Meshing with cfMesh
Training session

Franjo Juretić, Creative Fields, Ltd.

Creative Fields

Ann Arbor, June 2015

Outline

- What is cfMesh?
- Quick examples.
- Technology and best practices.
- Input geometry.
- Definition of salient features in the model.
- Hands-on session.

Background: What is cfMesh?

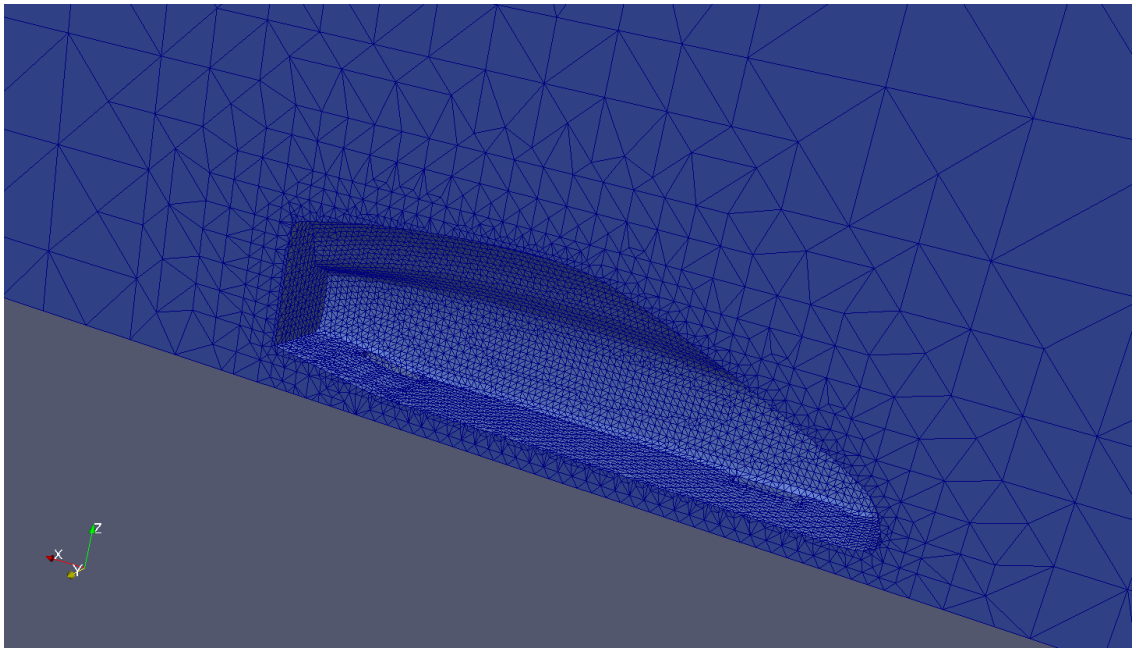
- cfMesh is a library for polyhedral mesh generation.
- The library consists of many meshing algorithms which can be reused to generate meshing workflows (meshers).
- It currently generates 2D and 3D cartesian meshes, 3D polyhedral, and 3D tetrahedral meshes.
- Meshing is a dynamic process in terms of memory allocation, and this adds extra complexity to the problem. The data containers available in cfMesh are implemented to reduce memory usage and improve performance.
- SMP and MPI parallelisation for performance and the ability to generate large meshes. SMP parallelisation is used on a single node. MPI is needed for large meshes, only.
- Currently generates meshes for manifold domains, only.

Meshing workflows (meshers)

- Simple syntax - applicable to all meshing workflows. Focus on minimising user input. Most tasks are automatic.
- All meshing workflows are based on the inside-out meshing approach.
- Mesh template is generated according to user's setting and adapted onto the input geometry.
- Require an input surface triangulation defining a domain and a meshDict file located in the system directory of a case. The template is generated automatically within the meshing process.
- Available workflows:
  - 3D Cartesian
  - Polyhedral
  - Tetrahedral
  - 2D Cartesian

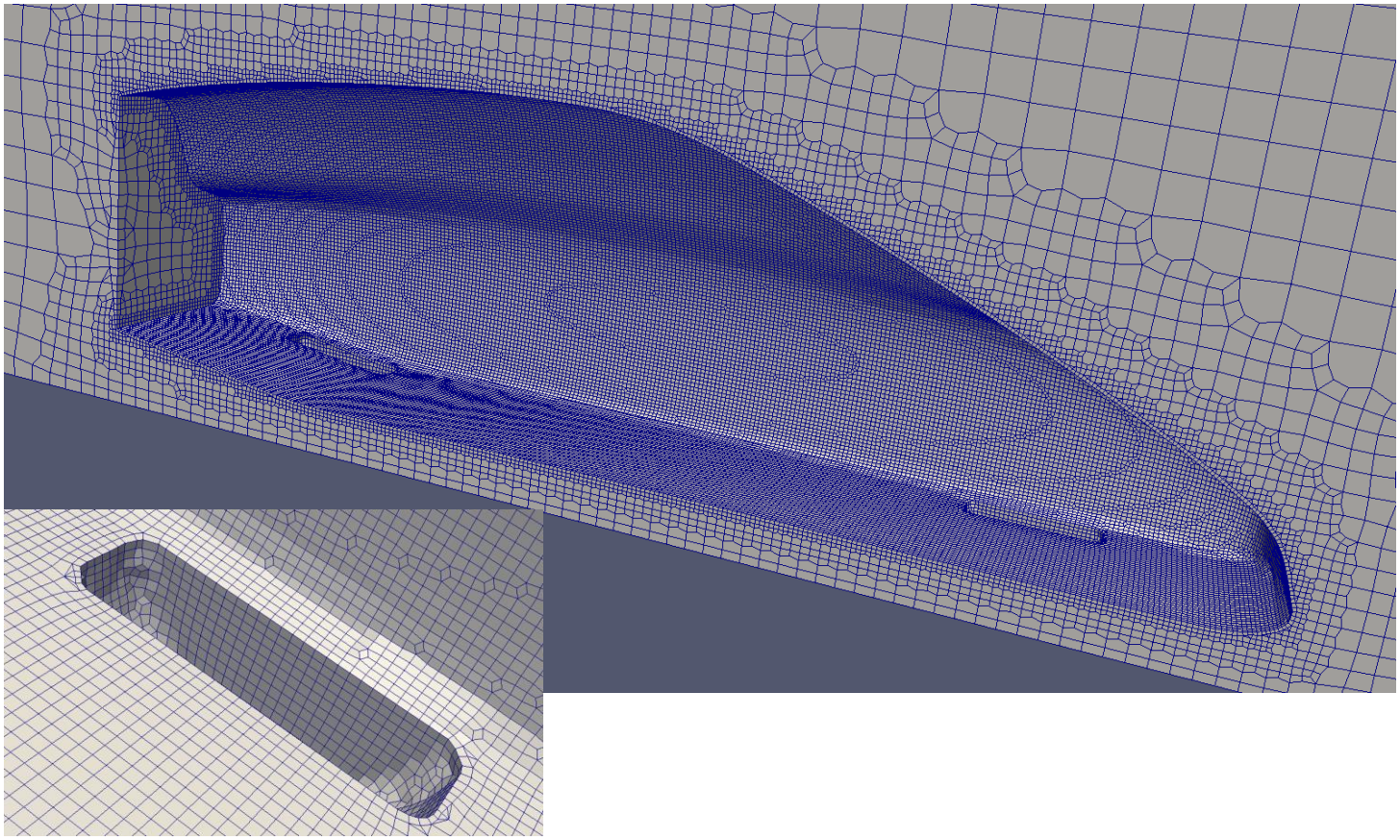
Examples: cartesianMesh - Asmo body

- Example located in: introductoryExamples/cartesianMesh/asmo.
- Surface triangulation: geom.stl.
- cfMesh handles edges at the boundary of a patch as feature edges.
- Steps:
  - *cartesianMesh* : starts the 3D Cartesian workflow and generates the mesh.



