CRANFIELD UNIVERSITY

MIGUEL MARQUES

A WEB BASED TERRAIN MODELLER USING FRACTALS AND CAD

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Computational and Software Techniques in Engineering
Digital Signal and Image Processing

MSc
Academic Year: 2015–2016

Supervisor: Dr Peter Sherar
August 2016

CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Computational and Software Techniques in Engineering
Digital Signal and Image Processing

MSc

Academic Year: 2015–2016

MIGUEL MARQUES

A Web Based Terrain Modeller using Fractals and CAD

Supervisor: Dr Peter Sherar
August 2016

This thesis is submitted in partial fulfilment of the requirements for the degree of MSc.

ABSTRACT

Type your abstract here.

Keywords

Procedural Generation; Terrain Modelling;

ACKNOWLEDGEMENTS

The author would like to thank . . .

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

fBm	Fractional Brownian Motion
PMHCI	Picewise Monotone Hermite Cubic Interpolation
HPF	High-pass Filter

1 INTRODUCTION

Procedural generation enables designers to create content algorithmically instead of manually. This technique is used in entertainment areas, such as, 3D animation and video game design.

One particular case in which procedural generation is used is the creation of random terrains, which is the focus of this thesis.

1.1 Problem

The main advantage of procedural content against manually created content is that the former is less time consuming and, as a result, enables designers to create detailed content faster. On the other hand the existent methods for random terrain generation are difficult to control and, sometimes, the parameters are unintuitive, which leads to a trial and error design approach, in which the designer changes the parameters and sees what happens.

1.2 Main Requirements

In terms of requirements they were divided in technical and functional requirements. The main technical requirements are:

- The implemented application must be web based
- The Random generation process must be based in fractal geometry
- Describe requirements with market tendencies

1.3 Proposed Solution

In this thesis a hybrid approach to content creation is proposed. This approach involves the modelling of a deterministic cad model that provides the basic topology of the terrain and the generation of a random terrain that will provide the details for that topology. This two surfaces are then blended together using a method that is explained in Chapter 3. The main focus of this project is the implementation

of the blending process and, as a result, the developed application does not deal with the cad modelling phase, but only with the random generation and blending phases.

1.4 Thesis Outline

This thesis is divided as follows:

- Chapter 2 contains the literature review.
- Chapter 3 explains the methodology.
- Chapter 4 presents the software details and constraints.
- Chapter 5 presents the results.
- Chapter 6 presents the conclusions.

2 LITERATURE REVIEW

Procedural generation of terrains has been the focus for many graphics researchers for some time now. Through the years this research has been mostly based on fractional Brownian motion and its similarity to a skyline of mountains, first noticed by Mandelbrot in [1].

2.1 Mathematical Background

2.1.1 Fractal Dimension

> "A fractal is by definition a set for which the Hausdorff Besicovitch dimension strictly exceeds the topological dimension." [1, p.15]

The Hausdorff Besicovitch dimension, also called fractional or fractal dimension, is a measure used to characterize a fractal. This dimension D_f does not need to be an integer and, for a surface in $\mathbb{R}^n$, $D_f \in [n, n+1)$.

2.1.2 Fractional Brownian Motion

Fractional Brownian Motion (fBm) was introduced by Mandelbrot and van Ness in [2] and it is an extension of Brownian Motion that uses the Hurst Exponent (H) as a parameter to control the correlation between successive values, such that, $0 < H < 1$. More specifically if $H = 0.5$ then fBm is just normal Brownian Motion and, due to that, the increments are independent and not correlated; if $H > 0.5$ the increments have positive correlation and this results in smooth curves while if $H < 0.5$ the increments have negative correlation and this results in erratic curves. [3]

