|  |
| --- |
| 2022 - 2023 |
| Graphics Programming: Mandelbrot Set |
| S2026838 |

|  |
| --- |
| Reid, Ben |

A close-up of a signature

Description automatically generated

*I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award.*

Contents

[1: INTRODUCTION 2](#_Toc135409654)

[1.1: BACKGROUND 2](#_Toc135409655)

[2: IMPLEMENTATION 5](#_Toc135409656)

[2.1: VERTEX SHADER 5](#_Toc135409657)

[2.2: FRAGMENT SHADER 6](#_Toc135409658)

[2.21: VARIABLES 6](#_Toc135409659)

[2.22: MAIN METHOD 7](#_Toc135409660)

[2.23: MANDELBROT FUNCTION 7](#_Toc135409661)

[2.24: APPLYING COLOUR 9](#_Toc135409662)

[2.25: FINAL RESULT 10](#_Toc135409663)

[3: NOTES 13](#_Toc135409664)

[3.1: FLOATING POINT PROBLEM 13](#_Toc135409665)

[3.2 Mandelbrot B 14](#_Toc135409666)

# 1: INTRODUCTION

This document covers an OpenGL implementation of The Mandelbrot set. A quick overview of the Mandelbrot set is provided for background that enhance the understanding of the concepts seen within the fractal to help with the final implementation process.

## 1.1: BACKGROUND

The Mandelbrot set is defined as a set of complex numbers (c) for which the corresponding orbit of 0 under does not escape to infinity. Fig 1 shows the visualization of this set.

A picture containing flower, symmetry, art, kaleidoscope

Description automatically generated

Fig 1: Visualization of the Mandelbrot Set

This is a fractal, it’s a complex self-similar shape on all scales that essentially repeats forever, as you continue zooming in more, smaller details will be revealed.

This set is generated through iterating the function below.

The Mandelbrot set is generated by iterating complex numbers. Starting with z = 0. The variable z acts as an iterator as it represents the current generation of the function. The current modulus of the current iteration of z is checked to see if the distance from origin is below a set value defined as 2, if it is less than this value the c is determined to be within the Mandelbrot set as otherwise the value will tend to infinity. To show this an example set of iterations below with being used. Remember that

This set of iterations has a modulus of 3 at z4. As defined before that if the modulus of z is greater than the set value 2, c would not be considered to be in the Mandelbrot set thus the equation will tend to infinity. Therefore would not be in the Mandelbrot set.

Values that would be acceptable and considered part of the Mandelbrot set would be something that bounces back and forth meaning the orbit won’t keep getting bigger leading towards infinity. C = -1 is such a number where the orbit would bounce between the values of 0, -1, 0, -1, 0, proving that c = -1 is part of the set.

These complex (c) numbers are typically represented as a plot on a 2D plane XY axis graph (Fig 2), that will act as our canvas. The X axis represents real numbers while the Y axis represents imaginary numbers. C represents a + bi, a represents a real number (x axis) while bi is the imaginary number (y axis). Below is a graph showing the point (3,5), a complex number 3 + 5i. Any point can be sampled like this to give a unique complex number, by using the Mandelbrot function each point will be coloured depending on if that number is in the set.

A picture containing diagram, line

Description automatically generated

Fig 2: Representation of a Complex Number

By now implementing into the function using c = a + bi the iterations will be as follows.

Here the imaginary number can be simplified to be that allows the overall function to be simplified, resulting in an easier implementation as it reduces the complexity of the imaginary number present with the below equation intended to be implemented within the code to represent the Mandelbrot set within a 2d canvas.

Finally, to prevent this equation from going out of scope by becoming unnecessarily detailed, due to the Mandelbrot set range not existing infinitely, essentially being bound to roughly -2 on the x axis to around ¼ on the y axis. Going forward for implementation a max amount of iterations along with checking if the modulus of c is greater than 2 is reached the current equation will stop and return either 0 if it has reached the max amount of iterations (in the set), or simply the current iteration value upon reaching greater than 2.

