Procedural Level Generation and Dynamic Path-finding A Combined Approach

Neil Notman - 40124066
Supervisor: Dr Benjamin Kenwright
Second Marker: Dr Neil Urquhart

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc Games Development (Hons)

School of Computing

April 27, 2016

Authorship Declaration

I, (Neil Notman), confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed:

Date:

Matriculation no:

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

As games evolve and expand with the times there is an increased demand placed upon the hardware of games consoles and personal computers in terms of storage space and memory usage. This project will analyse the approaches currently available and will examine the viability of developing a combined approach to level generation and path finding with the aim of producing a solution that will compete with current approaches in terms of the quality of the terrain generated tested through the results given by the path finding and the distances between nodes that will be placed in sane (flat, reachable) locations on the terrain this will also allow for testing the quality of the path finding approach by comparing this with different complexities of terrain and existing industry and commercial techniques.

Contents

1 Introduction 9
1.1 Computer Graphics . . . 9
1.1.1 The Graphics Pipeline . . . 9
1.1.2 Geometry . . . 9
1.1.3 Primitive shapes and polygons . . . 9
1.2 Procedural Level Generation . . . 10
1.3 Path-Finding . . . 10
1.4 Optimisation Techniques . . . 10
1.5 Aims and Objectives . . . 10
1.5.1 Design objective . . . 11
1.5.2 Implementation Objectives . . . 11
1.6 Report Layout . . . 11

2 Literature-Review 12
2.1 Introduction . . . 12
2.2 The history of computer graphics . . . 12
2.2.1 OpenGl . . . 12
2.2.2 DirectX . . . 12
2.2.3 Graphics Programming Development . . . 12
2.3 Background and inspiration . . . 13
2.4 Procedural Level generation Techniques . . . 13
2.4.1 The Diamond-Square algorithm . . . 13
2.4.2 The marching cubes algorithm . . . 14
2.4.3 Height-map generation and Rendering . . . 14
2.4.4 Perlin Noise . . . 15
2.4.5 Rigid Multi Fractal Noise . . . 15
2.5 Path-finding Techniques . . . 15
2.5.1 Dijkstra's algorithm . . . 15
2.5.2 Path finding in games . . . 17
2.5.3 The A* algorithm . . . 17
2.5.4 Analysing the quality of path-finding techniques . . . 17
2.5.5 Previous honours projects . . . 17
2.6 Optimisation techniques . . . 18
2.6.1 Parallelism and parallel programming approaches . . . 18
2.6.2 Graphics Processing Unit (GPGPU) . . . 19
2.7 Commercial packages . . . 19
2.8 Conclusions Drawn . . . 20

3 Technical Information 21
3.1 Software Used . . . 21
3.1.1 OpenGL . . . 21
3.1.2 Windows SDK . . . 21
3.2 Hardware . . . 21

4 Implementation Methodology 22
4.1 Level Generation . . . 22
4.1.1 Libnoise External Library . . . 22

4.2 Nodes and placement 23
4.3 Path finding implementation 23
4.4 Combining the terrain and path-finding metric 24
4.5 Additional Information / Knowledge Required 24
4.6 Experimental plan 24
4.6.1 Experiment 1 25
4.6.2 Experiment 2 25
4.6.3 After experiment 2 25

5 Testing and results **25**
5.1 Results 27

6 Conclusions **28**

7 Bibliography **29**

Appendices **31**

A Project Overview **31**
A.1 Overview of Project Content and Milestones 31
A.2 The Main Deliverables 31
A.3 Target audience for the deliverables 32
A.4 The work to be undertaken 32
A.5 Knowledge and skills required 32
A.6 Information sources that provide a context for the project 32
A.7 The importance of the project 33
A.8 The key challenges to be overcome 33

B Diary Sheets and project management **38**
B.1 Week 2 and before 38
B.2 Meeting 1 22/9/15 Week 3 38
B.3 Meeting 2 29/9/15 Week 4 38
B.4 Meeting 3 6/10/15 Week 5 38
B.5 Meeting 4 13/10/15 Week 6 39
B.6 Meeting 5/Group Thesis Review 20/10/15 Week 7 39
B.7 Meeting 6 27/10/15 Week 8 39
B.8 Meeting 6 3/11/15 Week 9 39
B.9 Meeting 7 10/11/15 Week 10 40
B.10 Meeting 8 17/11/15 Week 11 40
B.11 Meetings up to 21/3/16 40
B.12 Meeting 9 21/3/17 40

C Second Formal Review Output **41**

List of Tables

1 A table of tests carried out on the system 26
2 A table of results from pathfinding across different terrain types for Experiment 1 Note: for terrain quality lower is better due to the terrain being more level 27
3 The results with a combined score for Experiment 2 27

List of Figures

1 The 3D space of a marching cube 14
2 G= gradient vector P = input point Q = Neighbouring point 15
3 An image showing the output image that was generated by noise Note: left—- Perlin noise——————————right Rigid multi fractal noise . 22
4 The output from rendering a heightmap Note: left—- Perlin noise————————————right Rigid multi fractal noise 23
5 The node placement when drawn on to the terrain 23
6 The output given be pathfinding— Left:Dijkstra — Right:Astar 24
7 A time-line of path finding algorithms 34
8 An image of the project gantt chart 35
9 This shows how the program will execute 36
10 This is a mind map covering the scope of the project 37

Acknowledgements

I would like to thank

1 Introduction

1.1 Computer Graphics

Firstly before we discuss the main techniques that will be use within the project we will give a background of computer graphics techniques that will be utilised in the project as a basis for anyone without prior knowledge.

Computer graphics entails the process of drawing visuals to a screen using the graphics card of a computer the main technologies that allow this functionality include OpenGl (Open Graphics Library) and Microsoft's^{tm} DirectX framework.

1.1.1 The Graphics Pipeline

To utilise the graphics card to draw visuals we access the a pipeline within the card this began as a fixed set of steps that were followed to allow the creation of geometry and dictated how this was processed by the computer to draw this to a screen.

Later on the graphics pipeline was developed to allow us to customize how the computer processed this data these developments gave access to techniques such as procedural level generation and allowed programmers to improve the quality of the graphics that were produced.

1.1.2 Geometry

We create visuals to draw to the screen in the form of geometry this can include points which are known as vertices these represent a single point in 3d space and are made up of a coordinate in each of the X,Y and Z axes.

From these we can build up lines in the 3d space which can be used to create primitive or complex shapes such as triangles or spheres. Furthermore due to advancements in this field now full scale 3d models can be broken down into these allowing us to draw these to a screen.

The geometry is represented as a matrix inside the computer and can be moved,rotated and scaled through the multiplication of the matrix representing the geometry by a transformation matrix.

1.1.3 Primitive shapes and polygons

The basic way to create geometry is through the use of either individual or strips of triangles which are described as polygons these can be used to make up many complex shapes and the number of polygons that make up a shape or scene is generally used as a measurement of the complexity of a particular graphic.

The number of polygons can be a performance concern if this count is very large however with the increase in both memory and performance in graphics cards from the past where the maximum was in the low thousands for characters in games and game worlds with tens of thousands till now where we can have up to twenty thousand polygons per character with detailed worlds that can contain millions of polygons.

1.2 Procedural Level Generation

Procedural level generation is the process of creating a level through an algorithm contained within a computer program instead of using software to create this content then loading it into a program. The procedural level generation approach was mainly utilised in the early day of games development where storage space was a major concern.
The first game to contain procedurally generated levels was Richard Garriott's game Akalabeth: World of Doom which was published by California Pacific Computer Company in 1980 [Wikipedia, 2015].

This project is going to utilise procedural level generation to create a average sized game world by creating a height-map which is a grey-scale image then reading the data from a height-map and using this data against the depth buffer of a graphics programming language to generate the world.

1.3 Path-Finding

Path-Finding is an area of Artificial Intelligence that allows for definition what parts of a given level can be moved to by computer controlled players this is done by creating a set of nodes on walkable areas then linking these nodes together to form a path there are various algorithms that accomplish this task. A time-line of path-finding developments is given at figure: 7. The project will attempt to use path-finding on the generated level in small chunks to allow for faster processing and access to more optimisation techniques.

1.4 Optimisation Techniques

There are many factors that can affect the run-time performance of computer programs and a similar amount of approaches that can be utilised to improve performance some of the most prominent techniques in this area include use of multi-threading and parallelism frameworks such as OpenMP for use in single computer based solutions and MPI which allows for parallel implementation over a network.

There are other options that differ from a parallel approach that could be utilised these can include memory alignment of instructions or parallelisation of the project through CPU based frameworks such as OpenMP or for greater performance a graphics processing unit (GPU) based implementation.