In the field of fractal terrain generation fBm is approximated by $1/f^{\beta}$ noise, where β is the spectral exponent of the noise. Considering D_f as the fractal dimension and D_E as the Euclidean dimension, $1/f^{\beta}$ noise follows the following rule:

$$D_f = D_E + 1 - H = D_E + \frac{3-\beta}{2}$$

2.2 Terrain Modelling

In this section the different methods used for terrain generation are analysed. All the analysed methods try to approximate fBms and they are divided in six categories:

- Poisson Faulting
- Subdivision Methods
- Fourier Filtering
- Successive Random Additions
- Noise synthesis
- Generalized Stochastic Subdivision

2.2.1 Poisson Faulting

The Poisson Faulting technique, also known as Random cuts algorithm, consists in applying Gaussian random displacements to a plane at Poisson distributed intervals. [5]. This method was initially applied by Mandelbrot in [1]. Although this method has the advantage of working for both planes and spheres, in the creation of random planets, it is $O(n^3)$ complex in time.

2.2.2 Subdivision Methods

The methods presented in this section derive from the midpoint displacement algorithm that consists in successively subdividing a line and displacing the division points. These methods are classified in two categories [6]:

- Wire-frame Midpoint Displacement
- Tile Midpoint Displacement

2.2.2.1 Wire-frame Midpoint Displacement

In this type of methods the surface is thought of as a wire-frame mesh and the displacements are applied to the midpoints of the edges. This is the case of the Carpenter's Method [7] in which the wire-frame forms triangles which are recursively subdivided until there is no triangle with a side bigger than a specified length. Wire-frame methods are context independent as the only inputs that impact the shape of the generated surface come from the altitude values at the vertices. This context independence is the opposite of what happens with fBm, which incorporate an infinite span of context dependence [6].

2.2.2.2 Tile Midpoint Displacement

In tile midpoint displacement methods the surface is seen as a collection of tiles and the displacements are applied to points in the middle of every tile. This methods are context dependent.

2.2.2.2.1 Triangular tile midpoint displacement In [6] Mandelbrot proposes a method of tile displacement with triangles. In contrast with the Carpenter's method (section 2.2.2.1), the displacement is applied to the midpoint of the triangles using:

$$H(P) = \frac{H(A) + H(B) + H(C)}{3} + rand$$

where A, B and C are the triangle vertices, P is the triangle midpoint and $rand$ is the random displacement value.

2.2.2.2.2 Diamond-Square Algorithm The Diamond-Square algorithm [7] consists in the subdivision of quadrilaterals in two steps: in the first step, known as diamond, the midpoint of the square is displaced using a random value; in the second step, known as square, the midpoint of the original square sides are interpolated from the value of the two square vertices and the two closest diamond vertices and displaced by another random value. In practice this is a hybrid be-

tween wire-frame and tile displacement method as when the initial structure is composed of squares it is not enough to just displace the tiles midpoints but also interpolate the edges midpoints.

2.2.2.2.3 Square-square subdivision In [9] Miller proposes a method adapted from the CAD/CAM field. This method, denominated Square-Square subdivision, generates new points that form a square which is half the size of the existing one using the proportions 9:3:3:1, in which the nearest points have the greater weight.

2.2.2.2.4 Hexagonal tile midpoint displacement In [6] Mandelbrot also proposes the use of hexagonal tiles in the displacement method. This was due to his belief that the nesting properties of the displacement methods, that is when the generated points are nested in the old structure, was the cause of the creasing effect. Given the hexagon's properties, a structure like this will never nest but, instead, will create a crumpled boundary that fails to catch the eye, contrary to the "creases" that stand out.

2.2.3 Fourier Filtering

Another method to generate fBm surfaces is using the Inverse Fourier Transform. This is done by generating a two dimensional Gaussian white noise signal, applying a $1/f^{\beta}$ low-pass filter in the frequency domain, and using the result of the Inverse Fourier Transform of the filtered signal as a height field [5].

2.2.4 Successive Random Additions

The Successive Random Additions algorithm [10] builds on top of the midpoint displacement algorithm. If old points are reused in subsequent phases of subdivisions, they are displaced with a random variable with an appropriate distribution [5]. In terms of complexity this method is comprised of approximately twice the number of additions of the midpoint displacement algorithm [11].

