ERE 224 Project

Mesh Convergence and Refinement Options for 3D Geometries in OpenFOAM

Matthias Cremon

Fall 2015
Contents

I Introduction and Objectives .............................................. 1

II OpenFOAM ................................................................. 1
   1. Generalities ...................................................... 1
   2. Mesher: snappyHexMesh ........................................ 2
   3. Mesher: foamyHexMesh .......................................... 2
   4. Solver: simpleFoam .............................................. 3

III Meshes Display .......................................................... 4
   1. Level 0: Castellated ............................................. 4
   2. Level 1 Refinement .............................................. 5
   3. Level 2 Refinement .............................................. 6
   4. Partial Conclusion on Meshing .................................. 6

IV Flow Results .............................................................. 6
   1. Setup .............................................................. 6
   2. Maximum and Mean velocity ................................... 7
   3. Runtime ........................................................... 8

V Conclusion and Future Work ........................................... 9
I Introduction and Objectives

Pore-scale simulations play a key role in understanding small scale fluid behavior and its effect on the macro-scale properties we would like to accurately estimate. One can get many relevant information by doing direct numerical simulation of Stokes flow at the pore scale. This is particularly interesting for example in an upscaling framework, but the simulation cost can become prohibitive. As for any Computational Fluid Dynamics (CFD) application, meshing strategies play a key role in the quality of the results as well as in keeping the runtime in a feasible range. The obvious tradeoff between speed and accuracy needs to be studied, since as we will see the results are greatly modified when we change the meshes. We will both look at convergence studies, and also at new strategies to try to smartly refine the mesh to improve the quality of the results and/or the runtime. This work focuses on single phase, steady state simulations (corresponding to single phase parameter upscaling).

More information on pore scale flow simulation using microCT scanned rock, often called Digital Rock Physics, can be found in the following (very short) list of relevant papers: [6] [7] [10].

II OpenFOAM

1. Generalities

For this project we used an open-source CFD software named OpenFOAM [3] [11]. It is well known and widely used for a number of CFD applications, going from hypersonic, compressible Navier-Stokes flows to micro-scale Stokes simulations. Although its user base is very active on the internet, providing help and advice for new users, the lack of graphical user interface and the need to be familiar with linux shells requires a bit of time and practice to be able to make the most of its remarkable capabilities.

Cyprien Soulaine, a Research Associate in Supri-B working on pore scale simulations, wrote very helpful tutorials [2] on how to use OpenFOAM for these problems and also provided me with great advice during that project.

OpenFOAM contains many different solvers and utilities; we will just present two that we extensively used for this work: the mesher \textit{snappyHexMesh} and the solver \textit{simpleFoam}. 
2. Mesher: *snappyHexMesh*

Meshing a pore-scale geometry is not a trivial thing to do. *snappyHexMesh* is a meshing utility that allows the user to combine a regular, cartesian mesh generated by *blockMesh* with a surface-tracking file (.stl or .obj). First, we create a cartesian mesh for the whole domain. Then, *snappyHexMesh* locates the cells that are crossed by a surface from the .stl file, and removes one side (either the grains, which is what we do in fluid simulations, or the void space if one is interested in rock mechanics for instance). It also has refinement options, that we will investigate later on. Figure 1. (courtesy of Cyprien) illustrates what we are trying to accomplish:

![Figure 1: 2D illustration of pore-scale meshing](image)

Figure 1: 2D illustration of pore-scale meshing

More details and visuals can be found here: [1].

3. Mesher: *foamyHexMesh*

*foamyHexMesh* is a new mesher (released in OpenFoam 2.3.0, February 2014) that uses a completely different strategy. It creates fully unstructured meshes through the use of Delaunay triangulation and Voronoi tessellation. We wanted to use that new tool to generate meshes and compare the results with those of *snappyHexMesh*, that was the initial objective of this project. Unfortunately, we encountered numerous compiling and library-linking issues with that mesher. Thanks to Cyprien’s advice and Dennis Michael’s help, we managed to get a clean install of OpenFoam 3.0.0, and it appears that on this new version, *foamyHexMesh* did compile correctly, and runs in a sequential manner. However, Dennis had to use gcc-5 to compile it, and since the cluster’s basic setup for MPI jobs uses intel-14, we still need to do some adjustments in order to run parallel jobs.
We were able to get the built-in tutorials to run, and we are now in the process of adapting these to our geometry. It will take some time before we are able to use it to the full extent of its capabilities, but this tool is very promising and will give us some flexibility compared to *snappyHexMesh* (provided the quality of the meshes is good). Figure 2. shows one of the meshes from the tutorial, simply for illustration purposes and see what the unstructured meshes we could get will look like:

![Figure 2: 3D example of an unstructured mesh](image)

4. **Solver: simpleFoam**

We only solved single phase, incompressible and steady-state Stokes problems. Therefore we used a SIMPLE algorithm, first introduced in the early 70’s by Spalding and Patankar [9] [8]. The OpenFOAM implementation of this algorithm is *simpleFoam*. It is a semi implicit algorithm, using a predictor-corrector scheme. We compute the gradients of velocity and pressure, solve the momentum equation to get an intermediate velocity field. Then we solve the pressure correction equation, and update the pressure field using under-relaxation factors. Then the mass fluxes are corrected as well as the cell velocities.