1.5 Aims and Objectives

The main aim of this project is to examine and test the viability of combining level generation and path finding algorithms to create a double ended testing mechanism for quality to allow a performance based assessment between both level of detail in the terrain when compared with the time taken for a path finding technique to find a correct path.

The aim of this project will be achieved by completion of the following objectives which include both design and implementation as shown below.

1.5.1 Design objective

The main design objective will be the creation of a graphical user interface that provides a suitable visualisation of the implemented solution and displays all relevant data. There are various libraries that allow for and assist with the creation of such interfaces such as GLFW for Opengl based applications however the windows SDK also contains this ability and will allow for clear differentiation from the interface and any graphical code included in the application.

1.5.2 Implementation Objectives

The creation of an application using the chosen solution to allow a user to create various sizes of level.

The application will then build the terrain and perform path finding on a set of nodes positioned across a generated terrain.

The application will then assess the quality of both the level generated by and path finding technique taking into account the complexity of a given level such as obstacles slopes and caves.

The application will then assess how easy it is to build a path round that level and will give a score based on both the complexity of a generated level and the path-finding technique used.

1.6 Report Layout

The report goes on to give a review of various literature sources associated with the project Next there will be an analysis into the methodology used in implementation and finally the testing and results will be given alongside a conclusion to the project which will discuss the overall success and knowledge gained based on the results of testing.

2 Literature-Review

2.1 Introduction

We will now review a number of literary sources which fall within the context of path-finding methods and procedural content generation to explain how these techniques work and to establish how these may be used within the project to either help implementation or optimisation of the project in it's latter stages.

2.2 The history of computer graphics

As mentioned in the introduction there is a number of graphics based programming languages with the main competitors being OpenGl and DirectX we will now look at the fundamental differences between these and the history and developments to these that are of interest in the context of this project.

2.2.1 OpenGl

OpenGL was originally developed in 1991 from the existing IRIS GL that was created in 1982 by Silicon Graphics and OpenGL was released to the public in January of 1992 [Kilgard and Akeley, 2008] fierce competition from PEX which was a graphics language based on the X Window system drove the early development of OpenGL forward with later competition coming in the form of DirectX.

2.2.2 DirectX

DirectX is a full range of multimedia languages which included a graphics language component originally named Direct3D these tools were first released in 1995 by Microsoft^{tm} this was due to the release of windows 95 to entice game programmers as MS-DOS had been thought to be a better platform for games by the majority of developers another factor of this was to compete with OpenGL[Kerin, 2012].

2.2.3 Graphics Programming Development

As the competition between OpenGL and DirectX intensified this lead to the rapid development of both these languages and the graphics hardware.

Some of the main developments were the movement from a fixed graphics pipeline to one where the developer could program shaders to control area's within the pipeline this was first introduced in the form of vertex and pixel shaders and became usable in OpenGL 2.0 released September 2004[Wikipedia, 2016b] and Direct3D 8.0 which was earlier being released in November 2000 with stability being achieved in February 2001[Wikipedia, 2016a].

From this point forward the graphics pipeline became even more customizable with the introduction of the geometry shader which allowed for full sections of geometry to be generated from a set of vertices this was first introduced in DirectX 10.0 which became available in 2007 with OpenGL integrating these in 2009 with the release of version 3.2 in 2009.

The final main development was the move to general purpose applications that could run on the GPU (GPGPU) and this was released in DirectX 11.1 in 2008 under the name DirectCompute with OpenGL implementing this as a Compute shader in 2012.

Throughout the history of these competitors we can see that DirectX has generally been leading the developments in graphics programming and performance however this is due to the early focus of OpenGL being based towards design uses such as CAD and other hardware intensive uses whereas Microsoft^{tm} envisioned DirectX to be a streamlined games development based graphics language.

2.3 Background and inspiration

The inspiration for this project was based around pushing performance and analysing both level generation and path-finding approaches to find a common area where these methods can be measured together to determine a quality versus performance trade-off tool for games engines and to allow testing of these measurements of both the terrain and path that is built in terms of complexity against the amount of nodes required to build a logical path and any deviations that are made to the path due to constraints posed by the terrain.

The hope for this project is that the work could be integrated into a games engine to allow developers to view how changes to terrain may affect other measurements and give a scale that would be based on performance versus quality.

For a time line of various path-finding techniques please refer to figure 7

2.4 Procedural Level generation Techniques

There are a variety of ways to generate either 2d or 3d geometry within a program this section will discuss the documented methods and compare them in terms of usability and performance.

The paper [Hendrikx et al., 2013] examines the different layers of a game that can be procedurally generated this includes a section on game space for levels or maps and defines these both abstract or concrete methods for generation of both indoor and outdoor environments as well the paper gives a taxonomy of commonly used procedural content generation techniques.

2.4.1 The Diamond-Square algorithm

This method of level generation was proposed by [Fournier et al., 1982] is based off fractal subdivision to generate randomised terrain based on two steps which are split into the diamond step and the square step hence the algorithms name.

The diamond step of this algorithm uses the edge points of a square to generate a random value in the midpoint then the square step of this algorithm performs the same function however uses the edge points of the diamond made previously which gives a square as the result.

The algorithm is recursive meaning that it performs multiple passes through each step with the generated surface gaining detail every pass however the increase is due to more geometry being created each pass this means the algorithm can be memory intensive due to the size of array used to store the height values being a power of two + 1 this means for eight passes through the algorithm the array would need up to 256KB memory if storing floating point values [Miller, 1986].

2.4.2 The marching cubes algorithm

The marching cubes algorithm is another method that can be used to generate terrain it was first proposed in 1987 by [Lorensen and Cline, 1987] the algorithm was used to output three dimensional images from medical scan results and does this by creating polygons from the density of surfaces contained within a three dimensional array.

The algorithm works by first taking in a user defined start point within a cube in three dimensional space and starting to create triangles then using intersection testing against the cube to determine where the generated triangles end then we move or march to the next point within the cube an example of this from the paper [Lorensen and Cline, 1987] is shown below and shows how the 3D space and cube are set up.

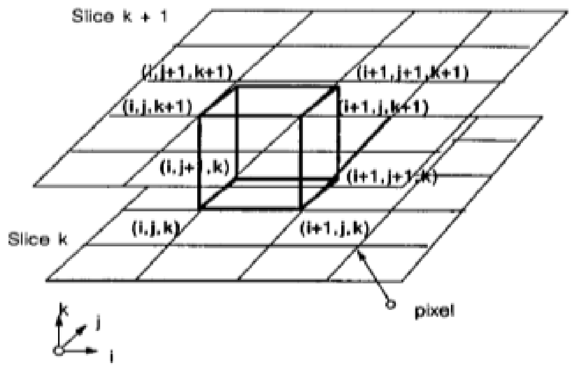


Figure 1: The 3D space of a marching cube

2.4.3 Height-map generation and Rendering

A height map is a grey scale image that makes use of depth to create a section of terrain with areas where the terrain to be raised being shown as brighter and lower areas shown as darkk for most height maps digital noise is used to produce a suitable terrain for an example of the output that is produced by a height map please refer to figure 3 the algorithm for creating digital noise will be detailed below.

2.4.4 Perlin Noise

To create the image 3 Perlin Noise was used this is a gradient based noise developed by Ken Perlin in 1983 [Perlin, 1985] to create procedural textures using the algorithm outlined below.

- Get an input point of the image
- Loop through the neighbours of this point
- Generate a pseudo-random gradient vector.

  - perform the following calculation

$$G \cdot (P - Q) \tag{1}$$

Figure 2: G= gradient vector P = input point Q = Neighbouring point

  This equation will give the value of P where G is equal to 0 at point Q.

- Finally you Interpolate between the the points down to your point, using an S-shaped cross-fade curve e.g($(eg : 3t^2 - 2t^3)$) this will give the weighting applied in each dimension.This step will require computing of the curve n times, followed by 2n-1 linear interpolations to get the final result.

2.4.5 Rigid Multi Fractal Noise

This type of noise is created by flipping the output of given by Perlin noise this is discussed in the thesis [Davis, 2013] the flipping of these values give a set of sharp high ridges and is useful for creation of mountains in the context of this project the use of rigid multi fractal noise would provide a suitable terrain to test against.

2.5 Path-finding Techniques

This section will analyse some of the key path finding techniques that would address the problem posed within the project including an analysis of past work done in this area will be given with the aim to analyse how this can be utilised and expanded within the project.

2.5.1 Dijkstra's algorithm

The algorithm for path finding presented by Edsger Dijkstra in 1959 [Cormen et al.,] allows searching of a weighted graph to determine the shortest route through a set of nodes by working out the shortest path from one node to every other connected node in the graph with only the shortest distance being chosen from one node to the next this technique is known as a uniform cost search. The nodes can be used to represent space within either a two dimensional grid or a point in three dimensions.

