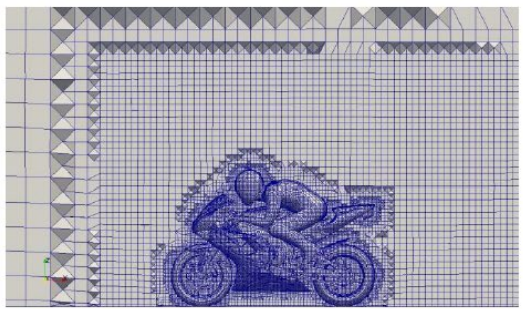
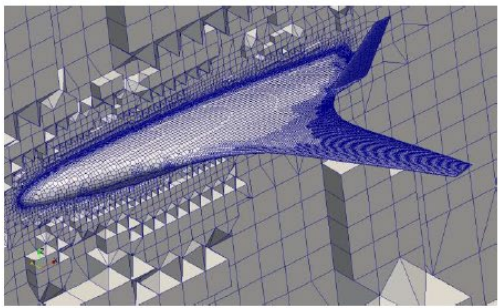
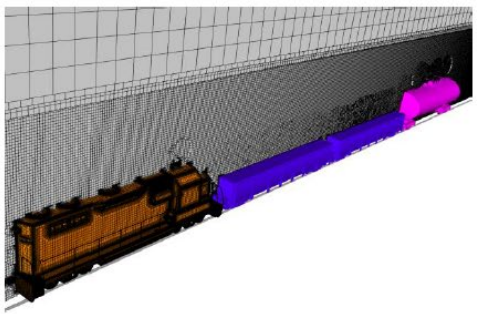


Lecture overview

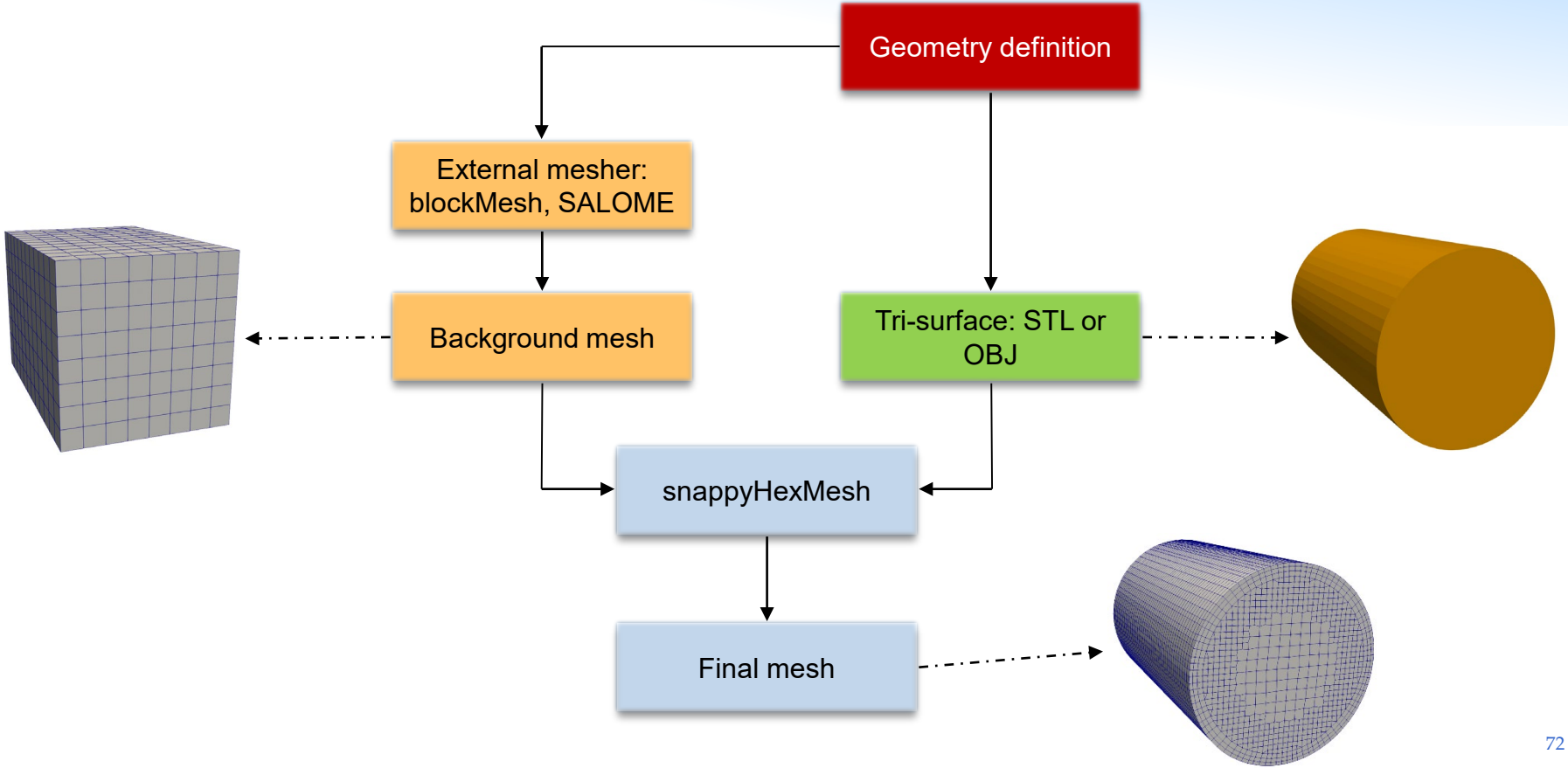
- Geometry preparation
- Finite volume meshing:
  - What is a mesh?
  - Mesh generation
  - Aspects of mesh resolution and quality
- Mesh manipulation in OpenFOAM
  - Grouping regions into zones
  - Dealing with multiple meshes
- Open-source tools for geometry preparation and meshing:
  - SALOME
  - blockMesh
  - snappyHexMesh

