CSC3003S

Mosaic Mecca: Capstone Stage 4 Report

Josh Buchalter
BCHJOS003
bchjos003@myuct.ac.za

Michael Sutherland
STHMIC006
sthmic006@myuct.ac.za

Luke Bell
BLLLUK001
blluk001@myuct.ac.za

Client:
Prof. James Gain
jgain@cs.uct.ac.za

Tutor:
Andrew van Rooyen
wraith.andrew@gmail.com

September 23, 2015

The result of this project has been the invention and creation of a standalone mosaic generator which users can use without the need for expensive photo-editing packages such as Adobe Photoshop. The program can take in any image chosen by the user and produce a mosaic representation of the image given user-defined features such as grout colour, tile size and edge features (see section 4 and more specifically sec 4.4.2). Once the mosaic is generated and shown on the screen, the user can then either add/remove/redraw edges as well as export the mosaic as a .jpg file.

Contents

1 Introduction **2**

2 Requirements Captured **2**
2.1 Use Case Narratives . . . 2
2.2 Other requirements . . . 3

3 Design Overview **4**

4 Implementation **6**
4.1 Framework . . . 6
4.2 Classes . . . 6
4.2.1 Frsutum . . . 6
4.2.2 EdgeCurve . . . 7
4.2.3 VoronoiDiagram . . . 7
4.2.4 DirectionField . . . 7
4.2.5 Mosaic . . . 7
4.3 GUI . . . 7
4.3.1 StartScreenGUI . . . 7
4.3.2 VectorFieldGUI . . . 7
4.4 Algorithm implementation . . . 7
4.4.1 Frustum . . . 7
4.4.2 EdgeCurve . . . 7
4.4.3 VoronoiDiagram . . . 8
4.4.4 DirectionField . . . 8
4.4.5 Mosaic . . . 8

5 Program Validation and Verification **8**
5.1 Testing Plan . . . 8
5.2 Testing Plan Details . . . 8
5.2.1 Class Testing . . . 8
5.2.2 Integration Testing . . . 9
5.2.3 System Testing . . . 9

6 Conclusion **11**

7 Appendix A: User Manual **11**

8 Appendix B: Member participation **12**
8.1 Michael Sutherland . . . 12
8.2 Josh Buchalter . . . 12
8.3 Luke Bell . . . 12

1. Introduction

The purpose of this project is to deliver a software program, equipped with an easily navigable UI which allows a user to convert ordinary images into decorative mosaics. The user will be required to define an edge map for the image to be mosaiced in order for the tiles of the mosaic to be adequately aligned and thus accurately portray objects in the image. The edge map will be defined by the user drawing lines which correspond to the lines that appear on the original image. The user may also select the tile size, number of iterations of the algorithm and the grout colour. The program will perform multiple algorithmic iterations on a voronoi diagram derived from the user-defined edge map, changing the positions and orientations of tiles according to their neighbours and vectors in the voronoi diagram. These iterations will each bring the mosaic to a form more realistic and accurate than the last, until the tiles are in such a configuration such that they meet predefined standards for tile orientation and size accuracy to sufficiently portray the original image as a decorative mosaic. The algorithm used was laid out and defined in Alejo Hausner, Simulating Decorative Mosaics, SIGGRAPH '01, pp. 573-580, 2001. This project had a Processing/Java implementation of the algorithm. Traditional analysis followed by design then implementation (agile approach) then testing was the process in which the software was developed.