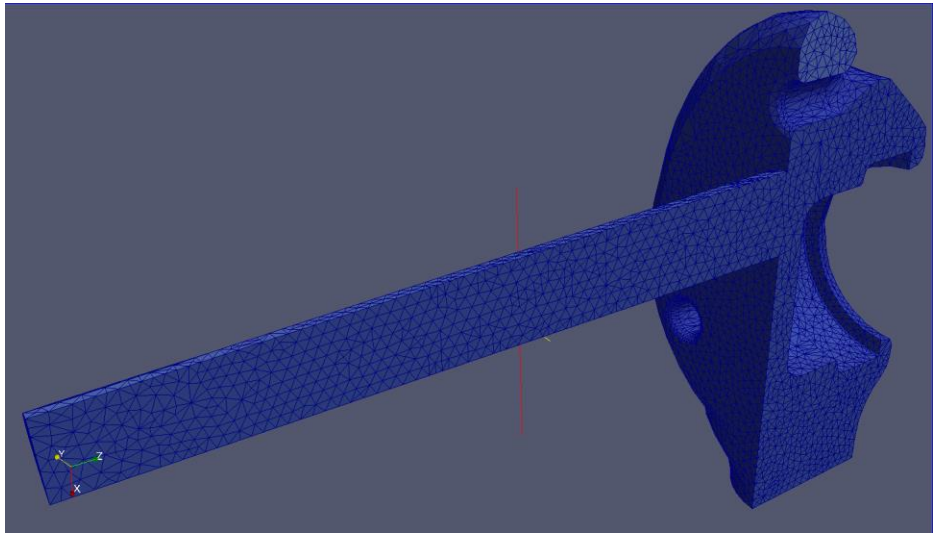
Examples: cartesianMesh – Asmo body

- Check the volume mesh in paraview.

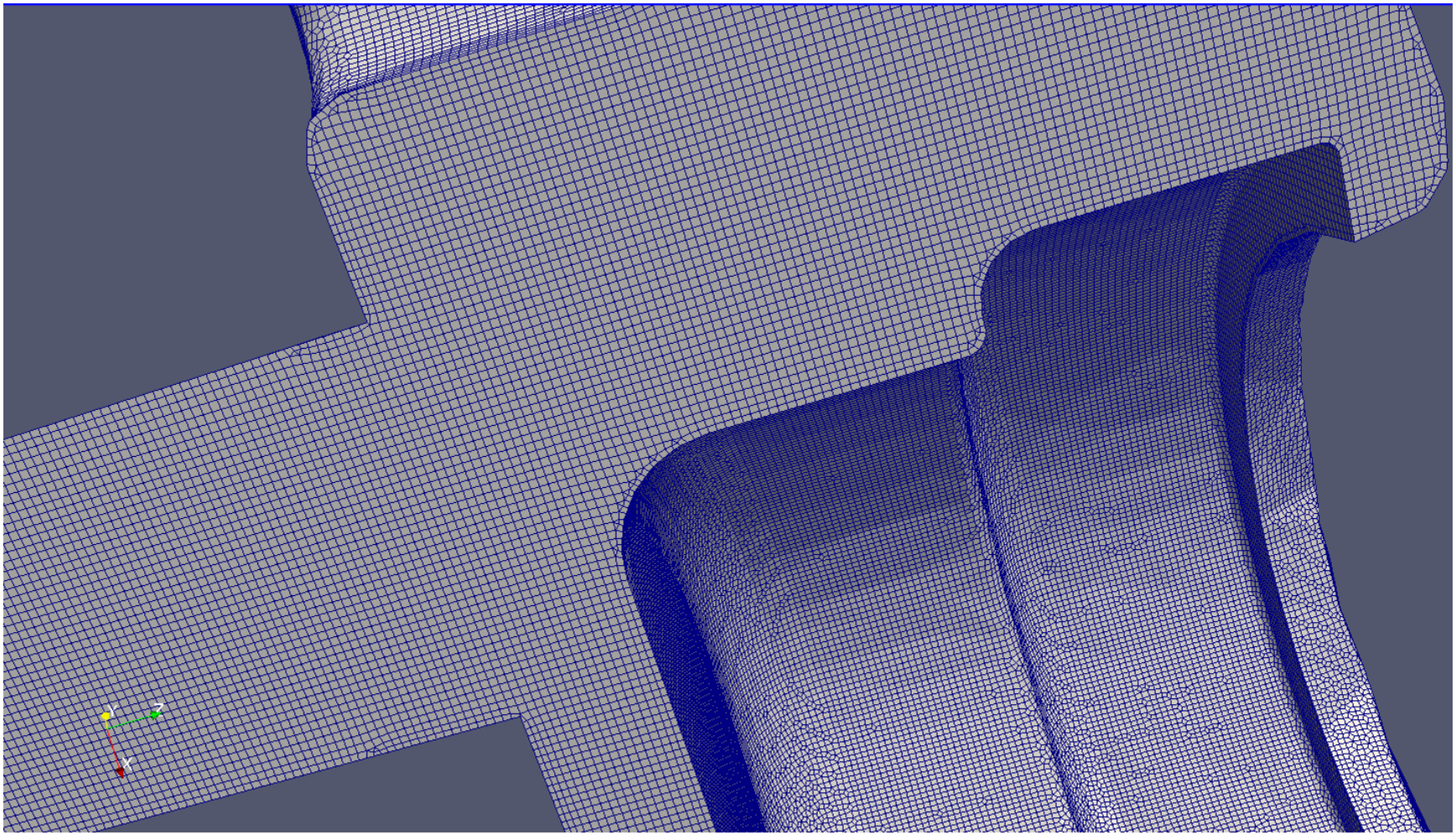


Examples: cartesianMesh - saw

- Demonstrate MPI parallelisation.
- Example located in: introductoryExamples/cartesianMesh/saw.
- Surface mesh: saw1.stl.
- Steps:
  - *preparePar* : generates processor* directories needed for MPI parallelisation.
  - *mpirun –np 4 cartesianMesh –parallel* : generates the mesh on four processor nodes.
  - *reconstructParMesh –zeroTime* : reconstruct the mesh.