The snappyHexMesh utility

- Generates 3-dimensional meshes containing hexahedra (hex) and split-hexahedra (split-hex) automatically from triangulated surface geometries (tri-surfaces)
- Ideal for complex geometries. Robust. Runs in parallel with load balancing
- In order to run snappyHexMesh, the user requires the following:
  - one or more tri-surface files located in a `constant/triSurface`. STL or OBJ formats.
  - a background hex mesh
  - a `snappyHexMeshDict` dictionary in the system sub-directory of the case.

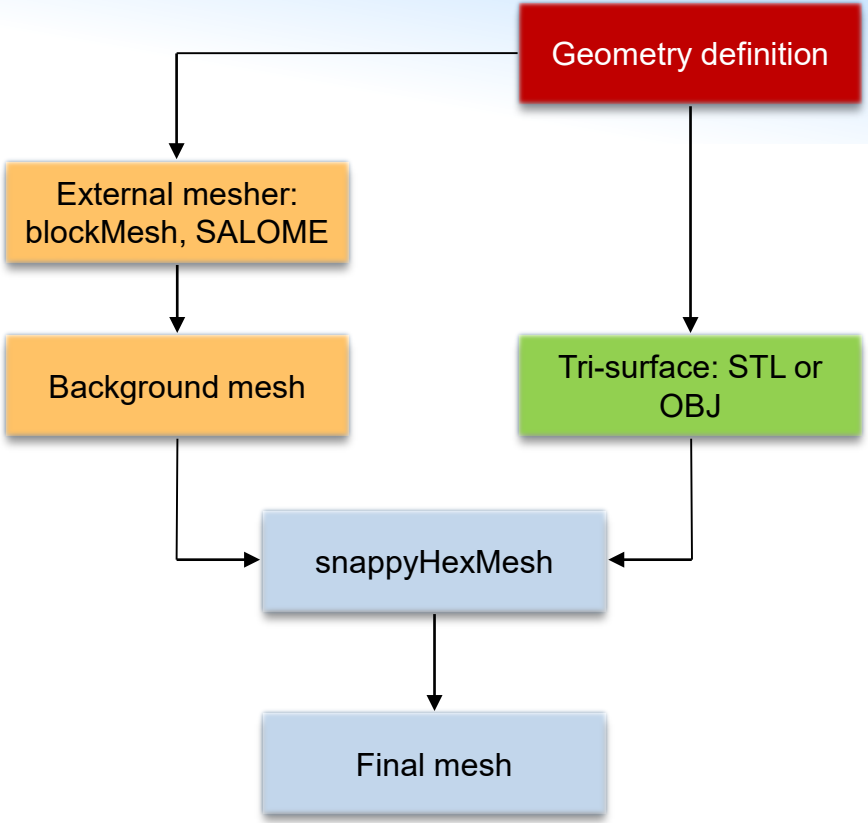




snappyHexMesh workflow



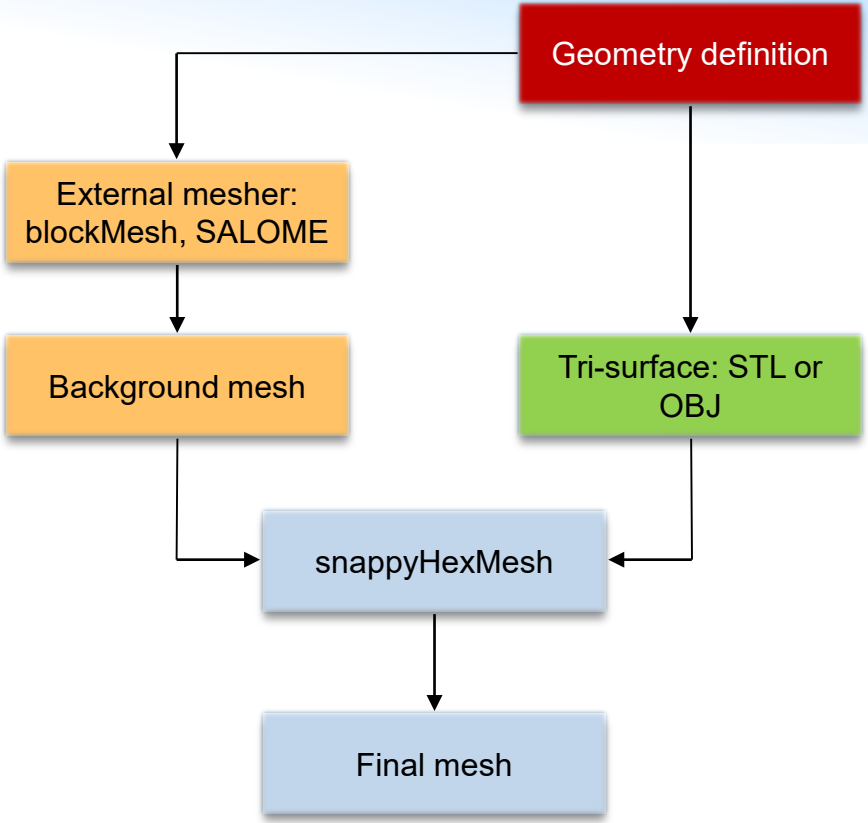
Workflow: tri-surfaces

- The STL geometry can be obtained from any geometry modeling tool, e.g. SALOME
- The STL file can be made up of a single surface describing the geometry, or multiple surfaces that describe the geometry.
- In the case of a STL file with multiple surfaces, we can use local refinement in each individual surface.