2. Requirements Captured

2.1. Use Case Narratives

Use Case: Upload Image
Actors: User/supervisor
Description:

1. The use case begins when the user chooses to upload an image
2. The system responds by opening its file browser
3. The user searches for and selects the desired image
4. The system formats the image and fits it into the screen for user interaction

Alternate flow of events:

3a

3.1 The user cancels the search and continues with the current image on the screen

4a

4.1 The system alerts the user that the image is not in the correct format or is corrupt

Use Case: Draw edge features describing an image
Actors: User/supervisor
Description:

1. The use case begins when the user initiates a drawing session
2. The system indicates to the user that it is in drawing mode with the paintbrush cursor
3. The user draws edges on the screen describing the underlying image
4. When the user is satisfied with the edges on the image, he/she finishes and the use case ends

Alternate flow of events:

3a

3.1 The system disallows drawing outside of the user's image

4a

4.1 The user undoes changes

Use Case: Generate decorative mosaic
Actors: User/supervisor
Description:

1. The use case begins when the user chooses to save all edge features and prompts the system to generate the mosaic
2. The system runs the algorithm based on the edge lines and outputs the result of the computation

Alternate flow of events:
None

Use Case: Edit edge features for an image
Actors: User/supervisor
Description:

1. The use case begins when the user initiates an editing session of the original image, discarding the generated mosaic
2. The user deletes or adds to the current edge features
3. The user, when satisfied with the changes, prompts the system to generate a new mosaic based on the changes

Alternate flow of events:

2.1 The user cancels the editing session and recovers the discarded mosaic

Use Case: Download decorative mosaic
Actors: User/supervisor
Description:

1. The use case begins when the user chooses to download mosaic they have generated
2. The user browses and chooses an appropriate directory to save their mosaic
3. The system downloads the mosaic and stores it in this location, to be accessed later or exported

Alternate flow of events:

1a

1.1 The user cancels the download and returns to the generated mosaic

2.2. Other requirements

There was an element of interactivity that was required. This was defined as

3. Design Overview

The architecture chosen for our project is a layered architecture consisting of a presentation layer, a logic layer and an application layer. For the purposes of our project, the simplicity of this architecture helped us to separate concerns easily, allowing us each to work within our own defined space at the same time, increasing our productivity and enabling us to make changes which did not have too much of a ripple effect throughout the rest of the system. We found great success in this approach and did not have to spend time designing a complex architecture due to the few classes we used in our implementation. Figure 2 below details the contents of each layer and contains the classes that we used within each of them. It was very useful separating the UI into its own layer as this was where all the user interaction took place and thus, the look and feel had to be focused on before integrating the functionality from the other layers into it.

In addition to the architecture, we also detailed each class and have compiled an analysis class diagram in Figure 1. This shows the methods and relationships of each class and gives some insight into how we satisfied the functional user.