The under-relaxation factors are governing the convergence. We set them to 0.7 for \( p \).

We used two built in OpenFOAM linear solvers for the velocity and pressure solves. We should note that we picked one called GAMG (for Generalized Algebraic Multigrid Solver) for the pressure equation. This is a multi-grid solver that excels at solving pressure/elliptic equation (more information can be found here [4]). This choice ensures a fast convergence of the algorithm.
III  Meshes Display

As we previously mentioned, *snappyHexMesh* offers different refinement levels on top of the normal case. In this section we show the resulting meshes.

First, we show in Figure 3. the .stl file that we used for all of the project. It is a 3D cube with 1mm edges and the surface file was obtained through a micro-CT scan:

![3D rendering of the pore (solid in red)](image)

Figure 3: 3D rendering of the pore (solid in red)

Note: most of the visuals in this report were obtained using ParaView [5].

1. Level 0: Castellated

The most basic option in *snappyHexMesh* is called castellation. The resulting mesh is composed only of hexahedral cells, all of the same size. This is very easy for the solver to deal with, but the main problem is that the size of the mesh is governed by how many cells we want to use to resolve the smallest capillary of the system. This gets prohibitive very fast, since we usually want \( \approx 10 \) cells to correctly resolve a Poiseuille flow. Doing that of course means that we have an enormous number of cells in big pores, which is not efficient. Figure 4. shows 2 meshes generated using this setup, respectively containing 1.7 and 38 million cells:
2. Level 1 Refinement

One level of refinement means that \textit{snappyHexMesh} will attempt to refine the cells near the surface once. Refinement in this context means using an octree-based scheme to divide each cell in 8 and then remove the ones not needed. However, this will make the mesh unstructured since some edges now arrive on faces. This is not a problem per say since OpenFOAM is perfectly capable of handling unstructuredness, but it may lead to poor approximations due to skewness and non-orthogonality. We did not investigate that part but surely it has an impact on the results. Figure 5. shows 2 of the meshes we used, containing respectively 3M and 6.6M cells.
3. Level 2 Refinement

If we allow another refinement step, the number of cells increases very fast due to the octree strategy. We still tested some of those meshes, and Figure 6. shows pictures of respectively 15.6M and 36.5M cells meshes:

![Figure 6: Level 2 Refined Meshes (left: 15.6M cells, right: 36.5M cells)](image)

4. Partial Conclusion on Meshing

Refining the mesh in that fashion, namely near the zero-velocity boundary conditions, should ensure that the small capillaries are well resolved without throwing a huge number of cells in areas where we do not need them. The next step is to test these concepts by running flow simulations.

IV Flow Results

1. Setup

We solve the single-phase, incompressible Stokes equations:

\[
\nabla \cdot \mathbf{u} = 0 \\
\mu \nabla^2 \mathbf{u} - \nabla p + \mathbf{f} = 0
\]

with \( \mathbf{u} \) velocity, \( p \) pressure, \( \mu \) viscosity and \( \mathbf{f} \) external forces (e.g. gravity).
Note that these equations are steady state due to the incompressibility. The boundary conditions are:

- Inlet/Outlet: fixed pressure.
- Walls: no-slip ($u=0$) and zero pressure gradient.

This setup corresponds to a single-phase parameter upscaling in the $X$ direction. As we usually do for CFD studies, we will use the biggest mesh as a reference solution (as we will see in the next section, this is valid since we do reach mesh convergence).

2. Maximum and Mean velocity

![Graph showing Maximum and Mean velocity](image)

Figure 7: Comparison of Maximum and Mean velocity in the pore, for several mesh sizes

The two most relevant quantities to look at to assess the accuracy of our results are the maximum velocity and the mean velocity. Indeed, the maximum velocity is a good indication of the quality of the resolution of the Poiseuille flows. The left hand side of Figure 7 shows those results. All meshes converge to the same value. A few more points would have been helpful to make sure that everything is fine, but due to time constraints we were not able to run more calculations.

Then, we need to look at the most important quantity: the mean velocity over the whole pore space. Indeed, in an upscaling framework this is the one we need to compute in order to be used...
as the effective property. We observe the same behavior as for the maximum velocity, with all meshes converging to a constant value.