  - In SALOME, select each surface (or group) and export in ASCII format. Then add the surface name in the first line of the STL file (solid […]) and concatenate the files
  - Important: always include the boundary names in the STL file, so snappy can add the names internally. Try to avoid utilities like `surfaceToPatch`
- The STL files are always located in `constant/triSurface`



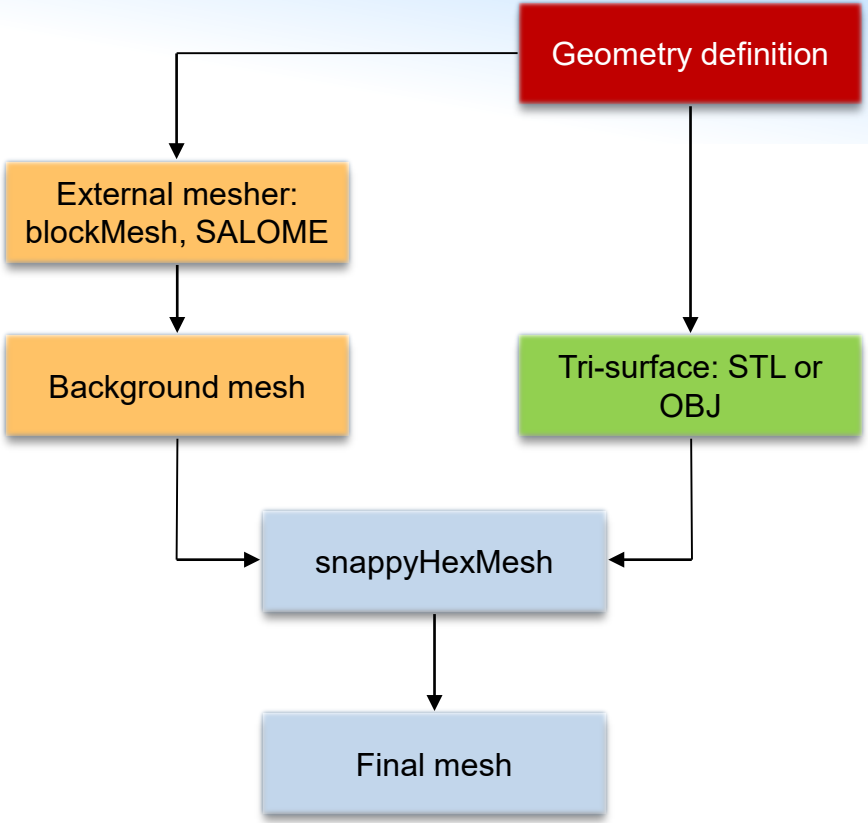
Workflow: background mesh

- The background or base mesh can be generated using `blockMesh` or an external mesher.
- The mesh must consist purely of hexes.
- The cell aspect ratio should be approximately 1, at least near the STL surface.
- There must be at least one intersection of a cell edge with the STL surface.
- It is extremely recommended to align the background mesh with the STL surface. However, most of the times this not trivial.