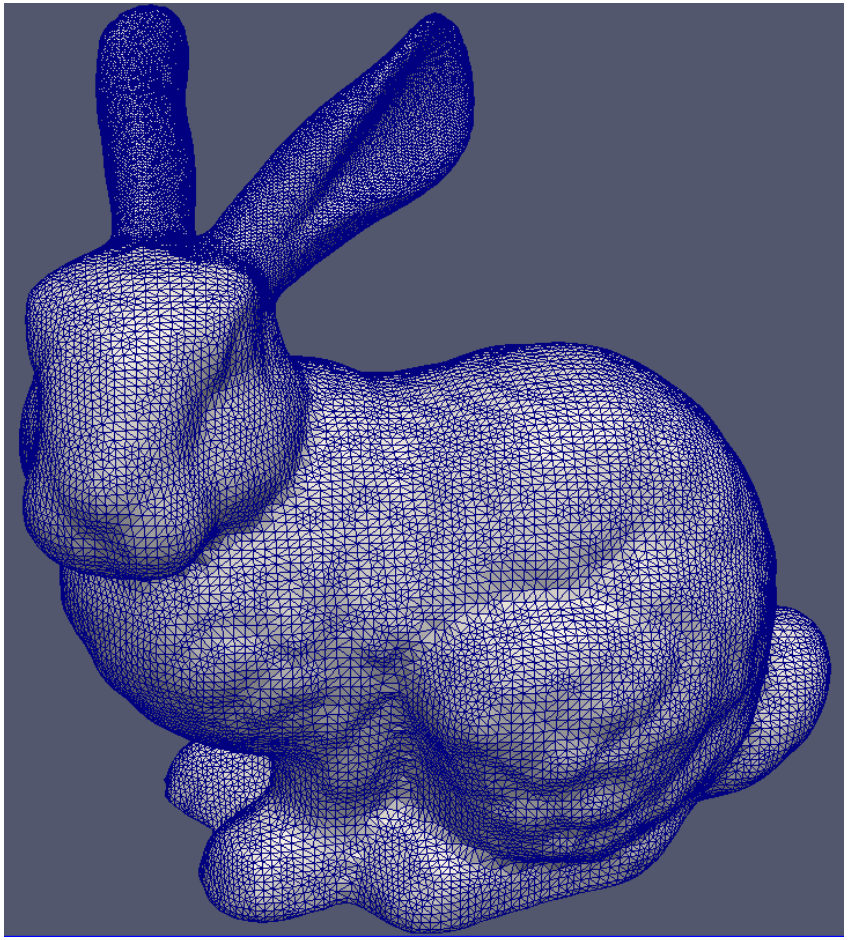


Examples: cartesianMesh - saw

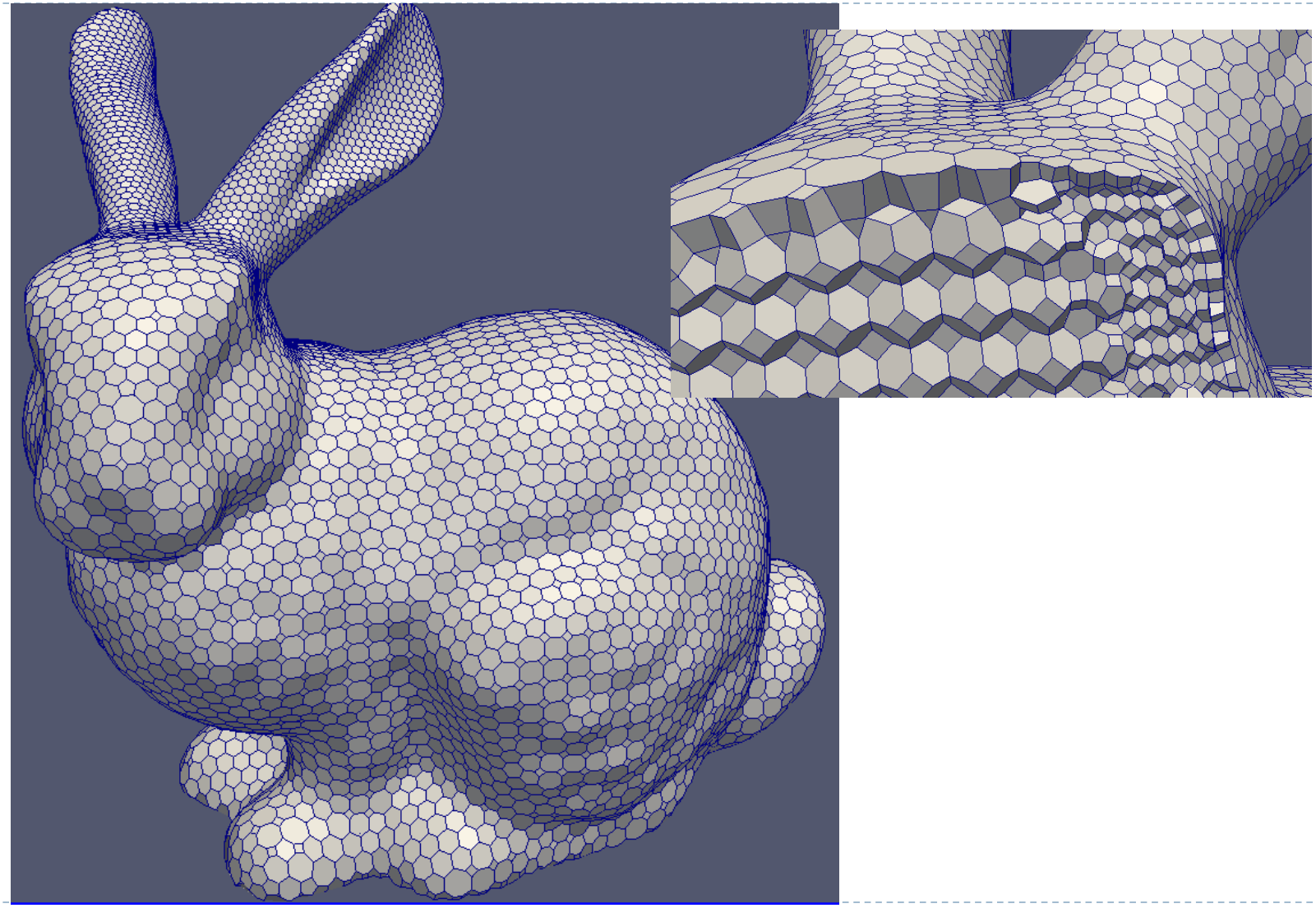


Examples: pMesh - bunny

- Example located in: introductoryExamples/pMesh/bunny.
- Surface triangulation: bunnyWrapped.stl.
- Generates a mesh consisting of arbitrary polyhedra inside the domain.
- Steps:
  - *pMesh* : starts the 3D polyhedral workflow and generates the mesh.