This method can be used within the project as the cost of travelling between nodes can be calculated as the euclidean distance between the nodes this will allow us to remain with vector calculations which would benefit a GPU based implementation. If we look at figure 7 it is shown there have been a number of developments to this method of path-finding the main adjustments that are beneficial to the project will be discussed below.

2.5.1.1 Algorithm details

Dijkstra's algorithm uses the following steps to calculate the shortest path from a starting node to any other node.

1. Set up graph of nodes with all distances being infinite except the starting node which is initialised at zero.
2. Make sure all distances are set temporarily with the starting node being set permanently
3. Find the neighbours of active node and from there calculate their distance to the starting node using the weights on the edges which are determined by distance.
4. The neighbouring node with the lowest distance is chosen to become the next node from the start we then update the distance at the starting node and set the next node to active.
5. set the chosen nodes distance to permanent and repeat steps three and four until there are no neighbouring nodes in the graph with temporary distances.

2.5.1.2 Developments to graph based path finding

There have been a multitude of developments to graph based path finding techniques these include some of the key methods mentioned in 7 the ideology of these techniques and their benefits in the context of this project will be discussed.

The first development to graph based techniques that is interesting is the article by [Goodchild, 1977] this was where the traditional orthogonal approach to building paths was modified due to the fact that errors were produced with straight line and smoothly curved paths the solution was to look at using both the orthogonal and diagonal steps to reduce the errors produced such as deviation from the path and this also prevents the path from becoming too long.

Another development of interest is the creation of a spread based algorithm for path-finding that was first proposed by [Califano, 2000] the origin of the splash (structural pattern localization analysis by sequential histograms) algorithm was to identify patterns in amino acids and other uses within the field of computational biology relying on sparse pattern matching techniques.

The splash is shown in [Califano, 2000] to be both memory and computationally efficient when compared with other path finding methods due to the utilisation of sparse pattern matching this is of value as it allows for faster processing of a path that contains pattern and using the splash algorithm could be beneficial due to the parallelizable nature of that this algorithm uses to search through data.

2.5.2 Path finding in games

Path finding is an area of artificial intelligence widely used in games to allow computer controlled characters to walk on the geometry that makes up a game world or level the paper by [Graham et al., 2003] describes the method for path finding on geometry by first building a navigational mesh which describes the parts of a given world or level can be traversed or using a node based waypoint system which is traditionally based on the visibility of one node to the next which allows for traversal of the geometry.

There is also a breakdown of path finding techniques into directed and undirected methods with directed methods such as the A* algorithm which combines the cost based searching of nodes or a navigational mesh with a heuristic search to a goal this allows for greater efficiency over Dijkstra's algorithm which is a important consideration for many games.

2.5.3 The A* algorithm

The A* or A star path-finding algorithm was first proposed in 1968 the paper [Hart et al., 1968] this method for path-finding uses a heuristic based search to find the shortest route for a given weighted graph to a goal node or set of goals.

There are many benefits to using this algorithm these include storing previously any visited nodes that do not lead to the goal this helps to prevent backtracking and eliminate infeasible solutions leading to a reduction in the search space thus an increase in runtime performance although this comes with an increased memory overhead as a result of using two lists over one list to store the path.

2.5.4 Analysing the quality of path-finding techniques

There are a number of models in place for the measurement of path finding techniques these utilise various properties of a given algorithm to determine it's quality in the context of this project we will be considering three main factors which are the run time performance of a particular algorithm, the total distance of the path created and if there is any difference in the number of nodes that make up a given path this will also be considered.

2.5.5 Previous honours projects

There has been a previous honours project based on performance of GPU based path-finding. The thesis that was authored by [McMillan, 2014] contains work on a variety of path-finding techniques including Dijkstra, A* and diffusion methods with both a sequential and parallelised implementation the optimised versions of these path-finding approaches which may be utilised in cases of performance bottleneck as this project will not be specifically related to performance but rather the production of an elegant constrained path with a given terrain then scoring this against any path finding technique based on a performance evaluation using the metric that will be established.

There was also a project that proposed a metric for picking good procedurally generated terrain this project looked at the creation of procedural terrain through the use of a height-map and considered what methods can be used for evaluating the quality of a given 3D terrain with descriptions on the implementation of these techniques.

The methods talked about within the thesis [Gray, 2012] for quality evaluation of 3D terrain include through the analysis of accessibility this is done by taking the average of a slope in the terrain along with the standard deviation this is then divided by the average slope.

This calculation if higher than one represents a better terrain than if the result is lower what this does is determine how much of the terrain is accessible by assessing the steepness of slopes on a terrain.

For the project [Gray, 2012] it was said that a point is accessible for a building if the slope is less than twenty six degrees or where the dot product of these points is greater than 0.9 whereas traversable terrain is defined with a dot product result of 0.7

2.6 Optimisation techniques

To optimise the project a wide range of factors and approaches are being considered the purpose and uses of these will be briefly discussed below along with a discussion on how these methods could benefit this project.

2.6.1 Parallelism and parallel programming approaches

The first approach to address is that of how to get the most performance out of the development platform (PC) one of the main ways to optimise for this platform is the use of parallel programming this will be implemented through the OpenMP API [Dagum and Enon, 1998]which is built into Visual Studio this will allow for multi-threading of any code running on the central processing unit (CPU) through the use of pre-processor directives to specify sections of code to incorporate parallelism this approach should give a performance increase by utilising the multiple cores in the central processing unit over a sequential implementation that uses a single core.

Other parallel frameworks have been considered however are not at this point set to be implemented these are the MPI (Message Passing Interface)[Snir, 1998] which allows for distributed parallelism over a network this would be extremely beneficial to performance as the host PC which is executing the program would have more resources available however this approach can have issues mainly due to the communication being network based meaning that bandwidth can affect the performance of a program to produce a performance decrease due to the time taken to communicate over the network it is unlikely that the project will utilise this technique however when testing this may be useful for running multiple instances of the project with different parameters.

2.6.2 Graphics Processing Unit (GPGPU)

GPGPU(General Purpose Graphics Processing Unit) programming allows the user to run non graphics based code on the GPU this is beneficial due to the amount of processing cores that the GPU contains over the central processing unit allowing dramatic performance increase however there are a number of drawbacks to this approach and it can only be used within certain problem area's.

GPGPU programming is a good fit to maths based problems over large data sets in comparison to problems that branch through sections of code this is due to the way data is processed on the GPU this is done by streaming data through the GPU cores in groups known as either wave-fronts or wraps which are groups of threads within the GPU and the terminology for these differs depending on the technology used instead of the normal fetch execution cycle of the CPU where instructions and resources are fetched from memory before they are executed or loaded.

To utilise the GPU for processing we must send the data from the processor to the GPU then perform our operations and finally transfer the answer back to the CPU this is one of the main drawbacks to GPGPU programming as the performance is determined not only by the hardware but also by the speed of data transfer between the CPU and GPU also the size that this data takes up in memory will impact on the time taken to transfer from one to the other.

The use of GPGPU techniques for parallel programming were only recently applied to AI because of the branching nature of the problems in this area it has always been considered that the AI system discussed in [Silver et al., 2016] used a distributed system to utilise the hardware resources of multiple machines to carry out consecutive parts of the processing which is then passed back to a host machine this is what allowed the massive amount of processing power required by the alphago system to process the best moves at every point of a match.

2.7 Commercial packages

There exist a number of commercial applications that fall within the context of this project most of these are contained within games engines which contain ways to generate levels and to build paths.

There are also a variety of evolutionary systems which can be used to find the best solution these can be found within lots of packages one example that is notable being journey planning software such as that contained within Google Maps this uses traffic information along with travel method and location to give you the fastest route to a given destination.

2.8 Conclusions Drawn

To conclude in this section we have looked at various methods to implement both path-finding and procedural level generation we also discuss how these techniques can be measured individually and look at methods such as multi-threading and GPGPU programming for optimisation of these techniques.

We have also looked at a number of previous projects based within the context of both level generation and path-finding and discussed the benefits and important features of these projects with the final section analysing commercial packages which implement techniques that may be of interest to the project.

3 Technical Information

In this section we will examine the software and hardware that will be utilised within the project and examine the purpose of these decisions and any benefits that these will give in terms of implementation of the project.

3.1 Software Used

To implement the project the choice was made to use the C++ programming language this is due to the familiarity with the syntax and functionality of this language as well this allowed for the utilisation of the vast array of tools that are either directly integrated or tools that can be used within the C++ language.

3.1.1 OpenGL

The choice to use OpenGL to implement the graphics based area's for the project was made to increase the cross platform capabilities of the project this also allowed for a familiar tool set to be used which should aid the speed of development and allow for some base performance optimisations to be made at an early stage.