Workflow: snappy

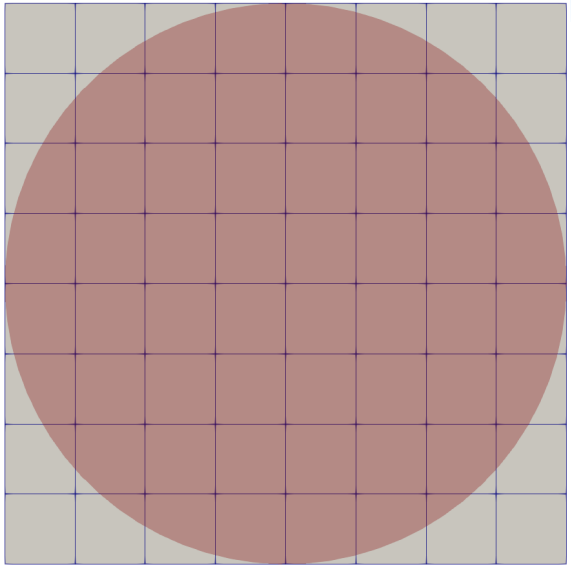
- `snappyHexMesh` uses the dictionary `snappyHexMeshDict` located in `system`
- Three steps: castellation, snapping, and boundary layer meshing.
- Mesh quality enforcement in snapping and layer addition
- The final mesh is always located in the directory `constant/polyMesh`
  - `runApplication snappyHexMesh -overwrite`: keeps only the final mesh
  - `runApplication snappyHexMesh`: writes intermediate meshes
- Refinement levels instead of mesh size



Meshing steps

Step 1. Creating the background hexahedral mesh

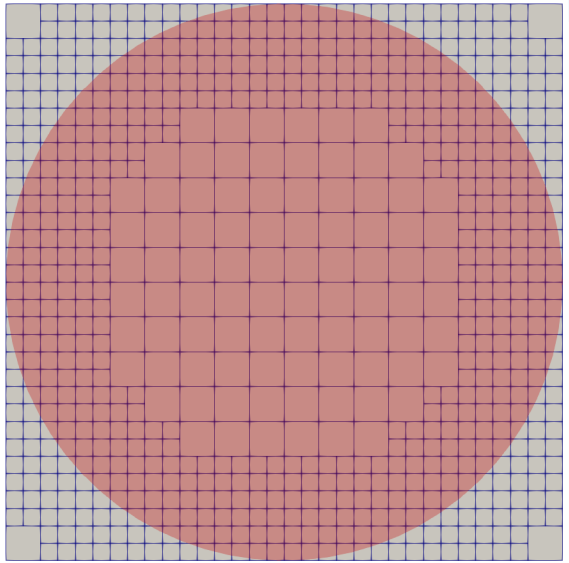
- Before `snappyHexMesh` is executed the user must create a background mesh of hexahedral cells that fills the entire region. This can be done by using `blockMesh` or any other mesher.
- The following criteria must be observed when creating the background mesh:
  - The mesh must consist purely of hexes. That is, around the surfaces and volume regions where you are planning to add refinement, the mesh must consist of pure hexes
  - The cell aspect ratio should be approximately 1, at least near the STL surface.
  - There must be at least one intersection of a cell edge with the STL surface.



Meshing steps

Step 2. Cell splitting at feature edges

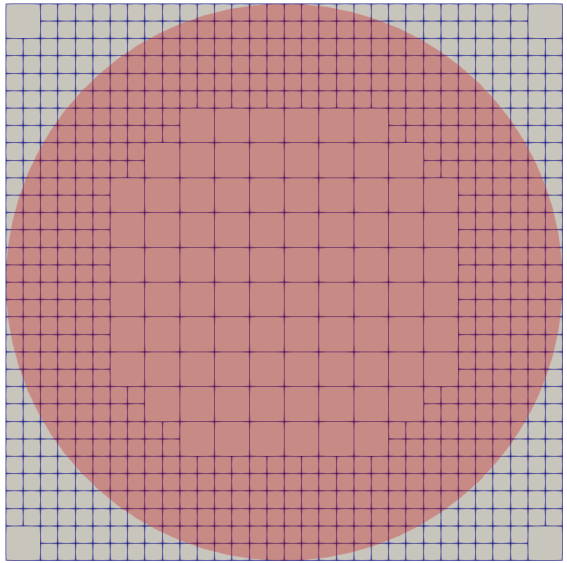
- Cell splitting is performed according to the specification supplied by the user in the `castellatedMeshControls` sub-dictionary in the `snappyHexMeshDict` dictionary.
- The splitting process begins with cells being selected according to specified edge features
- The feature edges can be extracted from the STL geometry file using the utility `surfaceFeatures`.
- The feature edges can also be extracted using `ParaView/paraFoam`.



