

Fractal Visualisation Application

Contents

Contents	1
Analysis	2
Background to the problem	2
The Problem	4
Solution Requirements	5
Project Plan	7
Necessary Libraries	8
Design	10
High Level Design	10
Additional Modules	12
Class Diagram	14
Program Flow Diagram	17
Interface Block-out	20
Algorithms and Data Structures	25
Technical Solution	29
Testing	119
Evaluation	135
User Feedback	140
Improvements and Further Evaluation	141
Appendices	142
Appendix 1 - Help Document	142

Analysis

In this project, I intend to create a simple but well-featured graphical interface for exploring fractals, a mathematical phenomenon arising from repeated iterations of mathematical equations. The project will therefore consist of two main parts, the backend (responsible for calculating and producing the fractal itself, and usable from a terminal for more advanced users) and the frontend (responsible for displaying and allowing interaction with the user in real-time, graphical and intended to be easy to use).

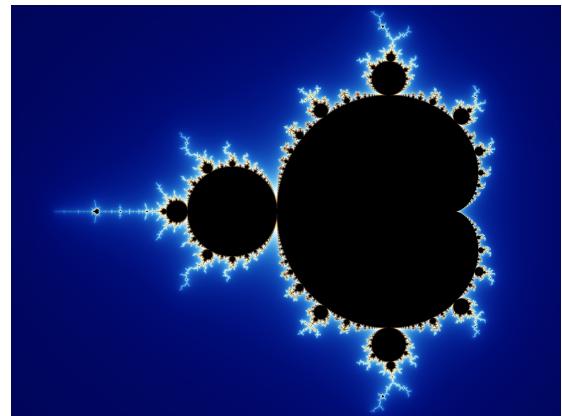
Background to the Problem

Fractals

Fractals are a mathematical phenomenon which arise from repeated iteration of a recursive mathematical equation, the most well-known being the Mandelbrot set. The equation responsible for producing it is $z = (z^2) + c$. A fractal is 'rendered' into an image by taking each pixel, calculating a position for it in the complex number space (the horizontal axis becomes the real coordinate, vertical axis becomes the imaginary coordinate), and assigning that complex number to both variables z and c initially. Then, the equation is iterated repeatedly, with c staying constant between iterations and z being recalculated each time. Iteration continues until z tends to infinity (usually defined as having a modulus greater than 2, after which it is guaranteed to tend to infinity), or until some defined limit is reached, at which point the number of iterations completed is counted or reported. The number of iterations performed then becomes the 'value' assigned to that point in the complex number space, and be used to select a colour for a particular pixel.

Fractals are interesting primarily due to their property of self-similarity: no matter how closely you zoom in or how high a resolution image is rendered (i.e. how fine the arbitrary 'pixels' are) there will always be an infinite depth of detail. The complex high-level shapes, such as the recognisable form of the Mandelbrot set, are repeated (although not necessarily identically) in smaller and smaller instances, with many other features such as spirals, 'cracks', and more. Thus, it is impossible to represent fractals using vector images, due to the fact that they have an infinite level of edge detail, and effectively an infinite circumference therefore.

Their self-similarity and detail properties has led them to be compared to (and indeed modelled against) natural shapes, such as the growth of fern leaves and coastlines (the coastline paradox is an observation that as a coastline is measured more and more precisely, its circumference seems to continue to increase towards infinity, known as having a fractal dimension). Thus, studying mathematics in this way can be very useful in understanding natural phenomena and real-world



Render of the Mandelbrot Set
(created by Wolfgang Beyer, via
Wikipedia)

problems. This also makes fractals a useful demonstration for mathematics students (or anyone in fact) as discussed further below.

Mathematics

Mathematics is often seen as an uninteresting topic or field, at least for anyone who doesn't already have an interest in it. It tends to be defined by equations and numeracy which many people find boring. However, fractals are an example of how mathematical ideas can apply not only to modelling nature and drawing interesting parallels with the real world, but also in producing aesthetically pleasing and beautiful images which everyone can enjoy and appreciate, perhaps even as an art form. Particularly, their use as an illustration of the applicability of mathematics in a classroom can make the subject more engaging, especially to school students.

Thus, this project aims to, as well as providing a useful fractal visualisation tool, be primarily a teaching aid which can give students the opportunity to experiment with fractals and learn about why they occur at the same time, in an engaging fashion.

The Problem

This brings into focus the problem which the project will address, namely the engagement of students or non-mathematical people with maths in general. Currently, tools for visualising fractals are limited, many are either demos with expensive full versions, outdated, slow, or difficult to use:

- Xaos - a free, interactive piece of software, but has a limited functionality and lacks the functionality to input custom equations and display them as fractals
- FRAX - cheap, but only runs on iOS devices and is thus limited in its processing and usability, making it difficult to render with more computationally expensive parameters and very hard to demonstrate to others
- Fractal to Desktop - less fully featured, simpler to use, but only supports Windows (might be inaccessible for some students who do not have access to a Windows-hosting device)
- JWildfire - although very powerful, also quite complex and has a steep learning curve to it
- Ultra Fractal - incredibly powerful and fully featured, but allows only a 30-day free trial before costing £25-80
- Apophysis - supports a limited range of fractals but lacks help/instructions for users

The user of my application is likely to be one of three main categories:

- A teacher looking to engage and excite students and make maths classes more practical (particularly for teaching topics such as complex numbers or iterative/numerical methods)
- A student learning maths interested in exploring mathematical phenomena or putting them to the test
- Ordinary, non-mathematical people looking to experiment in abstract design or maths

However, my primary **client** for the project will be a teacher from the mathematics department at my school. I interviewed them on their thoughts about the problem and basic requirements for a solution.

“

This tool would be very useful as a form of enrichment to the maths syllabus. I might expect to use it towards the end of term, or early on in the school to show students how exciting and beautiful maths can be, especially given the intriguing nature of fractals. This would also be an interesting extension exercise for sixth form students to investigate applications of complex numbers.

A solution would need to be quick to set up and responsive, to keep students engaged. It should also be visually appealing. There should be the ability to see how the images are generated for sixth form students interested in maths and computing.

”

Solution Requirements

Following my research and a brief discussion with my user, the following requirements are apparent. High level objectives are identified with a '1-xx', low level objectives with '2-xx'.

- 1-01: The program must be able to produce a rendered fractal at any resolution requested.
 - 1-02: The program must be able to display the output to the user
 - 1-03: The user must be able to move, navigate around, and zoom in and out of the fractal freely
 - 1-04: The user should be able to change the equation being used
 - a) The user should be able to input their own iterative equation (i.e. in place of `(z^2)+c`)
 - b) The user should be able to select an equation from a list of built-in presets
 - 1-05: The user should be able to save rendered images to a file
 - 1-06: The user should be able to save configured states (i.e. a particular equation with a particular offset, zoom, etc) and restore them
 - 1-07: The program should be able to generate a full screen rendered fractal in under 30 seconds
 - 1-08: The interface must be pleasant to use and intuitive, such that a maths teacher can use it easily without confusion
 - 1-09: The program should offer or come with instructions on how to get started and use each of its features
 - 1-10: The user must be able to change computation and display parameters, such as the iteration limit and colour scheme
 - 1-11: The program must be able to run on both Windows and Mac OS machines
-
- 2-01: The program should present some kind of preview render which has more minimal parameters (lower resolution and iteration limit) to allow the user to navigate quickly (without having to wait for a full resolution render in between every navigation event)
 - 2-02: The user should be able to navigate using both interface buttons and keyboard keys
 - 2-03: The program should be runnable both as a GUI or as a CLI tool for more advanced users
 - 2-04: The program must be able to parse mathematical expressions from a string including involving imaginary parts and evaluate them efficiently at runtime, including the following operations:
 - a) brackets
 - b) implicit multiplication
 - c) addition
 - d) subtraction
 - e) multiplication

- f) division
- g) indexing (powers)

2-05: The program must be able to utilise multithreading in order to speed up computation

2-06: The program must be able to store and load render configuration states in a database structure when the user clicks the relevant button

- a) When storing, the program should present a dialog which allows the user to save the render state
- b) The user should be able to name their render state
- c) The database should be stored in an appropriate location on disk
- d) When loading, the program should present the user with a dialog which allows the user to select a saved render state to load

2-07: The GUI must be able to resize and adapt its render resolution to match that of the screen for optimal user experience

2-08: The program should have an instructions area which is easy to find from the main window.

