ACSE 9 IRP: Project Plan

Duncan Hunter

June 26, 2019

Supervisor: Dr. A. Obeyesekara

1 Rationale and Project Objectives

Pipe and pipe network flow analysis is an important real world problem in areas such as water distribution and refining. The CFD code ICFERST (*Pain et al.*, 2001; *Gomes et al.*, 2017; *Salinas et al.*, 2017a,b) can simulate turbulent flows, yet understanding the ICFERST workflow is a barrier for its use. This project aims to make the cumbersome process of generating meshes for ICFERST easier, and to automate the ICFERST workflow for pipe/pipe network flow analyses.

The software produced in this project has several objectives. It needs to have the ability to create specific pipe network geometries (e.g. an engineer could specify a design, or batches of designs) easily, that are compatible with ICFERST. Geometrical information from meshes needs to be passed from the pre-processing software to ICFERST and the post-processing software, as part of automating the workflow of conducting analyses of one or several pipes/pipe networks.

If there is time in the project, optimisation of pipe networks could be developed, using the post-processing results of multiple simulations to reach an optimum. The software should be able to run on a high performance computer (HPC). This reduces the time taken for simulations significantly and is critical in determining whether optimisation is possible.

The desired entire process is shown in Figure 1, with information being passed between different pieces of software to create a more automatic process. An engineer could start at either a low or high level, and have an easy process conducting a pipe flow analysis.

2 Literature Review

2.1 ICFERST

ICFERST uses control volume, finite element methods with adaptive meshing to simulate fluid flow, with either single or multiphase flow, as well as porous media flow. The details of the code and numerical formulation are not the focus of this project. Mesh adaptivity is key, as it can increase accuracy in areas of high complexity (as found often in turbulent flow), and can allow the user to not need to know a priori the areas of mesh-refinement. It is also not used in many commercial CFD codes.

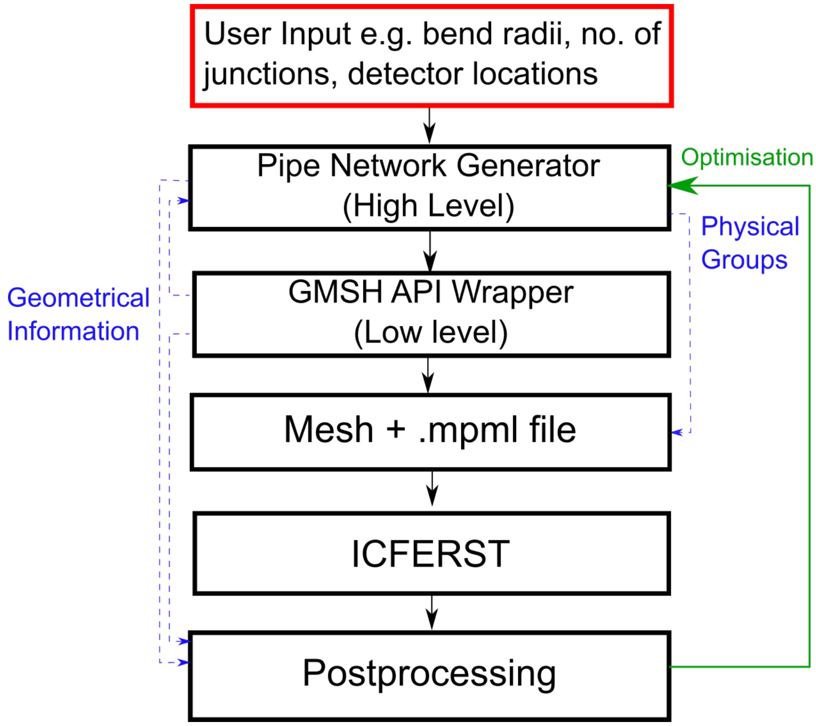

Figure 1: Ideal ICFERST process post project completion. Blue dashed lines show the passing of information between softwares. Green shows the optimisation loop.