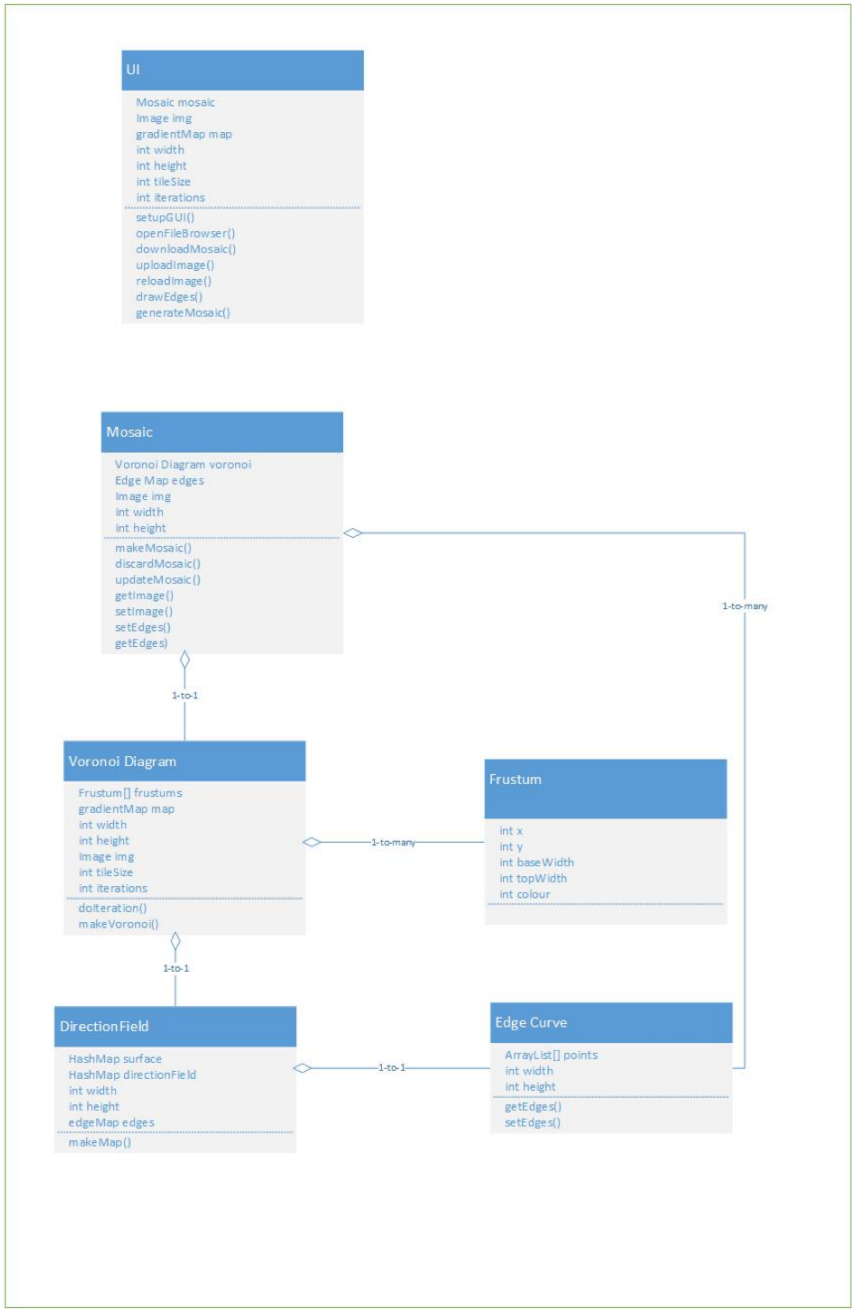

Figure 1: Analysis Class diagram for Mosaic Mecca

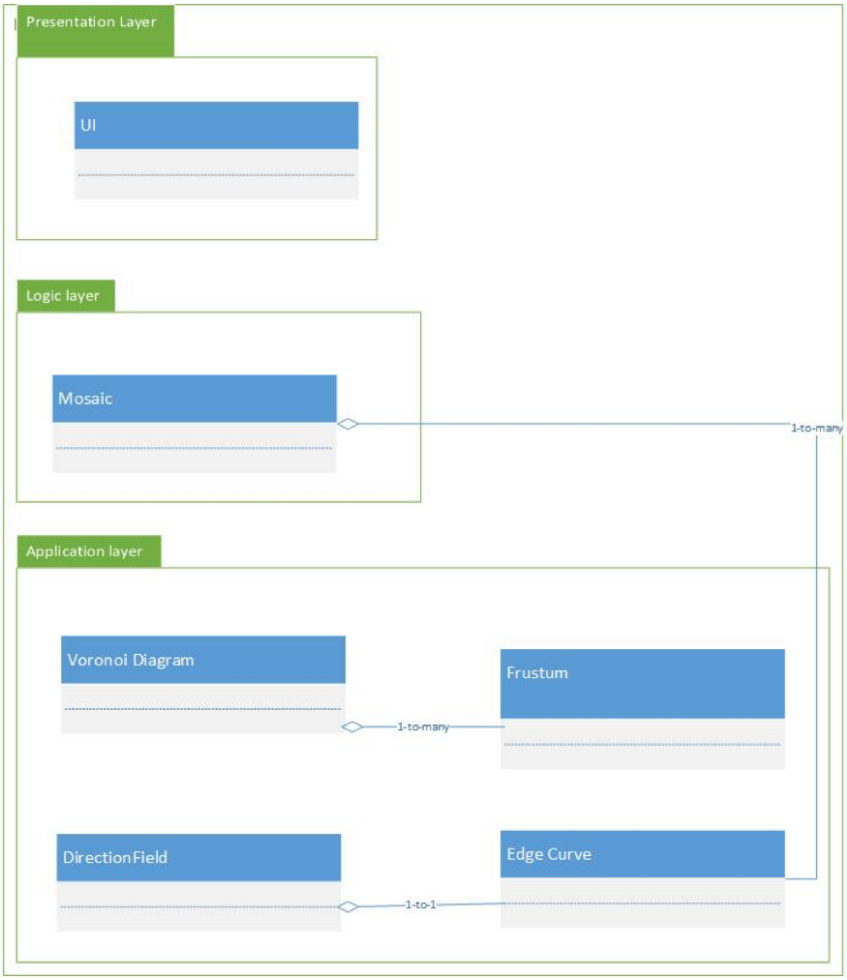

Figure 2: Layered Architecture diagram for Mosaic Mecca

4. Implementation

4.1. Framework

At the outset of the project, a graphics library had to be chosen to manage the algorithms needed to form a mosaiced image. Research was done and it was decided that the Processing library for Java would be used as the graphical framework of the project. Processing was chosen due to its relatively easy to understand methods and objects, as well various efficiency-related reasons. Processing has very fast draw and readback functionality which was needed for the drawing of numerous frustums and reading back the visual data generated from placing these frustums over many iterations of the program. The ability for orthographic view of data produced that is a part of processing was also needed for the mosaicing process. Since all graphics related needs were met by processing and because it was easy to learn, Processing was the obvious choice.

4.2. Classes

4.2.1. Frsutum

This is a class that represents a frustum object. A pyrimid like 3 dimensional shape with the top truncated to form a square surface. This shape finally forms a tile in the mosaic outputted at program climax. And is used in the algorithm that converges on tile orientation and placement. The Frustum class is very clearly necessary to the project.

4.2.2. EdgeCurve

This is a class that stores an array of 2 dimensional points. Given any other 2D point, it can calculate the closest point (using Euclidian distance) in the array to the input point. It does this by comparing the distance from the point to every point in the array (which is quite inefficient, but was decided to be the only realistic method in terms of finishing the project by the deadline). In the grand scheme of the project, this class is used to orientate the frustum array in VoronoiDiagram which eventually form the tiles of the outputted mosaic.

4.2.3. VoronoiDiagram

