University of Exeter coursework header sheet

ECMM102 Group Project (Meng) (A, TRM1+2 2016/7)

035794

1027442

630038328

Coursework: Individual contribution to the group achievement

Submission Deadline: Fri 12th May 2017 12:00

Personal tutor: Dr Prakash Kripakaran

Marker name: G_Tabor

Word count: 10686

By submitting coursework you declare that you understand and consent to the University policies regarding plagiarism and mitigation (these can be seen online at www.exeter.ac.uk/plagiarism, and www.exeter.ac.uk/mitigation respectively), and that you have read your school's rules for submission of written coursework, for example rules on maximum and minimum number of words. Indicative/first marks are provisional only.

I2 Report

Surface Representation for Optimisation using Catmull-Clark Subdivision

Samuel Hutchings

2016
4th year MEng Group Project

I certify that all material in this thesis that is not my own work has been identified and that no material has been included for which a degree has previously been conferred on me.

Signed …….Sam Hutchings…….

College of Engineering, Mathematics, and Physical Sciences
University of Exeter

I2 Report

ECMM102

Title: Surface Representation for Optimisation using Catmull-Clark Subdivision
Word count: 10,686
Number of pages: 37

Date of submission: Friday, 12 May 2017

Student Name: Samuel Hutchings
Programme: Electronic Engineering MEng
Student number: 630038328
Candidate number: 035794

Supervisor: Professor Gavin Tabor

Abstract

Optimisation of systems is a complex problem, requiring the alteration of numerous interacting parameters or degrees of freedom. Exploring such a problem space quickly becomes impractical to be undertaken by a human, evaluating single solutions at once. A problem particularly pertinent in the case of problems for which the evaluation of a potential solution is non-trivial or expensive (*eg.* the optimisation of a fluids system using *computational fluid dynamics*), making a undirected human search of a complex search space with many variables both ineffective, costly and slow. As a solution to this many optimisation techniques have been proposed such evolutionary algorithms and Bayesian learning approaches. While the latter can provide an intelligent exploration of a high dimensional problem space to find the optimal solution in the solution space, an increase in the number of dimensions (or parameters in the optimisation), intuitively reduces the amount of the problem space which can be effectively explored in the same number of objective function evaluations, reducing the likelihood of convergence on the optimal solution. As such it is beneficial to limit the number of parameters as much as possible. This report presents a novel approach using the Catmull-Clark subdivision surface algorithm, to generate detailed meshes for representing the boundary surface of a fluids problem (*eg.* optimising the pressure recovery of a draft tube) from a manipulable and parametrised low resolution mesh, consisting of connected primitive components, which can be simply constructed to represent a complex system. The details of a Python implementation are outlined, including a bespoke implementation of the half-edge polygon mesh data structure and the Catmull-Clark algorithm.

Keywords: Catmull-Clark, Half-Edge Data Structure, Optimisation, Parametric Modelling, Draft Tubes

Table of Contents

1 Introduction and Background

1.1 Overview

In a world with a rapidly inflating demand for resources and energy, the need for efficiency in the design of systems is greater than ever. Many problems however are vastly complex, depending on a vast set of parameters which can be challenging or even impossible for a human to intuitively alter to find an optimal solution. This issue is exacerbated when to evaluate such an individual solution is complex, computational intensive task, taking a significant amount of time such as a simulation using computational fluid dynamics (or CFD), where the execution time can be of the order of hours or days for complex problems. As such, there is significant value and motivation in developing an approach to optimise computationally complex problems where evaluation of potential solutions is expensive.

One example of such a system is a draft tube, the final element in a hydroelectric power station, where water from the turbine exits into the tube and is transferred into the lower reservoir in such a manner that the energy transfer into the turbine is maximised, improving the efficiency of the system. A draft tube is a complex and expensive construction and difficult to optimise, and as such would benefit from the application of a simulation-based optimisation, where the system can be designed optimally for the requirements, without discovering that it is sub-optimal after construction has been completed.

One element of the optimisation process is the design of the objective function or how the alteration of parameters by an optimiser (*eg.* a evolutionary algorithm or Bayesian optimiser) affects the efficiency of the system. In the case of this project, this consists of two stages: the generation of the geometry of the draft tube and the fluid simulation of the potential solution using the open source CFD software OpenFOAM.

This report will focus on the former, developing an approach to parametrising a geometry which can than be utilised in OpenFOAM to simulate and evaluate a potential solution. A system of Python modules has been developed to allow for the construction, parametrisation and generation of surface meshes of an arbitrary fluids systems, with an aim of reducing the number of parameters required to generate the necessary detail in the surface mesh to perform the required optimisation. This has been achieved by application of the Catmull-Clark subdivision algorithm, originally developed for character modelling and animation by Pixar, to a low

resolution polygon mesh, created by a simple structure of connected and parametrised 'pipe' sections.

1.2 Draft Tubes

A draft tube is the final element in a water powered turbine system, which transfers water from the outlet of the turbine into the outflow reservoir through a submerged diffuser mechanism. In a hydroelectric power station, the flow of water with potential energy or head, generally from a dam or reservoir higher than the turbine, is fed through a turbine to generate electrical energy. The lower section of this process is illustrated in figure 1.

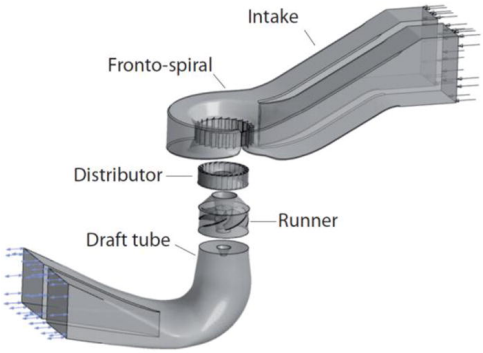


Figure 1: An illustration of a draft tube in the context of a hydroelectric power station [1].

The draft tube itself is situated below the runner (or blades of the turbine) providing a means of escape for water from the turbine. As most turbines are situated vertically, the direction of flow generally needs to be changed from vertical to horizontal, resulting in the characteristic elbow in many draft tubes (see Figure 2).

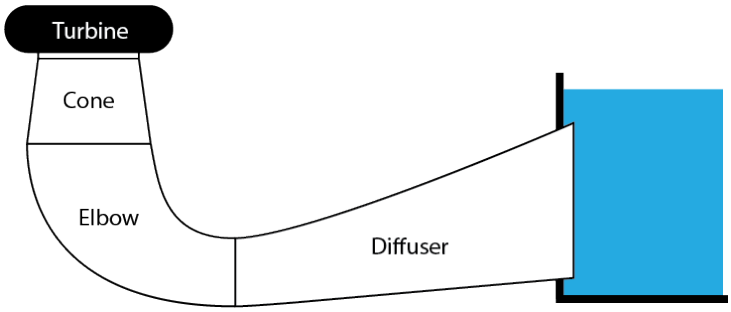


Figure 2: Diagram for a draft tube.

By expanding the cross sectional area of the draft tube, using an expanding diffuser, the dynamic pressure (due to the water having kinetic energy) is converted into static pressure, ensuring that the turbine can take advantage of the greatest possible head, or potential energy in

the water. By optimising a draft tube, ensuring the greatest possible recovery of pressure, the extraction of energy using the turbine can be maximised, improving the productivity and output of a hydroelectric turbine.

1.3 Optimisation

There are numerous algorithms available for the optimisation of a system. One of the most well known approaches is the evolutionary algorithm, which using a process inspired by natural evolution, a set of solutions can be mutated in such a manner that over numerous steps an optimal solution may be converged to, based upon the evaluation of solutions against an objective function. Whilst effective in many scenarios, an evolutionary algorithm requires a significant number of calls to that objective function with differing parameters. This be fine in contexts where the evaluation is inexpensive, but in the context of the optimisation of a draft tube, where the evaluation of an objective function will take the form of a simulation using the open source CFD software *OpenFOAM*. This prohibits obtain an effective solution.

One technique to improve this process, is to use an alternative optimisation algorithm such as Bayesian optimisation. This technique uses machine learning approaches to efficiently optimise an arbitrary black box function (or a function where the internal details are not known or are arbitrary). In the case of this project, the black box function would be formed of three core parts. The first, and focus of this report, is the generation of a STL file mesh to represent the boundary surface of the draft tube or other fluid system, based upon the values of a set of predefined parameters. Next, this file is passed to OpenFOAM, where it is processed to generate the cells necessary to undertake the simulation. Finally a simulation is executed, evaluating the objective from the resulting data. This result is then passed back to the optimiser as the resulting evaluation of the objective function for that set of input parameters.

1.4 Objectives and Deliverables

To perform a simulation of the draft tube, or any fluids system, it is necessary to form a mesh to act as the boundary surface. This report focuses on the development of an effective approach for the creation of mesh structures which can be parametrised to be altered by an optimisation method based upon the evaluation of an objective function. To achieve this the Catmull-Clark algorithm has been explored, which provides a means to trivially morph between circular and

rectangular cross section meshes. A means has been developed to represent a series of 'pipe' elements using simple syntax in a manner which exposes key parameters for each element which could then be altered by an optimiser. This structure can then be processed using the Catmull-Clark algorithm to create a refined surface mesh, exported as an STL file and suitable for further processing in OpenFOAM. Achieving this end objective has required passing along various stepping stones to the final goal, and occasionally having to backtrack and go down a different route when a particular pathway has been unsuccessfully. This report aims to outline and explain these successes and failures on the path to achieving the final deliverable.

1.5 Tools

Due to its prevalence in scientific computing, the Python programming language has been used for the development of software tools in this project. The development was undertaken on multiple machines using the same Anaconda version alongside Python version 3.5 and makes heavy use of the popular Python module *numpy* and makes use of a additional publicly available module for working with STL files with *numpy*. The development makes extensive use of the object oriented features of Python, using classes to encapsulate different components. The software developed as part of this project exists in a Git repository online and will be made available.