- a) The instructions should explain how to render an image
- b) The instructions should explain how to save and load render states
- c) The instructions should explain how to input custom equations, and also should describe the presets
- d) The instructions should explain the meaning of the colour scheme, zoom, offset and iteration limit parameters
- e) All of these should be explained in a way which a maths teacher or student can understand

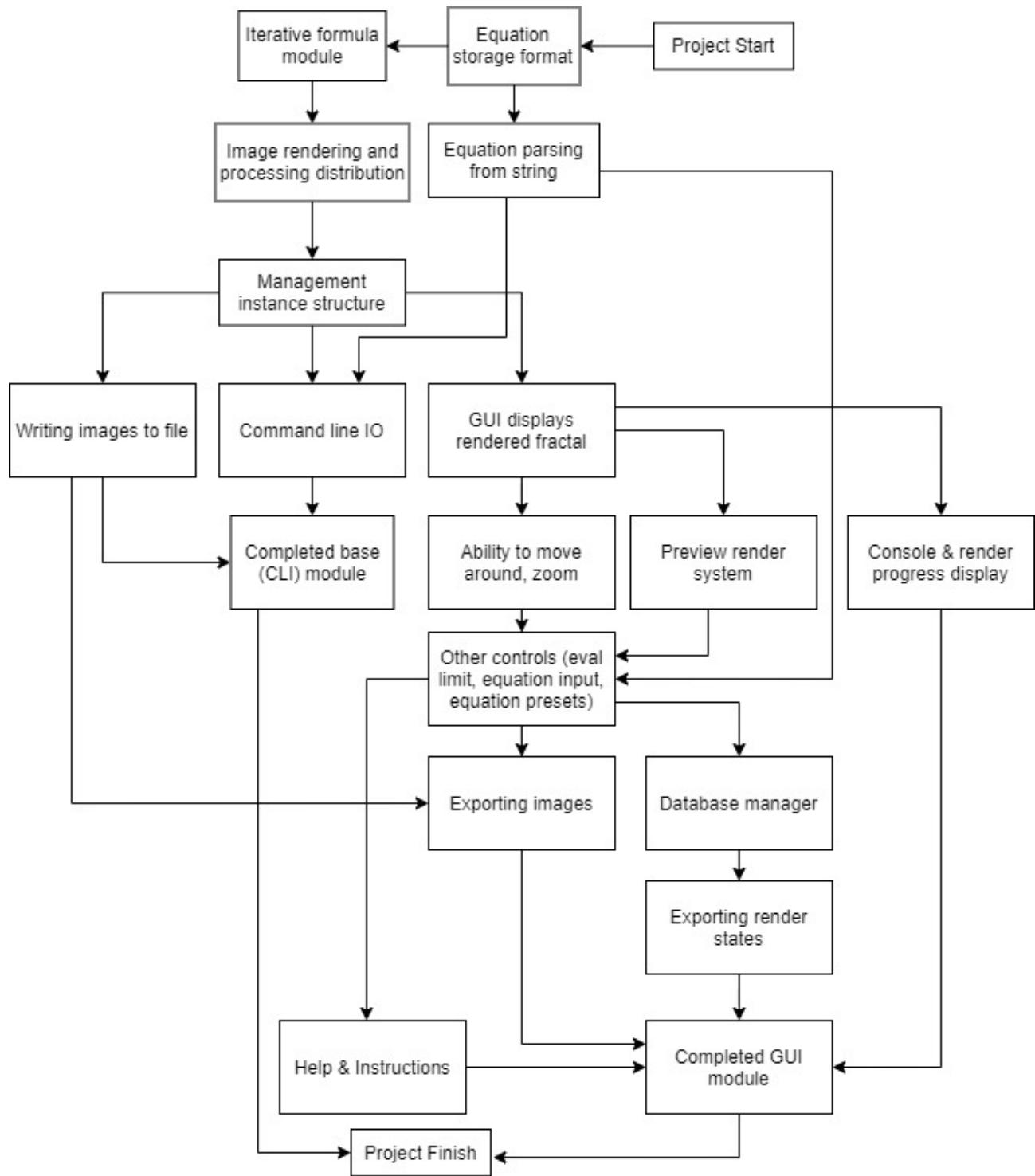
2-09: The program should have a sensible requirement of memory to run (less than 500MiB) at any one time, which should remain consistent even if the program is open for a large period of time (dependent on the resolution being rendered at)

2-10: The program should display its rendering progress (if there is a render currently happening) and its current status, to the user

2-11: The program should lock controls such as iteration limit, offset and zoom whilst a render is occurring

Project Development Plan

Below is a flow diagram showing the order in which I plan to carry out development tasks for this project. The order and requirements for each stage are flexible, as they may change due to further design improvement as I design and develop the project.



Necessary Libraries

The solution to the problem necessitates the use of a graphical user interface. For this, a GUI library is necessary in order to produce a consistent experience across platforms.

A wide range of solutions are available for this. My criteria for a package were as follows:

- free to redistribute
- free to use for commercial purposes
- configurable inside another project
- compatible with the target language, in this case C++
- cross-platform
- providing features such as displaying images, buttons, text

The potential solutions which I identified included Qt, GTK+, wxWidgets, FLTK, BoostUI, and gtkmm. However, many of these did not fit my criteria fully, so I finally chose to use 'raylib', a standalone header-only graphics library which meets all the requirements I set out with.

The C++ standard library fortunately has a built-in complex number handling library. However, during my analysis phase I initially experimented with configuring and using multiple-precision complex number libraries. Multiple-precision arithmetic, or arbitrary precision arithmetic, are methods of performing arithmetic which is not limited by the word size of the host system. The complex numbers involved in generating fractals must be very precise, so I considered using arbitrary precision arithmetic libraries such as:

- MPFR (Multiple Precision Floating-point Reliable) - provides extensible floating point data types which can be sized to any necessary precision
- GMP (GNU Multiple Precision) - a similar library but which determines to provide better performance
- MPC (Multiple Precision Complex) - library which builds off of GMP to provide complex number arithmetic at arbitrary precision
- Boost multiple precision complex - the Boost implementation of the same functionality

However, I encountered a number of problems with all or most of these during my experimentation:

1. They caused a huge performance hit when benchmarked compared with the C++ standard complex number library, on the scale of orders of magnitude slower (which is to be expected with any arbitrary precision arithmetic realistically), which would cause significant problems due to the number of calculations necessary in this project and the intended real-time nature of it.
2. They required the target machines to have libraries installed in order to use the project and application, which also would require configuring host machines (going against the aim of the project to provide a simple, standalone, easy to install and use application)

As a result I decided to stick with the C++ built-in complex number library, the only trade-off being the limited maximum precision of calculations as a result. This has the effect of effectively limiting the resolution or zoom depth possible with the application, as at a certain combination of resolution and zoom, pixels will no longer be able to be uniquely identifiable and their complex coordinates will be rounded to the nearest floating-point number which can be stored. In order to minimise this, in my application I will aim to use the highest precision float possible ('long double' in C++).

Design

High Level Design

As mentioned in the analysis section, I decided to allow both command line and GUI operation of my application. As such, my design can be effectively split into two main parts:

- the fractal generator (rendering environment)
- the GUI

The fractal generator is the core of the application which does the actual processing and creation of the rendered image. It will be accessible from both command line and GUI modes, whereby either mode can create a rendering environment with the necessary parameters (zoom, offset, equation, etc) which can be used for rendering universally.

In CLI mode, this can be done by parsing command line arguments and passing them into the rendering environment as render parameters.

In GUI mode, this can be done instead by simply reading out values from the interface which the user can manipulate (e.g. the user presses a button which zooms in and this command is processed by changing a parameter in the rendering environment).

The GUI is effectively a separate module which provides the graphical interface to the user and simply modifies and calls into the rendering environment provided by the core module and displays the results on the screen, as well as other features such as saving images and accessing the database of render parameters.

The rendering environment will be **encapsulated** in a class which can be instantiated to create an independent, contained environment for generating fractals. It will contain class variables which can be assigned or retrieved via getters and setters. It will also provide functionality to render fractals and to parse textual equations (see below), **hiding all the code** necessary to do this from anything which utilises the rendering environment (e.g. the GUI module).

The GUI module will also be **encapsulated** in a class, which is host to the entire graphical system. It will be responsible for creating and managing the main application window, handling button presses and key events, and will keep track of the rendering environments (two will be used, one for the low resolution preview render, the other for the main, full-resolution render). The GUI class will contain various methods for handling presses for each button, navigation events like moving the viewport, as well as the main-loop function which will manage drawing the interface each frame. The only externally accessible method will be the GUI class's main function, which will perform setup such as initialising the class, creating the GUI and starting the graphics main-loop, from which point the GUI will occupy the main program thread until the user closes the window, when the program will exit.

The ‘int main(int, char**)’ function, which is called when the program starts, will contain code to handle either a console start or a GUI start. For a console start, it will parse the command line arguments, configure a rendering environment, perform the render, save the result to a file, and exit. For a GUI start, instead it will simply construct a GUI class and transfer control to the graphics main method.

Additional Modules

Equation Parser

Other smaller subsystems will include the equation-parsing module, a set of static, **stateless** functions to convert a mathematical expression (e.g. " $(z^2)+c$ ", which represents the Mandelbrot set fractal) stored as a string into a form which can be evaluated rapidly at runtime. This module will define a number of data types to represent tokens which make up the parts of the equation.

More information about the equation storage and parsing is below, where the algorithm used and data structures required are described. In order to hide as much code as possible from other modules, only the outer-most of these functions will be accessible to the rest of the program, which will simply be called to perform the entire parsing and conversion operation and will return an equation object (see below).

Database

Another element of the application is the ability to save details of configuration states (effectively saving the configuration of the rendering environment, along with a name), via a simple **two-table database** stored in CSV format. This will have its own standalone module, with a class which can be used to manage, update, remove, or create records and handle saving and loading them automatically, again **encapsulating** the database system. This will involve two other data types, representing a record from each table. One table will contain configuration profiles, the other will contain the details of users. The configuration profile table will include fields for:

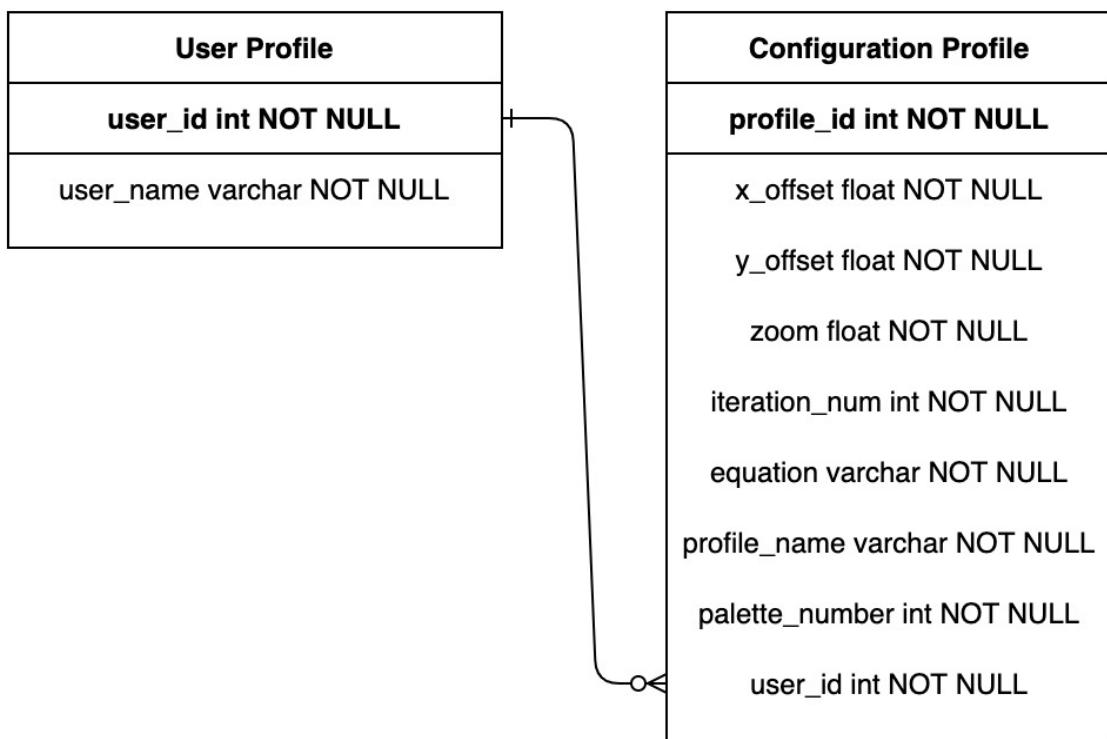
- Record ID (primary key)
- Config Profile Name
- Fractal X Offset
- Fractal Y Offset
- Fractal Zoom
- Fractal Equation
- Fractal Iteration Limit
- Palette Number
- User ID (foreign primary key of user profile table)

Meanwhile, the user profile table will contain fields for:

- User ID (primary key)
- User Name

The relationship between the configuration profile table and the user profile table will be many-to-one, since many configuration profiles can reference a single user profile (but only one user profile can be referenced by a particular configuration profile).

All of this is illustrated by the **entity-relationship** diagram below:



Evaluator

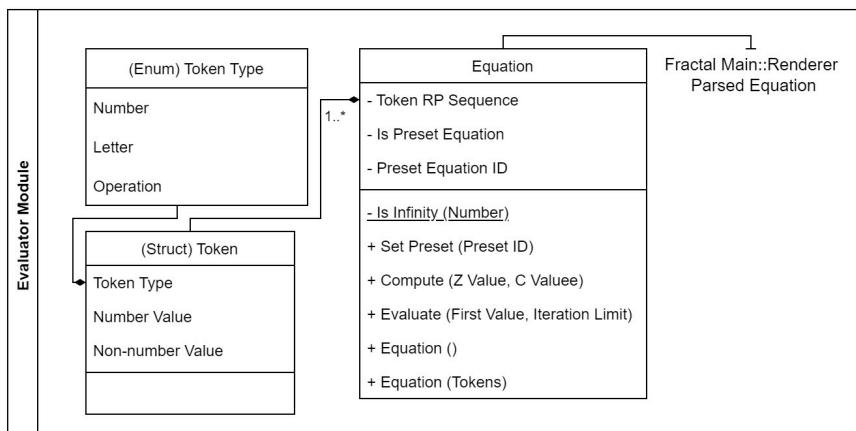
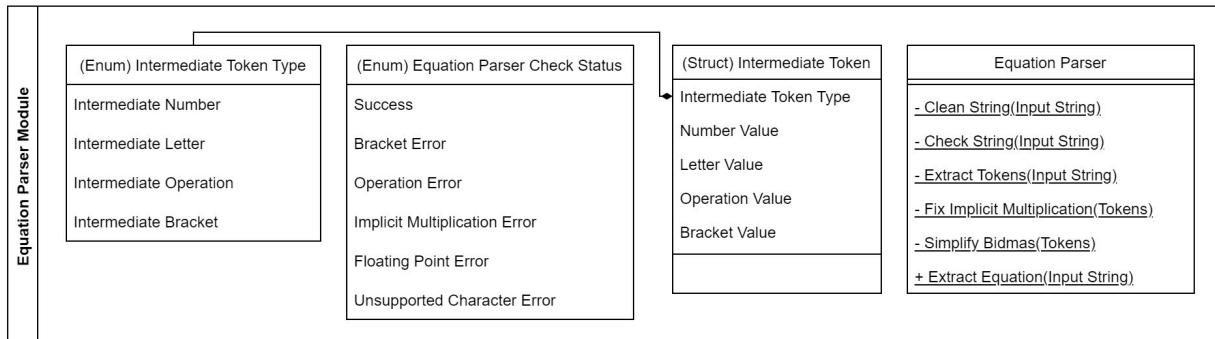
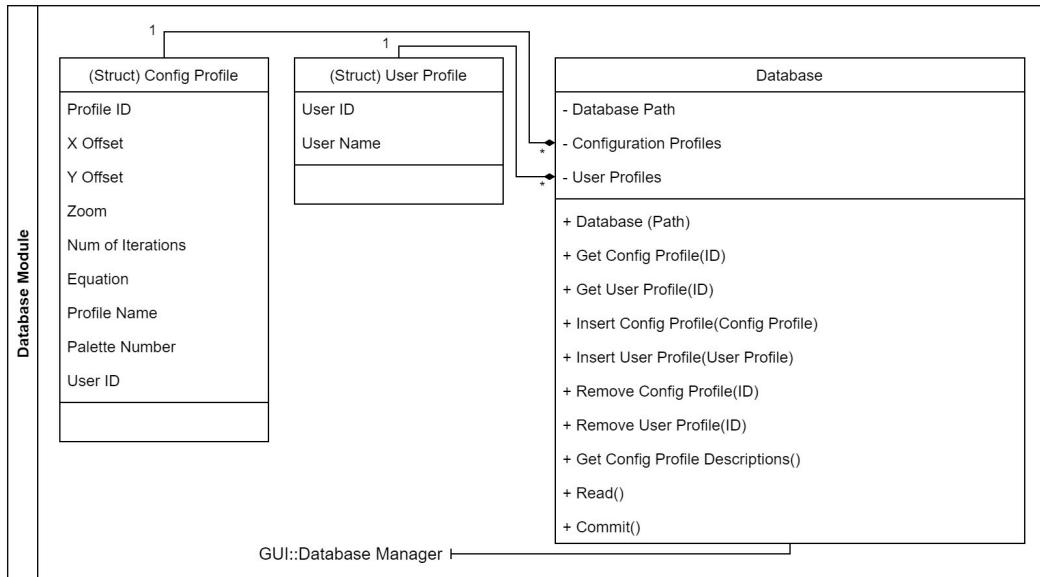
The final module will be the evaluator, which contains the code to handle and evaluate equations parsed by the equation parser. It will **hide the code** to actually evaluate values for any given complex coordinate given, providing functionality to do this which the rendering environment can call for every pixel which needs evaluating. In short, a function will substitute values for variables such as z and c into the stored equation and calculate the result in the form of a complex number. A second function will repeatedly perform this ‘single iteration’ as described in the analysis section (where the calculation is performed iteratively until the z value of the equation tends to infinity or until the specified iteration limit is reached), and an integer representing the number of iterations performed will be returned (which can then be used to colour each pixel).

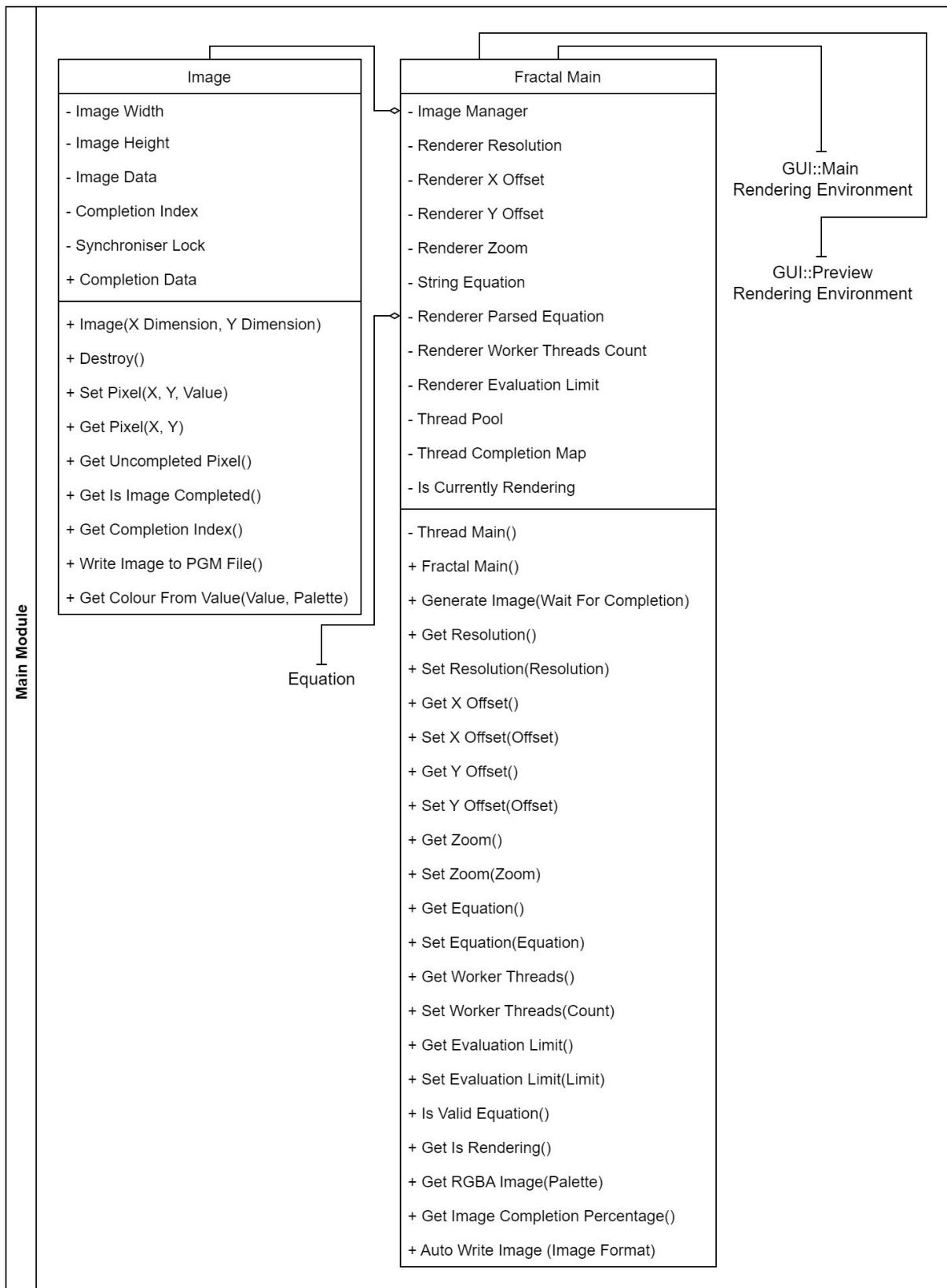
This will also be wrapped in a class to **encapsulate** the evaluation process as tightly as possible.