GLM is a maths based library which is made for use with OpenGL and other graphical based languages this library contains classes which allow for the creation of maths based variables such as vectors and matrices and functions that allow us to perform calculations based around these variables such as the dot and cross products of vectors and the inversion of matrices.

Glut(OpenGL Utility Toolkit) This library contains many functions that can help with basic graphics tasks these include the representation and generation of several shapes such as spheres and pyramids and support for font rendering and input from devices such as mouse or keyboard which will save time in the development process of the application.

3.1.2 Windows SDK

This allows for the creation of application built for windows based systems with the functionality for the creation of detailed user interfaces with a lot of basic functionality built in including the ability to integrate and use OpenGl within the user interface for development of graphics applications.

3.2 Hardware

The minimum requirements of a system to run the created application is as follows:

- Processor - Quad core Intel or AMD with over 2.0GHz clock speed
- Graphics card - Either Nvidia or AMD with OpenGL version of 440
- Memory - The memory usage is inconsequential requiring around 6 MB in RAM

4 Implementation Methodology

This section will analyse how the different areas of the implementation are approached and seek to explain how these approaches work at a technical level.

For a basic outline of the expected execution of the project please refer to figure 9

4.1 Level Generation

To generate the three dimensional terrain used in the project a height-map was generated through the use of the Libnoise external library which is discussed below.

4.1.1 Libnoise External Library

This external library generates different types of noise as shown by figure 3 this is achieved use of various modules that can produce different types of noise including Perlin which is the basis for our output shown in 4 another feature of this library is that it allows the noise generated to be written out to various graphical file types for the height-map in the project a Bitmap (BMP) graphical file type was produced which is then loaded into the project and stored in a data structure for manipulation and rendering.

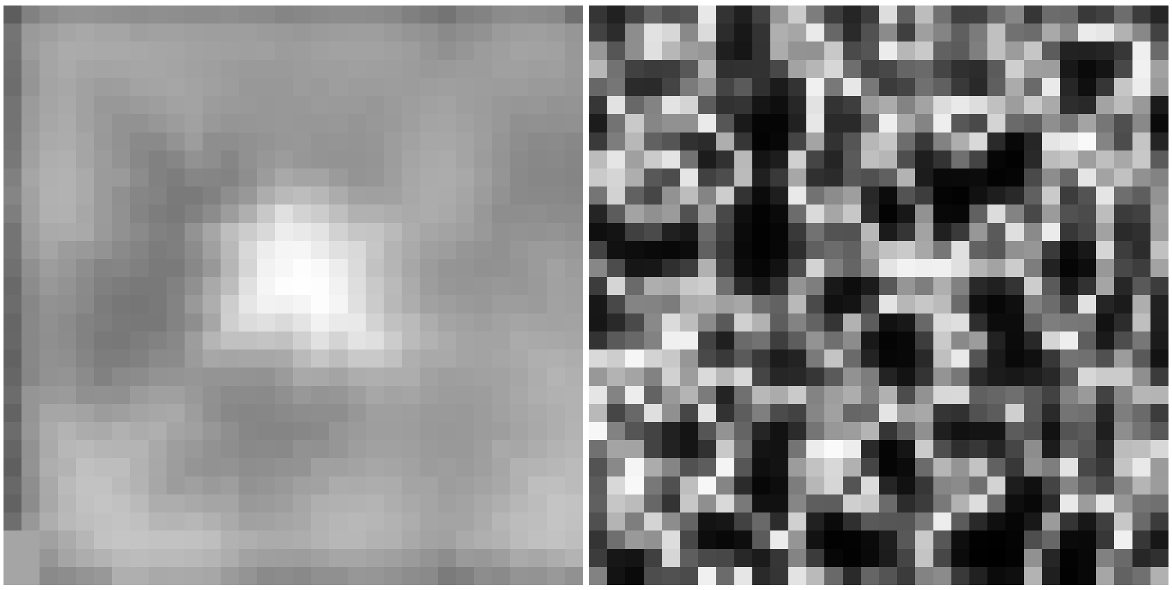


Figure 3: An image showing the output image that was generated by noise
Note: left—- Perlin noise——————————right Rigid multi fractal noise

When this image is rendered to the screen against a depth buffer on the GPU it gives the following output:

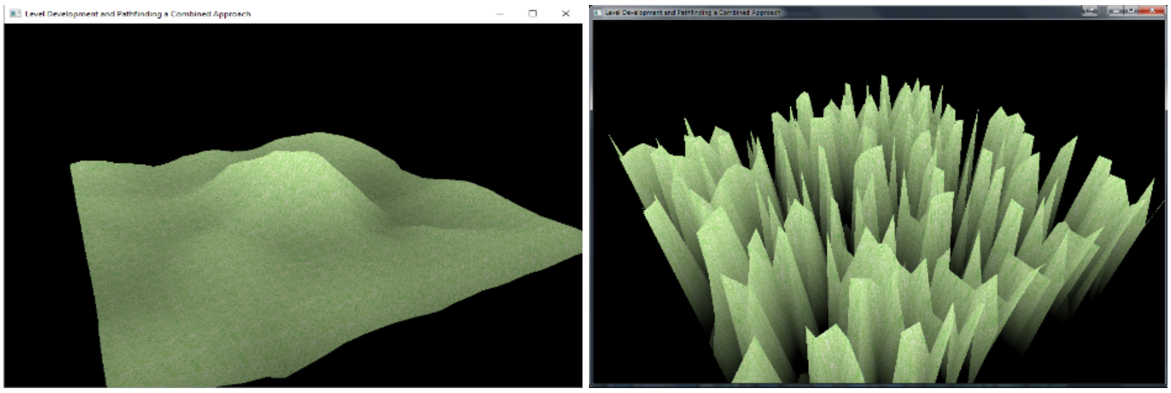


Figure 4: The output from rendering a heightmap
Note: left—- Perlin noise——————————right Rigid multi fractal noise

4.2 Nodes and placement

To place and visualise the nodes that will be utilised for any path finding technique to be used across a terrain. The glut (OpenGL Utility Toolkit) api was used to create a simple method for rendering a sphere these were then placed along the x and z axis of the terrain and position on the y-axis is checked against the height values read in from the height-map to either raise or lower the nodes to suit the terrain.

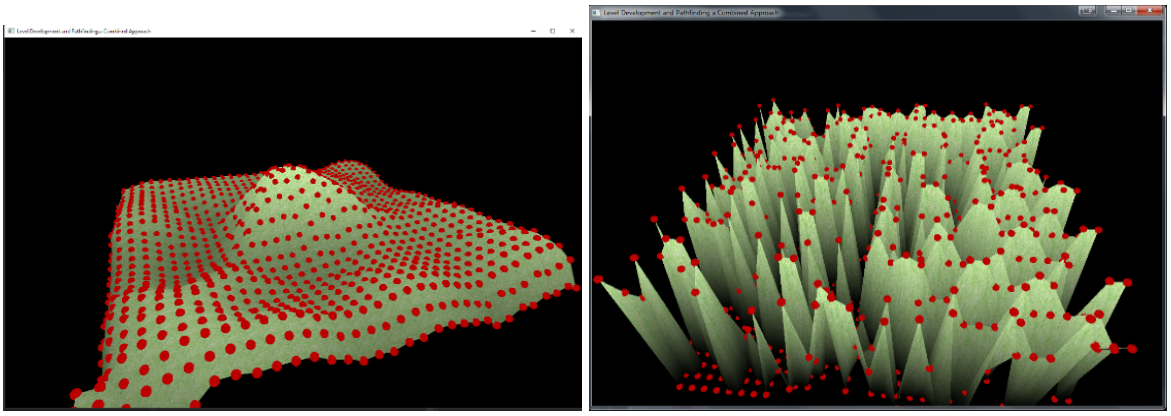


Figure 5: The node placement when drawn on to the terrain

We can see here for the example that utilises rigid multi fractal noise the node placement looks incorrect this is due to the floor of the terrain being so far away from the peaks of the terrain causing the geometry to be culled (not drawn).

4.3 Path finding implementation

To allow comparison between different path finding techniques we will implement both Dijkstra's algorithm and A-star when a path is found the nodes that make up the path will change from red 5 to green.