2.2.5 Noise synthesis

Noise synthesis consists in the addition of successive frequencies of tightly band-limited noises [5]. This can be done using several noise algorithms, such as, Perlin Noise [12], Simplex Nosie [13] or OpenSimplex Noise [14].

2.2.6 Generalized Stochastic Subdivision

All the methods previously presented have a basis function that is, usually, implicit in the generation algorithm. This basis function can affect the final surface in different ways: the saw-tooth wave in polygon subdivision methods explains the creasing effect and the sine wave in Fourier synthesis causes the terrains to become periodic. In [8] Lewis proposes a method that interpolates several local points based on a autocorrelation function [3], that is, this algorithm is able to generate surfaces with any basis function.

2.3 Terrain Representation

2.3.1 Height field

The height field, in some literature denominated as height map, is one of the most used forms of representing terrains in Computer Graphics. A height field stores an altitude value at regular intervals using a two-dimensional array [4], and, as such, can be saved as a grayscale image. Due to the fact that only one altitude value is retained for a pair of coordinates this method can only represent surfaces, that is, it does not support overhangs or caves.

3 METHODOLOGY

This chapter presents a detailed explanation of the proposed process.

- Detailed proposed process
- Control parameters available to the user

3.1 Process Overview

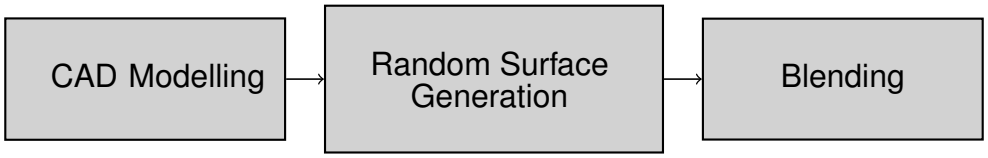

Figure 3-1: Process Phases

The proposed workflow consists of three high level phases which are presented in Figure 3-1. The first is the CAD modelling phase (section 3.2) which consists on the creation of a simplified model of the terrain, which will be denominated as base surface, containing only the main characteristics, such as, mountains and valleys. The second phase (section 3.3) consists in the generation of the random surface from which the details will be acquired. Finally the third and final phase is the blending phase (section 3.4) in which the base surface is combined with the random surface to create a more detailed version of the terrain.

3.2 Phase 1: CAD Modelling

The purpose of the CAD Modelling phase is to enabled the user to specify the main features of the terrain, such as mountains and valleys, using manual modelling. This phase results in the generation of a height map that contains the base surface.

3.3 Phase 2: Random Surface Generation

With the base surface obtained from the first phase it is now possible to generate a random surface with the same dimensions. This can be done using several

methods, as discussed in Chapter 2. For the purpose of this application the chosen methods were Fourier Filtering (section 3.3.1) and Noise Synthesis (section 3.3.2).

3.3.1 Fourier Filtering

The fourier filtering method, shown in Figure 3-2, consists in applying a $f^{-\beta}$ filter to a white noise surface in frequency domain.

This method only needs one parameter, the filter power (β).

Using a fork and join approach it is possible to parallelize this method as there is a parallel version of each of it's steps.