2.2 Mesh Generation

Before using any CFD code, meshes need to be generated to specify the domain, and the initial and boundary conditions. This project aims to build on an existing meshing software to create tools. The GMSH-API (*Geuzaine and Remacle*, 2009) has been chosen for the following reasons: it uses Python; it is code based, rather than graphical user interface (GUI) based; it can interface with ICFERST's requirement for *Physical Groups*; and can create specific, complex objects. Python is ideal as it is a commonly used programming language, and has useful built-in functionality. Being code based is critical, as this project hopes to produce software that can be used on HPCs, and built on by this project's pipe generators and potential optimiser. *Physical groups*, used commonly in GMSH, are needed to define initial and boundary conditions in ICFERST, which already has compatibility for GMSH files. The ability to create complex and specific geometries is critical, as it allows engineers to recreate designs accurately for analyses. GMSH contains features from OpenCASCADE, which allows it to create complex objects such as junctions and mitered bends through boolean operations such as fuse and intersect.

Other potential softwares that could have been used include SALOME (*Ribes and Caremoli*, 2007), PyGMSH, or creating a tool similar to PyGMSH that adds raw GMSH code to a file. It was found that these all had downsides greater than any found in the GMSH-API.

2.3 Pipe flow

This project aims to make analysis of specific pipes and pipe networks easier. Problems of turbulent flow in pipe bends are studied often, as they can be useful in determining how effective a pipe is, and how much wear the pipe will endure. An early analysis of pipe flow in a sharp bend is from *Tunstall and*

Harvey (1968). A good introduction and analysis of turbulent pipe flow in smooth bends can be found in *Hufnagel* (2016), where swirl switching is analysed. *Sierra-Espinosa et al.* (2000a,b); *Sakowitz et al.* (2014) have conducted analyses of junction flow. The tools this project will make creating geometries for these analyses easier, and may lead to more complex analyses. The cited analyses could be used as test cases.

2.4 Optimisation

Optimisation of networks could be investigated if there is sufficient time, and is related enough to the project. Optimisation could involve creating and evaluating (by simulation) many meshes, and generating fitness scores from post-processing results. Algorithms for optimizing pipe networks include genetic algorithms (*Dandy et al.*, 1996) and memetic algorithms (*Eusuff and Lansey*, 2003). These approaches are for large networks, but there is scope for optimization to be applied to smaller networks. There may not be enough time or computing power to do this successfully, but this project may create the tools to develop it in the future.

3 Proposed Approach and Prototype Program

This project aims to write code that can be used at both a high and low level, as in Figure 1. Thus an engineer without detailed knowledge could conduct flow analyses easily, but also a developer could create further tools. A GMSH-API wrapper will be low level, and will be used in a pipe/pipe-network generator which should be high level and easy to use.

3.1 GMSH Wrapper and Pipe Generator

Class based wrappers will be created, creating GMSH entities and storing useful and easy to access properties of GMSH entities, as well as useful functions. Many of these properties (examples listed below), are geometrical, and are often hard to access and manage when writing with the GMSH-API (or GMSH alone). Examples of entities that could be created include cylinders, smooth bends, mitered bends, and junctions.

- Location
- Centre of faces
- Centre of object
- Direction
- Direction of faces
- GMSH tags
- GMSH dimensions
- Radius[1]
- Length[1]

An example pseudocode of a GMSH entity is in Listing 1.

Listing 1: Cylinder class example with example functions.

```
class Cylinder():
        def __init__(self, properties of cylinder):
                self.properties = properties
                gmsh.create_object(properties of cylinder)
                if direction:
                        gmsh.rotate()
                if position:
```

[1]Where applicable

```python
                    gmsh.translate()
    self.update_properties()
    def rotate(self, new_direction, centre_of_rotation):
        gmsh.rotate()
        self.update_properties()
    def fuse(self, object):
        gmsh.fuse()
    def update_properties(self)
        self.properties = new_properties
```

Pipes or pipe networks could then be created in a modular, sequential fashion; either manually at a low level by using these classes (demonstrated in Figure 2), or the high level pipe/pipe network generator could be used. This would involve the engineer giving rules or requirements to the software, which then creates the pieces automatically through the GMSH-API wrapper. Rules could be simple, such as exits being 10 diameters from a curve, or more complex, perhaps involving multiple junctions.