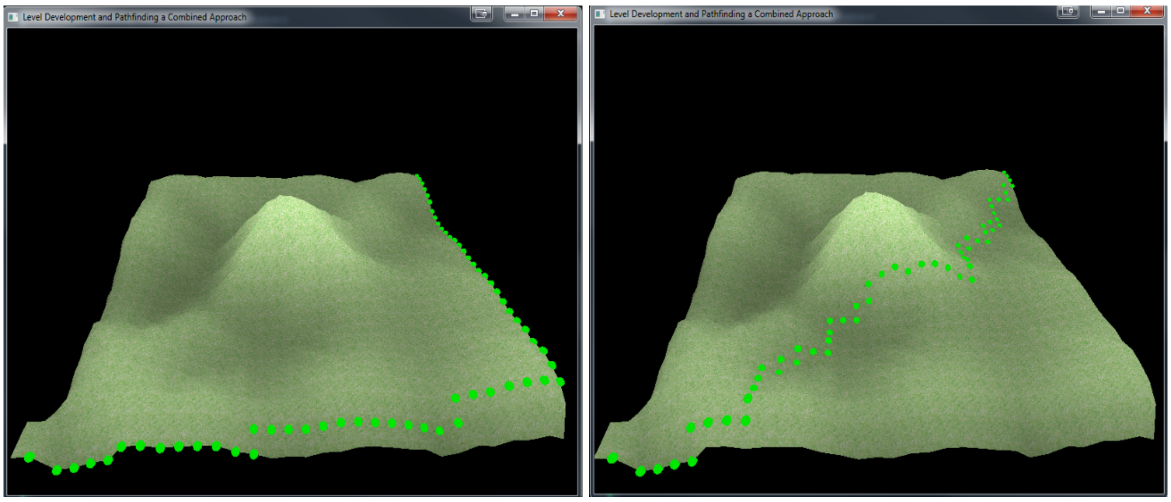

Level Development and Pathfinding a Combined Approach
Level Development and Pathfinding a Combined Approach

Figure 6: The output given be pathfinding— Left:Dijkstra — Right:Astar

4.4 Combining the terrain and path-finding metric

When it comes to the combination of measurements gathered by the project our first concern is what exactly we are measuring which for the path-finding will be the distance of path from start to finish and the time taken to build a path across the terrain.

To measure the quality of terrain we will look at the standard deviation of the normals across a given terrain to start we will combine the timing measurements as they will be on the same scale.

There is an issue when deciding how we measure the distance of a path against the standard deviation of the normals on the terrain as these will be scaled differently now there are a few options here firstly we could take lots of path-finding measurements and by working out how much difference there is between these we could establish a scale that would be able to integrate with the standard deviation of the terrain.

4.5 Additional Information / Knowledge Required

To implement the project a solid knowledge of a graphical programming language such as openGL or DirectX is required also perfomance based GPGPU programming API's such as OpenCL or Nvidia's CUDA could be useful for optimising the project.

The artificial intelligence aspects of the project will require knowledge of various path-finding techniques and also an understanding of machine based learning would be beneficial to allow training of the system for analysis of the terrain and path-finding metric.

4.6 Experimental plan

To determine how well the project works in terms of creating a combined metric for terrain generation and path-finding the following experiments will be run.

4.6.1 Experiment 1

For this experiment the aim is to gather the results of both path-finding techniques over a variety of terrain types generated using various types of noise as this will allow verification of the path finding techniques and lead on towards the integration of a combined metric.

4.6.2 Experiment 2

This will use the results from Experiment 1 to show how the combined metric works and discuss how the metric was established with examples this will be done to verify the reading from the system.

4.6.3 After experiment 2

As experiment two concludes this will show how the project meets with the aims laid out above 1.5 following on from here the main focus will be to increase the run time performance of the project and to generalise the implementation to increase the area's in which the project can be applied.

5 Testing and results

Test Id	Test Name	Test Rationale	results	Retesting result (if modified)
1	Height-map generation and loading	This is the first chosen level generation technique used within the project	pass	n/a
2	Storage and rendering of height-map	This is necessary to test to allow continued use of the height-map technique	pass	n/a
3	Height-map performance	This is needed to establish the viability of the height-map method for the project	pass	n/a
4	Other level generation techniques	This could be required if the height-map approach has issues or is not viable due to low performance	pass	may be revisited as additional functionality
5	Node placement and rendering	This will allow for visualisation of the data for testing and the nodes are required for the path finder	pass	n/a
6	Calculating the distance between all nodes on the generated terrain	This is required for the path finding algorithm	pass	n/a
7	Implementation of Dijkstra and A-star path-finding techniques	This will allow for a path to be built through the terrain	pass	n/a
8	Getting the normals of the terrain for quality measurement	This will be the last functionality that will be needed for the project	pass	n/a

Table 1: A table of tests carried out on the system

5.1 Results

Noise Type	Pathfinder	Size of Path in Nodes	Total Path Distance	TimeTaken	Terrain Quality
Perlin	class AStar	62	30257.46875	0.000949	981
Perlin	class Dijkstra	62	33461.98438	0.000728	981
Rigid Multi-Fractal	class Dijkstra	57	28352.38086	0.001816	1357
Rigid Multi-Fractal	class AStar	57	27775.75977	0.001203	1357

Table 2: A table of results from pathfinding across different terrain types for Experiment 1
Note: for terrain quality lower is better due to the terrain being more level

The results in 2 show us that for path-finding across a terrain that the A star algorithm creates a lower distance path across the terrain with faster processing times than Dijkstra's algorithm we can also see that the terrain quality metric is established with Perlin noise creating the best terrain in terms of the accessibility to area's of the map.

Noise Type	Pathfinder	Size of Path in Nodes	Total Path Distance	TimeTaken	Combined Score
Perlin	class AStar	62	30257.46875	0.000949	2.67
Perlin	class Dijkstra	62	33461.98438	0.000728	1.31
Rigid Multi-Fractal	class Dijkstra	57	28352.38086	0.001816	1.87
Rigid Multi-Fractal	class AStar	57	27775.75977	0.001203	2.25

Table 3: The results with a combined score for Experiment 2

As seen in 3 the terrain quality measurement has been removed and replaced with the combined score to create this score we take the three measurements of pathfinder performance, the total distance of the path created and the quality of terrain and scale these based on how much better the measurements were for a given type of terrain.

The key example of this is that the A star results over a terrain generated by Perlin noise scores higher than running the same parameters over a rigid multi-fractal terrain this is due to the other parameters being better apart from the terrain quality.

6 Conclusions

In summary this project set out to investigate the creation of a combined metric for both path-finding and procedural level generation this was envisioned as a tool for use within a games engine to fine tune either the level or path-finder created on a bias between performance or quality by allowing changes that were made to either the level or path-finder to affect the overall measurement produced.

This combined score was established by testing both Dijkstra and A star path-finding algorithms over terrain that had been generated with both Perlin and rigid multi fractal noise types.

These results show the individual values of each measured element and show the greater performance of the A star algorithm in comparison with Dijkstra even across different terrain types.

In terms of the terrain generated it was shown that using Perlin noise produces a smoother terrain than rigid multi fractal hence it having a smaller value in the terrain quality see the experiment 1 results 2.

From the combined metric results 3 we can see that the weighting system used to create the combined metric works through examination of the relation between these results and the data gathered in experiment 1 2.

7 Bibliography

References

[lev,] Dungeon delving: A short introduction to procedural level generation. https://www.youtube.com/watch?v=VGSHsh83_ns. Accessed: 2015-09-15. 32

[Califano, 2000] Califano, A. (2000). Splash: structural pattern localization analysis by sequential histograms. *Bioinformatics*, 16(4):341–357. 16, 34

[Cormen et al.,] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. Dijkstra's algorithm. pages 595–599. 15, 34

[Dagum and Enon, 1998] Dagum, L. and Enon, R. (1998). Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55. 18

[Davis, 2013] Davis, B. C. (2013). *Terrain generation engine using voxels*. PhD thesis, California State University, Northridge. 15

[Eastman, 1989] Eastman, J. R. (1989). Pushbroom algorithms for calculating distances in raster grids. In *Proc. Autocarto*, volume 9, pages 288–297. 34

[Fournier et al., 1982] Fournier, A., Fussell, D., and Carpenter, L. (1982). Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384. 13

[Goodchild, 1977] Goodchild, M. (1977). An evaluation of lattice solutions to the problem of corridor location. *Environment and Planning A*, 9(7):727–738. 16, 34

[Graham et al., 2003] Graham, R., McCabe, H., and Sheridan, S. (2003). Pathfinding in computer games. *ITB Journal*, 8:57–81. 17

[Gray, 2012] Gray, J. ((2012)). Evaluating procedural noise for generation of 3d terrain. *BEng (Hons) Games Development Dissertation). Edinburgh Napier University (Chalmers, K., Cummming, A.* 18

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107. 17

[Hendrikx et al., 2013] Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1):1:1–1:22. 13

[Huber and Church, 1985] Huber, D. L. and Church, R. L. (1985). Transmission corridor location modeling. *Journal of Transportation Engineering*, 111(2):114–130. 34