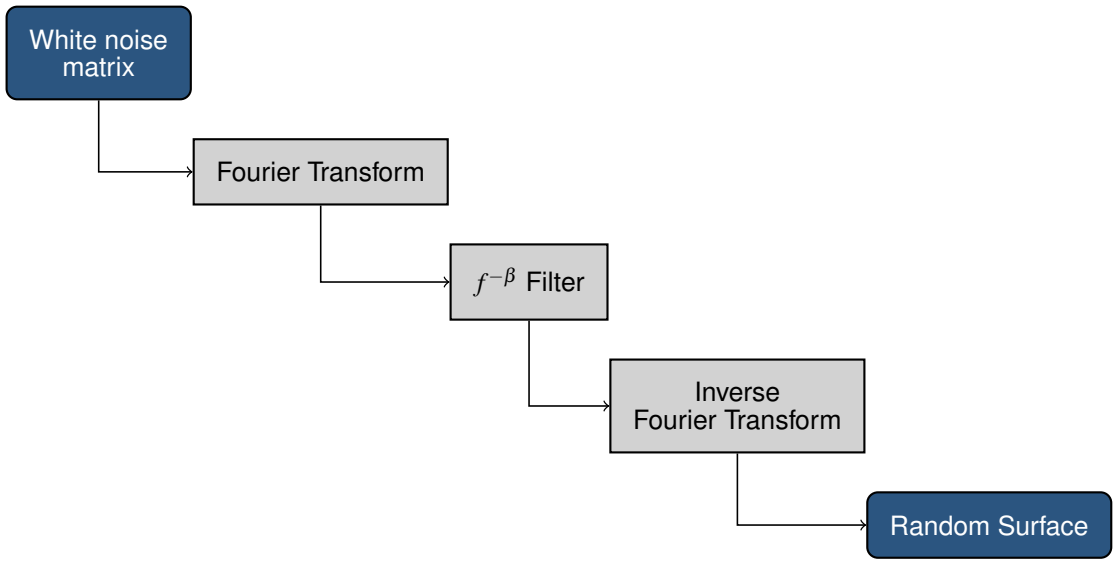

Figure 3-2: Fourier Filtering

3.3.2 Noise Synthesis

$$\frac{\sum_{i=0}^{octaves-1} noise(x \times lacunarity^i + base, y \times lacunarity^i + base) \times persistence^i}{\sum_{i=0}^{octaves-1} persistence^i} \quad (3\text{-}1)$$

The noise synthesis method consists on using a weighted average of several random values to approximate an fBm surface, as described in equation 3-1. This assumes that the random number generation is a structured noise function.

The implementation for this method is based on the code presented in [3]. There are several parameters used in the method, namely:

- **Number of octaves:** number of noise samples to use
- **Persistence:** contribution gap between successive octaves
- **Lacunarity:** gap between successive frequencies
- **Base:** base value for noise arguments
- **Noise:** noise function used to generate random numbers (eg: perlin noise [12], simplex noise [13])

3.4 Phase 3: Blending

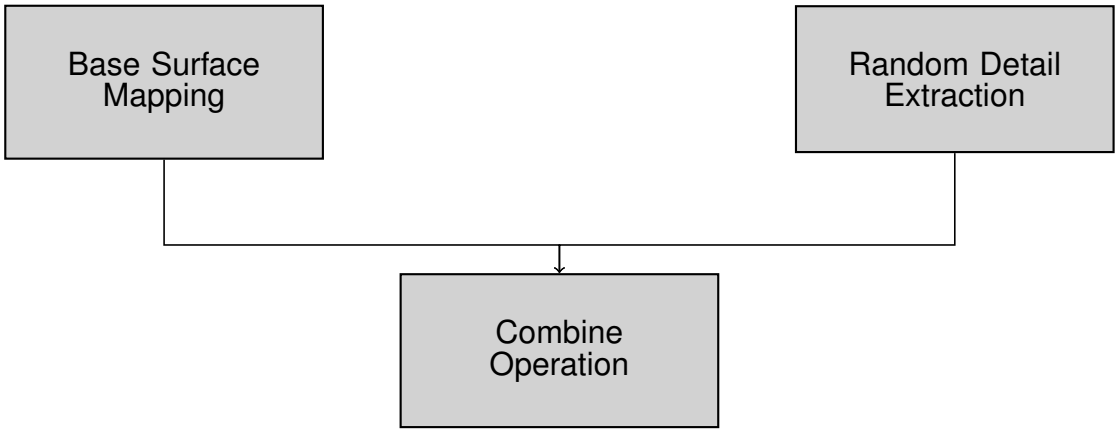

Figure 3-3: Blending Phase

The Blending phase is divided in three parts that are shown in Figure 3-3.

