FRASER BARKER

1600196

1600196@uad.ac.uk

ASSESSMENT REPORT

CMP305

Contents

[Features 2](#_Toc6948390)

[Controls 2](#_Toc6948391)

[Camera 2](#_Toc6948392)

[ImGui 2](#_Toc6948393)

[Smoothing 3](#_Toc6948394)

[Faulting 3](#_Toc6948395)

[Circle Algorithm 3](#_Toc6948396)

[Midpoint Displacement 3](#_Toc6948397)

[Simplex Noise 3](#_Toc6948398)

[Fractal Brownian Motion 3](#_Toc6948399)

[Voronoi Regions 3](#_Toc6948400)

[Terrain Picking 3](#_Toc6948401)

[L-Systems 3](#_Toc6948402)

[Organisation 4](#_Toc6948403)

[Terrain 4](#_Toc6948404)

[L-System 4](#_Toc6948405)

[Sizeable Quad 4](#_Toc6948406)

[Simplex Noise 4](#_Toc6948407)

[Terrain Shader 4](#_Toc6948408)

[Manipulation Shader 4](#_Toc6948409)

[Critical Appraisal 5](#_Toc6948410)

[Terrain Class 5](#_Toc6948411)

[L-System 5](#_Toc6948412)

[Tessellation 5](#_Toc6948413)

[Overall 5](#_Toc6948414)

[Changes 5](#_Toc6948415)

[Reflection 6](#_Toc6948416)

[Topics 6](#_Toc6948417)

[Further/Alternative Applications 6](#_Toc6948418)

[References 7](#_Toc6948419)

# Features

1. An outline of the general features of your program
2. A description of features such as visual and sound effects or interactivity.

## Controls

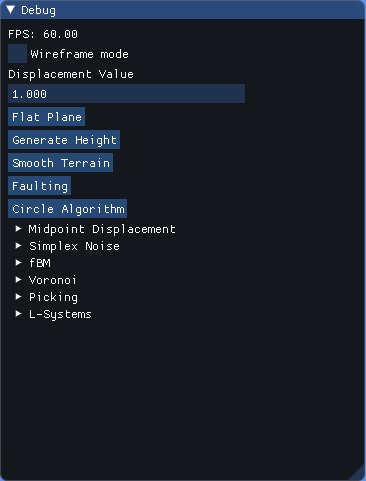
### Camera

The camera can be moved around the scene using the following controls.

* W – Move forward.
* S – Move backwards.
* A – Move left.
* D – Move Right
* Q – Move down.
* E – Move up.
* Spacebar – Toggle camera rotation via mouse/free mouse from screen boundaries.
* Up Arrow – Tilt up.
* Down Arrow – Tilt down.
* Left Arrow – Rotate left.
* Right Arrow – Rotate right.
* Mouse – Used to rotate the camera more freely around the scene. Can be toggled via Spacebar.

### ImGui

The application has a GUI integrated into which allows the user to alter specific values relating to the terrain in real-time rather than having to map specific function to certain keys.



When the user has the mouse unlocked from the centre of the screen, they can interact with the above ImGui menu. As you can see, the menu is composed of many sections which relate to specific terrain altering functions. These functions are detailed below.

## Basic Features

The ImGui menu show the current framerate of the application. This is useful as it really shows just how terrible that one function you wrote really was as you watch the framerate plummet.

It also has a “Wireframe mode” toggle which -as the name suggests- allows the user to see all the objects in the scene in their wireframe mode.

[WIREFRAME PICTURE]

The displacement value is a float that is passed into most -if not all- of the terrain altering functions to determine how much the terrain should be displaced by. This may have been a poor choice, as you can sometimes forget you set it to a high value when you begin experimenting with other features which start to cause bizarre results.

The button “Flat Plane” simply does as the name suggests, acting as a reset button it allows the user to revert the terrain back to its default flat state.

The button “Generate Height” was more of an introductory feature that kind of just stuck around. Very basic in that all it does is loop through the terrain’s vertices array assigning random values to the y-position at each point.