Meshing steps

Step 3. Cell splitting at surfaces

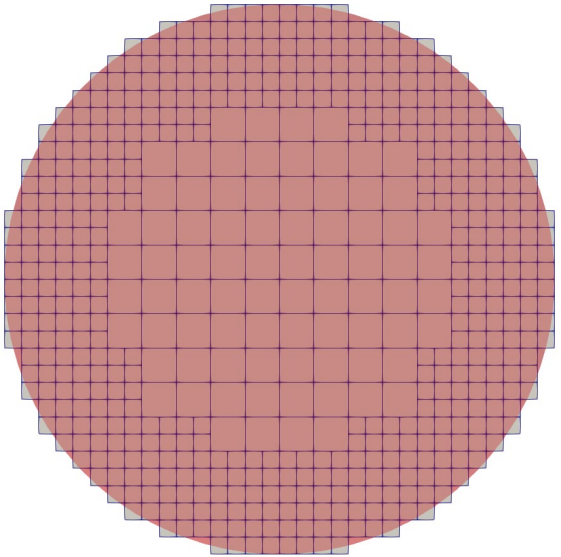
- Following feature edges refinement, cells are selected for splitting in the locality of specified surfaces
- The surface refinement (splitting) is performed according to the specification supplied by the user in the `refinementSurfaces` in the `castellatedMeshControls` sub-dictionary in the `snappyHexMeshDict` dictionary.



Meshing steps

Step 4. Cell removal

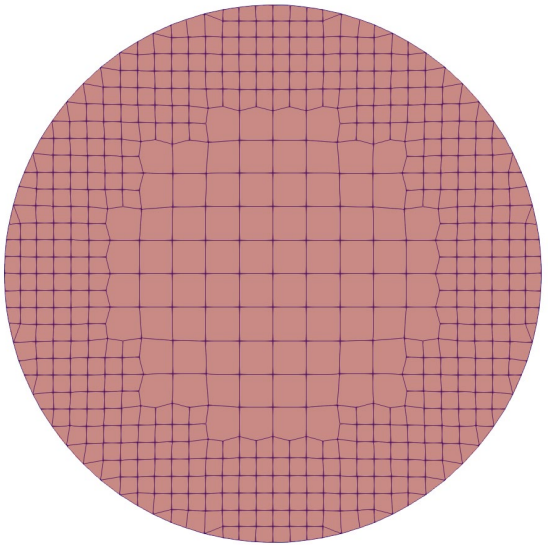
- Once the feature edges and surface splitting is complete, a process of cell removal begins.
- The region in which cells are retained are simply identified by a location point within the region, specified by the `locationInMesh` keyword in the `castellatedMeshControls` sub-dictionary in the `snappyHexMeshDict` dictionary.
- Cells are retained if, approximately, 50% or more of their volume lies within the region.
- Be careful to put the `locationInMesh` point in pure hexahedral regions. Do no put it in transition regions as you will get into problems.



Meshing steps

Step 5. Snapping to surfaces

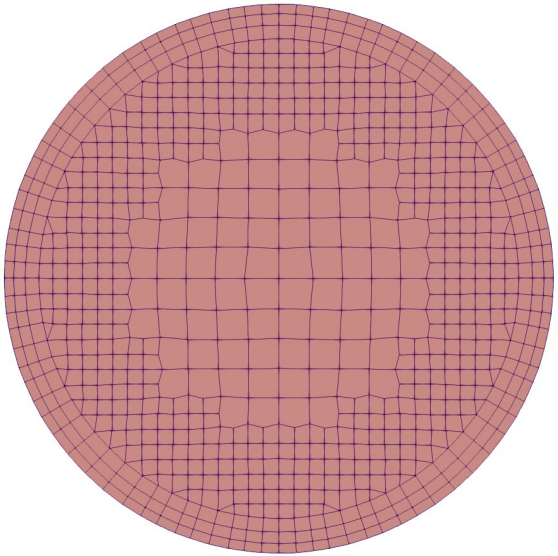
- After deleting the cells in the region specified and refining the volume mesh, the points are snapped on the surface to create a conforming mesh
- Mesh distortion to ensure quality restrictions
- Sometimes, the recommended `snapControls` options are not enough and you will need to adjust the values to get a good mesh. Save the intermediate steps with a high writing precision (`controlDict`)



Meshing steps

Step 6. Boundary layers

- Introduces boundary layer meshing in selected parts of the mesh
- This information is supplied by the user in the `addLayersControls` sub-dictionary in the `snappyHexMeshDict` dictionary
- This is the final step of the mesh generation process using `snappyHexMesh`



The `snappyHexMeshDict` dictionary

The dictionary `snappyHexMeshDict` consists of five main sections:

- `geometry`: sub-dictionary of all surface geometry used
- `castellatedMeshControls`: sub-dictionary of controls for castellated mesh.definition of feature, surface and volume mesh refinement. Definition of mesh location point.
- `snapControls`: definition of surface mesh snapping and advanced parameters.
- `addLayersControls`: definition of boundary layer meshing and advanced parameters. Only prismatic elements are added in this step, there is no refinement of the surface or volume mesh.
- `meshQualityControls`: definition of mesh quality metrics