3.4.1 Base Surface Mapping

In the Base Surface Mapping part the user has the possibility of adjusting the details that will be added to the base surface. This is done by manipulating a spline which represents a function that maps a height value to a multiplier. This mapping is then applied to the base surface generating a multiplier matrix.

The spline function is obtained using a Picewise Monotone Hermite Cubic Interpolation of the control points specified by the user.

3.4.2 Random Detail Extraction

In order to extract the details from the random surface, a high-pass filter is used. In the case of this application the HPF is implemented by subtracting a smoothed version of the surface to the surface itself. The smoothed version is obtained by applying a Gaussian Blur filter in frequency domain, using, as cut-off frequency, a user-specified value called Blend Strength.

3.4.3 Combine Operation

Using the outputs from the Base Surface Mapping and the Random Detail Extraction parts, the detailed surface can be computed. This is obtained by multiplying the mapped base surface by the random details resulting in an adjusted version of the details. The latter is then added to the Base surface.

3.4.4 Result Normalization

The final step in the blending phase is to normalize the result. This is done using user-defined bounds that need to be comprised between 0 and 255 in order for the surface to be stored in a single channel image using, for each pixel, an 8 bit unsigned integer.

3.4.5 Surface Normals Calculation

In addition to the previously described steps, a normal map is also computed for rendering purposes. This is done by filtering the final surface with a 3 by 3 Sobel filter in both x and y directions.

3.5 Parameters Summary

Phase		Name	Description	Minimum	Maximum
Random Surface Generation	**Fourier Filtering**	Filter Power	β value	1.0	2.9
	Noise Synthesis	Octaves	Number of noise samples		
		Lacunarity	Gap between successive frequencies		
		Persistence	Contribution gap between successive octaves		
		Base	Base frequency value		
Blending		Spline Control Points			
		Blend Strength			
Result Normalization		Result Bounds			

Table 3.1: Process Parameters

4 SOFTWARE ARCHITECTURE

4.1 Use Case View

The developed application enables the user to create a detailed terrain from a deterministic base surface. To achieve this goal several features, that aid the user in this process, were implemented. This features are specified in Table 4.1 in the form of user stories. Given the nature of this application only one actor, called User, is needed.

Code	Name	Description
US-001	Load Base Surface	As a User I want to load a surface from an image so that I can create a detailed version of it.
US-002	Export Result	As a User I want to export the result so that I can use it in other applications.
US-003	Add Detail	As a User I want to add detail to a previously loaded base surface.
US-004	See Result	As a User I want to see my result as I change the parameters so that I can adjust them more easily.
US-005	History	As a User I want to be able to consult a list of previously edited terrains so that I can track my work.
US-006	History Parameters	As a User I want to have access to the parameters used in previously edited terrains.

Table 4.1: User Stories

4.1.1 User Interface

The application was developed using Google's material design [17] for the user interface style. The editor is the main page of the application and it is shown in Figure 4-1. It contains a menu bar (1) where the user has access to the available functions, a details panel (2), which has the parameter controllers, a list with the previously results (3), a panel where the details of the selected previous result are displayed (4) and a canvas where the preview of the current current terrain is

rendered (5).