The main takeaway from these results are that snappyHexMesh refinement strategy does not degrade the quality of the results, but also as we can see on Figure 6., there is no miracle: we need at least $\approx 15$ millions of cells to get the correct values. Refining does not allow us to reduce the number of cells, which is a bit disappointing. But thankfully, in the next section we observe a more promising result.

## 3. Runtime

We now look at the runtime of all these calculations. Thankfully, OpenFoam’s domain decomposition routines are quite good and thanks to the cees cluster we were able to run jobs using 8 and 64 cores: 8 is a domain decomposition of 2x2x2, and 64 of 4x4x4. We tried to maintain a number of cells per processor below 250,000.

![Runtime](image)

Figure 8: CPU execution time for all the meshes tested.

The runtime results are quite interesting and promising. It shows that the more refined the mesh is, the faster it runs. Using a level 2 refinement gives a 3 times speedup compared to a non refined one, for roughly the same number of cells. Indeed, the number of iterations of the SIMPLE algorithm is greatly decreased, as shown in Table 1:
<table>
<thead>
<tr>
<th>Mesh Type</th>
<th>Num. Cells</th>
<th>SIMPLE It.</th>
<th>Cores</th>
<th>Clock Time (min)</th>
<th>CPU Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
<td>505,878</td>
<td>212</td>
<td>8</td>
<td>2</td>
<td>11.7</td>
</tr>
<tr>
<td></td>
<td>1,749,909</td>
<td>406</td>
<td>8</td>
<td>12.9</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>4,799,492</td>
<td>696</td>
<td>64</td>
<td>14.5</td>
<td>733</td>
</tr>
<tr>
<td></td>
<td>13,993,203</td>
<td>1,201</td>
<td>64</td>
<td>64</td>
<td>3,702</td>
</tr>
<tr>
<td></td>
<td>38,395,936</td>
<td>1,003</td>
<td>64</td>
<td>228</td>
<td>14,540</td>
</tr>
<tr>
<td>Level 1</td>
<td>3,081,174</td>
<td>237</td>
<td>8</td>
<td>14</td>
<td>109</td>
</tr>
<tr>
<td></td>
<td>6,639,004</td>
<td>380</td>
<td>8</td>
<td>119</td>
<td>568</td>
</tr>
<tr>
<td></td>
<td>16,904,704</td>
<td>664</td>
<td>64</td>
<td>41.5</td>
<td>2,654</td>
</tr>
<tr>
<td>Level 2</td>
<td>3,006,904</td>
<td>104</td>
<td>8</td>
<td>84</td>
<td>243</td>
</tr>
<tr>
<td></td>
<td>15,681,216</td>
<td>307</td>
<td>64</td>
<td>19</td>
<td>1,125</td>
</tr>
<tr>
<td></td>
<td>36,530,009</td>
<td>428</td>
<td>64</td>
<td>50</td>
<td>3,195</td>
</tr>
</tbody>
</table>

Table 1: Summary of the results

This is probably due to the fact that the cells are put in places where they are more 'useful': convergence of the SIMPLE algorithm is faster since we are able to capture the flow more accurately. We also looked at the cost per iteration, and although it depends on the number of cells used, it did not change with the refinement level (i.e. for all the ≈15 million cells meshes, they were similar). Thus, the speedup does not seem to come from improvement in the linear solves, but really from the fact that SIMPLE converges faster with a more refined mesh because it is easier to capture the 'hardest' regions of the flow, which are the ones that really govern convergence to the steady-state solution. With more cells, it is less likely that the algorithm will go back and forth between plausible solutions, and therefore convergence is faster.

We should note that these runtimes are only for the the simpleFoam solve. Generating the cartes- sian mesh, decomposing the domain, running snappyHexMesh (all of the aforementioned are pre-processing steps), reconstructing the solution and creating VTK visualization files (post-processing steps) also take time, but these steps mostly depend on the sheer size of the mesh, not on the refinement level we used.

V Conclusion and Future Work

Unfortunately, we were not able to conduct the initial study we hoped for, namely generate and use unstructured mesh for 3D geometries. This is definitely something we will look at in the future
and we think that it will give us new possibilities. The work we were able to conduct still yielded some interesting informations. We now can refine the meshes we use near the walls, which in most cases will be helpful. The case we studied was probably not the best suited to get important improvements concerning the number of cells we need to use and the speedup we can get. We did see an improvement in the convergence of the SIMPLE algorithm, which is still a good starting point. More heterogeneity (e.g., with carbonates) and/or going to multiphase flow (where the near-wall effects, such as contact angles for example) are even more crucial.
References