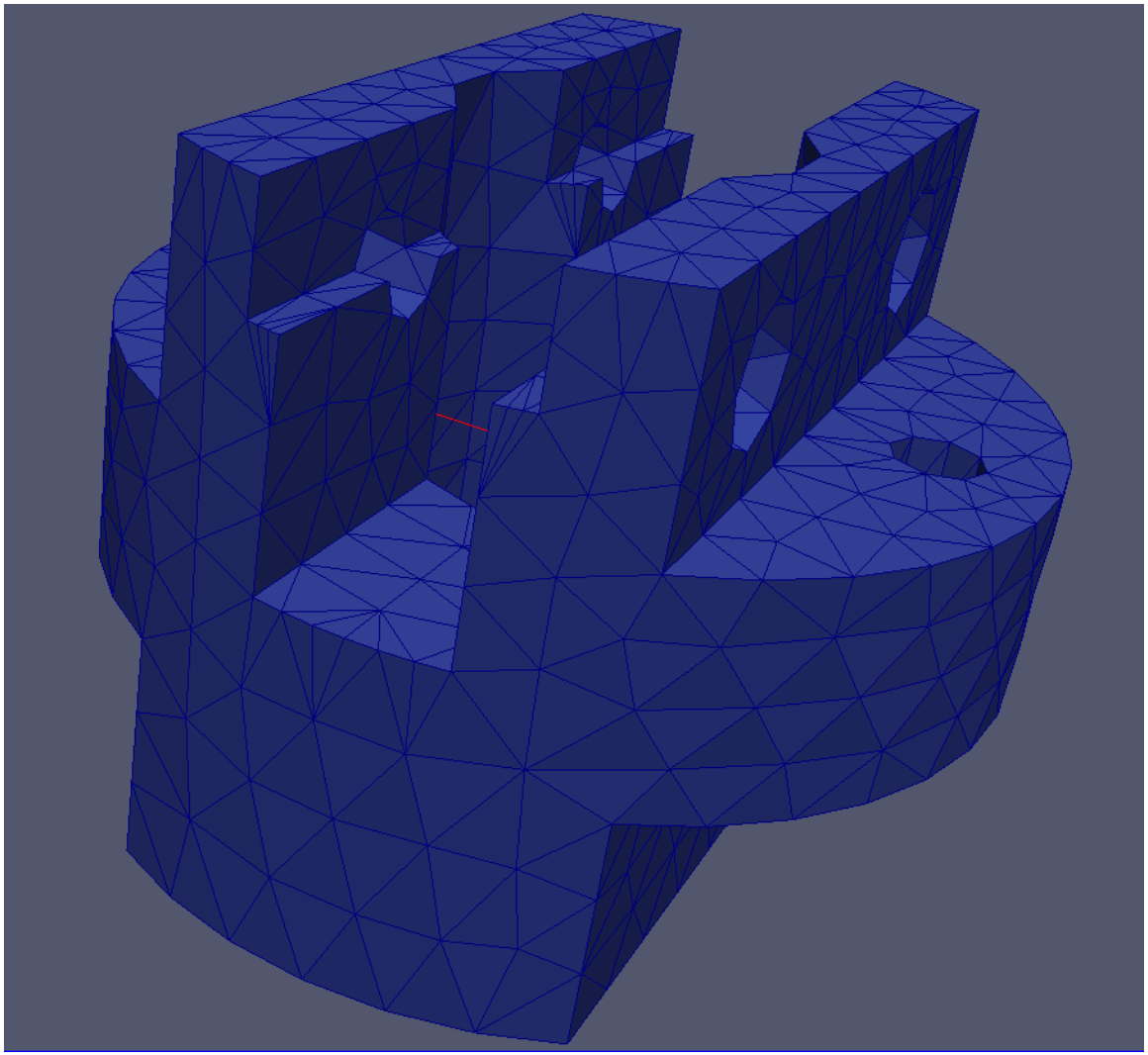


Examples: pMesh - bunny

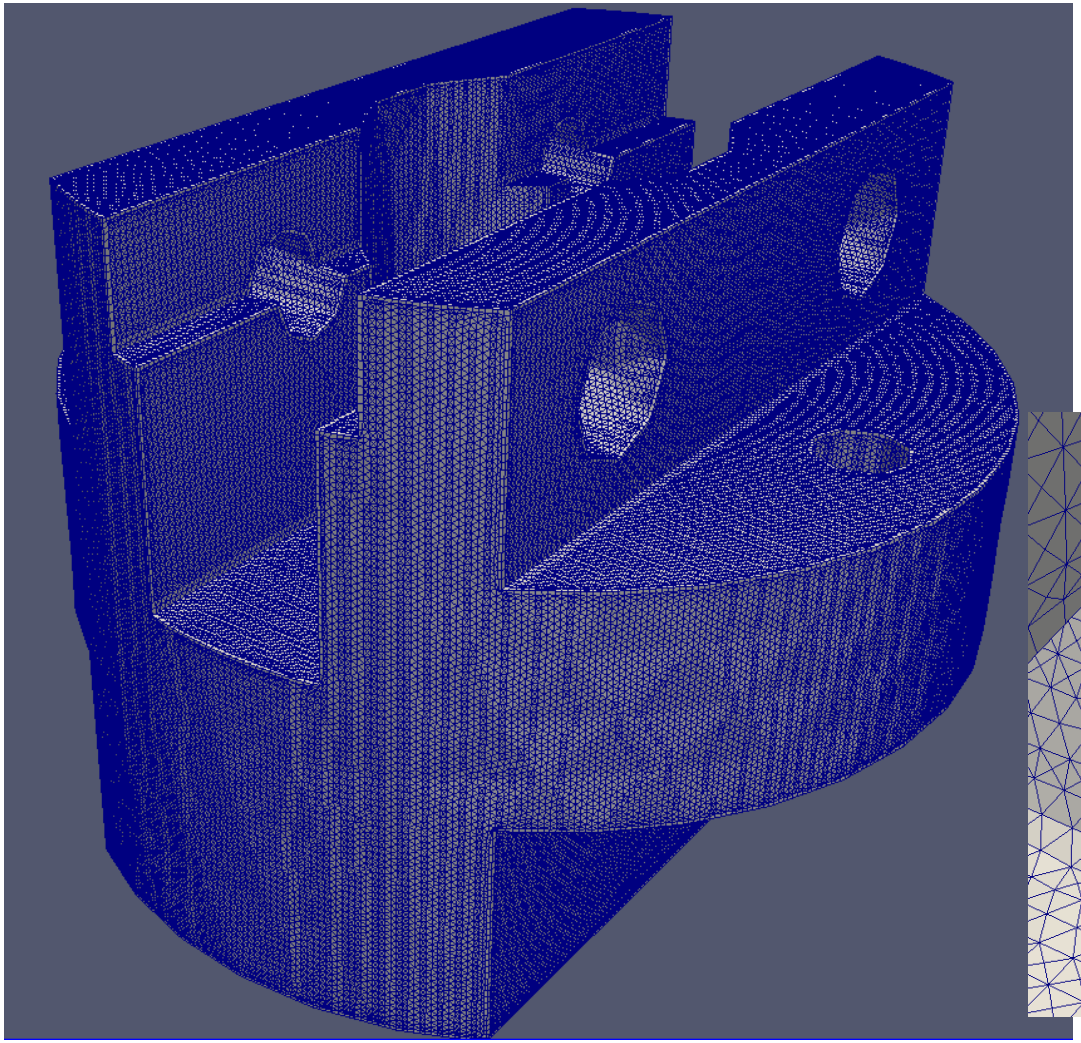


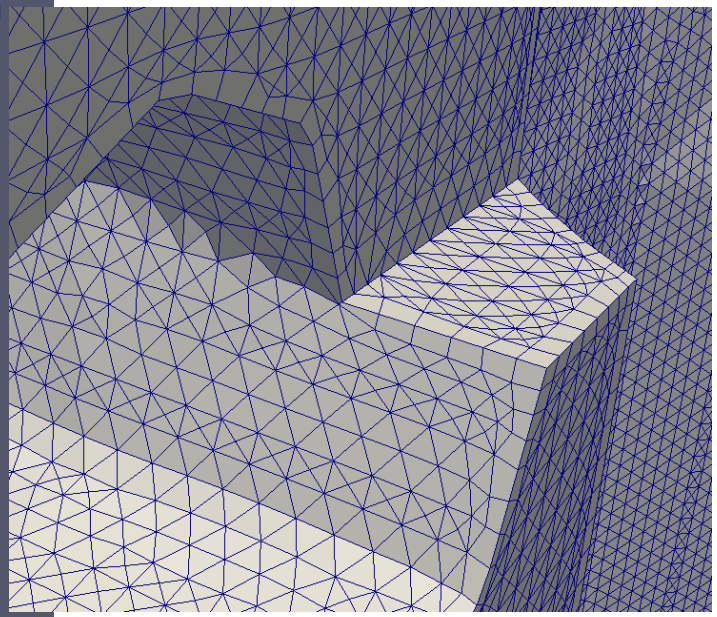
Examples: tetMesh - socket

- Example located in: introductoryExamples /tetMesh/socket.
- Surface mesh: socket.fms.
- fms format supports subsets and allows for definition of feature edges in the geometry.
- Steps:
  - *tetMesh* : starts the tetrahedral meshing workflow (mesher).