This is a class that stores information that represents an array of frustums placed next to each other on a plane. It generates and places theses frustums at its initialization on a 2 dimensional plane in the 3D space of a PApplet, which is an applet that uses the processing library; the library used as the graphics framework of the project. The class can calculate centroids of frustums and place new array of frustums over the newly calculated centroids. This represents the convergence of a voronoi diagram.

4.2.4. DirectionField

This is a class that takes an EdgeCurve as input and calculates for every point in a large 2D array, its associated closest point on the EdgeCurve, and the normalised vector pointing in the direction from the point to its associated closest point. This vector is used to orientate frustums in the VoronoiDiagram centered at this point.

4.2.5. Mosaic

This is the controller class for generating mosaiced images. It takes an EdgeCurve from which it generates a DirectionField. And it initialises a VoronoiDiagram object with a specified number of frustums. It then iterates over an algorithm that calculates the centroids of all the VoronoiDiagram frustums, as well as their orientations with respect to the DirectionField, and places the frustums at their new centroids with their new orientations. The number of iterations is specified. It then places tiles over the frustum objects to generate a mosaiced image of a supplied input image.

4.3. GUI

4.3.1. StartScreenGUI

This GUI allows for upload of an image to be mosaic-ed.

4.3.2. VectorFieldGUI

This class allows for the drawing of an EdgeCurve, the setting of iterations, tile size and grout colour. It also allows for the generation and saving of a mosaiced image.

4.4. Algorithm implementation

The algorithm first mentioned in Section 1 was implemented in the following way (divided into the classes used):

4.4.1. Frustum

In order to generate a tiling that is locally square, frustums were placed at each centroid in the voronoi diagram. When these frustums are orientated according to the direction field, the frustums overlap and using an orthogonal projection, these overlappings create a new voronoi diagram. After enough iterations, the algorithm creates a tilling of the space (with dimensions equal to that of the image). The Frustum class simulated a frustum defined in the algorithm.