1.6 Report Structure

This report has been split into numerous sections dealing with different aspects of the project process and execution. First, the some existing approaches to the optimisation are explored, focusing on the an approach making use of an evolutionary algorithm to optimise the PitzDaily tutorial case in OpenFOAM. This report drove the primary approach to optimisation of the draft tube for our project group, making use of a spline with control points to alter the geometry of the lower surface of the diffuser. Alongside this our chosen optimisation algorithm is outlined exploring its advantages over an evolutionary algorithm. Finally a review is conducted of the Catmull-Clark algorithm and various enhancements that have been introduced is undertaken, considering the impact this will have and what components will drawn upon for this project.

In the next section, the theory behind the implementation of three components of this project will be explored. First the methods used to represent polygon meshes will be evaluated and ex-

plained, along with an overview of the Python implementations made for this project. Next the theory behind the Catmull-Clark subdivision algorithm will be explored, along with the special cases necessary to evaluate certain regions of the mesh, such as the edges. Alongside this the implementation of sharp creases will be considered, using special sets of rules for the subdivision of edges tagged as sharp, providing a means to represent both rectangular and circular geometry. Finally, the design and implementation of a means to represent a set 'pipe' components will explored. These pipe components will be designed such that they can produce an easily manipulatable control mesh, which can be processed using the Catmull-Clark algorithm to produce a mesh suitable for passing to OpenFOAM for simulation.

Following on from this, the results from testing these algorithms will be explored and discussed, identifying potential improvements and exposing issues which could be explored should this approach be taken further. This includes renders from various versions of various component algorithms, assessments of their computational efficiency and an example of using the tools developed to represent a system based upon the geometry of a draft tube. Finally based upon these results conclusions will be drawn, highlighting areas of improvement and making suggestions for further work.

After conclusions, a section has been dedicated to the discussion of the project management and health and safety considerations undertaken during this project. Though in the case of the latter, the primarily computer based nature of the project limits its relevance, additional practical electronics work was also undertaken during this project in support of other team members and a short discussion about safety issue there has been included. Finally a section discussing the nature of my interaction and collaboration with the rest of my group members has been included.

2 Literature review

In any project, it is critical to explore and evaluate the work that has previously undertaken and the literature in the form of scientific papers and reports that this work has generated. As such, in this section, a thorough exploration of available existing material will be detailed, evaluating the efforts of previous research and exploring certain innovations. First, the OpenFOAM *PitzDaily* tutorial will explored and an approach to the optimisation of this using an evolutionary algorithm. Then, the groups chosen approach to optimisation will be considered and the impact this has on how the problem is represented and parametrised for the optimisation.

Finally, the Catmull-Clark subdivision algorithm will be discussed, along with a collection of enhancements developed by subsequent researchers.

Though the approach to optimisation used does not have any huge bearing on the focus of this part of the project, there are benefits to considering how the approaches handle the problem parameters and the effect the number of parameters has on the optimisation. Additionally, before my research path went down the direction of the parametrisation and representation of the problem I undertook some work with my colleagues working on an implementation of Bayesian optimiser. Ultimately my colleagues focus on this area, and their work can be reviewed in their individual reports [2, 3].

2.1 PitzDaily Optimisation

The PitzDaily case is a tutorial in the open source CFD package OpenFOAM based upon experimental work undertaken by Robert Pitz and John Daily whilst undertaking research for NASA in 1981 [4]. It is a relatively simple system, containing a simple pipe, with a downward step just beyond the inlet. Its simplicity has found it being frequently used in OpenFOAM for teaching and testing different models against as a known system. It was chosen by Daniels *et al.* in 2016 as a test case for an approach using an evolutionary algorithm to minimise the drop in pressure in the system [5].

An evolutionary algorithm uses a method derived from its namesake natural evolution, where organisms develop (or evolve) over numerous generations, with mutations resulting in changes in their genetic code and breeding with each other, resulting in mixes of genetic material of the parents in the descendants. These descendants then form a new part of the population, competing with its other members, with only the strongest surviving to have descendants of their own, thus leading to a passing on of the genetic material that makes these individuals the strongest. Over time, this process repeats with the population becoming more and more suited to their environment. An evolutionary algorithm (EA) aims to replicate this behaviour. In a simple EA, a random population of potential solutions for the optimisation is generated, spread out across the problem search space. From this a set of the strongest solutions may be selected and the parameters that make them up mutated or through some method, solutions are combined to create children. These children are then added back into the general population of solutions, where they 'compete' with the existing solutions, and only a subset of the strongest solutions in the population are considered when the algorithm then repeats, having been evaluated against

an objective function (in this case an OpenFOAM CFD simulation). Though it is by no means guaranteed, the progressive application of the algorithm to the population, should lead to a convergence towards a potential optimal solution.

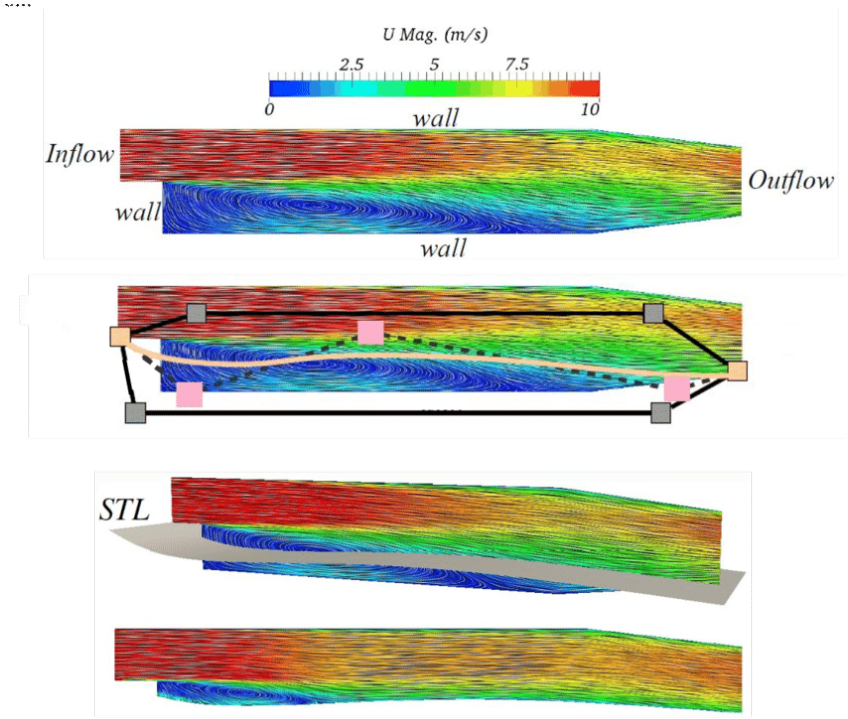


Figure 3: Visualisation of the PitzDaily case from OpenFOAM and the approach used by Daniels *et al.* to alter the geometry of the system based up parameters that can be mutated in the EA. Sourced from the paper under review in this section.

In the case of this investigation, the potential solutions were defined as a set of two dimensional control points (see Figure 3) within a bounding box. These were used to generate a spline curve, which was then used as an extruded cut through the 3D geometry of the PitzDaily mesh, cutting away the area below the spline, changing the geometry of the lower part of the system. To evaluate the objective function (the loss of pressure across the system) this geometry was then used for a CFD simulation in OpenFOAM, using the results from this to evaluate the pressure loss. The strongest solutions were those with the lowest pressure loss.

Despite having the potential to be more effective than an undirected random search of the search space and not requiring the intervention of a human to alter the parameters, an EA does have a number of issues. Primarily they require a large number of objective function evaluations, which in the case of an objective function requiring a CFD simulation, can add up taking considerable amount of time. For the PitzDaily case, the simulation time was of the order of a few minutes, but for a Draft Tube, it is likely that each individual objective function evaluation could take of the order of multiple hours. As such the use of such an approach on a

complex problem is likely to be prohibitively computationally intensive and take far too long to generate a result. Secondly, an EA is, like natural evolution, intrinsically random, potential solutions are randomly mutated by a random amount or combined together, or crossed over, in an arbitrary manner. There is no intelligence behind these decisions in the same way that natural reproduction is essentially a luck of the draw with no intelligence behind it driving the inherited characteristics or attributes in the genome. As such for an EA, the exploration of the mapping between the search and solution spaces is not planned, it is a random exploration.

Finally, and most pertinent to this report, the approach used to the alteration of the geometry used in the CFD simulation is simplistic and limited, only providing a means to alter a single aspect of the model in a two-dimension cross section. A draft tube has a more complicated geometry compared to the PitzDaily case, with elements such as the elbow introducing more complex surfaces and shapes. As such, it would be advantageous to develop a more flexible approach to altering a problem's geometry.

2.2 Bayesian Optimisation

An alternative approach to optimising a system is to use a Bayesian optimisation algorithm. Reviewed in detail by Shahriari *et al.* in their 2015 paper, the Bayesian optimisation methodology aims to improve the optimisation process by applying machine learning techniques to direct the optimisation and converge to an optimal solution with the minimal number of function evaluations [6]. Rather than stochastically exploring the problem space, the approach used in an EA, Bayesian optimisation uses a probabilistic approach using a model to consider the most efficient potential solution to evaluate, based upon what has been previously evaluated. This improved efficiency, whilst not requiring any knowledge of the objective function, makes this approach particularly suitable for this project, with the optimisation team working on this algorithm under the name of EGO (or Efficient Global Optimiser) as coined by Jones *et al.* in their 1998 paper, on which our implementation was based [7].