The snappyHexMeshDict dictionary

```
castellatedMesh true; //or false
snap true;            //or false
addLayers true;       //or false

geometry
{
...
}
castellatedMeshControls
{
...
}
snapControls
{
...
}
addLayersControls
{
...
}
meshQualityControls
{
...
}
```

- Five general sections
- In the first lines we can enable the different meshing steps. For example, if we want to generate a body fitted mesh with no boundary layer we should proceed as follows:
  - castellatedMesh true;
  - snap true;
  - addLayers false;

snappyHexMeshDict: geometry

```
geometry
{
    regionSTL.stl
    {
        type triSurfaceMesh;
        regions
        {

            inlet   { name inlet; }
            outlet  { name outlet; }
            wall    { name wall; }
        }
    }
};
```

Name inside the STL file

Internal name for snappy

- Definition of all surfaces.
- Surfaces are used to specify refinement for any mesh cell intersecting it:
  - to specify refinement for any mesh cell inside/outside/near
  - to 'snap' the mesh boundary to the surface

snappyHexMeshDict: castellatedMeshControls

```
castellatedMeshControls
{
    features
    (
        file "regionSTL.eMesh";
        level 0;
    );
    //Surface based refinement
    refinementSurfaces
    {
        regionSTL.stl
        {
            // Surface-wise min and max refinement level
            level (0 0);
            regions
            {
                inlet   { level (0 0); }
                outlet  { level (0 0); }
                wall    { level (2 2); }
            }
        }
    }
}
```

- `features`: the file is created with the utility `surfaceFeatures`
- `refinementSurfaces`: region name generated in the geometry section
  - Local refinement is possible on each surface
  - `level (2 2)` indicates minimum and maximum refinement of the background mesh. Determined by `resolveFeatureAngle`
  - Optional: `gapRefinementLevel`
  - The names in `regions` are the final patches names

snappyHexMeshDict: snapControls

```
snapControls
{
    // Number of patch smoothing iterations before finding
    // correspondence to surface
    nSmoothPatch 3;

    // Relative distance for points to be attracted by
    // surface feature point or edge
    tolerance 1.0;

    // Number of mesh displacement relaxation iterations.
    nSolveIter 100;

    // Maximum number of snapping relaxation iterations.
    // Should stop before upon reaching a correct mesh.
    nRelaxIter 5;

    // Feature snapping
    // Number of feature edge snapping iterations.
    nFeatureSnapIter 5;

    ....
}
```

- `nSolveIter`: The higher the value the better the body fitted mesh. The recommended value is 30. If you are having problems with the mesh quality (related to the snapping step), try to increase this value to 100. Have in mind that this will increase the meshing time.
- `nRelaxIter`: Increase this value to improve the quality of the body fitted mesh.
- `nFeatureSnapIter`: Increase this value to improve the quality of the edge features.

snappyHexMeshDict: addLayersControls

```
addLayersControls
{
    // Global parameters
    relativeSizes false;
    expansionRatio 1.2;
    finalLayerThickness 0.05;
    minThickness 0.0005;
    nGrow 0;

    // Per final patch the layer information
    layers
    {
        wall
        {
            nSurfaceLayers 3;
        }
    }

    // Advanced settings
    // When not to extrude surface. 0 is flat surface, 90 is
    // when two faces are perpendicular
    featureAngle 80;

    ....
}
```

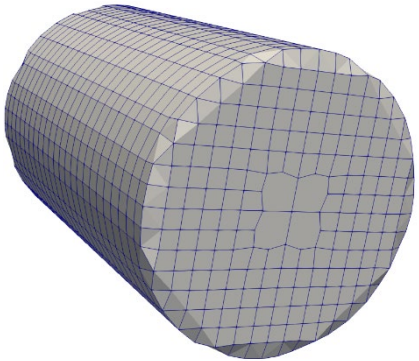
- `layers`: In this section we select the patches where we want to add the layers. We can add multiple patches (if they exist).
- `featureAngle`: Specification of feature angle above which layers are collapsed automatically.

snappyHexMeshDict: meshQualityControls

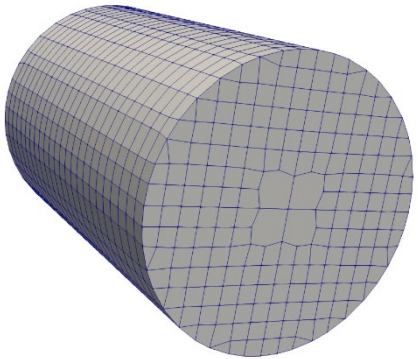
```
meshQualityControls
{
maxNonOrtho 75;
maxBoundarySkewness 20;
maxInternalSkewness 4;
maxConcave 80;
minVol 1E-13;
//minTetQuality 1e-15;
minTetQuality -1e+30;
minArea -1;
minTwist 0.02;
minDeterminant 0.001;
minFaceWeight 0.05;
minVolRatio 0.01;
minTriangleTwist -1;
minFlatness 0.5;
nSmoothScale 4;
errorReduction 0.75;
}
```

- During the mesh generation process, the mesh quality is continuously monitored.
- The mesher snappyHexMesh will try to generate a mesh using the mesh quality parameters defined by the user.
- If a mesh motion or topology change introduces a poor quality cell or face the motion or topology change is undone to revert the mesh back to a previously valid error free state.