3.2 ICFERST workflow and automation

The pipe generator should be able to pass geometrical data (such as locations of entries and exits of pipes) to ICFERST as well as post-processing software. As this data will be stored in python classes, it should be easy to access. The generator could also store information about the surfaces of the pipe, such as which one is the inlet, outlet or wall. This is useful as it can be passed to ICFERST as physical groups. Ideally, this could lead to using python to automatically generate the .mpml file (an xml style file with properties of the simulation) needed for simulations. If this is done, then the automated process of using ICFERST is greatly enhanced.

3.3 Optimisation

A simple genetic algorithm could be developed to optimise small pipe networks at first, and if time permitting, larger networks. This would likely involve using a physical attribute such as velocity or pressure from post-processing to create a fitness score for a network. This may not be feasible, as running simulations is time consuming, and genetic algorithms typically involve running many hundreds of possible configurations.

3.4 Code Sustainability

If possible, TravisCI will be used to perform continuous integration. It may not work perfectly, as Travis may not be able to use some of the libraries this project uses (such as GMSH-API). If so, then test scripts will be written that can be executed locally. Documentation will be written during development, both to ease the process of writing new software that builds on previous work, and to improve the quality (as freshly written code is easier to understand). Documentation will be in the form of doc-strings, and a LaTeX manual. Ideally, parts of this project could be uploaded to PyPI, so users can easily install it using pip-install.

4 Deliverables and deadlines

The timeline of this project is in Figure 3. There are three main phases of development: the GMSH-API wrapper, the pipe-network generator, and optimisation.