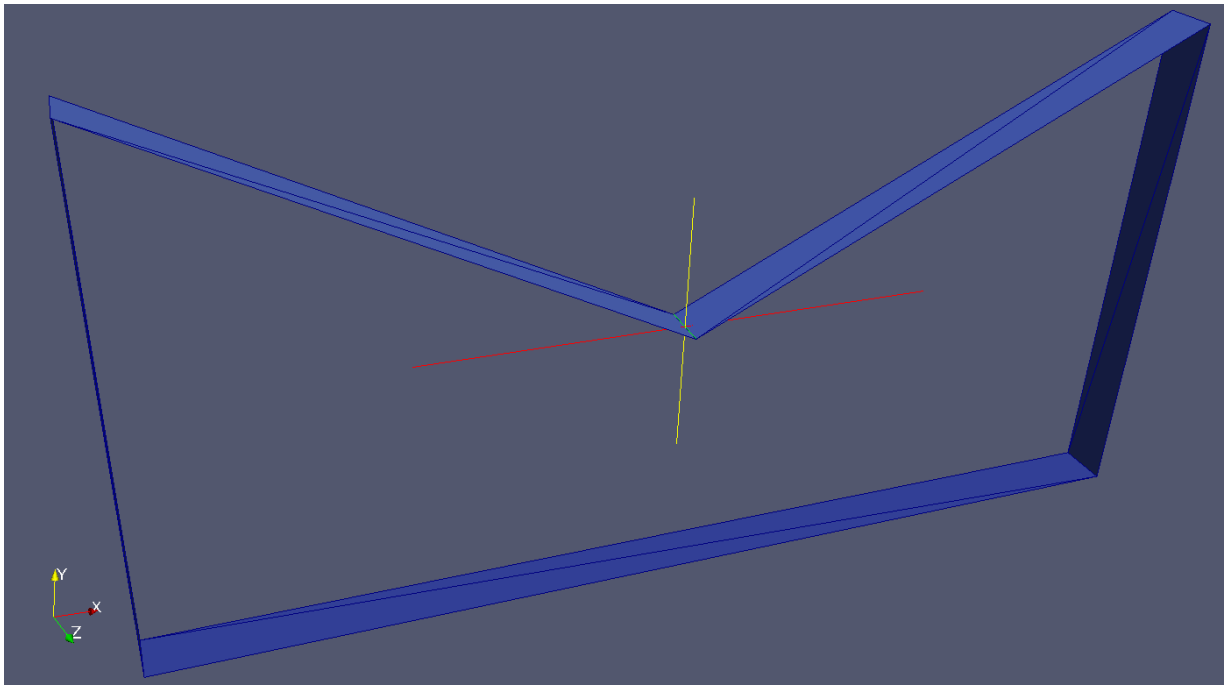
Examples: tetMesh - socket





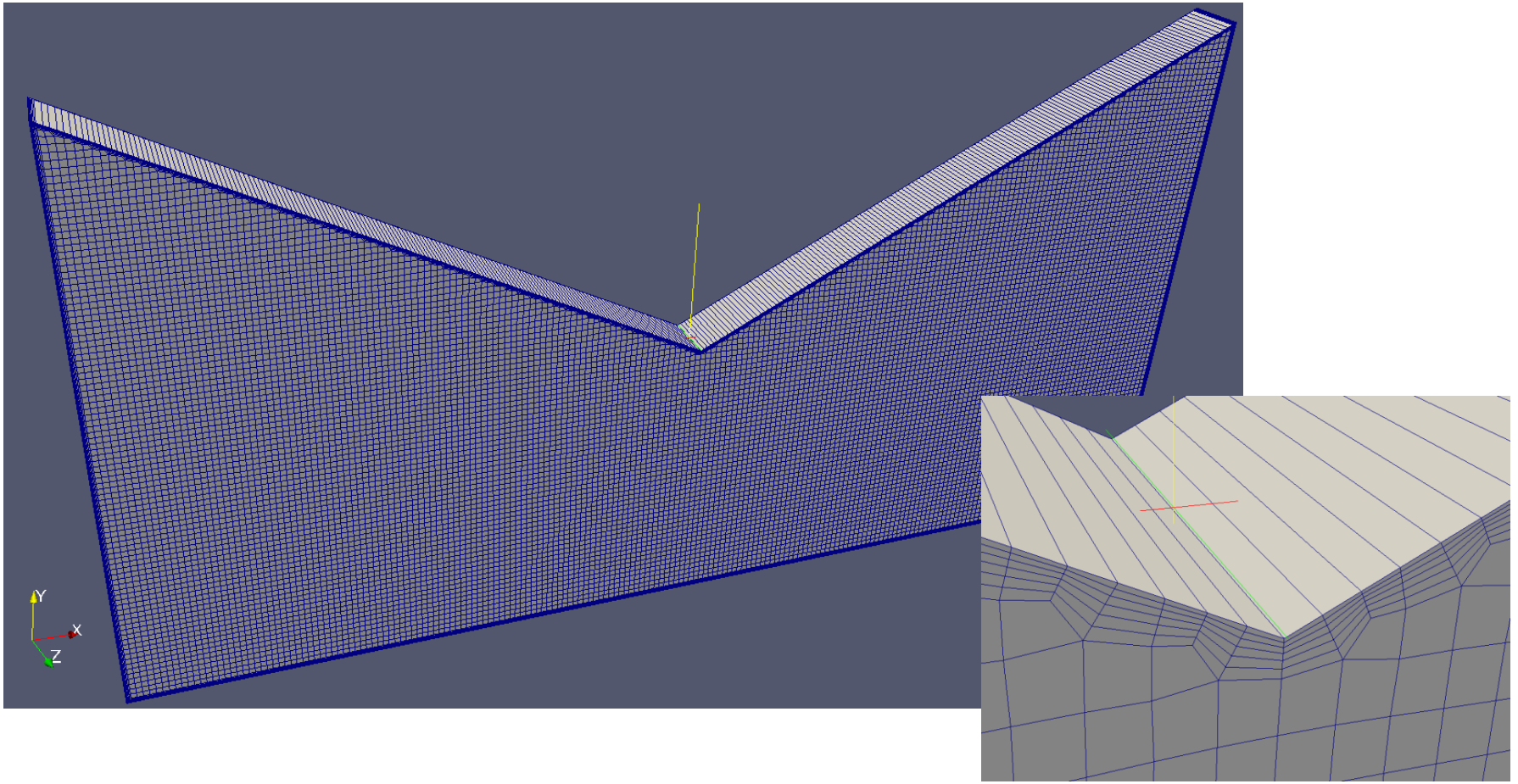
Examples: cartesian2DMesh - hat

- Example located in: introductoryExamples/cartesian2DMesh/hat.
- Input geometry is a ribbon of triangles in the x-y coordinates.
- Surface mesh: geom.fms.
- Steps:
  - *cartesian2DMesh* : starts the meshing workflow (mesher).



Examples: cartesian2DMesh - hat

- Mesh has one layer of cells in the z direction.



Technology in cfMesh - Overview

- Simple syntax – focus on minimising user input. Most tasks are automatic.
- Generates a manifold mesh inside a closed domain.
- Meshing algorithms – based on inside-out approach, which do not require watertight input geometry.
- Implemented lists and graphs resizable without re-allocating memory, to improve performance and reduce memory usage.
- Mesh modifiers:
  - Basic - (adding/removing of cells, etc.).
  - Advanced - (boundary layers, capturing of feature edges, etc.).
  - Re-usable – used to build other modifiers. Less prone to bugs, and easier to find and resolve problems.
  - Most algorithms are SMP and MPI parallelised.

