Working with cfMesh

Md Nazmul Azim Beg
Early Stage Researcher (quics.eu) and PhD Candidate
Department of Civil Engineering
University of Coimbra
1 June, 2017
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

Meshing with cfMesh Training session
Franjo Juretić, Creative Fields, Ltd.
Ann Arbor, June 2015
Examples: cartesianMesh-Asmobody

- Example located in: introductoryExamples/cartesianMesh/aso.
- 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

Meshing with cfMesh Training session
Franjo Juretić, Creative Fields, Ltd.
Ann Arbor, June 2015
Examples: cartesianMesh-Asmobody

Meshing with cfMesh Training session
Franjo Juretić, Creative Fields, Ltd.
Ann Arbor, June 2015
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

Meshing with cfMesh Training session
Franjo Juretić, Creative Fields, Ltd.
Ann Arbor, June 2015
Examples: tetMesh-socket

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

Meshing with cfMesh Training session
Franjo Juretić, Creative Fields, Ltd.
Ann Arbor, June 2015
Examples: tetMesh-socket

Meshing with cfMesh Training session
Franjo Juretić, Creative Fields, Ltd.
Ann Arbor, June 2015
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).
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: Overview

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

Meshing with cfMesh Training session
Franjo Juretić, Creative Fields, Ltd.
Ann Arbor, June 2015
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.

Meshing with cfMesh Training session
Franjo Juretić, Creative Fields, Ltd.
Ann Arbor, June 2015
Training Session

• Installation
• Hands on training
• Example at: manhole
• Surface mesh: manhole.stl
How to install

- Download cfMesh v1.1.2, works with OpenFOAM 4.1
  https://sourceforge.net/projects/cfmesh/?source=typ_redirect
- Unzip the file and install:
  - tar xzf cfMesh-v1.1.2.tgz
  - cd cfMesh-v1.1.2/
  - ./Allwmake
meshDict File

• The meshDict dictionary located in the system directory of the case provides necessary settings
• requires only two mandatory settings
• *surfaceFile:*
  • points to a geometry file.
  • The path to the geometry file is relative to the path of the case directory.
  • suggested file formats: fms, ftr, and stl.
  • *surfaceConvert* can be used if necessary (comes with OpenFOAM®)
• *maxCellSize* represent the default cell size used for the meshing job. It is the maximum cell size generated in the domain.
Topology creation

• **Purpose:** learn the basic options to preserve mesh parts.

• **Settings in meshDict:**
  - Maximum cell size: 0.004 m

• **Steps:**
  - Generate mesh: `cartesianMesh OR tetMesh OR pMesh`

```c++
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    location "system";
    object meshDict;
}

// **************************************************************************
<table>
<thead>
<tr>
<th>Field</th>
<th>Operation</th>
<th>And</th>
<th>Manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Operation</td>
<td>And</td>
<td>Manipulation</td>
</tr>
<tr>
<td>cfMesh: A library for mesh generation</td>
<td>Author: Franjo Juretic</td>
<td>E-mail: <a href="mailto:franco.juretic@c-fields.com">franco.juretic@c-fields.com</a></td>
<td></td>
</tr>
</tbody>
</table>

maxCellSize 0.004;
```
Topology creation

cartesianMesh

tetMesh

pMesh
Preserving feature edges

• Purpose: to have better shape at the intersecting edges

• Steps:
  • Generate new surface:
    `surfaceFeatureEdges manhole.stl manhole1.stl -angle 60`
  • Change the surface name at `meshDict`
    from `manhole.stl` to `manhole1.stl`
  • Create Mesh: `cartesianMesh`
Preserving feature edges
Preserving feature name and type

- **renameBoundary** dictionary has the following options:
  - **newPatchNames**: is a dictionary with the following inside the **renameBoundary** dictionary.
    - contains dictionaries with names of patches that shall be renamed.
    - **newName** keyword is followed by the new name for the given patch. The setting is not mandatory.
    - **type** keyword is followed by the new type for the given patch. The setting is not mandatory.
  - **defaultName**: gives is a new name for all patches except the ones specified in **newPatchNames**
  - **defaultType**: sets the new type for all patches except the ones specified in **newPatchNames** directory