As visualised in Figure 4, Bayesian optimisation uses a probability based process to generate a model to predict where the to evaluate a solution to converge towards a solution and gain a better 'understanding' of the relationship between the search and solution space. Whilst, the details of how the algorithm achieves this is beyond the scope of this report, there are some points worth considering. Though the example illustrated only shows the maximisation of a one-dimensional objective function, the algorithm can be trivially extended to allow for objective

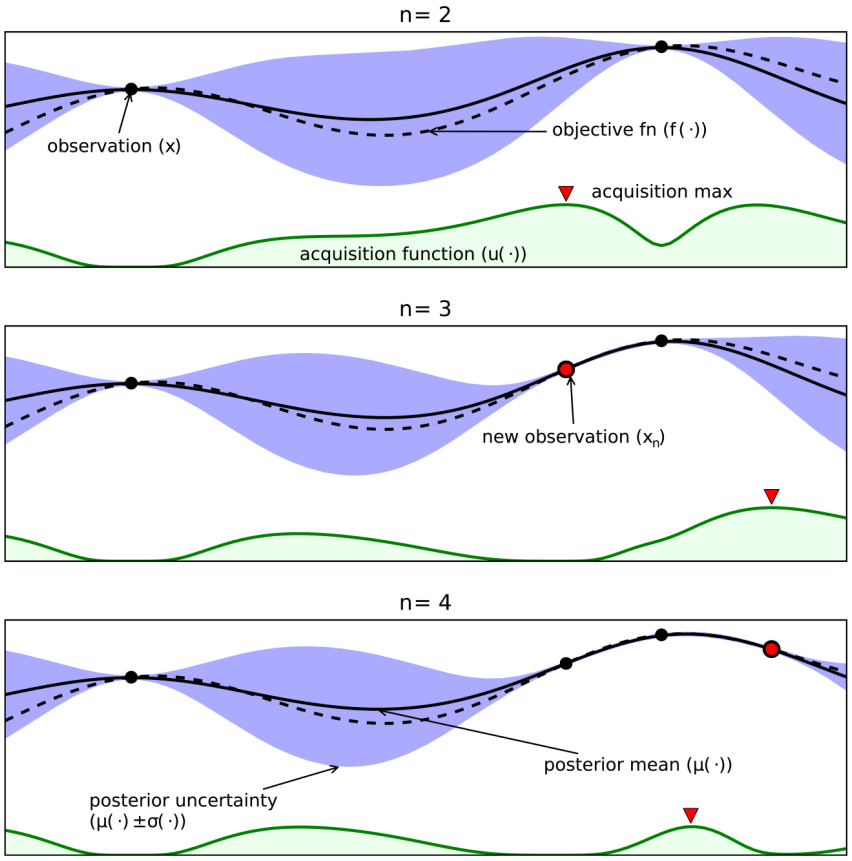


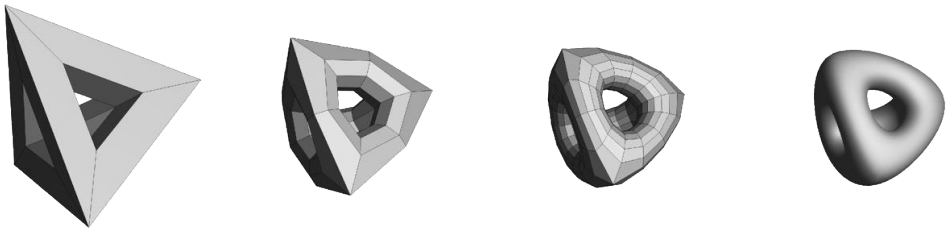
Figure 4: A visualisation of Bayesian optimisation of a simple one-dimensional objective function. Sourced from the 2015 paper by Shahriari *et al.* [6]

functions with more parameters (*i.e.* a higher dimensional objective function). However, as the number of parameters increases, the dimensionality of the search space increases, increasing the number of possible states. This will likely increase the number of objective function evaluations required before a solution is successfully converged to. As such it is important to attempt to minimise the number of separate parameters as much as possible when designing the objective function, which in the context of this report and project, is the generation of a mesh for use in an OpenFOAM simulation where the pressure loss can be evaluated, so the optimisation algorithm can minimise it.

2.3 Catmull Clark Subdivision

In 1978 Catmull *et al.* published a paper proposing an approach to recursively subdivide polygon meshes such that the limit surface (the resulting surface if the algorithm is applied infinitely) is a B-Spline surface, meaning it is a continuous smooth surface [8]. By recursively splitting edges and faces, and applying an algorithm which based upon the surrounding geom-

etry sets the location of the new points in the middle of each face, splits each edge surrounding the face creating a new vertex in between each existing vertex and updating the existing vertex locations, a mesh can be refined to a smooth surface, with the number of constituent polygons approximately quadrupling with each iteration (it is dependent on the number of edges on a face - a four sided face is split into four new four sided faces at each iteration). An example of this process is shown by Figure 5.



(a) Original Mesh (b) One Iteration (c) Two Iterations (d) Limit Surface

Figure 5: Successive application of the Catmull-Clark algorithm to a mesh [9].

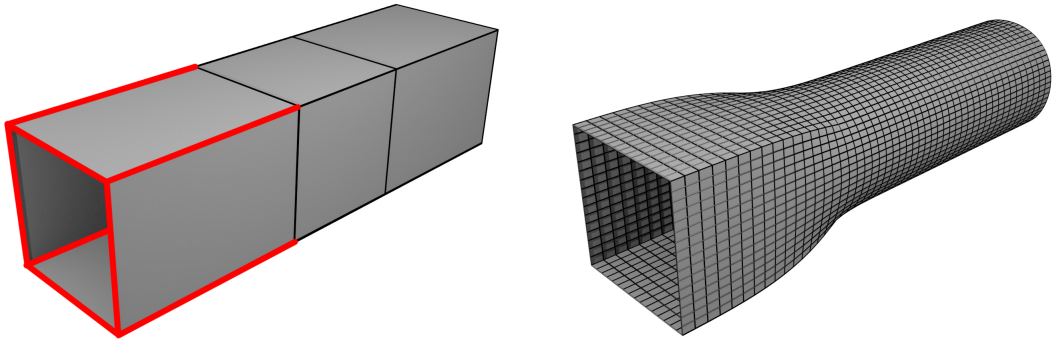
By using the Catmull-Clark algorithm, a low resolution mesh can be processed to become a high resolution and smooth mesh, potentially suitable for use as a boundary representation which may be used in OpenFOAM. By providing an optimiser with control of parameters in the low resolution mesh, the dimensionality of the optimisation problem may be reduced. An explanation to the operation of this algorithm is included in the theory section.

2.4 Enhancements on Catmull Clarke

On its own the Catmull-Clark algorithm is of limited use, fluids systems are not closed meshes, they have inlets and outlets, something which the basic Catmull-Clark algorithm does not have the capacity to deal with. Additionally, it does not in itself fix the issue of converting a round inlet to a rectangular outlet, a critical component in a draft tube.

In 1994, Hoppe *et al.* proposed a method for integrating infinitely sharp features in a different, but similar surface subdivision algorithm [10]. Following on from this work a team at Pixar research, DeRose *et al.*, developed it further, introducing infinitely sharp edges to the Catmull-Clark algorithm as demonstrated below in Figure 6, where the edges tagged to remain sharp are highlighted in red [9].

This behaviour is of value for this project as it allows for a simple way to smooth a mesh between a rectangular cross section and a circular cross section, an essential component in



(a) Original Mesh (b) Four Iterations

Figure 6: Infinitely sharp edges in the Catmull-Clark algorithm shown on geometry generated with Blender [11]. The original mesh highlights the edges tagged infinitely sharp in red.

representing a draft tube. Although it will not be delved into in this report, the paper also explores the use of variably sharp creases as a means to reduce the complexity of the original unprocessed meshes.

One of Pixar's primary motivations to under take this work was to allow eliminate issues that had arisen with the use of spline-based surfaces during animation work of the critically acclaimed animated feature film *Toy Story*. The animators working on the production had found that difficulties emerged when a model was distorted for facial and movement animations. The models consisted of patches of spline-based surfaces, and the seams between these started to become visible when the mesh was distorted, causing a break in a continuous surface. By using the Catmull-Clark approach, the low resolution mesh could be animated, and as will be explained in the next section, a polygon mesh can have arbitrary form, so character models could be formed without requiring the use of patches, eliminating the issue of seams. If 3D spline-based approaches were used in the context of a fluids system and the mesh distorted as part of the optimisation process, these same issues of seams cause defects in the continuous surface of the mesh, making it unsuitable for use. As demonstrated by the development work undertaken by Pixar, this issue can simply be eliminated by using the Catmull-Clark algorithm.

3 Theoretical Background and Design

In the attempt to develop an effective approach to the representation of geometry for an optimisation problem, techniques and technologies were pulled in from numerous disciplines of engineering and computer science, along with techniques developed by users of such technology,

such as the animation studio Pixar. In this section, the applied methods will be explored, first considering the theory and implementation of a custom Polygon mesh library, departing from the traditional use of Spline based techniques for boundary representation. Later the theory and methods behind the Catmull-Clark subdivision, will be explored, consider multiple implementations and enhancements of the algorithm providing features valuable to this project. Finally, an approach to the representation and formation of a fluids system will be proposed, making use of a linked-list data structure and a recursive approach to generating a readily parametrisable three dimension representation of the surface, which can be refined using the Catmull-Clark algorithm.

Where it is of value, Python code listings will be included in the body of this section of the report and will be referred to in the body text using as a 'listing' and its number in the document (*eg.* see Listing 4).

3.1 Polygon Modelling

3.1.1 Overview

The are many techniques through which a 3D surface can be represented and manipulated inside a computer. Historically one of the most common has been through the use of NURBS (or Non-Uniform Rational B-Splines), a technique with its origins in ship building, where a flexible piece of wood was bent around control points known as ducks. Since the advent of the computer age, and the use of computers in industrial design, NURBs have become a crucial part of the work flow and method behind many leading design packages. They do however have numerous issues attached to them. They can be particularly difficult to work with, especially when dealing with complex surfaces which will be distorted or altered during a process such as character animation (during the production of the first animated feature film, Toy Story, Pixar had to go to great lengths to ensure the animated NURBS surfaces did not exhibit visibly detractive seams). Secondly, compared to alternative technologies such as the polygon based modelling, or the Autodesk owned surface engine *T-splines*, they require a significantly larger number of control points to ultimately generate the same result. Finally, the structuring of NURBS makes it difficult to represent geometry where the necessary control mesh cannot be rectangular in form (*i.e.* the control point must maintain a grid structure). This can make it difficult to represent some geometric forms without resorting to piecewise methods and using separate patches to represent the geometry. The use of a polygon based modelling scheme can

remove this limitation, as the mesh can be formed in any manner.