# 2: IMPLEMENTATION

The Mandelbrot set will be displayed on a 2D plane using the 2 shader files Mandelbrot.vert and Mandelbrot.frag.

## 2.1: VERTEX SHADER

The vertex shader is setup to map the texture coordinates and the position of the object our shader will be applied to. This is simply allowing the given object to appear and be seen in the game world.

A screen shot of a computer program

Description automatically generated with low confidence

Fig 3: Vertex Shader

## 2.2: FRAGMENT SHADER

The fragment shader contains the logic that will result in the Mandelbrot Set being represented on the object it is applied to. For this example, a 2D plane is used to showcase the Mandelbrot set across a wide area.

### 2.21: VARIABLES



Fig 4: High Precision Float

This makes the precision value that this shader provides a 16-bit floating point. This is to allow for more detail to be presented when zooming into the fractal however the final output will still hit a limit with floating point precision if an infinite zoom / exploration is desired. An explanation of what precision does to this shader can be found later under the [Notes](#_Floating_Point_Problem) section as it can only be seen fully after the following implementation is carried out.



Fig 5: Output Variables

A picture containing text, font, screenshot, graphics

Description automatically generated

Fig 6: Variables Input from Vertex

The vertex will pass over the texture coordinates to be used for mapping onto the canvas, this is seen in the [Main Method](#_Main_Method).



Fig 7: Maximum Iterations

This is the max amount of iterations that will be done using the function. By controlling this the final results detail will be affected, the higher the value the more detail will be present.



Fig 8: Uniform Variables

This uniform variable passes in a time value where every frame adds 0.01f , This is used to progress the shader through a zoom effect to scale up and down the fractal.

For this example, Deltatime is not used to check the value is going up consistently, due to an external error that won’t affect this shader and any other demonstrations shown.

### 2.22: MAIN METHOD

A screen shot of a computer code

Description automatically generated with low confidence

Fig 9: Main Method

The canvas is gained by mapping our current texture coordinates on both x/y. This is modified by multiplying by 2 and displacing by (1,1).

Col is declared as (0,0,0) and will be added to using a hash function after the Mandelbrot has been generated.

A Mandelbrot result is stored by running the [Mandelbrot Function](#_2.23:_MANDELBROT_FUNCTION) while passing in the canvas.

Col is added to using the Mandelbrot result passed in to generate pseudo random numbers through the use of a [Hash Function](#_2.24:_APPLYING_COLOUR)

Once the colour has been generated the frag colour is set to it with a set hard coded alpha of 1.

### 2.23: MANDELBROT FUNCTION

C is defined as our canvas of coordinates, containing x and y of the current pixel the shader is calculating, with x and y also known as real and imaginary numbers. Here we multiply this value to change the amount of area that is taken up. Then by subtracting a vec2 the fractal origin in this shader can be displaced to be centred where preferred.

A picture containing text, font, screenshot, line

Description automatically generated

Fig 10: Defining The Value C

To make the fractal scale up to provide a zoom effect, C is divided by the time that has passed since the application started, Similarly another displacement vector is applied to control the movement. Although the power is set to 1 for time, for demo and experimentation purposes it makes it easy for adjustments to the speed of the effect to be made. Z is created and is set to 0 for the first iteration to be run through the equation.



Fig 11: Movement Effect, z Initialized At 0.

Below (Fig: 12) is the main Mandelbrot equation, here we get Z by multiplying out what is and adding c at the end and continuing for the max amount of iterations set before runtime. The equation used is the same as presented at the end of the introduction.

If the dot product of z is greater than 2 the value is determined to not be within the Mandelbrot set and will return the current n divided by the max iterations, used for colouring the current pixel. The reason the dot() is used is to get a single float value from the vec2 value of z. By returning 0 after reaching the max iterations, the colour will be black to represent the Mandelbrot itself.