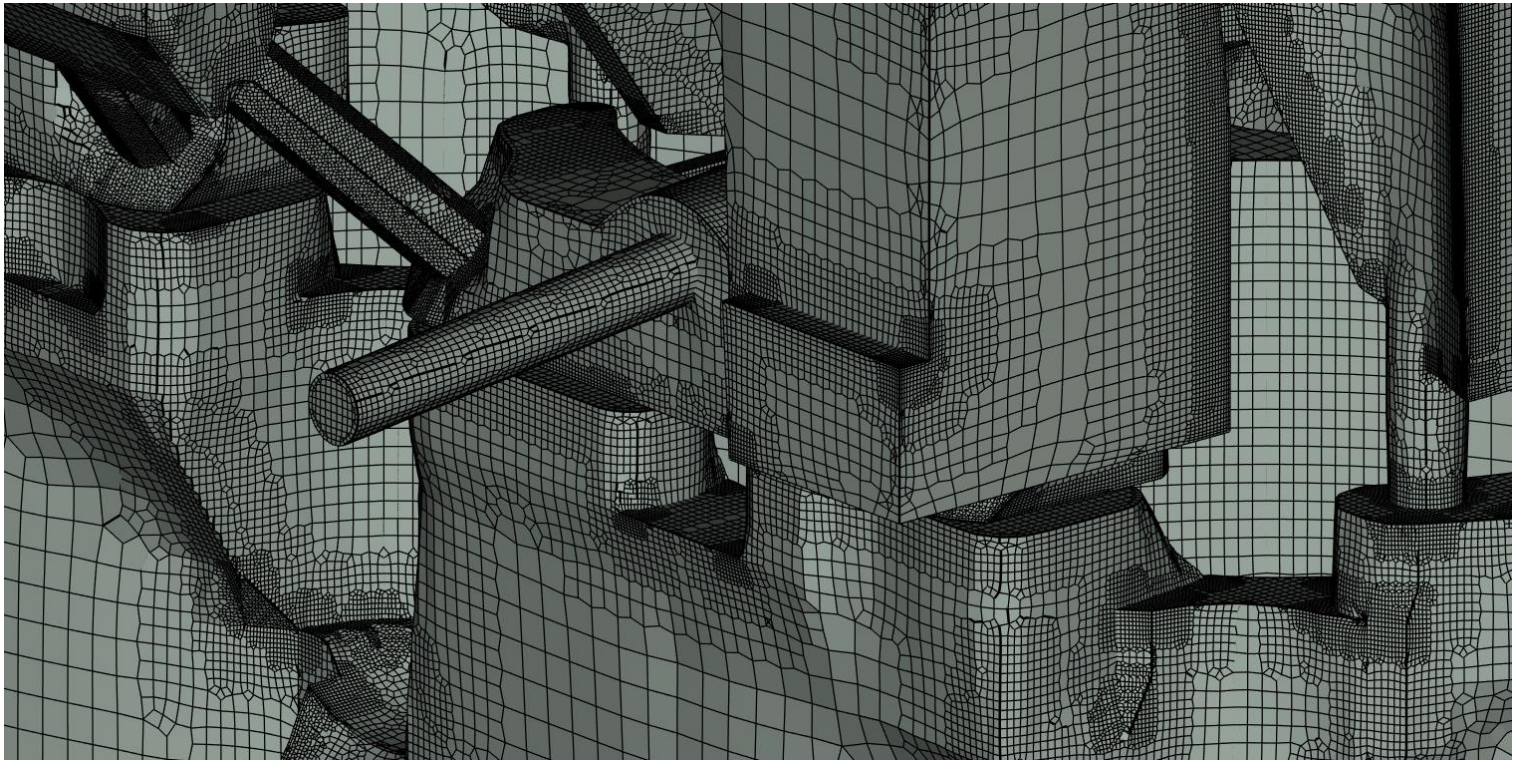
Technology in cfMesh – Meshing algorithms

- Inside-out approach – generate a mesh template based on input geometry and user's input (surface mesh, and meshDict).

```
boundaryCellSize    0.003;
keepCellsIntersectingBoundary   1;
maxCellSize 0.006;
minCellSize 0.0005;
removeGluedMesh 0;
surfaceFile "surfaceMeshes/coolantRef1.fms";

removeCellsIntersectingPatches
{

removeCells
{
keepCells   0;
}

removeGlued1
{
keepCells   0;
}
}
```

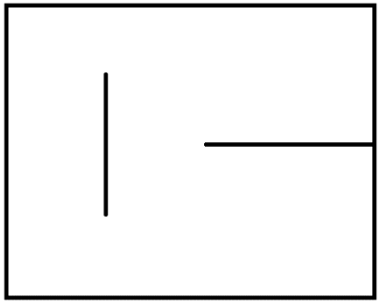
Technology in cfMesh – Meshing algorithms

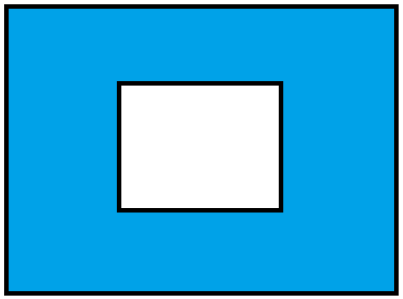
- Available templates: Cartesian 2D and 3D, polyhedral, tetrahedral.
- The final mesh is generated by modifying the mesh template to fit the input geometry. The surface of the template is projected onto the input geometry. The process is not very sensitive to input-surface quality.

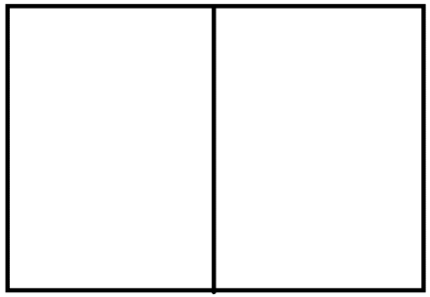


Input geometry: Domain definition

- cfMesh generates meshes within a manifold.