In a Polygon mesh, the mesh is represented using a collection of vertices, edges and faces which define the surface. Vertices are singular points in the domain, which can than be connected together with edges. Using a closed loop of at least three edges, the boundary of a face can be defined. Multiple faces can then be used to define a surface. Though now ubiquitous in the discipline of artistic computer graphics, polygon modelling approaches are not frequently used in 3D Computer Aided Design (CAD) packages, which more generally promote the use of spline-based design approaches.

Typically faces in polygon based meshes are either three or four sided (resulting in terms such as triangle meshes to describe the former and quads the latter), though in practice faces may contain more edges. Due to the ability to break any polygon into triangles and the inherent simplicity of triangular faces (three vertices will always be coplanar with each other) this is often the dominant form, finding implementations in everything from graphics processing units to the STL file format often used for the exchange of 3D files.

There are numerous approaches available for representing polygon meshes and several open source software libraries available for encapsulating data and methods. This most prominent of these is OpenMesh, developed by the computer graphics group at RWTH Aachen University [12]. This is a powerful library written in C++ and using the Half-Edge data structure for mesh representation, explored later in this section. Its use was explored for this project as it has Python bindings allowing the library to be used in Python in the same manner as Numpy. However, in the early stages of this project it was considered overly complex for the requirements of this project, not warranting the difficultly of compiling the library to be compatible with Numpy and Python 3.5. Additionally, as an individual with little experience working with 3D mesh representation, the learning curve to use such a library effectively was prohibitively steep. Ultimately, it was decided a custom implementation of a mesh representation data structure would be created.

3.1.2 Vertex Only Data Structure

The simplest method for representing a mesh is the vertex-vertex data structure. Each face in the mesh is defined by a loop of vertices (*eg.* the three corners of a triangle). For each face, the positions of the vertices are stored as vectors in an ordered list. Though extremely compact, and easily understandable, this method does not lend itself to manipulation of the mesh,

as necessary in this project. Instead, each face is independent of the rest of the mesh, with its component vertices perhaps occupying the same space as an adjacent face, practically providing a continuous mesh, if a component vertex of a face is moved altered, without additional processing the equivalent vertices will not be moved. The STL (or *STereoLithography*) file format is an example of the use of this data structure. As demonstrated in Listing 1, each face is considered as a facet with a loop of vertices with their Cartesian coordinate positions stored as fixed point decimals. Additionally the normal vector of the face is stored, although provided the vertices have a consistent order (*eg.* always clockwise looking at it from a point along the normal vector), this could be calculated at runtime using a vector cross product.

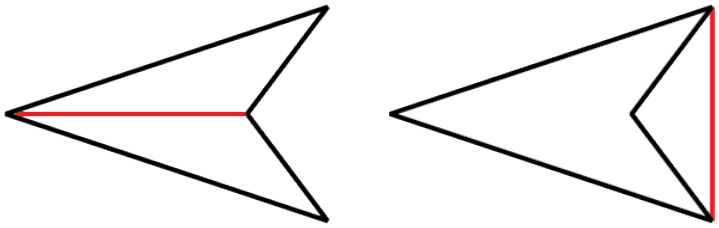
```
facet normal 0.712860 0.000000 -6.540000
  outer loop
    vertex 0.000000 1.090000 -1.090000
    vertex 3.000000 -0.763000 -0.763000
    vertex 0.000000 -1.090000 -1.090000
  endloop
endfacet
```

Listing 1: Sample from an ASCII encoded STL File

Based upon the work undertaken by other group members, the STL file format is the required format for geometry destined for use in OpenFOAM [2, 13]. The functionality required for working with STL files has been provided through the use of the STL module for Numpy on the Python package manager [14]. This provides an encapsulation of the methods required to work with and process STL files using methods and data types from the Numpy library. However, the limitation of the STL format to only represent triangles is a barrier to implementation of the Catmull-Clark algorithm, which requires the ability to represent quadrilateral faces. Consequently the STL library along with Numpy cannot be used for achieving the objectives of this project without further development. It is also worth noting that the

This does present the requirement of a method for converting quadrilateral faces in a mesh to triangles. At its most basic, this can be considered quite trivial - simply split a face in two between one of the two sets of opposite vertices. However this approach can break down in several situations as show in Figure 7.

Numerous alternative methods may be considered to address this, such as the use of the Delaunay triangulation algorithm or the ear clipping method. These methods are quite complex however and the likelihood of faces emerging where this approach would be necessary is limited. As such when triangulation has been implemented in this project, a simple fixed scheme has been used, splitting each quadrilateral face between the first and third vertex to create two



(a) Correct Case (b) Incorrect Case

Figure 7: Cases where a fixed approach to face triangulation can break down.

triangles. However, in further work, this may be a necessary addition to ensure generated meshes are of a sufficient quality.

3.1.3 Vertex-Face Mesh Data Structure

The Vertex-Face data structure is essentially an enhancement on the vertex only data structure previously described. The at the core of this structure is a repository of unique vertices in the mesh which have two parameters: their position and a set of faces that they form a part of. Alongside this, a set of faces is stored, with each face containing an ordered list of the vertices that define its boundary. Using this information, a mesh can be relatively simply manipulated, using the links between objects to evaluate querys (e.g. all the vertices around a vertex by considering the vertices connected to each surrounding face).

The Vertex-Face data structure was originally chosen for use in this project as it is simple to understand and implement in Python, simply uses classes with lists of pointers to other mesh elements. Additionally, though not of huge importance case, it is very low in memory use, using quite a compact approach to storing data.

Though the outline of this method and the approach to implementation has been briefly discussed, for full details please review the associated source code. Some features worthy of note were the ability to create a mesh from a set of face vectors, like those represented by the STL file. This is achieved by iterating through the vectors for each face and creating vertices if one does not already exist at that point in the mesh or referencing an existing vertex before each face is created. This ensures that faces are correctly connected together. It is however quite inefficient, requiring significant list searches.

In this project the vertex-face data structure was first implement to create the first implementation of the Catmull-Clark algorithm. However, due to issues of efficiency and stability, it became clear and different approach to the mesh engine was required.

3.1.4 Half-Edge Mesh Data Structure

Whereas the vertex-face mesh data structure stores most of its information in the vertices and faces, the half-edge mesh data structure is focused on edges. Each face is made up of a ring of half-edges (so named because an edge between two adjacent faces will be made up of two half edges). The half-edges have pointers for the next half edge in its face loop and the previous edge, along with pointers to its counterpart half-edge if there is an adjacent face and the vertex it points to (as illustrated by Figure 8. Using these pointers, the data structure can be quickly and efficiently traversed for manipulation and querying.

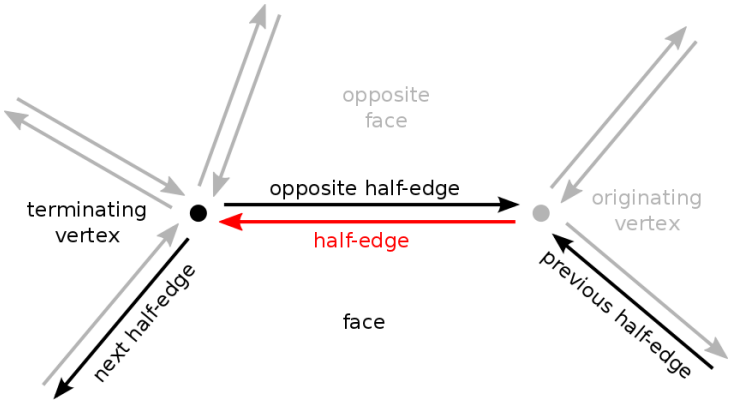


Figure 8: Diagram showing the geometric structure of the half-edge mesh data structure.

The choice of the half-edge data structure was motivated primarily by two factors. To incorporate creased/sharp edges into Catmull-Clark algorithm, it is necessary to store attributes against edges in the mesh. As the Vertex-Face structure lacks a list of edges, this isn't really possible, and as such an edge driven scheme is desirable. Secondly, the Vertex-Face data structure was difficult to make use of to manipulate the mesh in place, instead having to generate vectors and create a new mesh which was a significant drain on performance. The half-edge data structure has the facility for very efficient edge and face splitting, the former being where an extra vertex is added to an edge and the latter for splitting a face between a pair of vertices. Implementing these methods allowed for much more effective implementations of the Catmull-Clark algorithm, as will be explained later in this section.

For a more in depth explanation to the functioning of the Half-Edge mesh data structure, please review the documentation for the library OpenMesh, which uses the half edge data structure [15, 12]. As with the Vertex-Face data structure, the source code for the implementation used in this project will be made available.

3.2 Catmull-Clark Subdivision Surfaces

3.2.1 Overview and Motivation

As explained briefly in the previous section, the Catmull-Clark algorithm, developed by Ed Catmull and Jim Clark in 1978, is an algorithm which through successive subdividing of faces in the mesh and averaging of vertex positions, a mesh can be refined such that it approximates a smooth surface (the limit surface could be represented using spline-based patches which form a smooth surface). The following section will explore the theory behind the Catmull-Clark algorithm, along with details regarding the implementation of the Algorithm in Python.

3.2.2 Algorithm

The first step of the Catmull-Clark algorithm is to consider each existing face in the mesh, and calculate a new face point which will form the new vertex at the centre of each mesh. This is simply calculated by finding the centroid of a face by taking the mean of the position vectors of each vertex defining the boundary of the face. The new face point (Figure 9) is defined as f^{i+1} and its position is the centroid of f^i, where the subscript identifies the iteration rather than quantifying an exponent, and as such i is the current iteration of the algorithm and $i+1$ is the next.

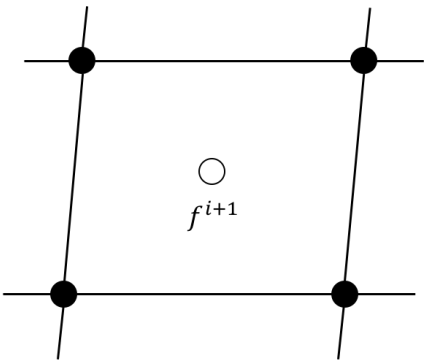


Figure 9: The new face point f^{i+1}, the first step of the Catmull-Clark algorithm (*Not to Scale*).