- This can also be changed manually from `/constant/polyMesh/boundary` file.
Preserving feature name and type

• Purpose: to assign proper name and type to all the boundary faces

• Steps:
  • Edit in `meshDict`:

```plaintext
renameBoundary
{
    newPatchNames
    {
        shape_0 { newName atmosphere; type patch; }
        shape_1 { newName wallManhole; type wall; }
        shape_2 { newName wallPipe; type wall; }
        shape_3 { newName wallBottom; type wall; }
        shape_4 { newName wallPipe; type wall; }
        shape_5 { newName inlet; type patch; }
        shape_6 { newName outlet; type patch; }
    }
    defaultName walls;
    defaultType wall;
}
```
Refinement settings in meshDict

• Quite often a uniform cell size is not satisfactory, and there are many options for local refinement sources in cfMesh.

• \textit{boundaryCellSize}: This option is used for refinement of cells at the boundary.
  • A global option and the requested cell size is applied everywhere at the boundary.

• \textit{boundaryCellSizeRefinementThickness} specifies the distance from the boundary until which the \textit{boundaryCellSize} is applied

• \textit{minCellSize} is a global option which activates automatic refinement of the mesh template.

• \textit{localRefinement}: a dictionary that allows for local refinement regions at the boundary.
  • is named by a patch or facet subset that needs refinement and uses following options
    • \textit{cellSize} keyword and a scalar value, or by specifying
    • \textit{additionalRefinementLevels} keyword and the desired number of refinements relative to the maximum cell size and \textit{refinementThickness} option
Refinement settings in meshDict

• Purpose: to have better mesh at the boundary

• Steps:
  • Edit in meshDict:

```plaintext
boundaryCellSize 0.002; //
boundaryCellSizeRefinementThickness 0.004;
minCellSize 0.0018;

localRefinement //
{
    shape_5 { additionalRefinementLevels 1; refinementThickness 0.006; }
    shape_6 { cellSize 0.001; }
}
```

• Perform cartesianMesh
Refinement settings in meshDict

- **objectRefinement** is used for specifying refinement zones inside the volume.
- The supported refinements are: lines, spheres, boxes, truncated cones, and hollow cones (annulus).
- **objectRefinement** dictionary represents the name of the object used for refinement.
- **refinementThickness** option specifies the thickness of the refinement zone away from the object.
- **cellSize** specifies the requested cell size inside the volume.
  Alternatively,
- **additionalRefinementLevels** specifies the number of additional refinement levels compared to **maxCellSize**, applicable inside the volume.
- Examples of uses as box, cone and hollowCone primitives are shown here.
Refinement settings in meshDict

• Purpose: to have better mesh resolution at certain zones

• Steps:
  • Edit in meshDict:
  • Perform cartesianMesh
Refinement settings in meshDict
Boundary layers

• Boundary layers are extruded from the boundary faces of the volume mesh towards the interior

• They cannot be extruded prior to the meshing process

• The thickness is controlled by the cell size specified at the boundary and the mesher tends to produce layers of similar thickness to the cell size

• Layers in cfMesh can span over multiple patches if they share concave edges or corners with valence greater than three

• Layer settings are provided inside a boundaryLayers dictionary
  • nLayers: specifies the number of layers to be generated. It is not mandatory. In case not specified the meshing workflow generates the default number of layers, which is either one or zero
  • thicknessRatio: is a ratio between the thickness of the two successive layer. not mandatory and must be larger than 1. Otherwise, the default value is 1.
  • maxFirstLayerThickness: not mandatory
  • patchBoundaryLayers: setting is a dictionary which is used for specifying local properties of specific boundary layers patches. possible to specify nLayers, thicknessRatio and maxFirstLayerThickness for each patch individually
  • allowDiscontinuity: ensures that the number of layers required for a patch shall not spread to other patches in the same layer
Boundary layers

- *optimiseLayer* can be enabled for large number of layers and for smooth variation of layers