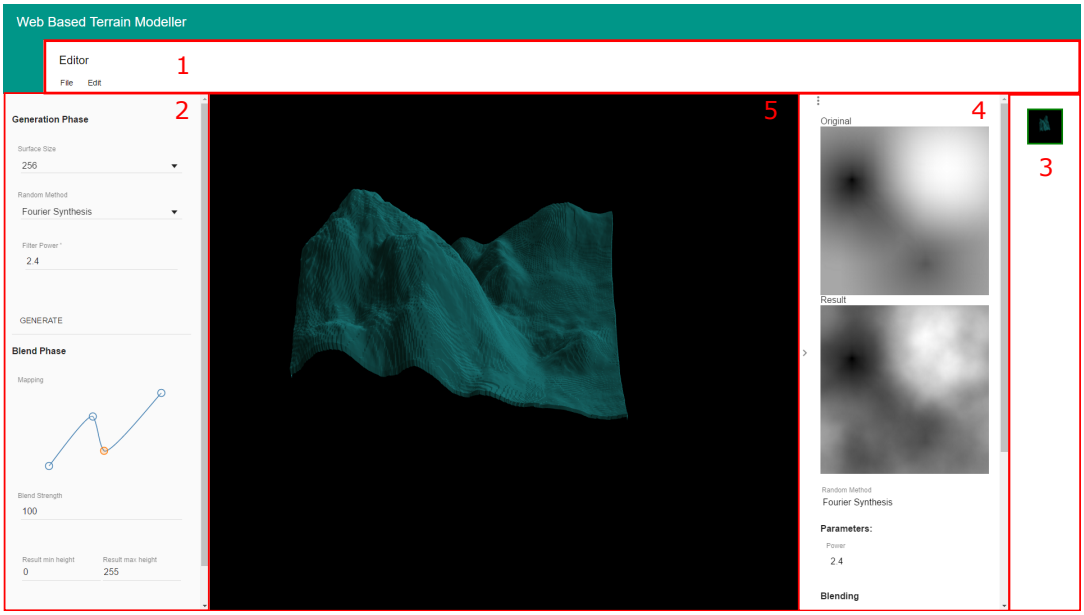

Figure 4-1: Editor Page

- Explain each panel in detail.

4.1.2 Input and Output Formats

In order to allow the user to load and save his terrains the application implements an import and export feature. The user can either import a Grayscale PNG image, which will be set as the base surface, or import a previously exported Zip file, which will add an entry to the history list and render the result contained in the file. In terms of export formats the user can: obtain: a 4-channel 8-bit per channel PNG containing the height map of the result; a Zip file that can be later imported; and a Grayscale 1-channel 16-bit per channel PNG image of the result height map, which can be import as a landscape in Unreal Engine 4.

4.2 Physical Architecture

The system was developed as a client-side single page web application and thus all the server-side content is static. Given this properties the application works as a static web site and has a simple deployment architecture which consists on a HTTP web server and a web browser, as shown in Figure 4-2.

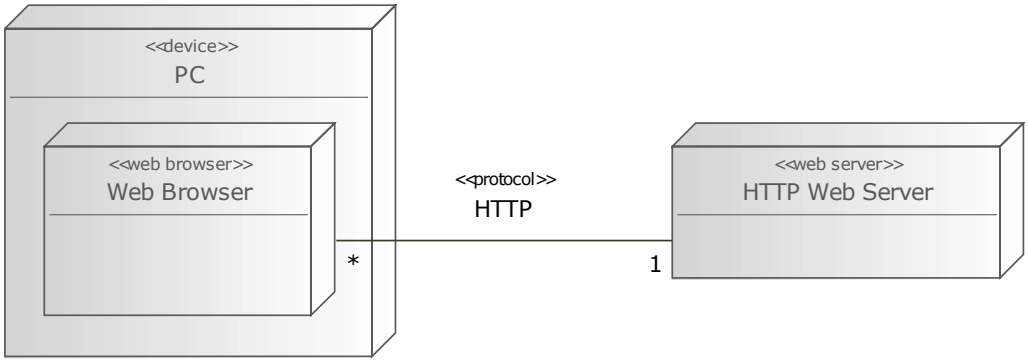

Figure 4-2: Deployment Diagram

4.3 Implementation Architecture

The application is composed by three submodules (Figure 4-3). The *components* submodule is responsible for configuring the page routing and, as such, contains, as submodules, the editor and the benchmarks pages. The *common* module contains services and directives that are generic to the application. The *imgproc* module contains the image processing utilities.