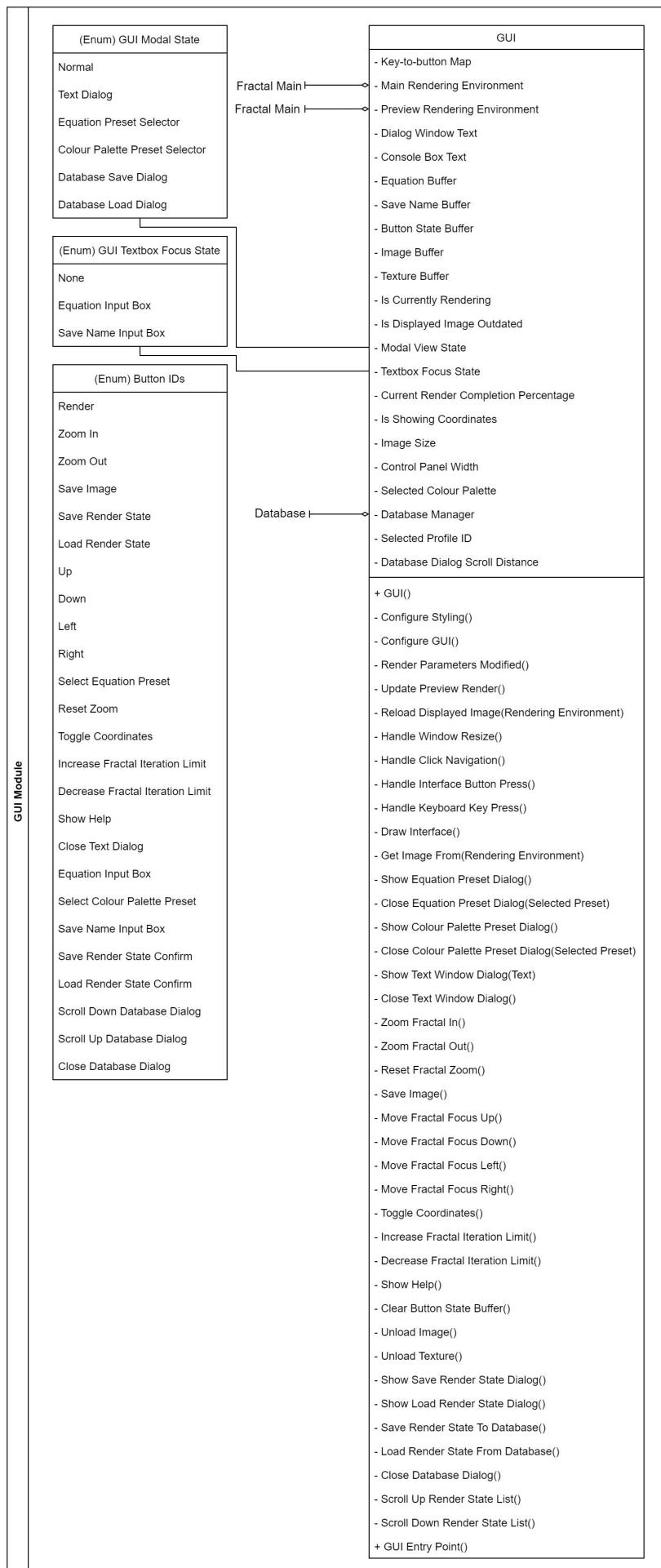
There will also likely be a utilities module which contains miscellaneous functions which multiple other modules might be using, such as a delay/wait procedure, fetching a system path, or retrieving information about an equation preset.

Class Diagram

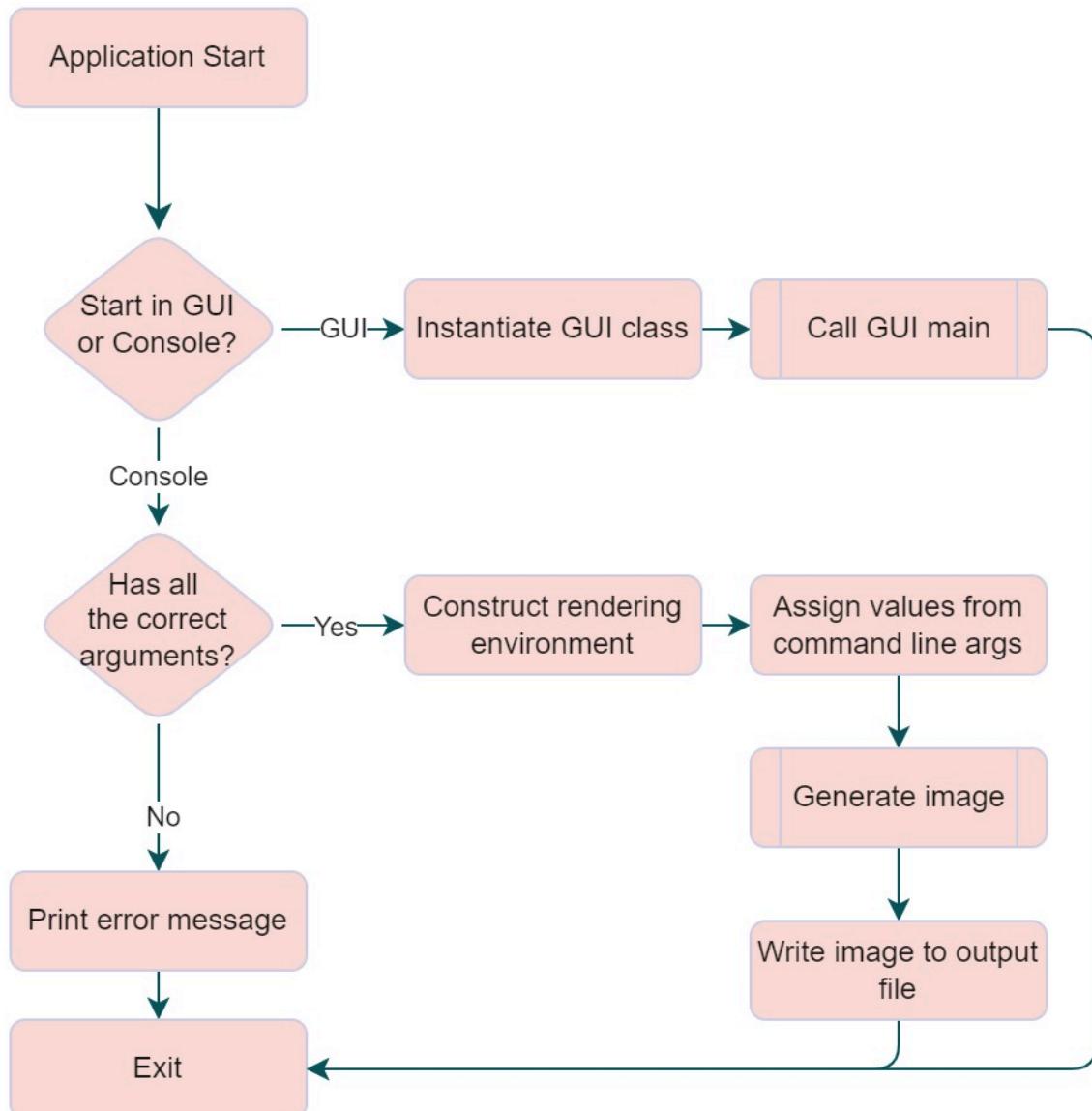
The following pages show comprehensive conceptual class diagrams, which have been separated into modules to make them easier to read. Due to the number of classes, structures and enumerations, the diagram has been split over multiple pages. Connections which split over page borders are shown with a label (as seen with 'GUI::Database Manager' which relates to the 'Database Manager' field on the 'GUI' class).