[Husdal, 2015] Husdal, J. (2015). Corridor analysis – a timeline of evolutionary development. *Unpublished coursework. University of Utah, USA*. 34

[Kerin, 2012] Kerin, A. (2012). The history of graphics: Software's sway over silicon. In *Handbook of Visual Display Technology*, pages 2511–2517. Springer. 12

[Kilgard and Akeley, 2008] Kilgard, M. J. and Akeley, K. (2008). Modern opengl: Its design and evolution. In *ACM SIGGRAPH ASIA 2008 Courses*, SIGGRAPH Asia '08, pages 13:1–13:31, New York, NY, USA. ACM. 12

[Lombard and Church, 1993] Lombard, K. and Church, R. (1993). The gateway shortest path problem: generating alternative routes for a corridor location problem. *Geographical systems*, 1(1):25–45. 34

[Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM. 14

[McIlhagga, 1997] McIlhagga, D. (1997). Optimal path delineation to multiple targets incorporating fixed cost distance. *Unpublished Thesis, Carleton University, Ottawa, ON.* 34

[McMillan, 2014] McMillan, C. ((2014)). Comparison of pathfinding algorithms using the gpgpu. *BEng (Hons) Games Development Dissertation). Edinburgh Napier University (Hart, E., Urquhart, N.* 17, 33

[Miller, 1986] Miller, G. S. P. (1986). The definition and rendering of terrain maps. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 39–48, New York, NY, USA. ACM. 14

[Perlin, 1985] Perlin, K. (1985). An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296. 15

[Shaker et al., 2015] Shaker, N., Togelius, J., and Nelson, M. J. (2015). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research.* Springer. 33

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489. 19

[Snir, 1998] Snir, M. (1998). *MPI–the Complete Reference: The MPI core*, volume 1. MIT press. 18

[Sturtevant and Buro, 2005] Sturtevant, N. and Buro, M. (2005). Partial pathfinding using map abstraction and refinement. *AAAI*, 5:1392–1397. 33

[Wikipedia, 2015] Wikipedia (2015). Akalabeth: World of doom — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Akalabeth:_World_of_Doom&oldid=670318824. [Online; accessed 13-October-2015]. 10

[Wikipedia, 2016a] Wikipedia (2016a). Directx — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=DirectX&oldid=714053122. [Online; accessed 20-April-2016]. 12

[Wikipedia, 2016b] Wikipedia (2016b). Opengl — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=OpenGL&oldid=711501805. [Online; accessed 20-April-2016]. 12

Appendices

A Project Overview

Keywords— Level-Generation,Path-Finding,Algorithm,Optimisation

A.1 Overview of Project Content and Milestones

This project will include research into current practices used in industry for path-finding and level generation with the emphasis being performance based compared against realism. The main milestone for this stage will be coming across a method to integrate both level generation and path-finding into an algorithm which will be carried forward to the development stage.

Development of an application which will utilise both level generation and path-finding either by the modification and optimisation of existing approaches or the creation of a new algorithm. The application will allow the user to create various sizes of level and the application will then give a visual representation of both the level and path that has been built.

The application will then be optimised and tested against some of the methods researched with the aim to show that a combined approach may achieve better results in a similar or faster run time with the paths created being checked for mistakes or inconsistencies based on various sanity checks (no path nodes on areas that should not be traversable such as extreme slopes or dips in the level).

Creation of a report that contains all project information and related work with a detailed analysis and testing of the approach used to provide an evaluation of the solution using performance figures to accurately examine the viability of this approach in comparison to the current industry standards and draw a conclusion over the project as a whole.

For a more detailed look at the time-line for this project please refer to Figure 8

A.2 The Main Deliverables

- A application that will allow the user to create various sizes of level and will give a visual output of the level and path built with a focus on the optimisation techniques used to integrate level generation and path finding.

- A detailed report containing an analysis on the approach taken and a comparison between the solution and other methods used currently using figures from both to evaluate the effectiveness and viability of the approach used.

- Test logs of the finished application and an industry standard practice to allow for an unbiased comparison to be made based on results obtained from these tests when documenting the project.

A.3 Target audience for the deliverables

The deliverables of this project may be of interest to people involved with both level generation and path finding in the games development industry. Researchers involved in this field may find the report and comparison of techniques used to be of interest. Other people may find this project helpful with regards to optimisation techniques. The main aim of the project is to show the performance of a combined approach to level generation and path-finding and an evaluation of the quality of paths built.

A.4 The work to be undertaken

- Research into current industry practices for level generation and path finding to allow for visualization of a solution to combine these algorithms.

- Development of the application and chosen approach with a focus on performance while maintaining a level of realism.

- Optimisation of the developed solution and gathering of test data from the solution and an industry approach.

- Creation of a report detailing the approach taken to implementing the project and providing an evaluation between the solution compared to industry practices.

There is a diagram showing expected steps of execution within the application shown at Figure 9

A.5 Knowledge and skills required

To be able to complete the project to a high standard I will first need to research the field's of both level generation and path finding as a solid knowledge will be needed to create a combined approach. In terms of development further research into optimisation techniques for C++ in Visual Studio will allow for a more polished and better performing solution also as a level of realism will be maintained experience with graphic programming would also benefit the project.

A.6 Information sources that provide a context for the project

The video available at [lev,] gives a few very basic examples of levels that can be generated and also details some methods to define areas within a level(slopes,walls,and others) this definition of areas within a level could be used for testing the path finding within the project and may give a basic concept on how to generate content within the level produced in relation with the project.

The paper[Sturtevant and Buro, 2005] analyses various path-finding algorithms however the purpose of their project was to look at an alternative approach to the A* path finding algorithm. Use of concepts such as map abstraction could be of interest to this project as it would allow a way to break the level into chunks which can then be processed individually by a path finding algorithm.

The textbook for level generation [Shaker et al., 2015] Provides extensive research into procedurally generated content in games from grid based levels to full landscapes there are multiple approaches detailed within the textbook that can provide a basis to generate a level within the project.

Previous honours projects to note
This paper[McMillan, 2014] analyses various path finding algorithms and discusses at optimisation options this will be the inspiration for optimising the project and will give a wide variety of path finding algorithms to research going forward.

For a timeline of research into path finding algorithms please refer to Figure 7

A.7 The importance of the project

This project is important as it will seek to integrate level generation and path finding which could allow for faster loading times in games while reducing the storage space taken by a game. When comparing the difference between developed solution and current industry practices - if the solution proves to be better in terms of performance and the quality of the path built is of a suitable standard for the complexity of the level that is generated.

A.8 The key challenges to be overcome

- Research of a variety of level generation and path-finding techniques to gain knowledge and propose a viable solution for development.

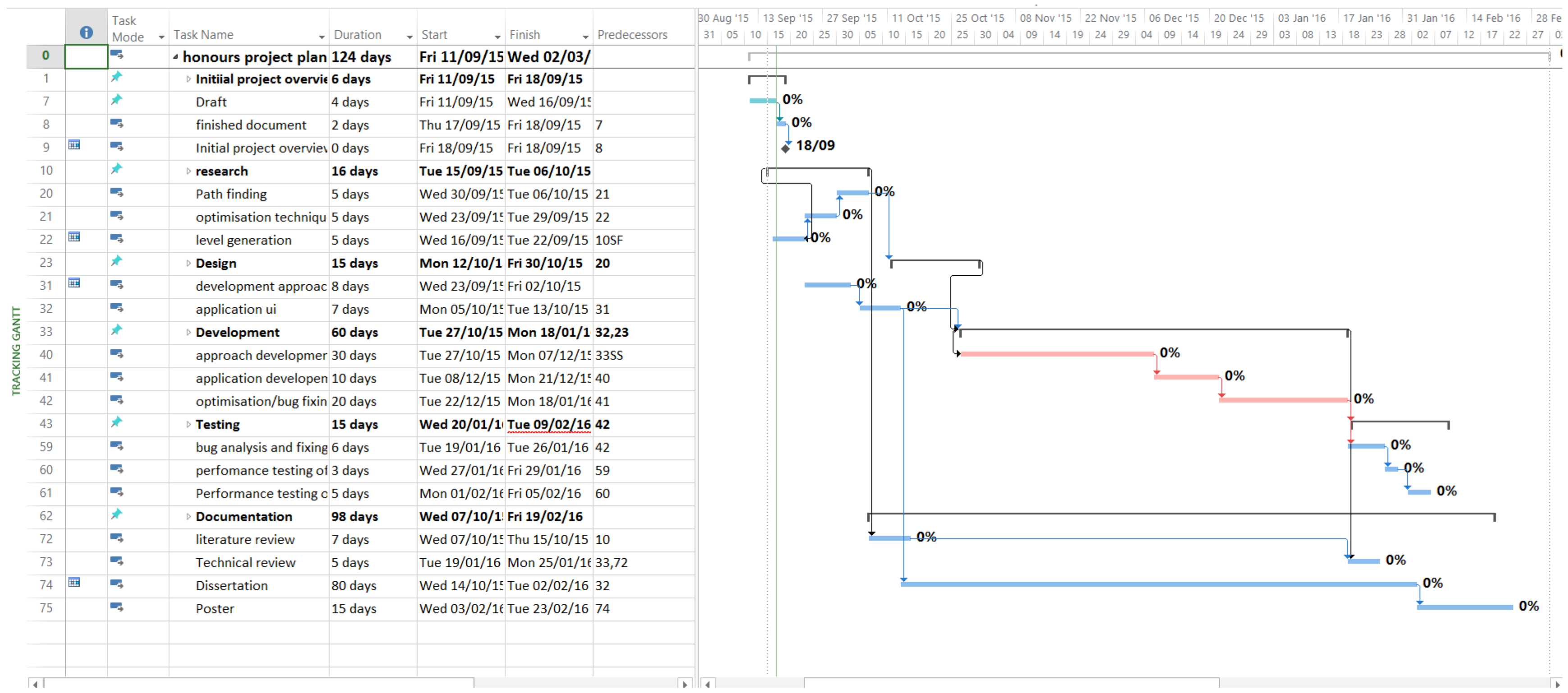
- Development of an application that will use the envisioned solution and give a visual output to the user of both level and path built.