Next, each of the edges bounding the face are split, creating a new vertex in between each existing pair of vertices bounding the face. The position of this new vertex e^{i+1} is given by:

$$e^{i+1} = \frac{v_a^i + v_b^i + f_c^{i+1} + f_d^{i+1}}{4}$$

Where v_a^i and v_b^i are the vertices at either end of the edge that is being split and f_c^{i+1} and f_d^{i+1} are the new face vertices for the faces on either side of the edge. See Figure 10 for a visual representation.

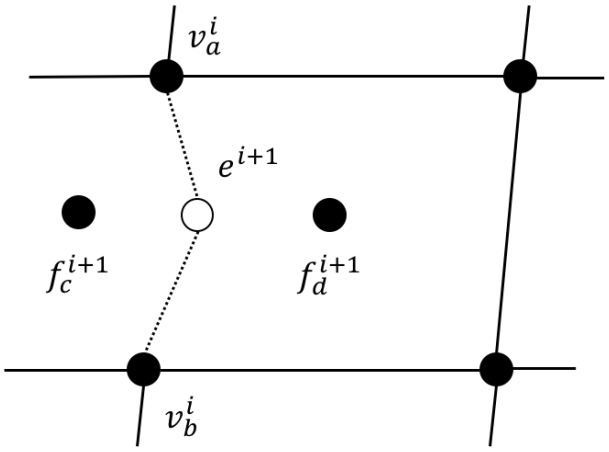


Figure 10: The splitting of an edge and creation of the new vertex e^{i+1}. (*Not to Scale*)

Now the existing vertices must be moved to a new position to smooth out the mesh. This rule was designed by Catmull *et al.* to achieve what they considered to be aesthetically best, resulting in the 'nicest' limit surface, but in the context of this application, this is not a significant problem. It should be noted however, that when a plane is subdivided using the algorithm, it does not form a perfect circle, instead being very slightly oblong. This is likely due to the arbitrary nature of this function, which could be rectified if further work is carried out. The following function was derived for computing the new position (v^{i+1}) of an existing vertex (v^i):

$$v^{i+1} = \frac{Q}{n} + \frac{2R}{n} + \frac{v^i(n-3)}{n}$$

Where Q is the average of all the new face vertices,f^{i+1}, for the faces around v^i and R is the average of all the midpoints of the old edges which are incident with the vertex. The variable n is the valence of the vertex in question (the number of incident edges with the vertex). This results with a ring of vertices on the edge of the existing face and vertex at the centre, similar to the structure shown in Figure 11.

From this ring of edges, new faces may be formed by creating a new edge between each new edge vertex and the new vertex at the centre of the face, so in the case of the example above, four new faces are formed. It is worth noting, that all new faces created will be quadrilateral after one iteration. This process can then be repeated to recursively refine the mesh, creating

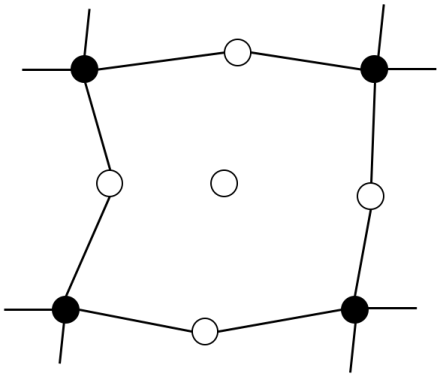
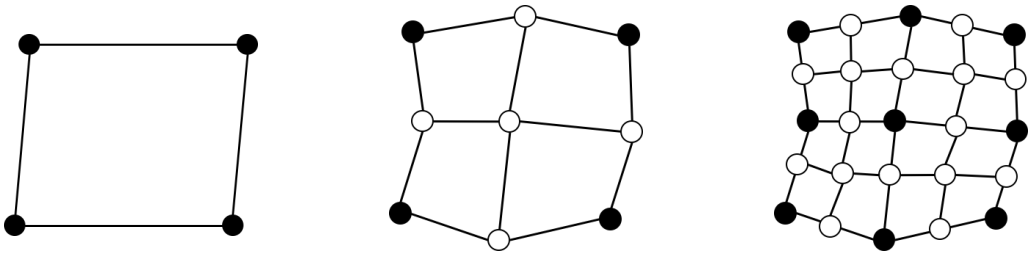


Figure 11: The ring of vertices around the existing face before the new edges are created subdividing the face. (*Not to Scale*)

new faces at each step as shown in Figure 12.



(a) Original (b) One Iteration (c) Two Iterations

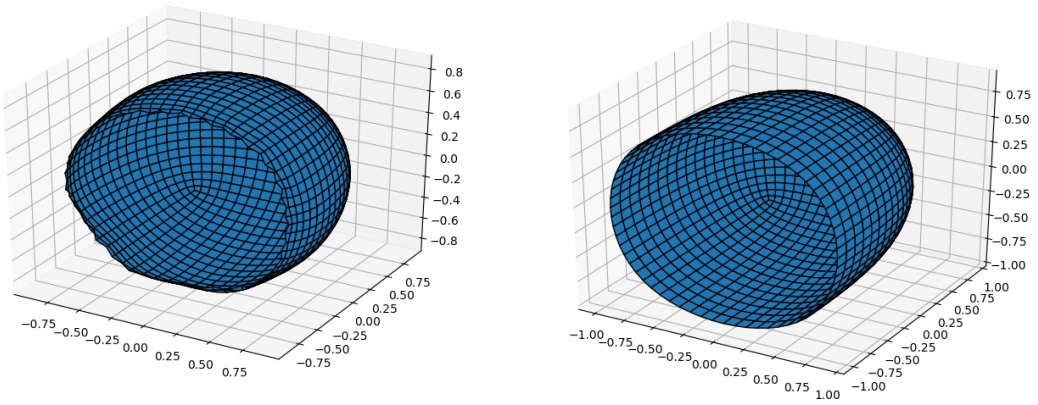
Figure 12: Representation of the resulting geometry from applying Catmull-Clark for two iterations with vertices that are new from that iteration being white with a black outline. (*Not to Scale*)

3.2.3 Special Cases and Sharp Edges

As shown in Figure 13, if the standard Catmull-Clark rules are applied at the edge of a mesh, on the boundary, the geometry is distorted, which is undesirable. To eliminate this issue, a special rule must be applied to boundary edges and vertices. In the case of new edge points, the influence of adjacent faces is ignored, and the new vertex is placed at the edge midpoint. In the case of a vertex which is on a mesh boundary, the new vertex position is given by the following function:

$$v^{i+1} = \frac{v^i}{2} + \frac{\Sigma e^{i+1}_{\text{boundary}}}{4}$$

Where $\Sigma e^{i+1}_{\text{boundary}}$ represents the sum of the new edge points on the boundary edges incident with the vertex. The application of this is shown in Figure 13b.



(a) Without (b) With

Figure 13: Boundary awareness and the Catmull-Clark algorithm on an cube with one side missing.

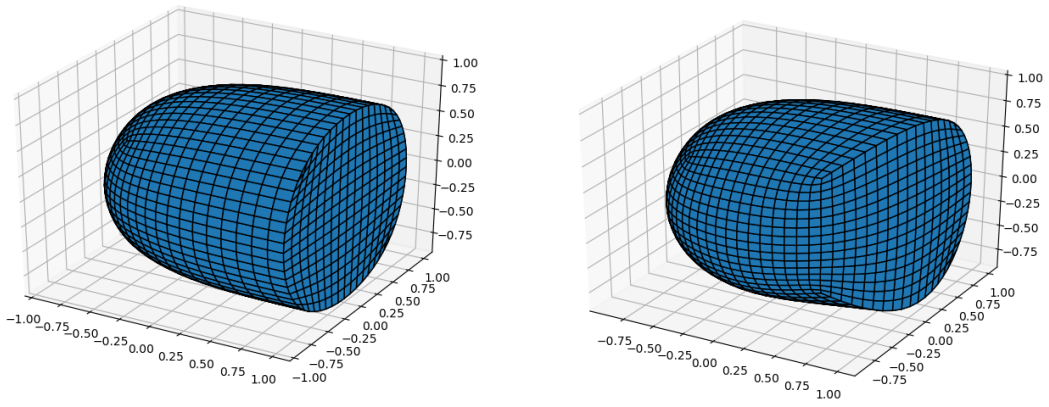
The final aspect of the Catmull-Clark algorithm that will be utilised in this project is the facility to add infinitely sharp edges or creases to the algorithm. This will allow for the smooth interfacing between rectangular and round cross-section geometry. Similarly to the boundary case, the new edge vertex position of any edge tagged as creased is simply its midpoint. The rules for new vertex positions are slightly more complex. If the vertex is incident with one sharp edge, no change is made to the existing algorithm. But if the vertex is incident with two sharp edges, a new function is used to calculate the new vertex position:

$$v^{i+1} = \frac{e_0^i + 6v^i + e_1^i}{8}$$

In this case e_0^i and e_1^i are the position vectors of the vertices at the the other end of the creased edges. Finally if three or more sharp edges are incident with a vertex, its position remains the same during that iteration:

$$v^(i+1) = v^i$$

The effect of the application of this addition to the algorithm on a cube with a face with four sharp edges and three sharp edges is shown in Figure 14.



(a) Four Sharp Edges

(b) Three Sharp Edges

Figure 14: Different sharp edges on subdivided cube.

3.2.4 Initial Python Implementation

For this project, this algorithm was first implemented making use of the previously described Vertex-Face data structure implementation. This provided the mechanisms for the manipulation and processing of data from the existing mesh at each stage and the creation of the new resulting mesh after each iteration. The algorithm held the following structure:

- Iterate through each face object stored in the list of faces in the mesh and add a parameter to each face called *new_face_point* which is the centroid of the face.
- Iterate through each edge of each face and add a parameter called *new_edge_point* which is calculated according to the algorithm previous explained. A check is conducted and the calculation is skipped if the edge already has the parameter set (faces share edges so this approach avoids unnecessary duplication).
- Iterate through each vertex, adding a *new_vertex_point* parameter calculated through an application of of the new point position function previously discussed.
- Create an empty list of vectors which will describe the subdivided mesh.
- Iterate through each face in the mesh, and then each edge in that face creating a new vector, starting from the new face point, going to the new edge point. Then evaluate the vertices at either end of that edge and check if it has already been used for a vector for this face. If it has not been used, add the new vertex point parameter to the new vector and then the new edge point on the next edge along in the face, completing a four location

vector defining the quadrilateral face. If this is repeated for each edge, vectors are created which represent all of the new faces which make up the existing face.