- Controlled by *optimisationParameters* dictionary inside the *boundaryLayers* dictionary with following options
  - *nSmoothNormals*: is the number of iterations in the procedure for smoothing normal vectors in the boundary layer. Not mandatory and the default value is five.
  - *maxNumIterations*: is the number of iterations in the smoothing procedure. Not mandatory and its default value is five.
  - *featureSizeFactor* is the ratio between the maximum allowed layer thickness and the estimated feature size. Used to limit layer thickness in the regions dominated by curvature. Not mandatory; and its valid range is between zero and one. Default value is 0.3.
  - *reCalculateNormals* calculates the surface normal vectors, and aligns boundary-layer edges to point in the normal direction. Not mandatory, and is active by default.
  - *relThicknessTol* controls the maximum difference of the layer thickness between the two neighbouring points, divided by the distance between the points. Not mandatory, and the valid range is between zero and infinity. Lower values reduce layer thickness thin and enforce uniform thickness distribution.

- *untangleLayers* setting is optional and is activated by default. It helps to get rid of any negative cell volume if created by *optimisationParameters* process
Boundary layers

• Purpose: to have better mesh resolution at certain or all boundaries

• Steps:
  • Edit in `meshDict`:
  • Perform `cartesianMesh`
Refinement settings in meshDict

- *surfaceMeshRefinement* allows for using surface meshes as refinement zones in the mesh.
- Specified as a dictionary of dictionary where each refinement zone is a sub-dictionary inside the *surfaceMeshRefinement*.
- *surfaceFile* provides the refinement zone.
- *cellSize* keyword controls the cell size.
- OR *additionalRefinementLevels* keyword gives desired number of refinements relative to the maximum cell size.
- Also possible to control the thickness of the refinement zone via *refinementThickness* keyword.
- Example is similar to the *localRefinement*. 
Refinement settings in meshDict

• *edgeMeshRefinement* allows for using surface meshes as refinement zones in the mesh

• Specified as a dictionary of dictionary where each refinement zone is a sub-dictionary inside the *edgeMeshRefinement*

• *surfaceFile* provides the refinement zone

• *cellSize* keyword controls the cell size

• OR *additionalRefinementLevels* keyword gives desired number of refinements relative to the maximum cell size

• Also possible to control the thickness of the refinement zone via *refinementThickness* keyword

• Example is similar to the *localRefinement*
Anisotropic meshing

• The settings for anisotropic meshing are given in `anisotropicSources` dictionary.

• Primitive objects applicable are: Box and Plane

• For *box* the options are:
  • `centre`: coordinates of the centre.
  • `lengthX`: length of the box in the x-direction.
  • `lengthY`: is the length at y-direction.
  • `lengthZ`: is the length at z-direction.
  • `scaleX`: is a scaling factor applied in the x-direction. The valid range is between zero and infinity. Values smaller than 1 make the cells smaller in the x-direction, and value greater than 1 make the cells larger, respectively.
  • `scaleY`: is a scaling factor applied in the y-direction.
  • `scaleZ`: is a scaling factor applied in the z-direction.

• For *plane* the options are:
  • `origin`: coordinates of the plane origin.
  • `normal`: specifies the normal vector of the plane
  • `scalingDistance`: specifies the distance from the plane in the positive direction of surface normal. The mesher generates anisotropic cells in the in the region between the plane and plane translated by a scaling distance.
  • `scalingFactor`: is applied in the normal direction. The valid range is between zero and infinity. Values smaller than one make the cells smaller in the direction of the normal vector, and value greater than one make the cells larger, respectively
Anisotropic meshing

• Purpose: to have different mesh resolution at certain zone

• Steps:
  • Edit in meshDict:
  • Perform cartesianMesh

```plaintext
anisotropicSources
{
  boxSource
  {
    type box;
    centre (0 0 0.3);
    lengthX 0.05;
    lengthY 0.05;
    lengthZ 0.05;
    scaleX 1.0;
    scaleY 1.0;
    scaleZ 0.5;
  }
}
```
Anisotropic meshing
Refinement settings in meshDict

- `enforceGeometryConstraints` option stops the meshing process when it is not possible to capture all features of the input geometry. When active, it stops the meshing process and writes a subset of points that had to be moved away from the geometry in order to yield a valid mesh.

```
enforceGeometryConstraints 1;
```
Reference


Thank you for your attention!

Md Nazmul Azim Beg
mnabeg@uc.pt; nazmul.azim@gmail.com

Md Nazmul Azin Beg is funded by the QUICS project. This project has received funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no 607000.