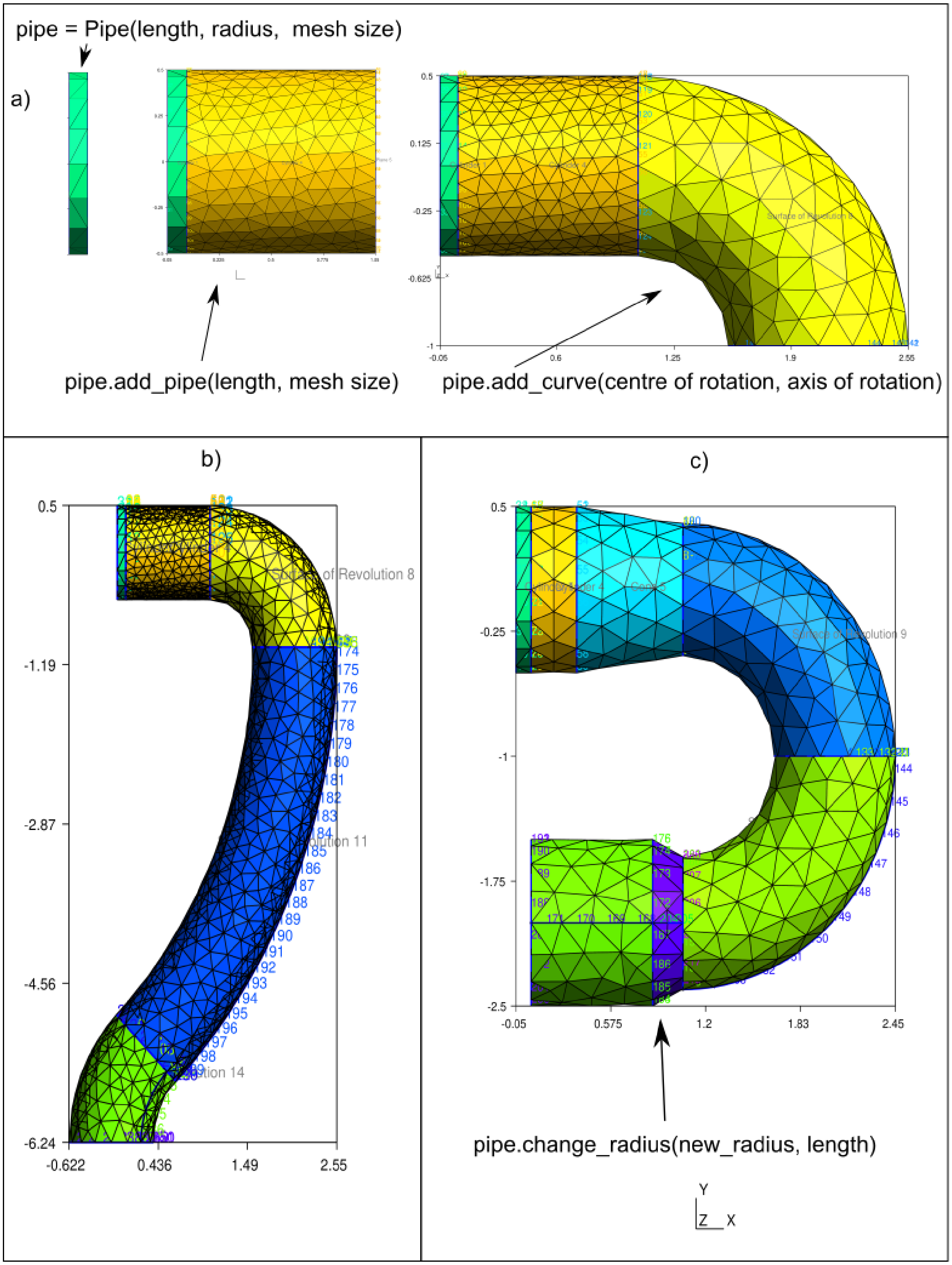

Figure 2: An early prototype, showing that specific geometries can be created. a) Sequential, modular pipe building. b) Arbitrary radii can be created. c) Radius can be changed.