A screenshot of a computer

Description automatically generated with medium confidence

Fig 12: Mandelbrot Equation Loop

### 2.24: APPLYING COLOUR

To generate the ‘wild’ colour pattern that creates a variation of colour a pseudo random method is used through the use of a hash function. This function takes in the float value m to generate a set of 3 numbers to generate a colour pattern.

The fract() function is used to get the value for the colour pattern by returning the fractional part of the number, To confine this to 0-1 for a colour value to be represented, the fractional part of sin is gained as shown in Fig 13.

The number that is multiplied after is a constant based on what is seen in similar rand functions and can be changed for varying results in colour / pattern.

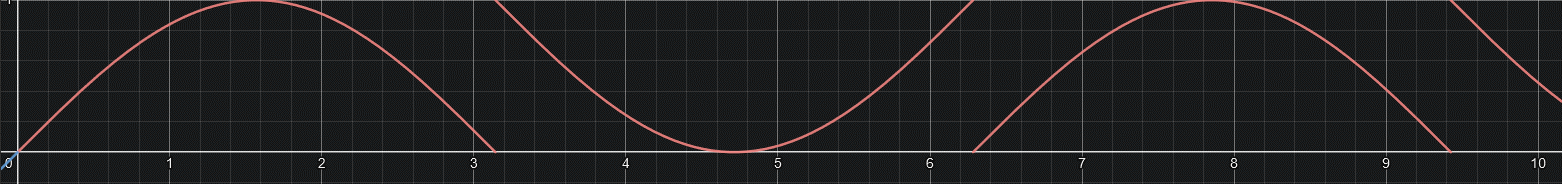


Fig 13: Visual of the fractional part of sin (0-1 values).

This value is a vec3 to store the R, G, B values that will be used. The alpha value is set in the final frag colour in the [Main Method](#_2.22:_MAIN_METHOD).

A picture containing text, screenshot, font

Description automatically generated

Fig 14: Hash Function, Generating Vec3

### 2.25: FINAL RESULT

Without the use of zoom or time is set to 0, the final result of the shader appears as below. A picture containing circle, graphics, colorfulness, graphic design

Description automatically generated

Fig 15: Final Result

After 10 seconds or so the shader can be expected to look like similar to this while continuing to move.

A picture containing art, colorfulness, reef, fractal art

Description automatically generated

Fig 16: Early Shader Progress (Time)

A picture containing colorfulness, art, lilac, psychedelic art

Description automatically generated

Fig 17: Later Shader Progress (Time)

Texture can be added for background / tint if desired by multiplying the colour by a texture() containing diffuse and the texture coordinates.

A picture containing art, painting, pattern

Description automatically generated

Fig 18: Texture / Use of diffuse.

# 3: NOTES

This section includes any extra explanations of the GLSL shader, relating to the final output results.

## 3.1: FLOATING POINT PROBLEM

There is an issue with GLSL shaders by default, The maximum precision you can get is a 16-bit floating point which requires the following declaration.

precision highp float

This means that upon reaching a certain zoom level the fractal will start to look like the resolution has been lowered resulting in a compressed image although this is not the case. This results in there being a render limit within the GPU requiring a more complex implementation (not present in this project) to allow for unlimited precision that would provide a greater level of zoom.

Below are 2 examples (Fig 19/20) of what the floating-point problem results in the fractal looking like after a certain level of zoom. This results in the Mandelbrot appearing to somewhat fade out despite being able to see the path the fractal still takes through the amount of detail lost.

A picture containing colorfulness, art, graphics

Description automatically generated

Fig 19: Floating Point Limits

A picture containing colorfulness, art, lilac, psychedelic art

Description automatically generated

Fig 20: Further Floating-Point Limits

## 3.2 Mandelbrot B

Another, less efficient attempt at the set is present in the program from implementation. It was left uncompleted however this is partially due to the unique effect present, due to the canvas not being mapped properly this set is squished and compressed however this led to a shape similar to an inkblot being present, so it has been left as is as another perspective of how alterations are interesting to play around with.

A white blot on a black background

Description automatically generated with low confidence

Fig 21: MandelbrotB Shader.