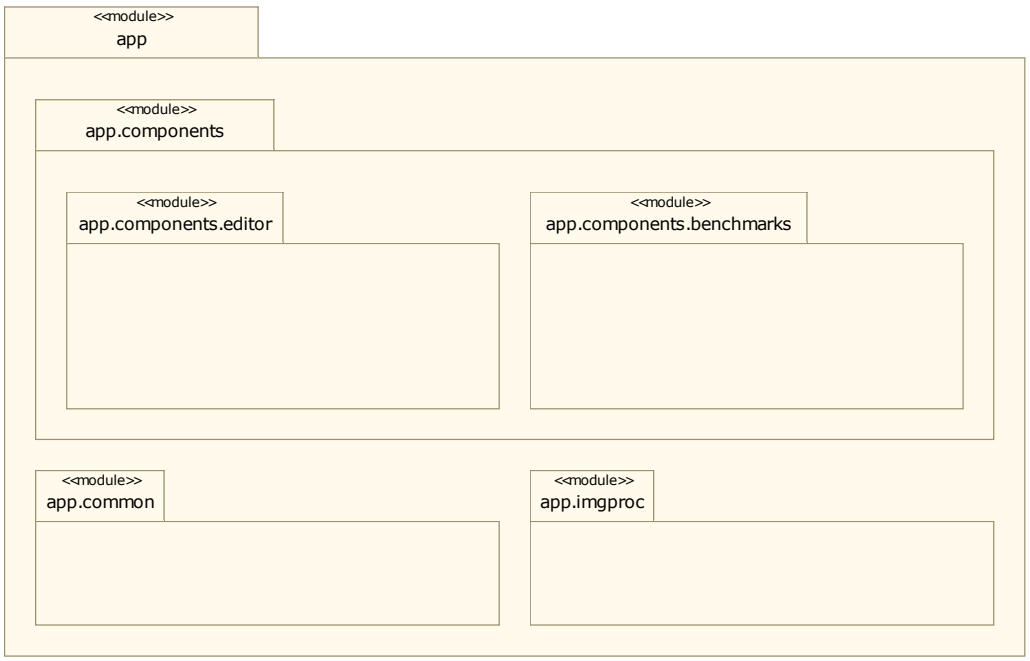

Figure 4-3: Component Diagram

4.3.1 Technologies

Given the web-based requirement the application needed to be in Javascript. For convenience the Babel transpiler was used so that the development could be done in ECMAScript 6, which is a newer standard of Javascript. Additionally, to simplify the management of the required dependencies, JSPM and System.js were used, which, integrating with Babel, enable better support for ECMAScript 6 modules.

To aid the development of the application, Angular.js was used. This framework implements the MVC pattern, which was used to better modularize the source code.

The terrain viewer was implemented using three.js, which is a 3D rendering Javascript library that allows the developer to create 3D Scenes in a straight forward way.

In order to parallelize some operations, WebGL 2.0 was used. This implementation will be detailed in section 4.5.1

4.4 Logical View

- Angular.JS oriented design
- Routes (Directives)
- Services
- Views

4.5 Process View

4.5.1 GPU Computation Framework

- Small introduction to WebGL pipeline
- Generic algorithm execution (Framebuffers & Textures)
- Matrix Type System

- Implemented Features
  - FFT & IFFT
  - Element-wise Operations
  - Normalization
  - Frequency domain filters
  - Spatial domain filters
- Radix-2 algorithm for FFT
- Reduction based algorithm for finding minimum and maximum

4.5.2 Blending Process Pipeline

- Pseudo-reactive design
- Process diagram

5 RESULTS

5.1 Visual Presentation

- Show some comparison between base surfaces and results with different parameters
- Show some unreal engine 4 renderings of the results

5.2 GPU vs CPU Computations Benchmarks

- Hardware Specification
- Software Specification (including chrome version and flags)
- Operations/second or time comparison
- Ratios