[UGLY DEFAULT HEIGHT GENERATION PICTURE]

## Smoothing

Following the “Simple terrain smoothing” blogpost, the Smoothing function again loops through the terrain’s vertices array checking the neighbours of each vertex and averages the value for the y-position.

[BEFORE SMOOTHING PICTURE] [AFTER SMOOTHING PICTURE]

Its current implementation could do with a bit of refinement, but I’ll cover this in the critical appraisal section.

## Faulting

The Faulting function utilises the fault line algorithm (as detailed in *The Fault Algorithm*). This works based on choosing two random points at each side of the plane, drawing a line between them and depending on what side of the line a vertex is, its position is either raised or lowered.

[FAULTING PICTURE]

## Circle Algorithm

Quite possibly my favourite function of the application. There are two variations of this function; a random circle algorithm and a refined circle algorithm which is used in the terrain picking function. The random circle algorithm loops one hundred times choosing random points on the plane to alter based on whether a vertex is within a circle of random radius around the random point.

[CIRCLE ALGORITHM PICTURE]

## Midpoint Displacement

The Midpoint Displacement function (as adapted from *Terrain Generation with Midpoint Displacement*) alters the terrains height by getting the “corners” y-position value and setting the mid-points y-value of these two “corners” to be an average of the two y-position values. This is the recursively done as it moves closer to the centre of the terrain.

Note: The terrain **must** be an odd value dimension for this function to work. E.g. 3, 5, 7, 9 etc.

[MIDPOINT DISPLACEMENT PICTURE]

## Simplex Noise

The Simplex Noise function (as detailed in *Simplex Nosie Demystified*) utilises the simplex noise algorithm – a replacement for Ken Perlin’s “Classic Perlin Noise” algorithm.

This is an improvement of Perlin’s Class Noise algorithm in that it is less computationally complex, scales to higher dimensions with less computational cost and has no noticeable directional artefacts.

It works by sample an array for a gradient value that you use to displace the entire terrain as you displace it based on its position.

[SIMPLEX NOISE PICTURE]

## Fractal Brownian Motion

More of an extension of conventional noise function in that it would use something like Simplex Noise as a base (this application does) and merely repeats the noise function based on set values.

[fBM PICTURE]

## Voronoi Regions

Like the fault line algorithm, the Voronoi Regions function splits the terrain into sections. This works by picking random points on the terrain, deciding (based on nearest distance to these random points) which “region” these points belong to, and then finally displacing the height based on what region it is in.

[VORONOI PICTURE]

## Terrain Picking

Essentially, ray-casting. When it is enabled, and the user clicks on the terrain, the screen co-ordinates of the mouse are converted into world co-ordinates via calculating a directional vector from the cameras position to the terrain. Where this directional vector intersects the terrain, a check is done to see what specific vertices it is nearest to. For the vertices near this intersection point, the refined circle algorithm is used to displace them if they are near this point and in the radius of the circle in the algorithm.

[TERRAIN PICKING PICTURE]

## L-Systems

Linden-Mayer systems or L-Systems are used to generate self-similar fractals. In this applications case, it generates a 2D river as it follows two specific rules. The L-System could be better integrated with the terrain, but this will be explored in more detail in the Critical Appraisal section. Based on a starting character, some input rules and an interpreter to determine what should be done when coming across certain character (E.g. F means move forward), a variety of shapes can be made.

[L-SYSTEM PICTURE]

# Organisation

1. A description of your code organization: classes, data structures, separate code files etc. Include diagrams as required.

## Terrain

## L-System

## Sizeable Quad

## Simplex Noise

## Terrain Shader

## Manipulation Shader

# Critical Appraisal

1. A critical appraisal of your program in relation to code organization, efficiency of strategies and the decisions regarding alternative strategies and a critical evaluation of the success of your procedural content.

## Terrain Class

Non-utilisation of the GPU for semi-expensive calculations.

## L-System

More central integration with the terrain.