- Research into a way of determining the quality of path finding and level generation techniques.

- Writing of a detailed report that will clearly explain the solution chosen and give a view into past work done within this field then the report will provide an unbiased comparison between the approach and current industry practices to evaluate the project in terms of the quality the path built when compared to the complexity of the level generated.

Creator	Method	Year
Dijkstra [Cormen et al.,]	Graph searching	1959
Goodchild [Goodchild, 1977]	Orthagonal and diagonal movement	1977
Huber and Church [Huber and Church, 1985]	analysis of neighboring cells	1985
Eastman [Eastman, 1989]	Pushbroom procedure	1989
Lombard and Church [Lombard and Church, 1993]	GSP(Gateway-shortest-path)	1993
Mcilhagga [McIlhagga, 1997]	Fixed cost distance	1997
Andrea Califano [Califano, 2000]	Splash algorithm	2000

Figure 7: A time-line of path finding algorithms
[Husdal, 2015]



	Task Mode	Task Name	Duration	Start	Finish	Predecessors
0		**honours project plan**	**124 days**	**Fri 11/09/15**	**Wed 02/03/**	
1		**Initiial project overvie**	**6 days**	**Fri 11/09/15**	**Fri 18/09/15**	
7		Draft	4 days	Fri 11/09/15	Wed 16/09/1	
8		finished document	2 days	Thu 17/09/15	Fri 18/09/15	7
9		Initial project overvie	0 days	Fri 18/09/15	Fri 18/09/15	8
10		**research**	**16 days**	**Tue 15/09/15**	**Tue 06/10/15**	
20		Path finding	5 days	Wed 30/09/1	Tue 06/10/15	21
21		optimisation techniqu	5 days	Wed 23/09/1	Tue 29/09/15	22
22		level generation	5 days	Wed 16/09/1	Tue 22/09/15	10SF
23		**Design**	**15 days**	**Mon 12/10/1**	**Fri 30/10/15**	**20**
31		development approac	8 days	Wed 23/09/1	Fri 02/10/15	
32		application ui	7 days	Mon 05/10/1	Tue 13/10/15	31
33		**Development**	**60 days**	**Tue 27/10/15**	**Mon 18/01/1**	**32,23**
40		approach developmer	30 days	Tue 27/10/15	Mon 07/12/1	33SS
41		application developen	10 days	Tue 08/12/15	Mon 21/12/1	40
42		optimisation/bug fixin	20 days	Tue 22/12/15	Mon 18/01/1	41
43		**Testing**	**15 days**	**Wed 20/01/1**	**Tue 09/02/16**	**42**
59		bug analysis and fixing	6 days	Tue 19/01/16	Tue 26/01/16	42
60		perfomance testing of	3 days	Wed 27/01/1	Fri 29/01/16	59
61		Performance testing c	5 days	Mon 01/02/1	Fri 05/02/16	60
62		**Documentation**	**98 days**	**Wed 07/10/1**	**Fri 19/02/16**	
72		literature review	7 days	Wed 07/10/1	Thu 15/10/15	10
73		Technical review	5 days	Tue 19/01/16	Mon 25/01/1	33,72
74		Dissertation	80 days	Wed 14/10/1	Tue 02/02/16	32
75		Poster	15 days	Wed 03/02/1	Tue 23/02/16	74

Figure 8: An image of the project gantt chart

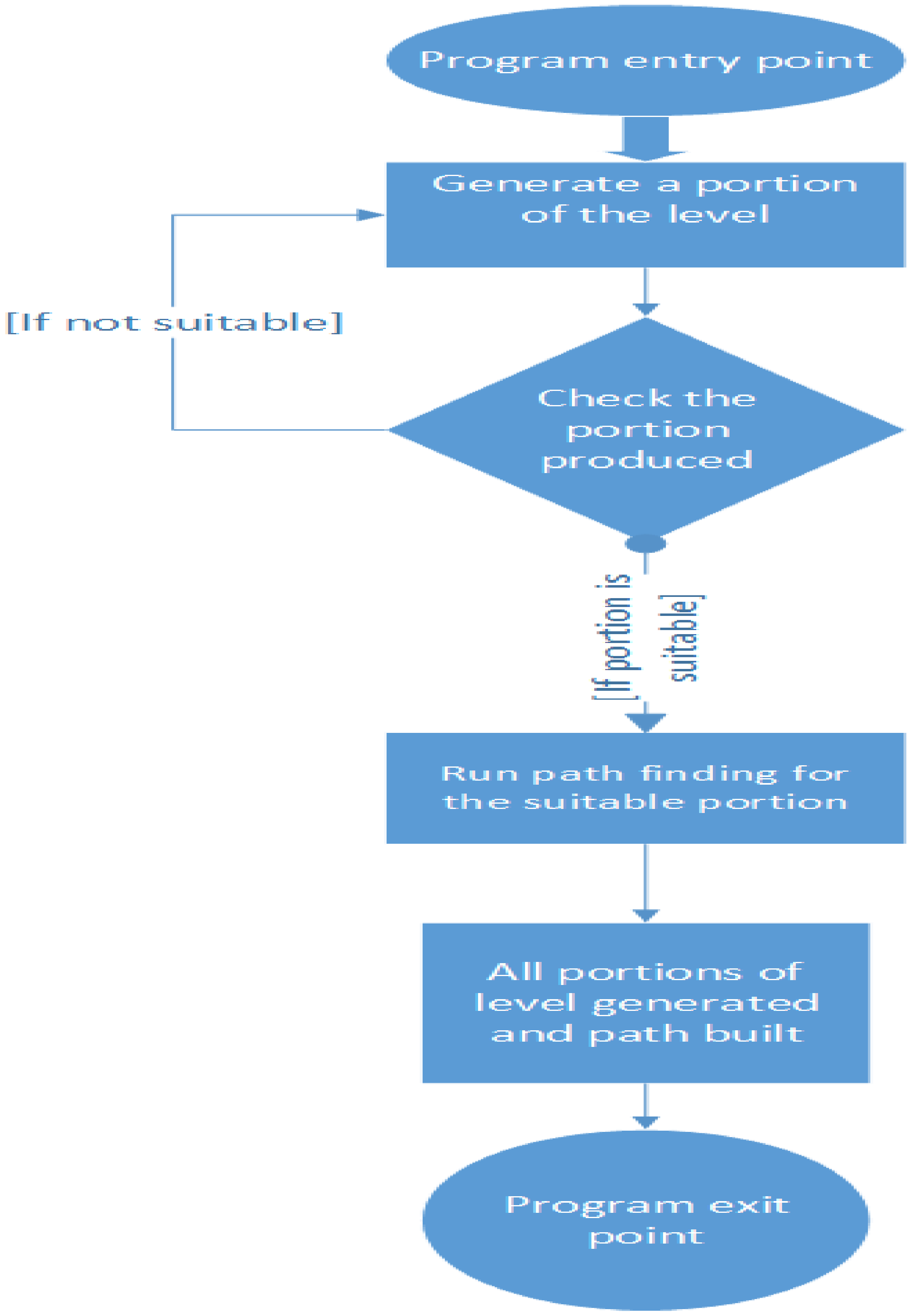

Program entry point
Generate a portion of the level
[If not suitable]
Check the portion produced
[If portion is suitable]
Run path finding for the suitable portion
All portions of level generated and path built
Program exit point