- Create a new mesh using these vectors. This is the subdivided mesh.

Though this approach proved to work for basic cases (e.g. the first few iterations of the subdivision of a cube), the creation of a new mesh using vectors was incredibly inefficient, with the time taken to create the new mesh growing exponentially which each iteration, due to the list search required to check if a vertex already exists at that location. This made the algorithm impractically slow after two to three iterations. Additionally, as shown in Figure 15, after the fourth iteration, the algorithm displayed instability, resulting in an incorrectly formed mesh.

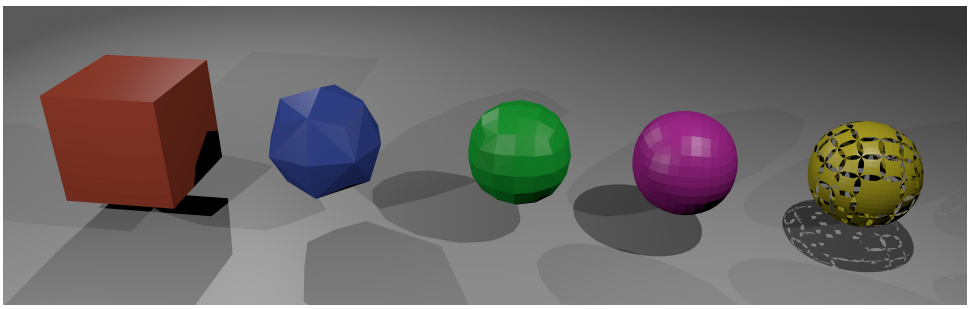


Figure 15: Successive iterations of the Catmull-Clark algorithm on a Cube using the initial Python implementation, exported as an STL and rendered in Blender.

Investigations into this issue have come up short and the cause of this has not been identified (mostly due to the fourth iteration taking around 40 seconds to calculate), but it, along with the need for an algorithm which could better represent edges to further enhance the Catmull-Clark implementation, lead to the development of tools for using a half edge mesh data structure and a revision of the Catmull-Clark algorithm implementation to use this mesh representation.

3.2.5 Enhanced Python Implementation

After redeveloping the base mesh methods from scratch, moving from a Face-Vertex data structure to a half-edge data structure, the a new Catmull-Clark algorithm implementation was created, also implementing boundary case handling and infinitely sharp edges. The notable changes from the previous version were:

- Updates were made to the part of the algorithm calculating the new edge points. Rather than iterating over edges, it iterated over half-edges. If the half-edge was tagged as

creased, the new edge point was simply the existing edge midpoint. Additionally, if the half-edge did not have an opposite half-edge, it must be on a boundary, and consequently the faces are not considered.

- Similarly the methods described for boundary vertices and vertices with vary numbers of incident sharp/creased edges has been implement based upon the algorithms expressed previous.

- A new approach is used however to form the new faces on the mesh, taking advantage of the features of the half-edge mesh data structure. First each of the edges of a face is split, creating a new vertex at the new edge point position (logic has been implemented to ensure that adjacent faces don't lead to the splitting of the edge again). Then the face being subdivided is split between the first two new edge vertices, creating a bisector across the face. This bisector is then split to have a vertex at its midpoint which is then moved to the new face point for the original face. The faces are then split between this new point and each of remaining new edge points. This is far, far more efficient than the generation of a new mesh from vectors and has the added benefit of being able to simply copy meta data (*eg.* whether or not an edge is crease) to the new edges created during the division process.

3.3 Parametrised Surface Model

3.3.1 Overview and Motivation

The primary objective of this project is to form a method through which the geometry of a fluids system (*eg.* a draft tube) may be represented simply in such a manner that its parameters may be manipulated by an optimisation algorithm. So far this report has focused on the theory and implementation behind the representation of polygon meshes and the Catmull Clark algorithm, but not yet the initial formation of the mesh itself.

The method proposed here takes inspiration for the loft feature in many computer aided design tools (*eg.* Solidworks, Autodesk Fusion 360) by essentially linking control profiles (two-dimensional parametrised drawings) together. In this case predefined system elements can be created using a polygon mesh approach, allowing for the trivial construction of potentially complex geometries, such as the elbow of a draft tube. Provided an element can expose a ring of four vertices representing the inlet and another ring representing the outlet, elements can be

simply knitted together.

3.3.2 Knitting Components Together

The proposed structure for representing components is to use a linked list. In a linked list the first element, or head, contains a pointer to the next element in the list. This next element then has a pointer to the next element and so on and so forth (see Figure 16). This structure can be easily implemented in Python by creating a class to encapsulate a geometry section and that class can contain an instance variable, which is a pointer to the next section in the system. For example a draft tube could be considered as a Cone to a Elbow to a Diffuser.

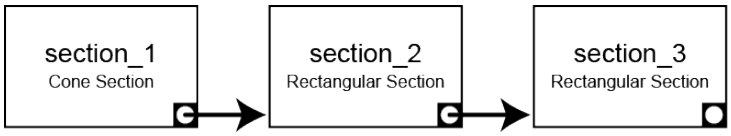


Figure 16: Linked-list of fluids system elements. Each section has a pointer assigned to the next section.

By using an abstract class, which is then inherited in a class for a particular type of section, a consistent structure and interface can be defined. For example, or sections will have a origin location in Cartesian coordinates, then their own geometry can be defined locally within itself.

Further more, the linked list structure allows for a recursive algorithm for constructing the low resolution mesh to process with the Catmull-Clark algorithm. By calling a function called *generate_mesh* on the first section in the structure, the first element can call a method implemented in its child class for generating the geometry for that section. This function has a template in the abstract class, which is overridden in the child class to create the geometry required for that section type. This function returns three things: a mesh object containing the geometry for that section, and a ring of vertices for the inlet and outlet of the section. This geometry can then be merged into the main mesh which is being generated and a connection built between the outlet vertices of the last element and the inlet vertices of the new component. This process can continue, recursively calling the *generate_mesh* function on each next section, passing the mesh as it currently stands, until the last section.

3.3.3 Component Sections

For this prototype to demonstrate this concept, two elements have been implemented: a cone element, which is circular in profile and can have different diameters for the inlet and outlet, and a rectangular section, which is like the cone but resulting geometry will be rectangular in form.

The cone element has expanded the abstract class to create a structure able to represent a cone shaped geometry with different inlet and outlet sizes. In the resulting low resolution mesh, a cone should be represented by a box shaped element without creasing along its side but with creasing at its end loops (though this may be disabled if a smooth change from the cone is required. Additionally the box mesh must be of a size such that when it subdivides with the Catmull-Clark algorithm, its diameter is the one defined in the object parameter. This was found empirically to be a factor of around 1.09, so a square of with sides 1.09 units long will subdivide to be approximately a circle of 1 unit in diameter. A function was implemented in this class to generate this particular geometry.

The rectangular section (the same geometry as the diffuser component in the draft tube) allows for the representation of a component with a rectangular cross section with the ability to change the dimensions on both the inlet and outlet section. Generally speaking all edges in this geometry will be sharp, but there may be a reason that a smooth geometry could be required entering or leaving the component so the edge loops at the inlet and outlet can both be made smooth if necessary. The outlet can also be moved relative to the input. The resulting structure from both of these elements can be seen in the next section along with more details regarding their functionality.

4 Presentation of Results and Deliverables

4.1 Overview

To present the results and evaluate the successes and shortcomings of this project, this section will present an example of the construction of a fluids system problem including an example on the use of the library to form the system, a discussion as to how an optimiser could be integrated with the mesh generator to alter parameters and how the Catmull-Clark algorithm implementation may be applied to produce an output mesh. This mesh will then be exported as an STL file at various different levels of subdivision and evaluated using Blender. Unfortunately, due to time restrictions, it has not been possible to implement the necessary elements to represent a draft tube mesh, instead a mesh has been designed which incorporates components that exist within a draft tube.

4.2 System Definition

The following section will explore the assembly of the system step by step, using the small library of pipe sections implemented. The code listings are taken from the same file and the line numbers indicate the position in the file. By wrapping this code in a function, which returns the STL file, and takes the optimisation variables as parameters and passes these to the require geometric properties, this could be quite trivially be hooked into an optimisation tool.

The first step is to import the necessary modules developed: the mesh library and the pipe sections library along with the Catmull-Clark implementation, as shown in Listing 2 below:

```
1 import app.pipes as pipes
2 from app.catmull_clark import catmull_clark
3 import app.mesh as m
```

Listing 2: Importing the necessary modules to create the system.

Next, the first element in the system is defined, in this case a Cone element, similar to the inlet of a draft tube connected to the turbine outlet (see Listing 3).

From this the next section is defined, a rectangular section (see Listing 4). Note that the previous section's next attribute is set to be this new element.

```
section_1 = pipes.ConeSection()
section_1.outlet_radius = 0.7
section_1.length = 3
section_1.smooth_out = True

# Other avaliable parameters

# section_1.inlet_radius = 1
# section_1.outlet_translation_horizontal = 0
# section_1.outlet_translation_vertical = 0
# section_1.smooth_in = False
# section_1.x  = 0
# section_1.y = 0
# section_1.z = 0
```

Listing 3: Defining the first section in system.

```
section_2 = pipes.RectangularSection()
section_1.next_element = section_2
section_1.x = -1
section_2.y = 1
section_2.z = 6
section_2.inlet_width = 1.6
section_2.inlet_height = 2
section_2.outlet_width = 2
section_2.outlet_height = 1
section_2.outlet_translation_horizontal = 0
section_2.outlet_translation_vertical = 0.5
section_2.length = 3
section_2.smooth_in = True
section_2.smooth_out = True
```

Listing 4: Defining the second section and setting it to be the next element after the first element.