## Tessellation

Should have been utilised for both terrain and l-system.

## Overall

Giant success!

## Changes

Better texturing, lighting, use of tessellation.

# Reflection

1. Reflection on what has been learned, technically in terms of programming and also about procedural methods from undertaking the work, concentrating on being informative to someone else who might take on a similar task.

## Topics

## Further/Alternative Applications

# References

Elias Daler. 2019. Dear ImGui: Bloat-free Immediate Mode Graphical User interface for C++ with minimal dependencies. [ONLINE] Available at: <https://github.com/ocornut/imgui> [Accessed 23/04/2019]

Nic. 2013. Simple terrain smoothing. [ONLINE] Available at:

<http://nic-gamedev.blogspot.com/2013/02/simple-terrain-smoothing.html> [Accessed 23/04/2019]

Antonio Ramires Fernandes. 2019. The Fault Algorithm. [ONLINE] Available at:

<http://www.lighthouse3d.com/opengl/terrain/index.php?fault> [Accessed 23/04/2019]

Steve Losh. 2016. Terrain Generation with Midpoint Displacement. [ONLINE] Available at:

<http://stevelosh.com/blog/2016/02/midpoint-displacement/> [Accessed 23/04/2019]

Stefan Gustavson. 2005. Simplex noise demystified. [ONLINE] Available at:

<http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf> [Accessed 23/04/2019]

Patricio Gonzalez Vivo & Jen Lowe. 2015. Fractal Brownian Motion. [ONLINE] Available at:

<https://thebookofshaders.com/13/> [Accessed 23/04/2019]

Glare Technologies. 2019. Advanced Fractional Brownian Motion Noise. [ONLINE] Available at:

<https://www.indigorenderer.com/indigo-technical-reference/indigo-shader-language-reference/built-functions-%E2%80%93-procedural-noise-fun-0> [Accessed 23/04/2019]

Matej Zabsky. 2016. Procedural Terrain with Ridged Fractal Noise. [ONLINE] Available at: <https://stackoverflow.com/questions/36796829/procedural-terrain-with-ridged-fractal-noise> [Accessed 23/04/2019]

Red Blob Games. 2016. Making maps with noise functions. [ONLINE] Available at:

<https://www.redblobgames.com/maps/terrain-from-noise/> [Accessed 23/04/2019]

Paul Bourke. 1989. Efficient Triangulation Algorithm Suitable for Terrain Modelling. [ONLINE] Available at: <http://paulbourke.net/papers/triangulate/> [Accessed 23/04/2019]

Iedoc. 2015. Picking. [ONLINE] Available at: <https://www.braynzarsoft.net/viewtutorial/q16390-24-picking> [Accessed 23/04/2019]

Przemyslaw Prusinkiewicz & James Hanan. 2016. Lindenmayer Systems, Fractals, and Plants. [ONLINE] Available at: <http://algorithmicbotany.org/papers/lsfp.pdf> [Accessed 23/04/2019]

Richard Hawkes. 2016. Unity Algorithms – The Lindenmayer Series (L Series). [ONLINE] Available at: <https://www.youtube.com/watch?v=uBEA6VSUybk> [Accessed 23/04/2019]

Sher Minn Chong. 2018. L-Systems Renderer. [ONLINE] Available at: <http://piratefsh.github.io/p5js-art/public/lsystems/> [Accessed 23/04/2019]

Rastertek. Unknown. Slope Based Texturing. [ONLINE] Available at: <http://www.rastertek.com/tertut14.html> [Accessed 23/04/2019]

Xing Mei, Philippe Decaudin & Bao-Gang Hu. 2007. Fast Hydraulic Erosion Simulation and Visualization on GPU. [ONLINE] Available at: <http://www-evasion.imag.fr/Publications/2007/MDH07/FastErosion_PG07.pdf> [Accessed 23/04/2019]

Florian Bosch. 2011. WebGL Landscaping and Erosion. [ONLINE] Available at: <http://codeflow.org/webgl/craftscape/> [Accessed 23/04/2019]