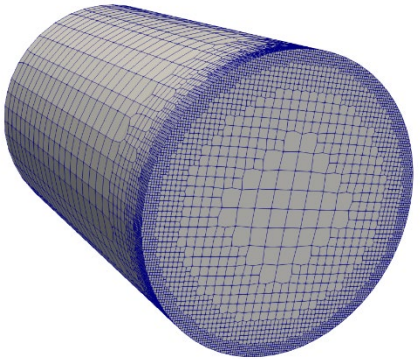
Effects of refinement parameters



- Edge refinement 0
- resolveFeatureAngle 110
- Surfaces refinement (1 1)



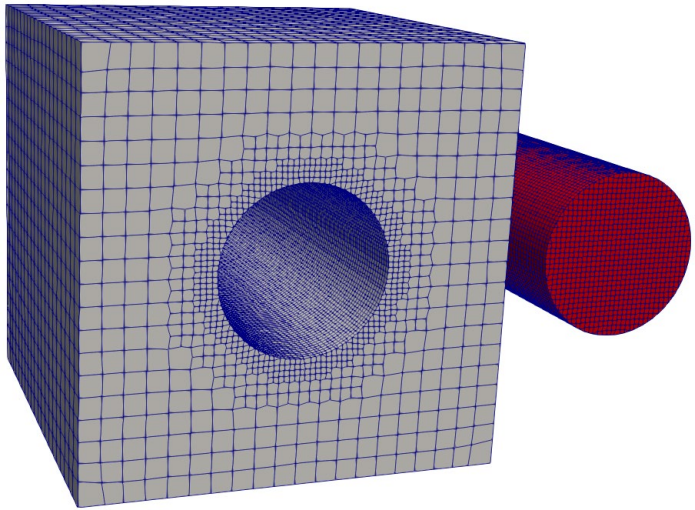
- Edge refinement 0
- resolveFeatureAngle 30
- Surfaces refinement (1 1)

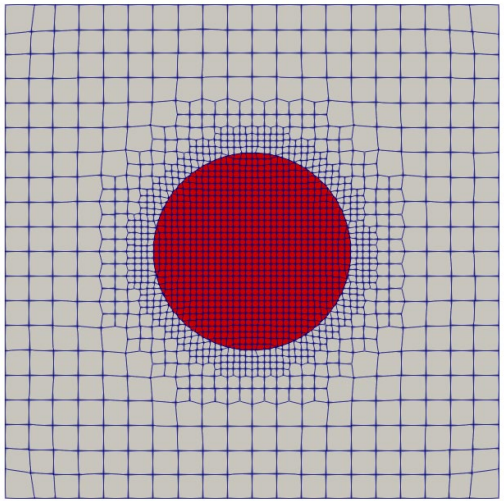


- Edge refinement 4
- resolveFeatureAngle 30
- Surfaces refinement (1 1)

Multi-region meshing

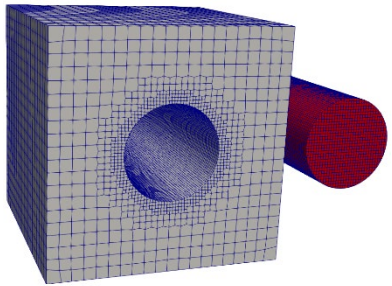
- `snappyHexMesh` is also capable of handling multi-region meshing
- Can mesh each region with different properties
- It's a clear advantage over `snappy` + `topoSet`





Multi-region meshing

```
geometry
{
    regionSTL.stl
    {
        type triSurfaceMesh;
        regions
        {
            inlet   { name inlet; }
            outlet  { name outlet; }
            wall    { name wall; }
        }
    }

    Hole.stl
    {
        type triSurfaceMesh;
        name Hole;
    }

    Box.stl
    {
        type triSurfaceMesh;
        name Box;
    }
};
```



Add each region independently

```
refinementSurfaces
{
    regionSTL.stl
    {
        level (1 1);
        regions
        {
            inlet  { level (1 1);}
            outlet { level (1 1);}
            wall   { level (1 1);}
        }
    }
    Hole
    {
        level (3 3);
        faceZone    Hole;
        cellZone    Hole;
        cellZoneInside inside;
        boundary    internal;
    }
    Box
    {
        level (1 1);
        faceZone    Box;
        cellZone    Box;
        cellZoneInside inside;
        boundary    internal;
    }
}
```