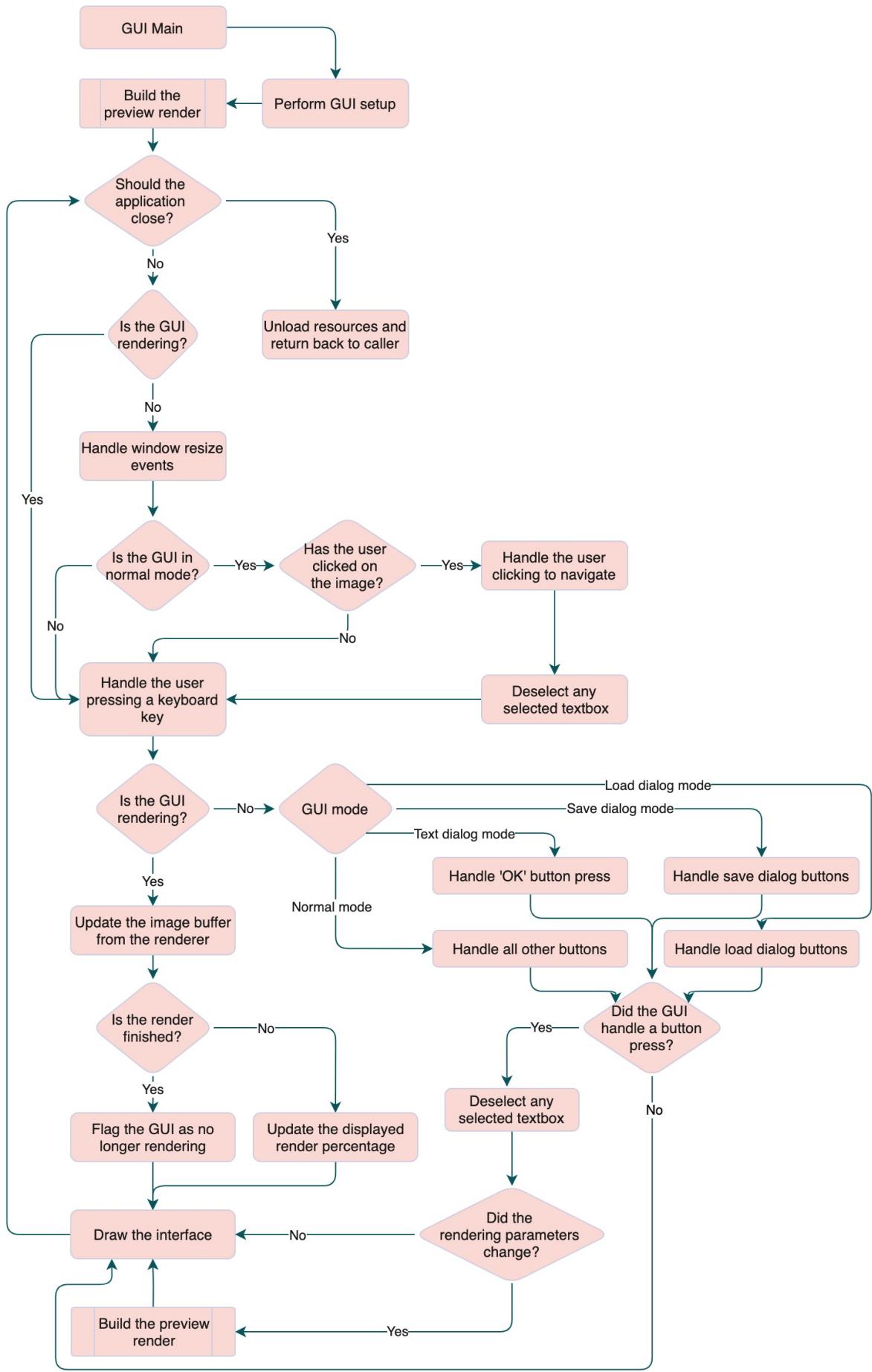


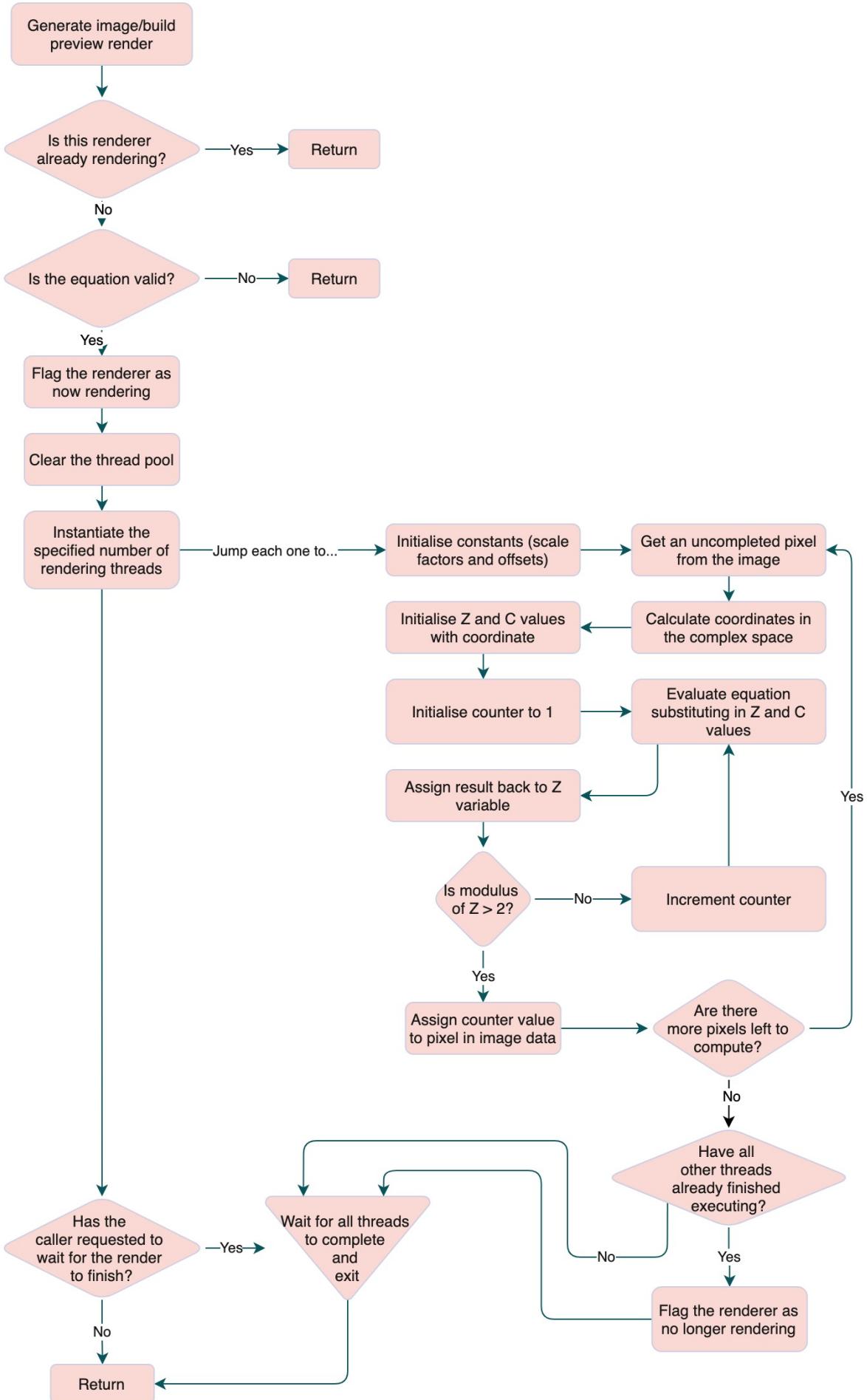




Program Flow Diagram

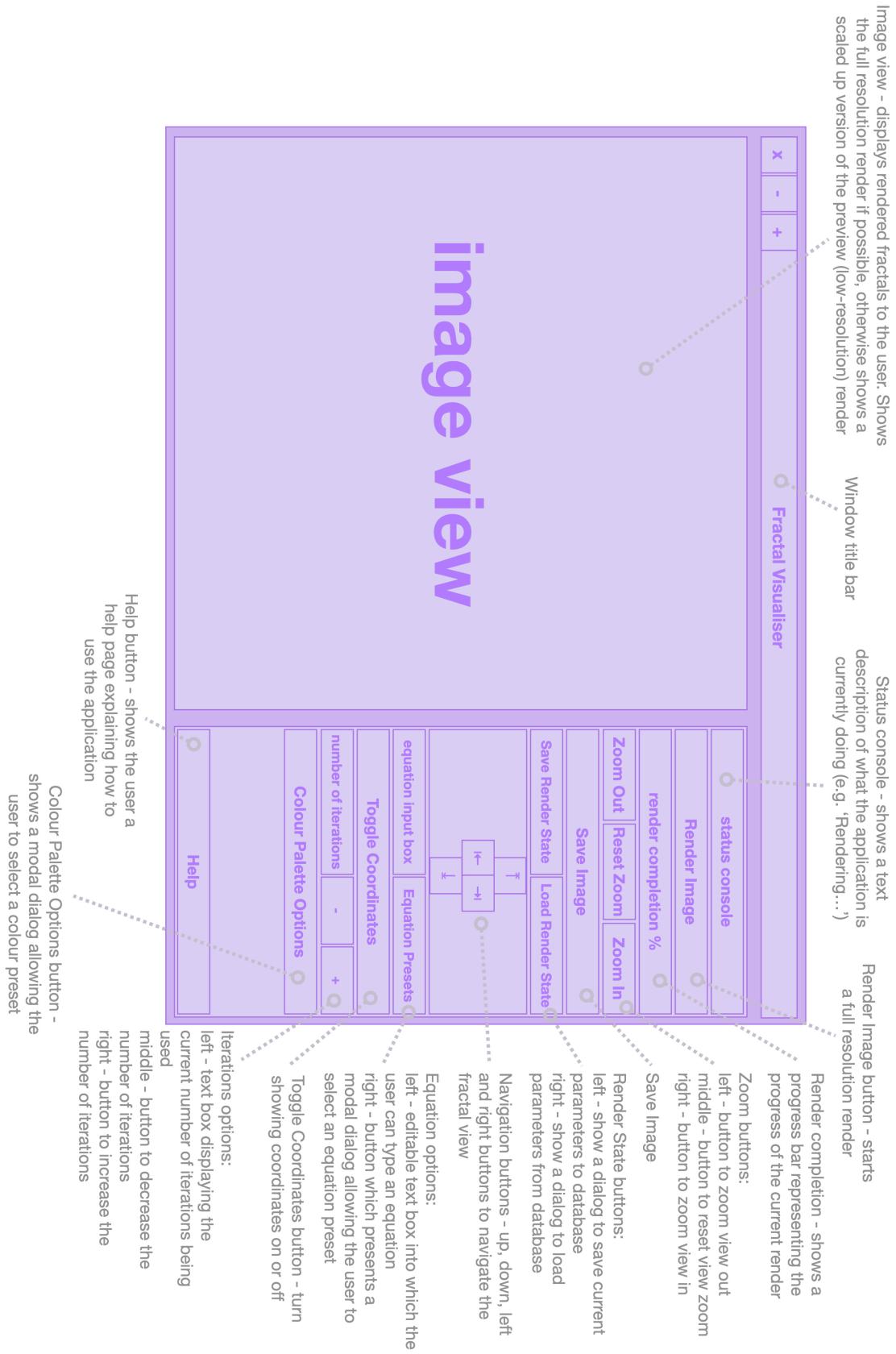




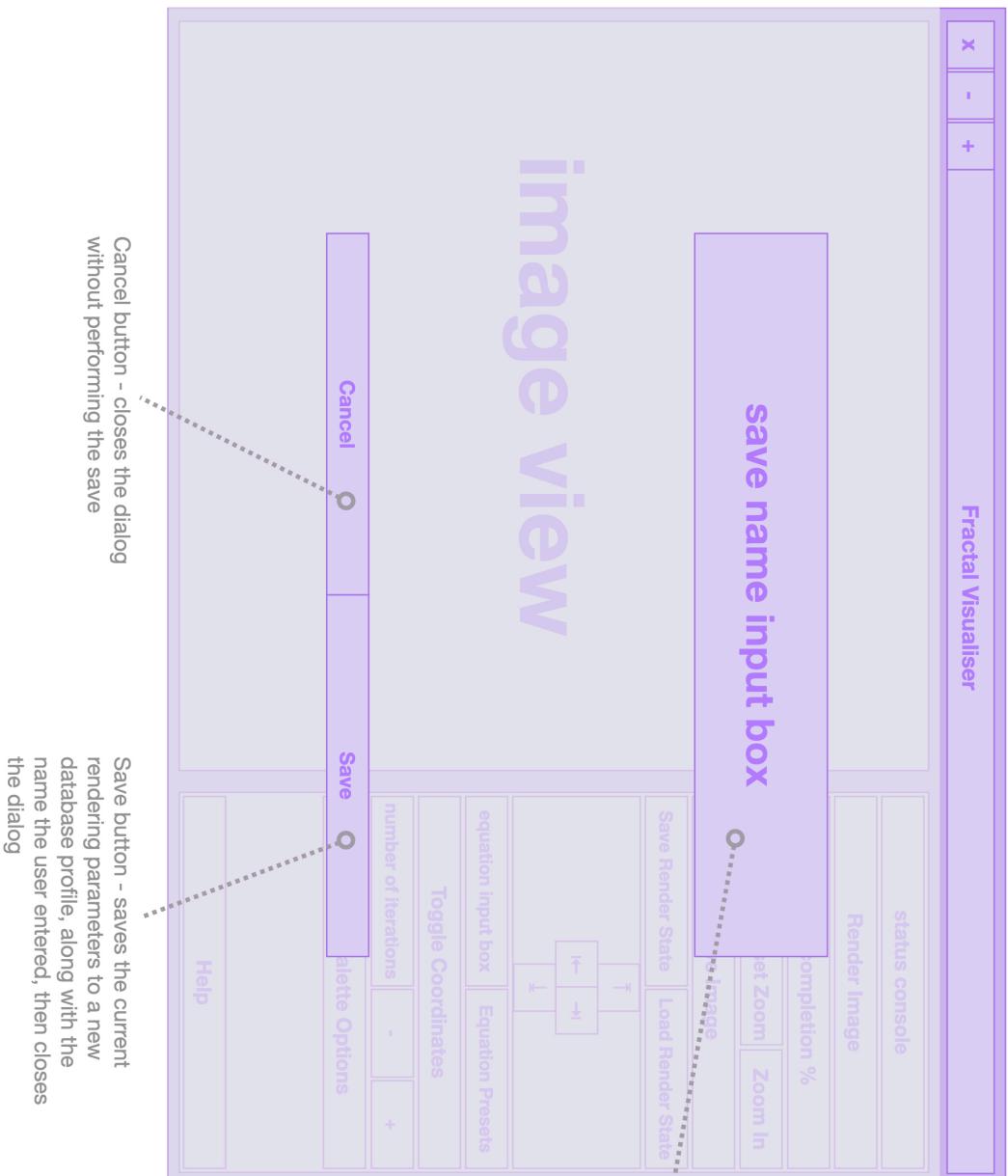


Interface Block-out

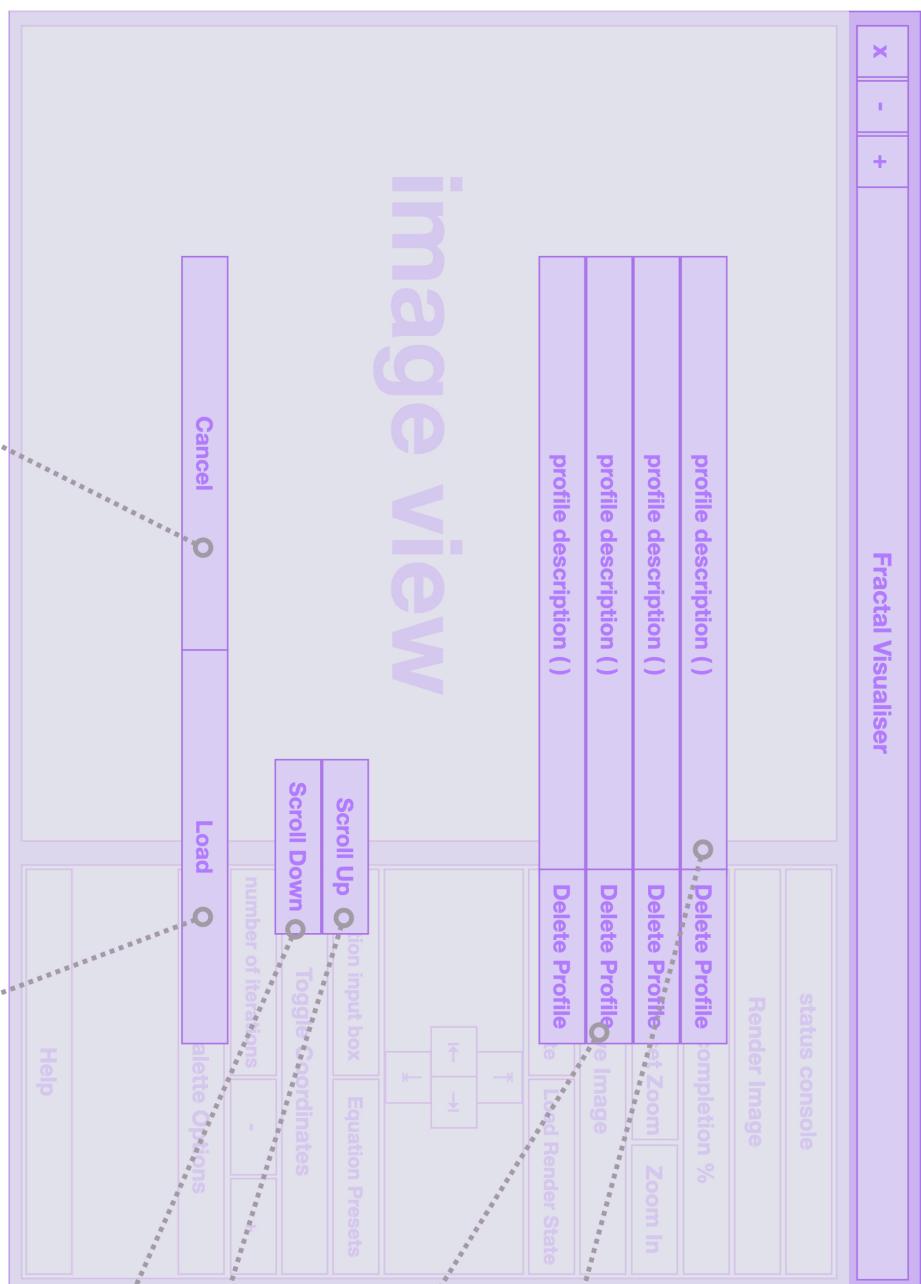
Base GUI State



Database Save Dialog GUI State



Database Load Dialog GUI State



Cancel button - closes the dialog
without performing the load

Load button - loads the selected
database profile as a set of
rendering parameters, overwriting
current ones