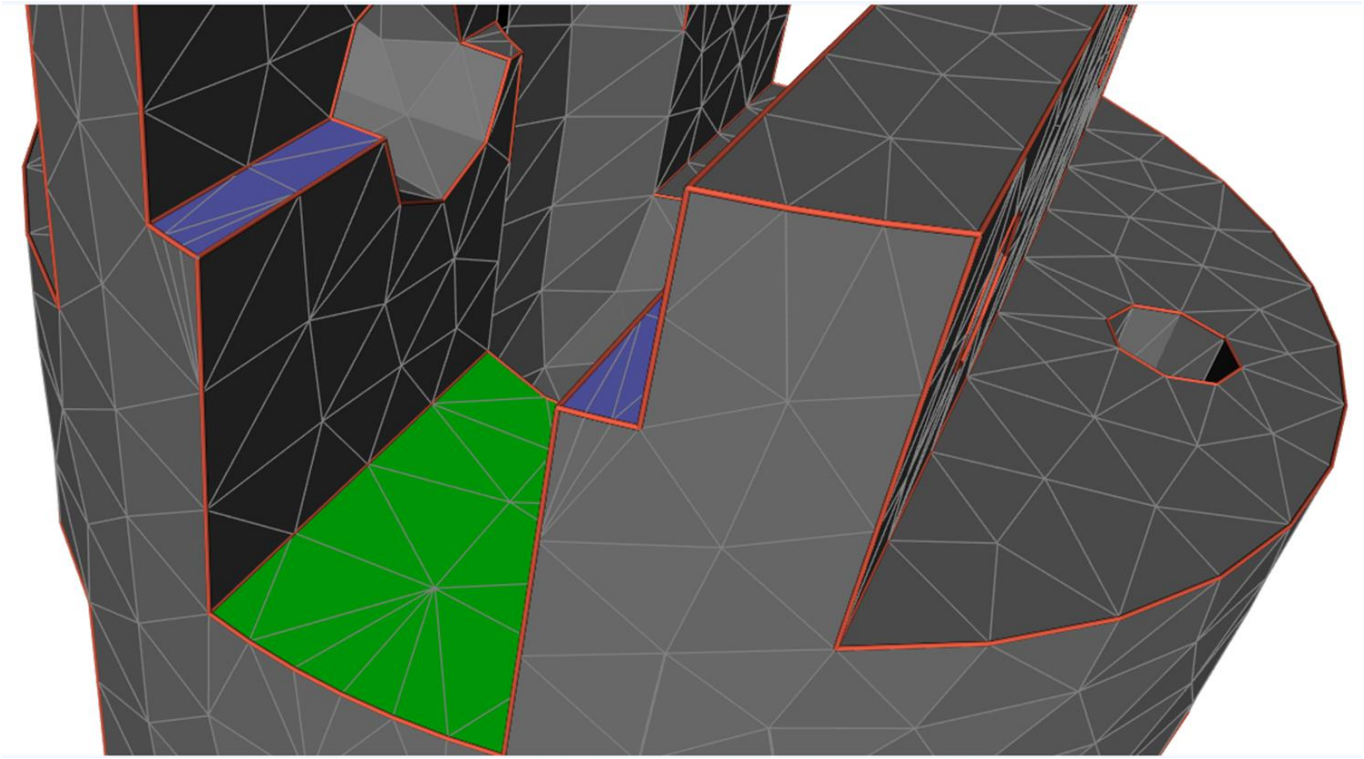


Valid configurations

Currently not supported

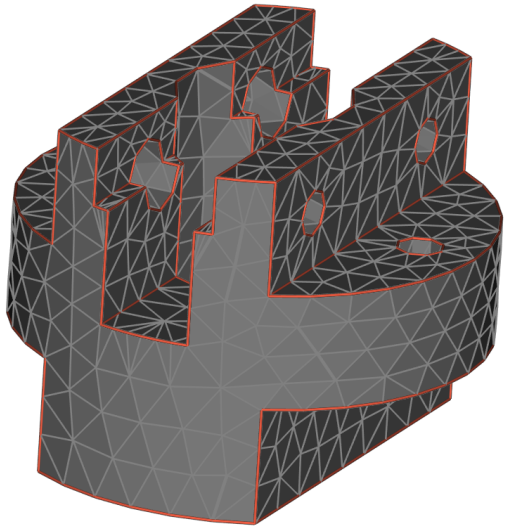
Input geometry: Patches and subsets

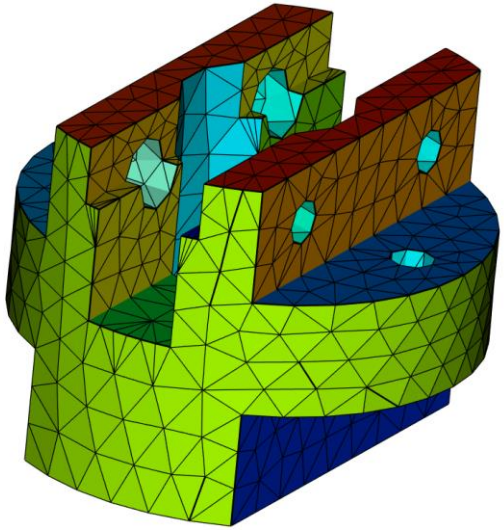
- Patches (green) are entities transferred on the volume mesh in the meshing process.
- Subsets (blue) are entities that are only used for definition of settings and are not transferred onto the volume mesh.



Input geometry: Salient features

- Feature edges which shall be preserved in the meshing process are selected by the user.
- *surfaceFeatureEdges <inputSurface> <outputSurface> -angle <val>*
- There exist two available modes:
  - Create edges – this is achieved by exporting to fms format. Patches remain unchanged.
  - Create patches – exporting to stl or ftr, other formats do not support patch information. ftr is suggested because it supports patch types. stl supports patches, but it does not support patch types.





Technology in cfMesh: Parallelisation

- SMP parallelisation:
  - By default, most modifiers use all available cores for the meshing job.
  - Uses openMP available with most modern C++ compilers.
  - OMP_NUM_THREADS is a system variable which can be set to limit the number of threads.
  - Requires little modification of the code base.
- MPI parallelisation:
  - Available for cartesianMesh.
  - It is intended for generation of large meshes which do not fit into the memory of a single computer.
  - It is difficult to maintain load balancing in when the mesh changes in the process.

Technology in cfMesh: Memory usage

- The slowest, and the most “dangerous” operation in every program is memory allocation!
- Meshing is dynamic in terms of memory resources, and memory allocation play an important role for its performance.
- Common problems:
  - Appending of elements : data containers in cfMesh do not re-allocate memory and copy all data every time a new element is added.
  - Adding vertices, faces and cells: solved by transferring pointers to the existing data instead of copying data.
- It may cause the problem that the computer requires more memory to generate a mesh than it needs to run a solver.
- The problem is solved by using data containers which reduce the need for memory allocation, in order to improve performance and reduce memory usage.

Hands-on session: preserving of topology

- Example located in: trainingExamples/socket.
- Surface mesh: socket.stl.
- Purpose: learn the basic options to preserve mesh parts.
- Settings in meshDict:
  - Maximum cell size: 5 m.
- Steps:
  - Generate mesh: *cartesianMesh*

Hands-on session: preserving of topology

- Example located in: trainingExamples/socket.
- Surface mesh: socket.stl.
- Purpose: learn the basic options to preserve mesh parts.
- Settings in meshDict:
  - Maximum cell size: 5 m.
- Steps:
  - Generate mesh: *cartesianMesh*

```
maxCellSize 5;

surfaceFile "socket.stl";
```

Hands-on session: preserving of topology

- Example located in: trainingExamples/socket.
- Surface mesh: socket.stl.
- Solution 1: Refine cells that intersect the boundary.
- Settings in meshDict:
  - Maximum cell size: 5 m.
  - Boundary cell size: 2.5 m.
- Steps:
  - Modify meshDict: use boundaryCellSize option
  - Generate mesh: *cartesianMesh*

Hands-on session: preserving of topology

- Example located in: trainingExamples/socket.
- Surface mesh: socket.stl.
- Solution 1: Refine cells intersecting the boundary.
- Settings in meshDict:
  - Maximum cell size: 5 m.
  - Boundary cell size: 2.5 m.
- Steps:
  - Modify meshDict: use boundaryCellSize option
  - Generate mesh: *cartesianMesh*

```
boundaryCellSize    2.5;

maxCellSize 5;

surfaceFile "socket.stl";
```

Hands-on session: preserving of topology

- Example located in: trainingExamples/socket.
- Surface mesh: socket.stl.
- Solution 2: Use octree cells intersecting the surface in the template. Keeps the number of cells low, at the expense of increased meshing time due to mesh smoothing.
- Settings in meshDict:
  - Maximum cell size: 5 m.
  - Use cells intersecting the boundary.
- Steps:
  - Modify meshDict: use keepCellsIntersectingBoundary option
  - Generate mesh: *cartesianMesh*

Hands-on session: preserving of topology

- Example located in: trainingExamples/socket.
- Surface mesh: socket.stl.
- Solution 2: Use octree cells which intersect the surface in the template. Keeps the number of cells low, at the expense of increased meshing time due to mesh smoothing.
- Settings in meshDict:
  - Maximum cell size: 5 m.
  - Use cells intersecting the boundary
- Steps:
  - Modify meshDict: use keepCellsIntersectingBoundary option
  - Generate mesh: *cartesianMesh*

```
maxCellSize 5;

surfaceFile "socket.stl";

keepCellsIntersectingBoundary 1;
```

Hands-on session: preserving features

- Example located in: trainingExamples/socket1.
- Surface mesh: socket.stl.
- Purpose: learn possible ways to capture feature edges in the generated mesh.
- Steps:
  - Generate patches: *surfaceFeatureEdges socket.stl socket1.stl –angle 60*
  - Settings in meshDict:
    - Maximum cell size: 3 m.
    - Boundary cell size: 1.5 m.
  - Generate mesh: *cartesianMesh*