6 CONCLUSIONS

- Implemented Features
- Future work
  - Viewer
  - Additional Pages (Help & About)
- Emphasise Web-based implementation

REFERENCES

1. Mandelbrot BB. The Fractal Geometry of Nature. 1983.

2. Mandelbrot BB, Ness JV. Fractional Brownian motions, fractional noises and applications. SIAM review. 1968; Available at: http://epubs.siam.org/doi/abs/10.1137/1010093

3. Musgrave FK. Methods for Realistic Landscape Imaging. Thesis. Yale University; 1993. pp. 1–276. Available at: http://www.kenmusgrave.com/dissertation.pdf

4. Ebert DS, Musgrave FK, Peachey D, Perlin K, Worley S, Mark WR, et al. Texturing and Modeling. Texturing and Modeling. Elsevier; 2003. 9 p. Available at: DOI:10.1016/B978-155860848-1/50050-4

5. Musgrave FK, Kolb CE, Mace RS. The synthesis and rendering of eroded fractal terrains. ACM SIGGRAPH Computer Graphics. New York, New York, USA: ACM Press; 1989; 23(3): 41–50. Available at: DOI:10.1145/74334.74337

6. Mandelbrot BB. Fractal landscapes without creases and with rivers. The science of fractal images. 1988. pp. 243–260. Available at: http://dl.acm.org/citation.cfm?id=61160 http://portal.acm.org/citation.cfm?id=61160

7. Fournier A, Fussell D, Carpenter L. Computer rendering of stochastic models. Communications of the ACM. ACM; June 1982; 25(6): 371–384. Available at: DOI:10.1145/358523.358553

8. Lewis JP. Generalized stochastic subdivision. ACM Transactions on Graphics. 1987; 6(3): 167–190. Available at: DOI:10.1145/35068.35069

9. Miller GSP. The definition and rendering of terrain maps. ACM SIGGRAPH Computer Graphics. 1986; 20(4): 39–48. Available at: DOI:10.1145/15886.15890

10. Saupe D. Algorithms for random fractals. The Science of Fractal Images. New York, NY: Springer New York; 1988. pp. 71–136. Available at: DOI:10.1007/978-1-4612-3784-6_2

11. Voss RF. Random Fractal Forgeries. Fundamental Algorithms for Computer Graphics. Berlin, Heidelberg: Springer Berlin Heidelberg; 1985. pp. 805–835. Available at: DOI:10.1007/978-3-642-84574-1_34

12. Perlin K. An image synthesizer. ACM SIGGRAPH Computer Graphics. 1985; 19(3): 287–296. Available at: DOI:10.1145/325165.325247

13. Perlin K. Improving noise. ACM Transactions on Graphics. New York, New York, USA: ACM Press; 2002; 21(3): 2–3. Available at: DOI:10.1145/566654.566636

14. Spencer K. OpenSimplexNoise.java. 2015. Available at: https://gist.github.com/KdotJPG/b1270127455a94ac5d19

15. Musgrave FK. 2 Procedural Fractal Terrains. Texturing and Modelling. A Procedural approach. 1994; 2.

16. Evertsz CJG, Mandelbrot BB. Multifractal measures. Chaos and Fractals. 1992. pp. 921–953. Available at: http://lipenreferences.google-code.com/svn/trunk/Papers/Multifractals/1992Multifractal Measures.pdf

17. Google. Material design. Google design guidelines. 2016. Available at: https://material.google.com

APPENDICES

Whilst Heading 1 to Heading 6 can be used to number headings in the main body of the thesis, Heading styles 79 have been modified specifically for lettered appendix headings with Heading 7 having the Appendix prefix as shown below.

Appendix A User Manual

A.1 User Section 1

A.1.1 User Subsection 1

Something

A.2 User Section 2

Appendix B Appendix 2

B.1 Appendix Section 2