4.4.2. EdgeCurve

Since edge detection was out of scope for this project, the user was requested to give the edge features (as a set of curves drawn with the mouse) as input to the algorithm. This class stored the curves as a set of points taken from the mouse location on the screen as they dragged. These points were then used in DirectionField to calculate the direction field(see sec 4.4.4).

4.4.3. VoronoiDiagram

The voronoi diagram is made up of a list of Frustums and each one is given a different colour. After each tile is placed and oriented, the centroid of the new region needs to be calculated. This is done by averaging the location of all pixels with the same colour. Once these new positions are calculated, the next set of tiles are placed at these new centroids. Edge avoidance - where tiles that straddle the edge curves are moved off them - was implemented by drawing the edge curves over the voronoi diagram in a colour different to all colours used in each voronoi region. The centroid is calculated using the colour of the region, but if part of the region straddiling the edge curve were to be changed to a different colour (colour of the edge curve) then the centroid would be moved such that the new region is shifted off the edge curve.

4.4.4. DirectionField

In order for the mosaic to accurately portray the curves of the image, the tiles need to be aligned to the edge curves that the user draws. By creating a surface $z(x, y)$ such that for any point $P(x, y)$ not on an edge curve and $C_s(x, y)$ the closest point to P on an edge curve:

$$z(x, y) = |P(x, y) - C_s(x, y)| \tag{1}$$

The vector field produced by finding the gradient:

$$\phi(c_i) = \nabla z(c_i), \quad c_i \in \text{centroids} \tag{2}$$

results in orientations of the tiles such that tiles near to the edge curves are perpendicular to them. This orientation allows the mosaic to portray the key features of the image as defined by the user.

4.4.5. Mosaic

This class implements the algorithm by controlling all the constituent parts of the algorithm (see sec 4.2.5).

5. Program Validation and Verification

Testing done on the system can be broken down into three distinct sections: class testing, integration testing and system testing. These account for testing done at all levels of the system and the success of these tests is a good indicator of the success of the system and project as a whole. They range from tests done at the lowest level: individual methods within individual classes, to the highest: system tests of various possible paths through the program.

5.1. Testing Plan

Table 1: Testing plan

Process	Technique
1. Class Testing	Random, Partition and White-Box Tests
2. Integration Testing	Use-case, Behavioral and Boundary testing
3. System Testing	Use-case, Blackbox and User testing

5.2. Testing Plan Details

5.2.1. Class Testing

This consisted of a suite of unit and whitebox tests done on various methods throughout all of the projects classes. As well as performance tests for some methods required to meet user efficiency requirements. The test results shown below in Figure??. All Class tests were passed except one which was the performance test on the createSurface() method in the DirectionField class. This indicates that the performance user requirements for DirectionField generation were not met, and due to the project deadline, this had to be accepted in order to deliver the program on time.

5.2.2. Integration Testing

Behavioral and Boundary testing were executed on the various parts of the system to test their individual success and readiness for integration into the system. Various use-cases were tested as detailed in table?? below.

5.2.3. System Testing

Users were asked to carry out steps in various use-cases as well as explore as many paths through program as they could. Users reported on their actions and resulting states of programs. All errors and program crashes reported by users were subsequently fixed so that the program can handle all inputs and actions that were supplied by users.