Parallel meshing

- `snappyHexMesh` can be executed in parallel
- Some extra setting is needed

```
# Extract main features from STL files
runApplication surfaceFeatures

# Base mesh
runApplication blockMesh
rm -rf 0
cp -r 0.orig 0

# Decompose mesh
runApplication decomposePar –copyZero

# Run snappyHexMesh
runParallel snappyHexMesh -parallel –overwrite

# Create final patches
runParallel createPatch -parallel -overwrite

# Check mesh quality
runParallel checkMesh -writeSets vtk –parallel

# Renumber mesh
runParallel renumberMesh -overwrite
```

- Extract geometry features
- Needs surfaceFeaturesDict

```
surfaces (“Box.stl");

// Identify a feature when angle between faces < includedAngle
includedAngle      150;

subsetFeatures
{
    // Keep nonManifold edges (edges with >2 connected faces)
    nonManifoldEdges      no;

    // Keep open edges (edges with 1 connected face)
    openEdges      yes;
}
```

Parallel meshing

- `snappyHexMesh` can be executed in parallel
- Some extra setting is needed

```
# Extract main features from STL files
runApplication surfaceFeatures

# Base mesh
runApplication blockMesh
rm -rf 0
cp -r 0.orig 0

# Decompose mesh
runApplication decomposePar –copyZero

# Run snappyHexMesh
runParallel snappyHexMesh -parallel –overwrite

# Create final patches
runParallel createPatch -parallel -overwrite

# Check mesh quality
runParallel checkMesh -writeSets vtk –parallel

# Renumber mesh
runParallel renumberMesh -overwrite
```

- Create background mesh
- Keep initial time step in a backup folder. It will be overwritten!

Parallel meshing

- `snappyHexMesh` can be executed in parallel
- Some extra setting is needed

```
# Extract main features from STL files
runApplication surfaceFeatures

# Base mesh
runApplication blockMesh
rm -rf 0
cp -r 0.orig 0

# Decompose mesh
runApplication decomposePar –copyZero

# Run snappyHexMesh
runParallel snappyHexMesh -parallel –overwrite

# Create final patches
runParallel createPatch -parallel -overwrite

# Check mesh quality
runParallel checkMesh -writeSets vtk –parallel

# Renumber mesh
runParallel renumberMesh -overwrite
```

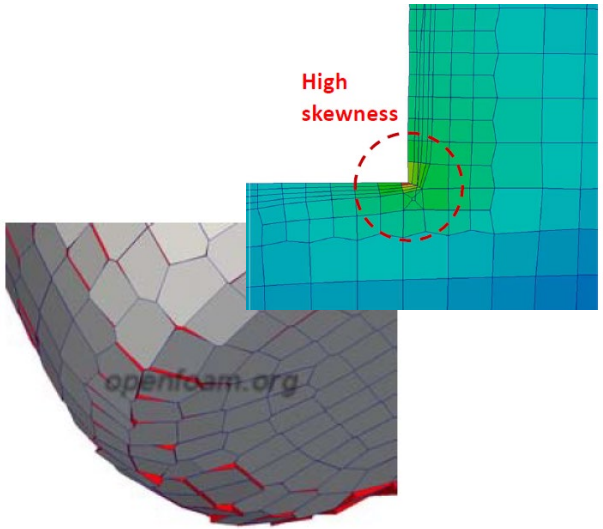
- Output boundaries from snappy are of type `patch`
- Set the correct type with `createPatch`

```
// Patches to create.
patches
(
    {
        name Walls;
        patchInfo
        {
            type            wall;
        }
        constructFrom patches;
        patches (snappy_Walls);
    }
);
```

Personal comments about snappy

- `snappyHexMesh` is a great and powerful tool for meshing complex geometries
  - It is included in the OpenFOAM suite
  - Robust: you will get a mesh
  - It is "automatic". You can start with a simple setup from the tutorials
  - It runs in parallel. Saves meshing time and memory limitations

- But, proceed with caution
  - The snapping process can be problematic
  - You need some effort to produce good boundary layers
  - Always check your mesh, and remesh if it is necessary



Concluding remarks

- This lecture was about geometry and meshing
- Solid modeling and meshing is an important part of a simulation
- Think about your problem first. Try to reduce the complexity if possible
- Be patient. You may need to iterate
- There are several open-source tools available
- GIGO is almost always valid
- It is not a waste of time. It's an investment

IAEA
International Atomic Energy Agency

Multi-physics modeling and simulation of nuclear reactors using OpenFOAM

30 Aug 2022 – 6 October 2022 (every Tuesday & Thursday)

Contact: ONCORE@iaea.org

Thank you!

Contact: ONCORE@iaea.org

Course Enrollment : Multi-physics modelling and simulation of nuclear reactors using OpenFOAM

ONCORE: Open-source Nuclear Codes for Reactor Analysis