Figure 9: This shows how the program will execute

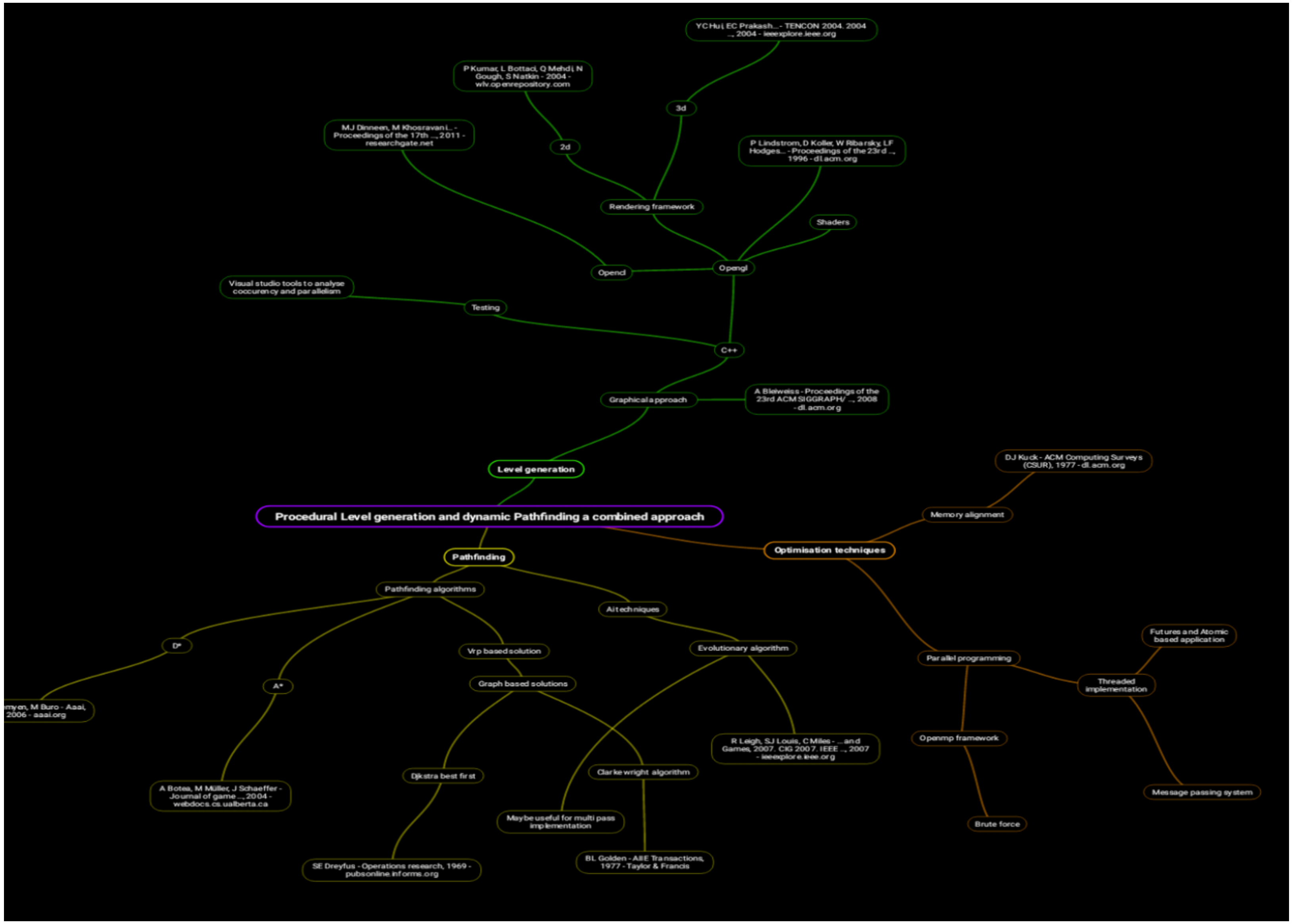

Procedural Level generation and dynamic Pathfinding a combined approach
Level generation
Graphical approach
A Bleiweiss - Proceedings of the 23rd ACM SIGGRAPH/ ..., 2008 - dl.acm.org
C++
Testing
Visual studio tools to analyse coccurency and parallelism
Opengl
Opencl
MJ Dinneen, M Khosravani ... - Proceedings of the 17th ..., 2011 - researchgate.net
Shaders
P Lindstrom, D Koller, W Ribarsky, LF Hodges... - Proceedings of the 23rd ..., 1996 - dl.acm.org
Rendering framework
2d
P Kumar, L Bottaci, Q Mehdi, N Gough, S Natkin - 2004 - wlv.openrepository.com
3d
YC Hui, EC Prakash... - TENCON 2004. 2004 ..., 2004 - ieeexplore.ieee.org
Pathfinding
Pathfinding algorithms
D*
myen, M Buro - Aaai, 2006 - aaai.org
A*
A Botea, M Müller, J Schaeffer - Journal of game ..., 2004 - webdocs.cs.ualberta.ca
Vrp based solution
Graph based solutions
Djkstra best first
SE Dreyfus - Operations research, 1969 - pubsonline.informs.org
Clarke wright algorithm
BL Golden - AIIE Transactions, 1977 - Taylor &amp; Francis
Ai techniques
Evolutionary algorithm
Maybe useful for multi pass implementation
R Leigh, SJ Louis, C Miles - ... and Games, 2007. CIG 2007. IEEE ..., 2007 - ieeexplore.ieee.org
Optimisation techniques
Memory alignment
DJ Kuck - ACM Computing Surveys (CSUR), 1977 - dl.acm.org
Parallel programming
Openmp framework
Brute force
Threaded implementation
Futures and Atomic based application
Message passing system

Figure 10: This is a mind map covering the scope of the project

B Diary Sheets and project management

B.1 Week 2 and before

This was the point where a supervisor for the project was established and formal meetings were begun from week 3.

B.2 Meeting 1 22/9/15 Week 3

Objectives
update literature/background
Polish IPO for submission
Block diagram

Progress
First draft of IPO and Project plan
First meeting

Comments
On Track/looking good

B.3 Meeting 2 29/9/15 Week 4

Objectives
start drafting literature review
Mind map
Add keywords
setup version control with basic renderer
Comparison grid

Progress
IPO handed in
Collecting papers for research
Setup blog

Comments
On Track

B.4 Meeting 3 6/10/15 Week 5

Objectives
Thesis template
Add main bib file
Screen-shots from implementation
Add mind map
Write introduction
Comparison grid

Progress

Skeleton Framework
Continued Collecting papers for research
Added keywords to IPO

Comments
Starting to slow down

B.5 Meeting 4 13/10/15 Week 6

Objectives
Screenshots
Add mind map and comparison grid to thesis
Fix and complete level generation implementation
Draft lit review
Write introduction
Blog updates
Print and bring thesis to next meeting

Progress
Height-map generation code implemented
Continued Collecting papers for research
Drafted thesis and started introduction

Comments
Need to improve time management
On track-good work

B.6 Meeting 5/Group Thesis Review 20/10/15 Week 7

Feedback
Report looking good so far minor changes to be made to already completed sections.

Progress
Mind map finished and added to thesis.
Loading of height-map code fixed.

B.7 Meeting 6 27/10/15 Week 8

Meeting was missed however general discussion was had with supervisor during the week to state goals for this week and record progress informally.

B.8 Meeting 6 3/11/15 Week 9

Objectives
Update thesis
Look at commercial packages for comparison
Implementation work (Test cases)
Print and bring thesis to next meeting

Progress
Numerical experimentation for level generation
Started writing implementation methodology
almost finished lit review(Draft version)

Comments
Coursework week(busy)
Second marker meeting next week

B.9 Meeting 7 10/11/15 Week 10

Objectives
Update thesis
Fix rendering
Implementation work (Test cases)
Blog/Website updates Print and bring thesis to next meeting

Progress
Started to fix development issues(rendering)
Thesis Work
Re-done blog/website (more structured)

Comments
Review thesis progress next week
Week 10-Second marker meeting organisation

B.10 Meeting 8 17/11/15 Week 11

Second marker meeting.

Feedback Following the meeting it was agreed that the project would be based around the creation of a testing mechanism instead of being solely performance based the thesis work to this point was said to be under what was expected however as the project aims shifted this allowed less changes to be made overall there was also guidance on existing projects within the university.

B.11 Meetings up to 21/3/16

There has been a lack of formal communication with weekly updates being discussed at any point where we happened to meet however there has been no meetings recently this is due to the absence of my supervisor.

B.12 Meeting 9 21/3/17

This meeting was a review of the dissertation to this point after the meeting constructive feedback on structure and progress of the dissertation was given also a discussion of testing the finished project took place.

C Second Formal Review Output