All Tests

Class	Name	Status	Type	Time(s)
TestSuite	testGetClosestPointError	Success		0.002
TestSuite	testToString	Success		0.001
TestSuite	testGetSize	Success		0.000
TestSuite	testContainsPoint	Success		0.001
TestSuite	testGetClosestPoint	Success		0.001
TestSuite	testGetVector	Success		0.001
TestSuite	testGetEdgeCurveSize	Success		0.009
TestSuite	testGetImageHeightWidth	Success		0.005
TestSuite	testCalcGradOfSurface	Success		0.007
TestSuite	testSurfacePoint	Success		0.007
TestSuite	testGetEdgeCurveVector	Success		0.003
TestSuite	testDirectionFieldPoint	Success		0.013
TestSuite	testCreateSurfaceTime	Error	test timed out after 1000 milliseconds	1.037
			java.lang.Exception: test timed out after 1000 milliseconds at main.EdgeCurve.getClosestPoint(EdgeCurve.java:33) at main.DirectionField.getSurfaceValue(DirectionField.java:87) at main.DirectionField.createSurface(DirectionField.java:43) at main.DirectionField.<init>(DirectionField.java:27) at tests.DirectionFieldTest.testCreateSurfaceTime(DirectionFieldTest.java:49)	
TestSuite	testGetRandomColours	Success		0.570
TestSuite	testPlaceFrustums	Success		14.796
TestSuite	testGetRandomPoints	Success		0.002

Figure 3: Class Test results

Table 2: Integration Tests

Test Case	**Details**
Test case 1: Image with at least 3 different edge curves	Input: image with three different user-drawn edge curves. Explanation: This will test how well and efficiently the program can calculate distances from multiple curves to a particular point. We expect that the direction field be perpendicular near all curves and have some form of smooth flow (i.e, the direction does not vary rapidly for small change in position) in places far away from any edge curve. Results: DirectionField calculated with acceptable speed (1-2 seconds for EdgeCurve of size 200, 1-3 seconds for EdgeCurve of size 500). DirectionField not always perpendicular to curve at points close to curve (curve not continuous, but made up of distinct points). This is a failure and does not meet user requirements, but a sacrifice had to be made with regard to fixing this so as to be able to complete project and project report before deadline.
Test case 2: The scale by which the direction field produced varies is much smaller than tile size	Input: image and edge curve. Explanation: The convergence of the algorithm (i.e, how uniform the voronoi diagram is) is being tested. The expected output would be a voronoi diagram that accounts for small angles between EdgeCurve lines and thus has tiles placed over corners of EdgeCurve, not throwing errors related to tile placement. Results: all tests passed with meeting user requirements. No errors thrown and no large areas in mosaic with no tiles.
Test case 3: There are at least two areas in which tile size differs	Input: image with a set of edge curves and at least one area that the user has highlighted in which the tiles size will be different from the originally set size. Explanation: The programs ability to produce more detailed areas will be tested and we expect the output to be very detailed in the area of smaller tile size. Results: The ability for varying tile size within mosaic was not implemented in system and hence all tests failed. This was not, however, a system requirement but rather a bonus functionality.

Test case 4: No edge curves drawn	Input: image with no EdgeCurve. Explanation: The programs ability to produce a mosaic at all will be tested. The expected output is that the direction field is undefined and thus the tiles never get aligned creating a mosaic where all tiles are oriented the same way. Results: System displays error message explaining that no mosaic will be generated without a drawn EdgeCurve and asks user to draw some EdgeCurve. This meets user requirements.
Test case 5: Images colours are not well defined	Input: image and arbitrary edge curves. Explanation: We are looking for how well the mosaic represents the image when the tile colours do not differ greatly. The expected output should be a mosaic with no real definition or detail of the original image. Results: undetailed mosaics that align to drawn EdgeCurve. Meets user requirements.

6. Conclusion

We were tasked with creating a user-controlled decorative mosaic simulator using specific algorithms and our choice of an appropriate Java graphics framework. This was achieved with the use of Javas Processing framework.

We have delivered a complete, robust system based on the user requirements and have tested this system to see how well we met these requirements. We broke these tests into categories and have concluded that our system failed only 4 of the tests and thus, we have met almost all of the user requirements.

Our system is separated into three well structured layers, making our productivity concurrent and separating the important aspects of the system from each other making the code easier to understand, easier to write, able to evolve independently of each other and most importantly, easier to extend functionality once each of us had programmed our sections and integrated them together.