The final section is another rectangular section, with a definition shown in Listing 5. Please note that parameters that have not been altered from default have not be shown here to save space.

```
section_3 = pipes.RectangularSection()
section_2.next_element = section_3
section_3.z = 11
section_3.inlet_height = 2
section_3.length = 2
```

Listing 5: Defining the final section.

Now, by creating a new mesh using the imported mesh library and passing that to the generate mesh function on the first element, resulting in a recursive assembly of the system mesh, and then calling the draw function for the mesh (Listing 6), a visualisation of the mesh can be seen (Figure 17).

```
mesh = m.Mesh()
section_1.generate_mesh(mesh=mesh)
mesh.draw()
```

Listing 6: Creating a drawing the low resolution mesh.

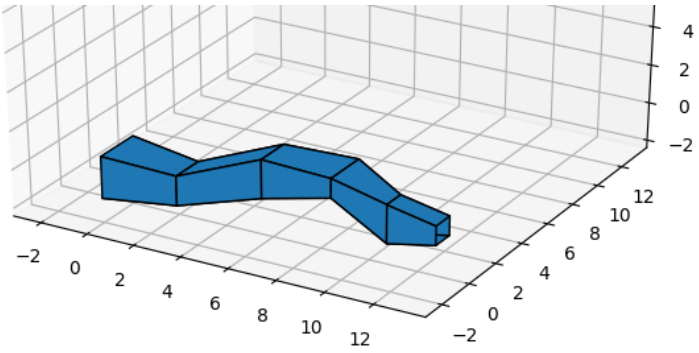


Figure 17: Matplotlib plot of the low resolution mesh generated using the above method.

At this point no smooth surfaces can be been as it is simply a low resolution mesh of quadrilateral faces. Some edges have been tagged as sharp according to the definitions provided in each sections respective generate geometry function, which will emerge when the Catmull-Clark algorithm is applied. Listing 7 shows the code used to apply the Catmull Clark algorithm to the mesh for four iterations, generating a STL file at each level. It should be noted that as the triangulate function mutates the mesh (due to difficulties in duplicating the mesh object), the mesh is being regenerated from scratch each time. The time module has also been imported to provide timing information.

```
mesh = m.Mesh()
section_1.generate_mesh(mesh=mesh)
mesh.draw()

import time
iterations = 4

# Loop for number of required iterations. Prints stats to console.
for iteration in range(iterations):
    start_time = time.perf_counter()
    catmull_clark(mesh)
    end_time = time.perf_counter()
    print("Iteration", iteration + 1, end_time - start_time)
    print("Face Count", len(mesh.faces))

# Save STL
mesh.save_stl('example' + str(iterations) + '.stl', mutate_mesh=True)
```

Listing 7: Generating STL Files using the Catmull Clark algorithm. The interactions variable is changed depending on the number of required iterations. Note that this duplicates some lines from the previous listing.

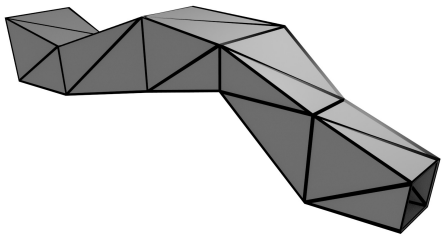
The results for running this code for four iterations is shown by Figure 18 on page 30. Renders of the generated surface from two points of view on the left and right hand side of the mesh are shown, with the mesh wire frame composited on top, along with the iteration number, the time it took for that iteration to execute and the resulting of quadrilateral faces (it was necessary to triangulate for exporting to STL to render in Blender so the renders have double the number of faces than the actual geometry straight from the Catmull-Clark function).

As expected, the mesh is progressively smoothed as the algorithm further subdivides faces on the mesh. There are some points worth drawing attention too where either the Catmull-Clark algorithm, or this implementation of breaks down.

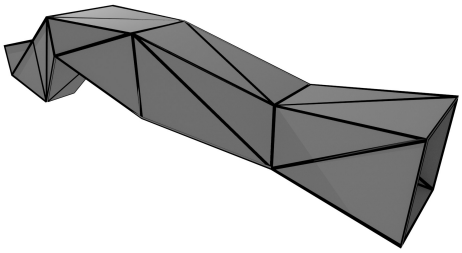
In certain circumstances the boundaries of do not seem to maintain their correct form, distorting when they should be a sharp creased rectangular form (see Figure 19). This may be due to a fault in the implementation of the Catmull-Clark algorithm in how it has been set-up to handle boundary edge cases, but as will be built upon over the next paragraphs, it is more likely there is a fault in the implementation of the mesh engine, which is failing to identify when a half-edge is on a boundary correctly.

A similar issue is the appearance of abnormally flat or bulged areas without a smooth transition into them or smoothing across there surface. The exact origin of this issue has also eluded investigation, but there are some indications that the mesh engine is incorrectly forming faces at some points of mesh creation process. When the number of half-edges without opposite half-edges is counted, twenty four are found where there should only be eight (the eight edges at either end of the system). If the flaw is in the mesh engine, a suitable course of action if this approach is taken in future work, would be to focus on the integration of a well maintained and comprehensive mesh library (*eg.* OpenMesh or CGAL [12, 16]).

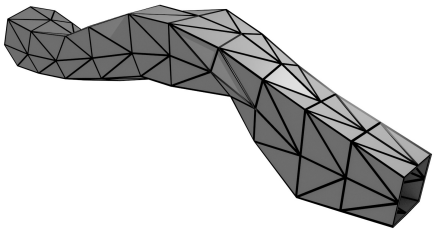
Additionally, there may be a need for more complex interpolation in the Catmull-Clark, subdivision algorithm or simply just a more robust implementation. There are various potential options for third party libraries implement the Catmull-Clark algorithm, the most promising of which is the Pixar R&D project *OpenSubdiv* which was born out of paper by DeRose *et al.* and is now a highly featured software library implementing many complex features such as adaptive subdivision (only subdividing faces where the geometry complexity deems it necessary) along with the ability to use variably sharp creases [17].



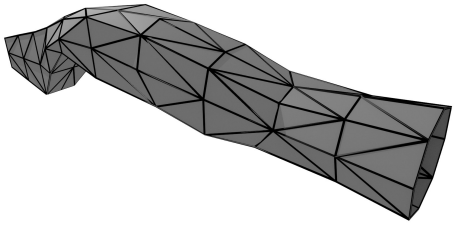
(a) Left, Iter: 0, Time: N/A, Faces: 20



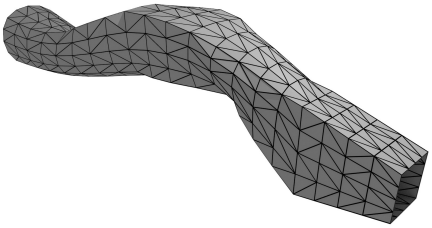
(b) Right, Iter: 0, Time: N/A, Faces: 20



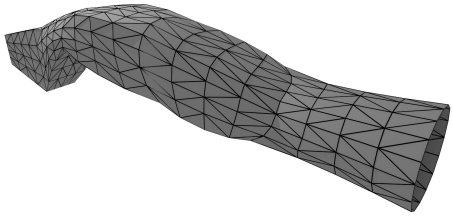
(c) Left, Iter: 1, Time: 7ms, Faces: 80



(d) Right, Iter: 1, Time: 7ms, Faces: 80

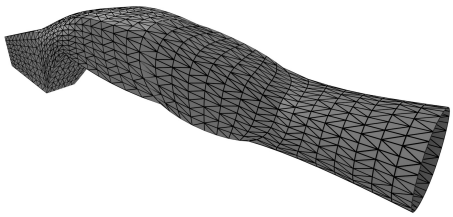


(e) Left, Iter: 2, Time: 32ms, Faces: 320

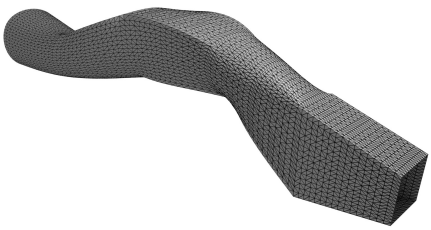


(f) Right, Iter: 2, Time: 32ms, Faces: 320

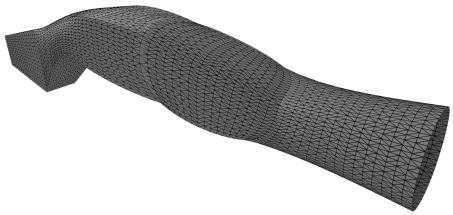
(g) Left, Iter: 3, Time: 134ms, Faces: 1280



(h) Right, Iter: 3, Time: 134ms, Faces: 1280



(i) Left, Iter: 4, Time: 567ms, Faces: 5120



(j) Right, Iter: 4, Time: 567ms, Faces: 5120

Figure 18: Successive applications of the Catmull-Clark algorithm to the mesh described and created using the previously shown source code.

Figure 19: Breakdown of sharp creases at mesh boundaries.

5 Discussion and Conclusions

This project and report has successfully demonstrated the feasibility of using a generic but customisable approach for the definition of fluid systems to generate a low complexity mesh which can be refined using the Catmull-Clark algorithm to then be used with OpenFOAM and an optimisation tool. There is however significant further development necessary before this concept could be deployed as a usable production tool.

The pathway on this project has led to an assessment and implementation of two different types of polygon mesh data structures, the Face-Vertex and Half-Edge structures and the implementation of the capability to convert either two and from a STL file, a standard file format for exchanging 3D geometry. The Catmull-Clark algorithm has been reviewed and explored along with several extensions to it. It has provided a means to create a high resolution mesh, potentially suitable for integration with OpenFOAM, from a low resolution mesh which can readily be parametrised such that it can be incorporated into a optimisation process. Finally an approach to parametrically creating a fluids system has been developed and undergone some limited testing. Some flaws were uncovered in the algorithms as a result, the causes of some of which has eluded investigation, but some possible cause have been posited.

Though this project has not truly delivered upon its objective of finding an approach to optimise the geometry of an entire draft tube, it has made steps towards that goal, providing a frame work and approach that may be of interest to explore in the future.