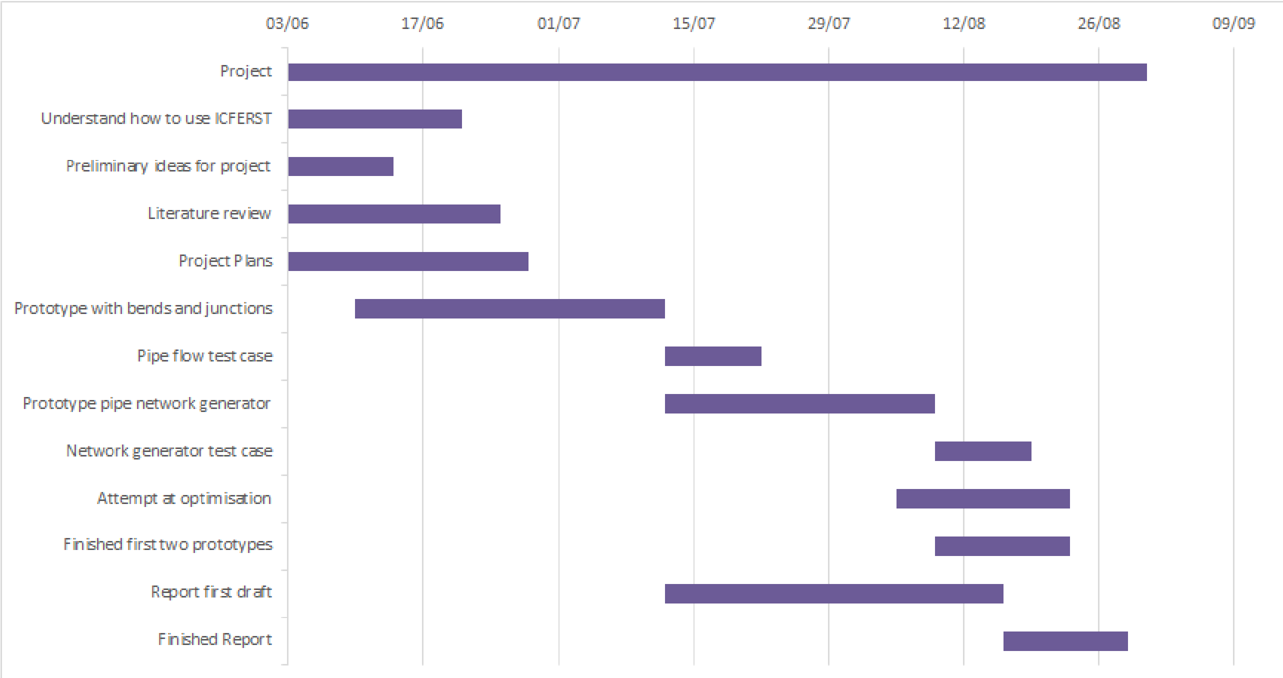

Figure 3: Gantt chart showing ideal project timeline

References

Dandy, G. C., A. R. Simpson, and L. J. Murphy (1996), An Improved Genetic Algorithm for Pipe Network Optimization, *Water Resources Research*, *32*(2), 449–458, doi:10.1029/95WR02917.

Eusuff, M. M., and K. E. Lansey (2003), Optimization of Water Distribution Network Design Using the Shuffled Frog Leaping Algorithm, *Journal of Water Resources Planning and Management*, *129*(3), 210–225, doi:10.1061/(ASCE)0733-9496(2003)129:3(210).

Geuzaine, C., and J.-F. Remacle (2009), Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities, *International journal for numerical methods in engineering*, *79*(11), 1309–1331.

Gomes, J. L. M. A., D. Pavlidis, P. Salinas, Z. Xie, J. R. Percival, Y. Melnikova, C. C. Pain, and M. D. Jackson (2017), A force-balanced control volume finite element method for multi-phase porous media flow modelling, *International Journal for Numerical Methods in Fluids*, *83*(5), 431–445, doi:10.1002/fld.4275.

Hufnagel, L. (2016), On the swirl-switching in developing bent pipe flow with direct numerical simulation.

Pain, C., A. Umpleby, C. de Oliveira, and A. Goddard (2001), Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations, *Computer Methods in Applied Mechanics and Engineering*, *190*(29-30), 3771–3796, doi:10.1016/S0045-7825(00)00294-2.

Ribes, A., and C. Caremoli (2007), Salome platform component model for numerical simulation, in *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, vol. 2, pp. 553–564, IEEE.

Sakowitz, A., M. Mihaescu, and L. Fuchs (2014), Turbulent flow mechanisms in mixing T-junctions by Large Eddy Simulations, *International Journal of Heat and Fluid Flow*, doi: 10.1016/j.ijheatfluidflow.2013.06.014.

Salinas, P., D. Pavlidis, Z. Xie, C. Jacquemyn, Y. Melnikova, M. D. Jackson, and C. C. Pain (2017a), Improving the robustness of the control volume finite element method with application to multiphase porous media flow, *International Journal for Numerical Methods in Fluids*, *85*(4), 235–246, doi: 10.1002/fld.4381.

Salinas, P., D. Pavlidis, Z. Xie, A. Adam, C. C. Pain, and M. D. Jackson (2017b), Improving the convergence behaviour of a fixed-point-iteration solver for multiphase flow in porous media, *International Journal for Numerical Methods in Fluids*, *84*(8), 466–476, doi:10.1002/fld.4357.

Sierra-Espinosa, F., C. Bates, and T. O'Doherty (2000a), Turbulent flow in a 90 pipe junction. Part 2., *Computers & Fluids*, *29*(2), 215–233, doi:10.1016/S0045-7930(99)00005-5.

Sierra-Espinosa, F., C. Bates, and T. O'Doherty (2000b), Turbulent flow in a 90 pipe junction. Part 1., *Computers & Fluids*, *29*(2), 197–213, doi:10.1016/S0045-7930(99)00004-3.

Tunstall, M. J., and J. K. Harvey (1968), On the effect of a sharp bend in a fully developed turbulent pipe-flow, *Journal of Fluid Mechanics*, *34*(3), 595–608, doi:10.1017/S0022112068002107.