Other than these few failures, our system met all the initial aims based on the user requirements and the overall outcome of the project was deemed a great success. We are very excited to showcase what we have created and are proud of our team dynamic and effort throughout the project.

7. Appendix A: User Manual

Introduction

This Manual is intended for users of the MOSAIC MECCA system as well as the projects clients. It contains instructions pertaining to the installation and basic use of the software. However, this manual is by no means it it a full step by step guide as to what can be achieved within MOSAIC MECCA, as such a guide would be endless. The manual intends only to introduce you to the wonders of the software product, and leaves it up to you to find out what you can do with it.

Installation Guide

Make sure you have Java 1.8 installed on your Machine. MOSAIC MECCA is unfortunately only compatible with apple operating systems. Other than that, there is no installation required. The MOSAIC MECCA system can be run by simply opening the mosaicmecca.jar file, provided you have correctly installed java 1.8 on your machine. Program can also be run by navigating to the correct directory in terminal and typing java -jar mosaicmecca.jar

If you wish to run the system on a non apple operating system, you will have to run it through an IDE. For running within an IDE, download the project as a compressed file and extract it to an appropriate

directory on your system. For easiest installation, ensure eclipse IDE is installed on your system and import the DecorativeMosaic project into your workspace. MOSAIC MECCA is now installed. Run the project from StartScreenGUI.

NOTES:

1. Make sure you only upload jpeg or png files for mosaic generation.
2. Make sure you have at least one edge curve drawn before attempting to generate a mosaic.
3. Experiment with program by changing iterations, tile size and grout colour in File drop down menu in main GUI frame.
4. Save mosaic after generation to your machine after generation for storage.
5. If you forget to save mosaic. All generated mosaics can be found in output directory within the projects home directory.

8. Appendix B: Member participation

8.1. Michael Sutherland

As the Information Systems major of the group, I was in charge of documentation, making sure the planning and specifically the Gantt Chart, was up to standard and that, from our discussions, we had given ourselves enough time to accomplish the tasks we chose for ourselves. The first three phases were purely documentation; I used my knowledge of the documentation techniques learnt in IS and in the software development lectures to fairly break the deliverables into sections for each member to write, providing examples from my own past projects for the group to get an idea of the standard expected. The documentation phases were greatly successful due to the team effort put in and the past experience providing good examples to learn from.

In the fourth stage of the project I was put in charge of the user interface, as I am only the third strongest mathematician of the group (the weakest, by very far). This was a convenient section for me as I have had extensive experience in designing prototypes, wireframes and other user interface diagrams in my informations systems courses. This meant that I was able to use the design principles I have learnt and make efficient use of my time and resources in order to create the design for the final product. During the integration phase the team were of great assistance to me, helping me fit everything together and linking the user interface with the controller and model classes that they had designed as well as suggesting improvements to the UI design and a whole array of extra features such as sliders, drop down menus and user response text. A feature I am quite proud of is my custom pencil cursor. This was fun to design and creates an exciting look and feel to the user interface.

8.2. Josh Buchalter

My second major is Applied Mathematics and I have done a fair amount of numerical analysis which deals with numerical methods for performing differential operations such as those found in sec4.4.4. Thus I focused on implementing the edge features and orientation parts of the program. I created the EdgeCurve and DirectionField classes as well as helped in various parts of the deliverables including formating this report in Latex. I also found a way to speed up the rendering of the frustums and tiles in the voronoi and place tiles parts of the program by using a Processing construct called a PShape which utilized openGL's Vertex Buffer Object thus converting the rendering process to a retained mode instead of the intitial immediate mode.

8.3. Luke Bell

I was positioned as the lead developer for the project and oversaw the developement process from a programming point of view. I was in charge of coming up with the basic architecture of the system as well as delegating various programming taks to the other group members. The code for which I am solely responsible included the voronoi component and the controller class for mosaic generation. The task of writing class tests for the project was also assigned to me and the test plan and results were all completed

by myself. On top of these tasks, I also helped the other two group members with their code when they needed and ensured the integration of all the various components within the projects.