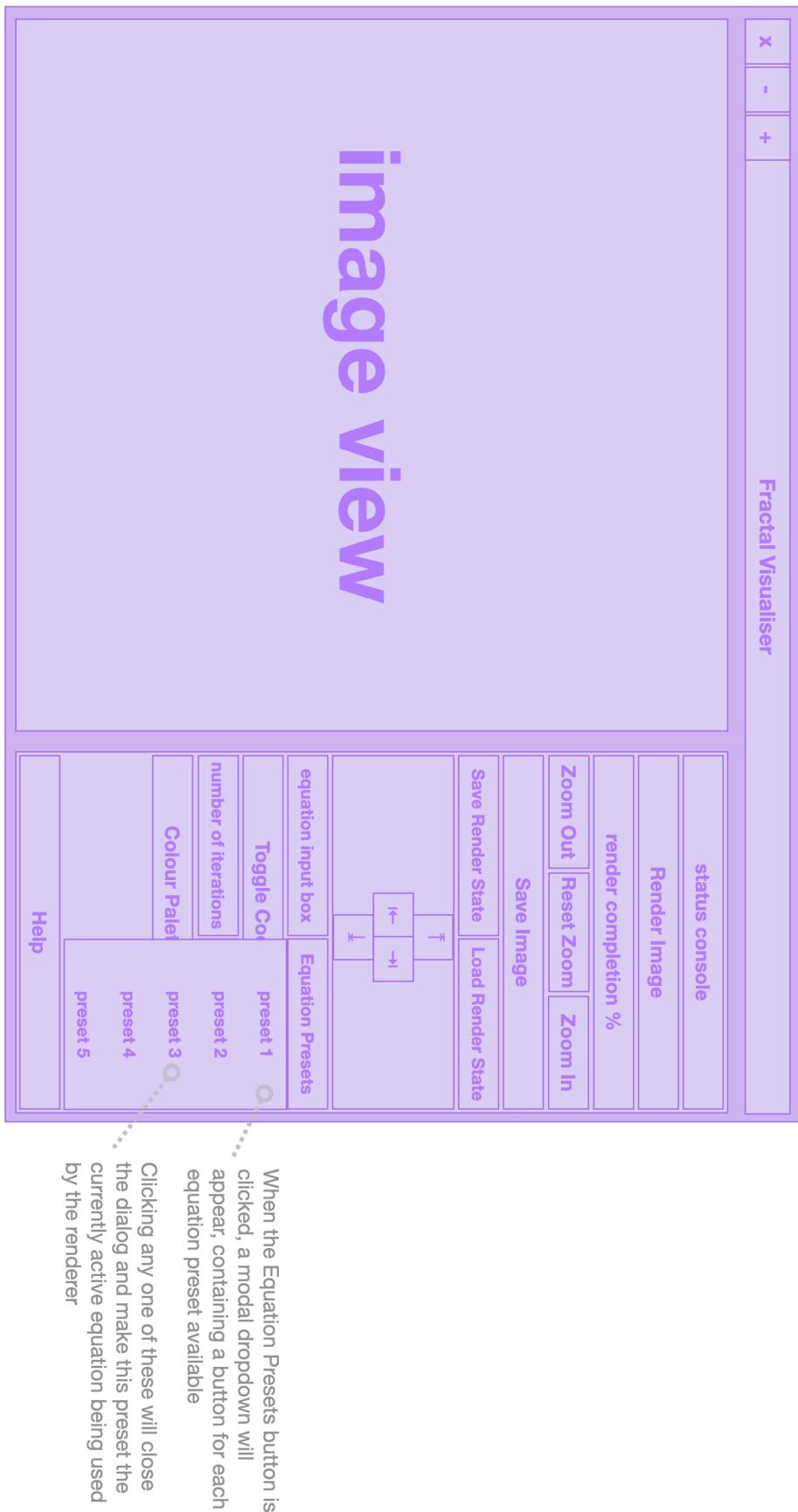
Scroll Up button - scrolls the profile
list view one item further towards the
top

Scroll Down button - scrolls the
profile list view one item further
towards the bottom

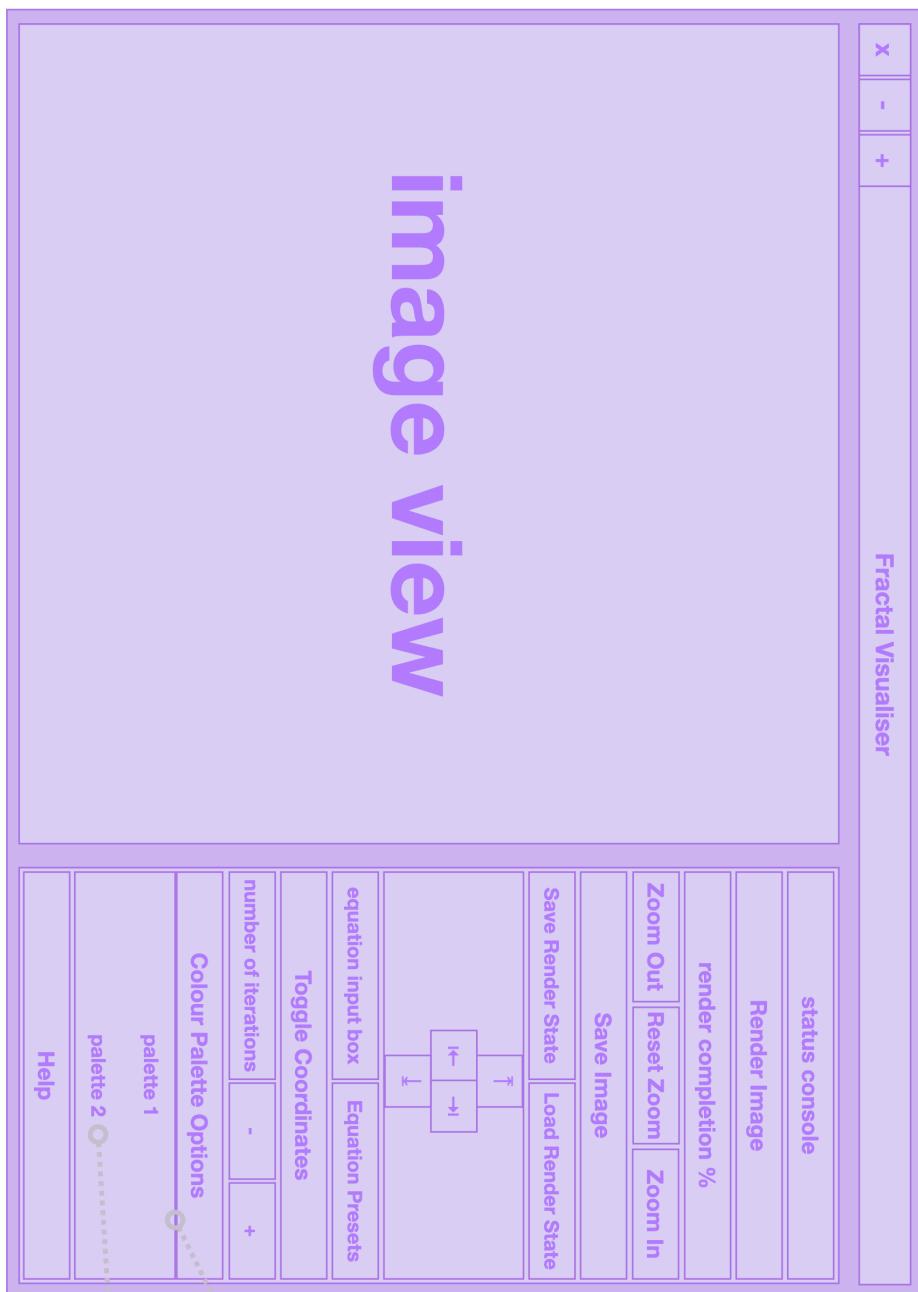
Delete Profile button - each profile is
accompanied by a button which
deletes it from the database when
clicked

Each list item is a button which
selects the profile it represents when
clicked

Equation Preset Dialog GUI State



Colour Palette Dialog GUI State



Toggle Coordinates

number of iterations - +

Colour Palette Options

palette 1



When the Colour Palette Options button is clicked, a modal dropdown will appear, containing a button for each colour palette available to choose from. Clicking any one of these will close the dialog and make this palette the currently active palette being used to interpret rendered images.

Algorithms & Data Structures

Parsing A Mathematical Expression with Recursion

This is one of the most difficult elements of this project, simply due to the variability of possible cases because this is a complex user-specified parameter.

Equations will be stored using **Reverse Polish Notation**, also known as **postfix notation**. This can be easily stored in an array, and can then be evaluated using a **stack** method.

Extracting the equation from a string is much harder however. I decided to do this in a step-by-step process, as can be seen in the program flow diagram. The string is first cleaned to remove whitespace, then checked to detect errors and invalid equations which would otherwise cause the parser to encounter problems. The parser can then step through the string and convert the sequence of characters into a sequence of intermediate tokens, which are much easier to process afterwards.

This is done by tracking the type of the character currently under inspection (i.e. whether it is a digit, letter, '.', operation, etc) and using changes in the current character type to decide when a new token begins (e.g. when the last character was a digit and the next is a '+', the parser can determine that a numerical constant has finished being read and that an operation token has just begun).

Handling brackets is the ideal situation to use a **recursive function**; this can be done by simply extracting the contents of each set of brackets, and feeding it back into the same tokenising function. The result can then be inserted into the resultant array of tokens of the outer function (and so on as far as necessary until the base case, the outermost function call being performed over the whole equation, is reached again), so that the entire equation is processed including all sets of brackets.

The next step of this is to turn the sequence of 'intermediate' tokens into a finalised array of tokens formatted as postfix notation. Postfix notation is laid out with the operator after the operands for each binary operation.

Before this finalising step, mathematical processing must be performed, which means obeying mathematical rules, specifically the BIDMAS rule for order of operations. This is done simply by iterating over the sequence of tokens and pairing them up over an operation, and wrapping this in brackets (making post-processing of the equation very easy). This is done in multiple passes, one per operation type, so first indices, then division, and so on, following the order of operations in mathematics.

However, another issue must be fixed as well. A common practice in mathematical notation is to use 'ab' instead of writing 'a * b'. This is known as **implicit multiplication**, and for the purposes of the equation parser it needs to be translated from the shortened form into the explicit form, by replacing instances of the form 'ab' with the form '(a*b)'. Order of operations still needs to be taken into account, so fixing implicit multiplication must be done in between the 'division' and 'multiplication' passes of the BIDMAS simplification phase.