6 Project Management and Health and Safety

6.1 Project Management

6.1.1 Time Management

Maintaining an effective use of time and keeping focused is often a challenge. In truth, this project has been particularly challenging during the duration of which many distractions and issues arising which have been both a distraction and barrier to work. Additionally, as a research focused project with no fixed objective or deliverable from the start, it can be difficult to find and maintain an effective direction. Despite these difficulties, I have endeavoured to apply myself and make the most of my time available. This has also unfortunately been hampered by personal health issues over the last term, and as such the progress made has been limited. It is also worth noting that particular difficulties that arose, such as originally attempting to build OpenMesh and being unsuccessful, and consequently going down the path of a custom implementation, could have eliminated major difficulties later if they had been overcome, resulting in greater exploratory work rather than time being spent unnecessarily on a derivative implementation. Ultimately however, it has provided the opportunity to learn from mistakes in the future, and develop new skills and hone existing ones which will aid future work as an individual.

6.1.2 Source Control and Development Tools

As a project focusing on the development of software tools in Python, it was important to use a tool for keeping track of changes and taking backups of source code as the project progressed, and on some occasions facilitating collaboration with other group members. The software tool of choice for this was to use the Git version control system, and over the course of this project I have made use of Git with remote depositories at both my own personal GitHub account and on the university's GitLab installation. Git has also been used for writing this document in LaTeX, taking advantage of Overleaf, an online tool for writing LaTeX documents which exposes a Git repository for each project to allow for off-line editing and revision tracking.

Alongside Git for managing and versioning my work, I have made use of other software tools to aid in development. The code itself has been written in Microsoft's Visual Studio code, taking

advantages of its tools such as a Python debugger. For the evaluation of generated meshes, the open source 3D modelling and animation tool Blender has been used, often comparing its implementation of the Catmull-Clark algorithm against the one developed for this project.

6.2 Health and Safety

6.2.1 Soldering

At one point during the project, I was asked by the experimental team to help with assembling the electronics used to collect data during the experiment. This involved the use of soldering equipment in the electronics laboratory and some power tools. When soldering, care was taken to ensure that work area was kept clear of potentially flammable materials which could be ignited by the heat of the soldering iron. Safety glasses were worn, to protect eyes from debris from cutting wires or drips of solder, and care was taken with the iron to prevent accidentally burns. During breaks, or when leaving the room was required, the soldering iron was turned off and placed safely in its holder. Additionally, a sacrificial board was placed on the workshop desk to protect the table from burns for the soldering iron head.

6.2.2 Health

Though there were no significant necessary safety considerations in the undertaking of my individual project, it is pertinent to make a consideration of health (which is increasingly starting to consider individual well being), the often sidelined sibling of safety, as over the course of the project this has affected both myself and other group members. Such situations are often difficult to handle and even harder to predict, from the perspective of both an individual and the group they are part of. Over the course of the past year, I have come to learn that I am autistic and I am starting to understand both the gifts and handicaps this has brought to my life. I can find it extremely difficult to focus at times, a difficulty particularly exacerbated by a fatigue related issue I seem to have been suffering from. I am thankful to my group and supervisors for being understanding of this issue and doing their best to be supportive. As the work I was undertaking was not on the critical path, the impact of my health related issues on the larger project was minimal. Ultimately an impossible balance must be struck between maintaining ones health and well being and ensuring the project successfully progresses.

7 Contribution to Group Functioning

As a group project, my work and activities fitted into a wider picture in a multidisciplinary group with all of my colleagues working on other areas and facets of the project. Our group of eight was initially split into three sub-groups, with three of us working on CFD models and meshing techniques in OpenFOAM, two of us working on the experimental tools and approaches and the remaining three of us working on the the optimisation and machine learning section of the project.

As a group we held weekly meetings, generally on the Friday, to review our progress and to discuss emerging issues, challenges and discoveries that had a bearing on the work undertaken by other team members. These meetings were operated formally with a rotating chairperson and secretary position, the first to manage and coordinate the meeting, and the other to take notes in the form of meeting minutes which were then distributed digitally after the meeting. It was also the responsibility of the chairperson to write the agenda for each meeting - although ultimately this became quite generic, only being altered when extraordinary points of discussion needed to be considered. These positions were rotated to ensure that each group member got the opportunity to fulfil each role at least once during the course of the project.

A primary feature of these meetings was a progress report from each sub-group, feeding back and reporting on the work they had undertaken during the previous week. Often, this would lead to the identification or appearance of issues both technical and logistical and the meeting space would provide a forum to discuss these issues and for others in the group to provide input to help with the formation of a solution.

Alongside the group members, these meetings were also attended by our project supervisor, Professor Gavin Tabor, and two post-doctoral researchers working working with Professor Tabor, Alma Rahat and Steven Daniels, who are working on projects related to ours and helped to advise us.

Personally, I was part of the optimisation sub-group, with our central goal being the development and refinement of a Bayesian optimiser for efficiently optimising the draft tube geometry, to maximise pressure recovery. After receiving an introduction from Alma, we initially worked together to improve our understanding of the optimisation process and to implement our own Bayesian optimiser. After this we started proceeding down our own areas of interest and research pathways, with Carol Ng working on the primary implementation of the Bayesian

optimiser and Rhys Gilbert working on the use of adjoint methods to enhance the optimisation [2, 3].

Alongside my primary work path, I also spent some time during March and early April helping the experimental team. Due to unforeseeable circumstances in the health of one of the experimental team members, the implementation of the electronic systems for data gathering and processing during the experiments was running behind. Given my degree programme is Electronic Engineering and that my work was not on the project critical path for the project (in that not prioritising it was not going to adversely affect any other parts of the project), I was tasked with helping with the assembly and soldering of the electronic components (see Figure 20). This primarily involved the completion of some differential amplifier circuits using operational amplifiers on strip board and soldering together Arduino-based components. The differential amplifiers were latter connected to pressure sensors, which connected to pressure tappings in the 3D printed draft tube models, to collect pressure information [18]. For further details on this aspect of the project, see Tom Dye's individual report [19].

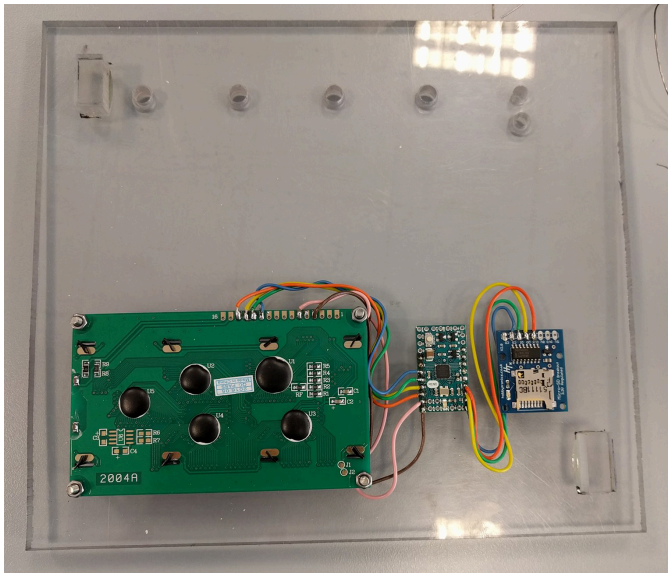


Figure 20: Part of the electronics instrumentation used by the experimental team. The component of the left is a LCD screen for displaying live information, the middle an Arduino Nano and on the right a SD card Arduino shield for storing data on an SD card.

Additionally, on a couple of occasions, I assisted the CFD team with some Linux and Python issues: utilising a significant amount of Python and Linux system administration experience, I was able to quickly resolve problems quickly which may have otherwise taken significant time to solve. Most notably, I was able to help Carol Ng and Patrick Burns with configuring the Python environment on the simulation machine to match the former's development environment using the requirements functionality in the Python package manager, *PIP* [13].

8 References

[1] Jean-Mathieu Gagnon, Vincent Aeschlimann, SeÌĄbastien Houde, Felix Flemming, Stuart Coulson, and Claire Deschenes. Experimental Investigation of Draft Tube Inlet Velocity Field of a Propeller Turbine. *Journal of Fluids Engineering*, 134(10):101102, 2012.

[2] Carol Ng. Implementation of Bayesian Optimiser on CFD Modelled Draft Tube. Technical report, University of Exeter, 2017.

[3] Rhys Gilbert. An investigation into Bayesian methods for CFD optimisation. Technical report, University of Exeter, 2017.

[4] R Pitz and J Daily. Experimental study of combustion in a turbulent free shear layer formed at a rearward facing step. In *19th Aerospace Sciences Meeting*, Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, 1 1981.

[5] Steven Daniels and Alma Rahat. Shape Optimisation Using Computational Fluid Dynamics and Evoluionary Algorithms. In *OpenFOAM Workshop*, number April 2017, Portugal, 2016.

[6] J. Melorose, R. Perroy, and S. Careas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 1(1):148–175, 1 2015.

[7] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13:455–492, 1998.

[8] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 11 1978.

[9] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques SIGGRAPH 98*, pages 85–94, 1998.

[10] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*, pages 295–302, 1994.

[11] Blender Foundation. Blender, 2017.

[12] RWTH Aachen University. OpenMesh, 2017.

[13] Patrick Burns. The development of an automatic numerical draft tube model. Technical report, University of Exeter, 2017.

[14] Rick van Hattem. numpy-stl, 2017.

[15] M Botsch, S Steinberg, S Bischoff, and L Kobbelt. OpenMesh âĂŞ a generic and efficient polygon mesh data structure. *OpenSG Symposium*, 2002.

[16] The CGAL Project. *{CGAL} User and Reference Manual*. CGAL Editorial Board, 4.9.1 edition, 2017.

[17] Pixar. OpenSubdiv, 2017.

[18] James Angus. Experimental Investigation of Flow Inside Draft Tubes with Varied Diffuser Geometry. Technical report, University of Exeter, 2017.

[19] Thomas Dye. Design and Prototyping of a Custom Data Collection System for testing Scale Model Draft Tubes. Technical report, University of Exeter, 2017.