Hands-on session: preserving features

- Example located in: trainingExamples/socket1.
- Surface mesh: socket.stl.
- Purpose: learn possible ways to capture feature edges in the generated mesh.
- Steps:
  - Generate patches: *surfaceFeatureEdges socket.stl socket1.stl –angle 60*
  - Settings in meshDict:
    - Maximum cell size: 3 m.
    - Boundary cell size: 1.5 m.
  - Generate mesh: *cartesianMesh*

```
surfaceFile "socket1.stl";

boundaryCellSize    1.5;
maxCellSize 3;
```

Hands-on session: preserving features

- Example located in: trainingExamples/socket1.
- Surface mesh: socket.stl.
- Purpose: usage of fms surface format. It preserves patches defined on the input surface.
- Steps:
  - Generate patches: *surfaceFeatureEdges socket.stl socket1.fms –angle 60*
  - Settings in meshDict:
    - Maximum cell size: 3 m.
    - Boundary cell size: 1.5 m.
  - Generate mesh: *cartesianMesh*

Hands-on session: preserving features

- Example located in: trainingExamples/socket1.
- Surface mesh: socket.stl.
- Purpose: usage of fms surface format. It preserves patches defined on the input surface.
- Steps:
  - Generate patches: *surfaceFeatureEdges socket.stl socket1.fms –angle 60*
  - Settings in meshDict:
    - Maximum cell size: 3 m.
    - Boundary cell size: 1.5 m.
  - Generate mesh: *cartesianMesh*

```
surfaceFile     "socket1.fms";

boundaryCellSize    1.5;
maxCellSize 3;
```

Hands-on session: preserving features

- Example located in: trainingExamples/socket1.
- Surface mesh: socket.stl.
- Purpose: usage of ftr and renameBoundary option.
- Steps:
  - Generate patches: *surfaceFeatureEdges socket.stl socket1.ftr –angle 60*
  - Settings in meshDict:
    - Maximum cell size: 3 m.
    - Boundary cell size: 1.5 m.
    - Rename all patches to meshSurface of type wall.
  - Generate mesh: *cartesianMesh*

Hands-on session: preserving features

- Example located in: trainingExamples/socket1.
- Surface mesh: socket.stl.
- Purpose: usage of ftr and renameBoundary option.
- Steps:
  - Generate patches: *surfaceFeatureEdges socket.stl socket1.ftr –angle 60*
  - Settings in meshDict:
    - Maximum cell size: 3 m.
    - Boundary cell size: 1.5 m.
    - Rename all patches to meshSurface of type wall.
  - Generate mesh: *cartesianMesh*

```
surfaceFile     "socket1.ftr";

boundaryCellSize    1.5;
maxCellSize 3;

renameBoundary
{
   defaultName   "meshSurface";
   defaultType   "wall";
}
```

Hands-on session: boundary layers

- Example located in: trainingExamples/sBend.
- Surface mesh: geom.stl.
- Purpose: usage of boundary layers in cfMesh.
- Settings in meshDict:
  - Maximum cell size: 0.05 m.
  - innerWall: set local cell size to 0.025 m.
  - Global number of boundary layers is 1.
  - 3 layers at innerWall and outerWalls with thickness ratio 1.2.
  - innerWall: set maximum layer thickness 0.01 m.
- Steps:
  - Generate mesh: *cartesianMesh*

Hands-on session: boundary layers

- Example located in: trainingExamples/sBend.
- Surface mesh: geom.stl.
- Purpose: usage of boundary layers in cfMesh.
- Settings in meshDict:
  - Maximum cell size: 0.05 m.
  - innerWall: set local cell size to 0.025 m.
  - Global number of boundary layers is 1.
  - 3 layers at innerWall and outerWalls with thickness ratio 1.2.
  - innerWall: set maximum layer thickness 0.01 m.
- Steps:
  - Generate mesh: *cartesianMesh*

```
maxCellSize 0.05;

surfaceFile "geom.stl";

boundaryLayers
{
    patchBoundaryLayers
    {
        innerWall
        {
            maxFirstLayerThickness  0.01;
            nLayers 3;
            thicknessRatio  1.2;
        }

        outerWall
        {
            nLayers     3;
            thicknessRatio      1.2;
        }
    }
}

localRefinement
{
    innerWall
    {
        cellSize    0.025;
    }
}
```

Hands-on session: boundary layer optimisation

- Example located in: trainingExamples/sBend.
- Surface mesh: geom.stl.
- Purpose: usage of boundary layers in cfMesh.
- Settings in meshDict:
  - Maximum cell size: 0.05 m.
  - innerWall: set local cell size to 0.025 m.
  - Global number of boundary layers is 1.
  - 3 layers at innerWall and outerWalls with thickness ratio 1.2.
  - innerWall: set maximum layer thickness 0.01 m.
  - Activate optimisation of boundary layers.
- Steps:
  - Generate mesh: *cartesianMesh*

Hands-on session: boundary layer optimisation

- Example located in: trainingExamples/sBend.
- Surface mesh: geom.stl.
- Purpose: usage of boundary layers in cfMesh.
- Settings in meshDict:
  - Maximum cell size: 0.05 m.
  - innerWall: set local cell size to 0.025 m.
  - Global number of boundary layers is 1.
  - 3 layers at innerWall and outerWalls with thickness ratio 1.2.
  - innerWall: set maximum layer thickness 0.01 m.
  - Activate optimisation of boundary layers.
- Steps:
  - Generate mesh: *cartesianMesh*

```
maxCellSize 0.05;

surfaceFile "geom.stl";

boundaryLayers
{
    untangleLayers 0;

    optimiseLayer 1;

    optimisationParameters
    {
        nSmoothNormals 5;
        relThicknessTol 0.1;
        featureSizeFactor 0.3;
        reCalculateNormals 1;
        maxNumIterations 5;
    }

    patchBoundaryLayers
    {
        innerWall
        {
            maxFirstLayerThickness  0.01;
            nLayers 3;
            thicknessRatio  1.2;
        }

        outerWall
        {
            nLayers     3;
            thicknessRatio      1.2;
        }
    }
}
```

Hands-on session: anisotropic meshing

- Example located in: trainingExamples/ship.
- Surface mesh: shipHull_kvlcc2Box_3.stl.
- Purpose: show anisotropic meshing using box primitives.
- Settings in meshDict:
  - Maximum cell size: 120 m.
  - shipHull.stl: refine by two additional levels from maxCellSize.
  - Global number of boundary layers is 10.
  - Global thickness ratio 1.2.
  - Activate optimisation of boundary layers.
  - Create a refinement box for free surface cells.
- Steps:
  - Generate mesh: *cartesianMesh*

Hands-on session: anisotropic meshing

- Example located in: trainingExamples/ship.
- Surface mesh: shipHull_kvlcc2Box_3.stl.
- Purpose: show anisotropic meshing using box primitives.
- Settings in meshDict:
  - Maximum cell size: 120 m.
  - shipHull.stl: refine by two additional levels from maxCellSize.
  - Global number of boundary layers is 10.
  - Global thickness ratio 1.2.
  - Activate optimisation of boundary layers.
  - Create a refinement box for free surface cells.
- Steps:
  - Generate mesh: *cartesianMesh*

```
anisotropicSources
{
    boxSource
    {
        type box;
        centre   (2600 -10 170);
        lengthX  25000;
        lengthY  5000;
        lengthZ  230;
        scaleX   1.0;
        scaleY   1.0;
        scaleZ   0.2;
    }
}
```

Thank you for your attention!

Creative Fields, Ltd.

X. Vrbik 4

10000 Zagreb, Croatia

E-mail: info@c-fields.com

Web: www.c-fields.com