Another mathematical practice which must be observed is the use of '-' as a unary operator. Binary operators are operators (like '+', '-', '*', and so on) which take two arguments and return a single output. Unary operators however only take a single argument. As a result, instances of '-a', where 'a' is any other value (e.g. a bracket, a number, etc), with '0-a'. This transforms the '-' into a binary operator, making it easier to process.

However, once all this is done, the finalising/post-processing of the equation is relatively simple. Because of the simplification phase, we can be sure that the sequence of 'intermediate' tokens will be formed only of a single $\langle \text{token} \rangle \langle \text{operation} \rangle \langle \text{token} \rangle$, where each token may be a bracket (which contains another set of $\langle \text{token} \rangle \langle \text{operation} \rangle \langle \text{token} \rangle$), a number, or a letter (out of z, x, y, c, a, b, i). In this way, tokens can be easily reordered into postfix notation; if they are simple tokens such as an operation or a number they can simply be copied out, otherwise they will be post-processed **recursively**, once again by calling the post-process function and inserting the result into the postfix token sequence from the outer function (again, this process of recursion is called down each set of brackets, until each sub-call has returned its results and the outermost function has been reached again).

This will generate a simple, linear, no-parsing-required sequence of instructions effectively which can be easily evaluated many times over by the fractal evaluator module.

Evaluating Mathematical Expressions using a Stack

After the majority of the work has been done by the parser, the fractal evaluator has minimal processing to do. This is deliberate, as the module must evaluate the equation many times (a maximum of $\text{ImageSize} * \text{ImageSize} * \text{IterationLimit}$ times, which on an average screen with average settings is very large). This is done by reading along the array of tokens produced by equation parser and treating each as an instruction.

A **stack** will be used to keep track of the computation state. Different tokens are treated differently:

- Number tokens will push a constant onto the stack with the value specified
- Letter tokens will be replaced with the relevant value (e.g. the letter token 'z' will be replaced with the current value for the z variable, see analysis for explanation of fractals) and pushed onto the stack
- Operation tokens will be executed over the top two items on the stack. These top items will be popped and the result of the operation will be pushed
- At the end of the token sequence, there will be only one value left on the stack, which can be copied off and returned.

This is done many times over for each pixel, until the computed value z is greater than the threshold, which is typically set at 2. Again, see analysis for explanation.

A major optimisation which can be made for this system is for built-in equation presets. The speed of evaluating equations in the way described above is limited by the overhead of performing the stack operations. However, if the equation being used is already known (i.e. it is one of the application's presets) evaluating equations in this way is unnecessary, and so instead for the equation presets (such as Mandelbrot and Julia sets) the evaluation function can be hard-coded and switched to at runtime, providing a major performance boost. Such hard-coded calculations avoid the overheads associated with stack operations and iterating over an array, and can also be optimised considerably in terms of machine code at compile-time by the compiler.

Dividing Up Workload with Multi-threading

Generating the entire fractal image requires a lot of processing, and would take an extremely long time on a single execution thread. Nearly all modern computers have more than one logical processor, meaning they have the capacity to run multiple operations simultaneously, allowing multiple pixels to be computed at the same time. Excluding overheads, this would effectively divide required computation time for the whole image by the number of logical processors, vastly increasing performance (especially on more powerful machines).

Multi-threading is key to achieving as realtime a result as possible. The command line arguments will allow the user to specify the number of worker threads to use, whilst the GUI mode will automatically detect the optimal number of threads to use for the given host system and configure the rendering environments accordingly.

From here, the workload of rendering the image must be divided up between the worker threads. The mechanism to be used is very simple: a custom image management class keeps track of not only the image data containing computed pixel values (i.e. the outputs for each pixel from the evaluator module), but also of whether or not a pixel needs to be calculated or not. This way, each worker thread simply requests a new pixel coordinate from the image manager, which would then mark that pixel as 'in-progress'. The image class should keep track of which pixel should be assigned next to minimise wait times when threads request new pixels to compute. After each pixel is computed, it can be assigned back to the image data of the image class, which will flag the pixel as completed at the same time.

Some locking is required on the image manager class to prevent collisions, but this will be minimised in order to maintain performance without encountering collisions.

When the image is complete, the threads can be automatically halted as computation is complete.

Generating Colours with a Colour Converter

In order to display the fractals, they must be converted from their 'raw' form (stored simply as the number of iterations required for each pixel) into a 'prettified' colourful image. The solution will provide a set of several different colour presets, which algorithmically convert the raw data into a 32bit RGBA image which can be

displayed. Although not every colour palette will use this algorithm, a **colour conversion** algorithm must be implemented, to convert between different colour formats.

One format for representing colour is ‘HSV’ - Hue, Saturation, Value. Hue represents the particular colour group of a colour, measured around the colour wheel. Saturation represents how strong that colour is (i.e. between pure hue and pure white). Value represents how light the colour is. This format gives good artistic control over colour, as different parameters can be changed more easily (for instance, you can make a colour ‘lighter’ with a single value, without having to manually increase each RGB channel).

Another format is ‘RGB’ - Red, Green, Blue. This format balances the strengths of three components, red, green and blue primary colours to produce another colour (i.e. all colours fully shown is white). This format is more common for representing colour by computers, and is used to render images to the screen, due to the way physical pixels work (they balance the brightness of red, green and blue LEDs).

For images to be displayed, they must be in RGBA format. The A here represents Alpha, which is simply the opacity of a pixel and here is set to always be 100% (fully opaque). However, handling colours as HSV data is much easier to produce nice-looking gradients between colours. Thus, there is a need to write a **mathematical colour conversion algorithm**, to convert HSV defined colours into RGBA colours which can be displayed on the screen. For this the solution will include a simple HSV-to-RGBA function, which implements the standard mathematical algorithm for such an operation.

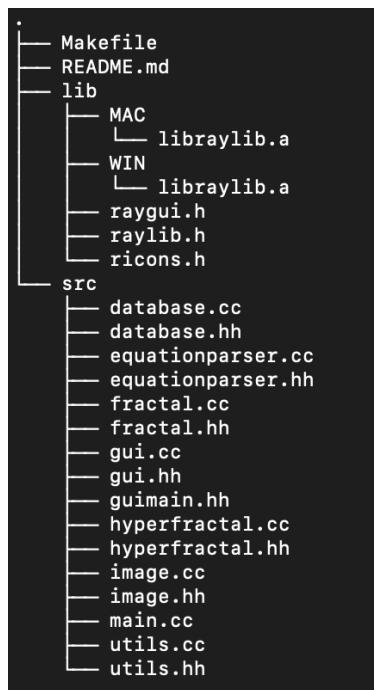
Technical Solution

On the following pages are inserted the source files for the implementation, but not including the Makefile. A configured, buildable version of the project is available on <https://github.com/JkyProgrammer/HyperFractal>.

Each of the files below has been formatted with a comment at the top which states the name of the file it originates from.

Mini-Contents for Implementation

On the right is the directory structure for the project, including libraries and source files. See the contents table below for the page numbers of specific source files in the following insert.



database.cc	30
database.hh	39
equationparser.cc	41
equationparser.hh	52
fractal.cc	54
fractal.hh	59
gui.cc	61
gui.hh	89
guimain.hh	94
hyperfractal.cc	95
hyperfractal.hh	101
image.cc	104
image.hh	110
main.cc	111
utils.cc	113
utils.hh	117

```

1 // src/database.cc
2
3 #include "database.hh"
4
5 #include <vector>
6 #include <string>
7 #include <iostream>
8 #include <fstream>
9
10 using namespace std;
11
12 /**
13 * @brief Modify a string so that it can be written to CSV properly. This
14 * means replacing double quotes with a pair of consecutive double quotes
15 *
16 * @param s Input string
17 * @return A ready-to-write result string
18 */
19 std::string HFractalDatabase::forCSVInner (std::string s) {
20     string fixed_quotes = "";
21     for (char c : s) {
22         fixed_quotes += c;
23         if (c == '\"') fixed_quotes += '\"';
24     }
25     fixed_quotes = "\"" + fixed_quotes + "\"";
26     return fixed_quotes;
27 }
28 /**
29 * @brief Break a record from a CSV file into a sequence of raw string
30 * fields
31 *
32 * @param line Line from CSV file to process
33 * @return std::vector of raw string fields (require additional processing)
34 */
35 std::vector<std::string> HFractalDatabase::componentify (std::string line) {
36     vector<string> ret_value;
37     string current_component;
38     int start_index = line.find ("\"");
39     int end_index = -1;
40     while (start_index < line.length ()) {
41         end_index = line.find ("\",\"", start_index);

```

```

41     if (end_index == string::npos) end_index = line.find ("\"", start_index+1);
42     current_component = line.substr (start_index+1, end_index-(start_index+1));
43     current_component = fixDoubleQuote (current_component);
44     ret_value.push_back (current_component);
45     start_index = end_index+2;
46 }
47 return ret_value;
48 }
49
50 /**
51 * @brief Remove double-double quotes from a string. Effectively the reverse
52 * of forCSVInner
53 *
54 * @param s String field to process
55 * @return Cleaned-up field string
56 */
57 std::string HFractalDatabase::fixDoubleQuote (std::string s) {
58     string result = "";
59     char current_char = '\0';
60     char next_char = '\0';
61     for (int i = 0; i < s.length(); i++) {
62         current_char = s[i];
63         next_char = (i+1 < s.length()) ? s[i+1] : '\0';
64         result.push_back (current_char);
65         if (current_char == '\"' && next_char == '\"') i++;
66     }
67     return result;
68 }
69 /**
70 * @brief Construct a database instance using a base path to CSV files
71 *
72 * @param path Base path to the target CSV database. Individual tables must
73 * be stored as separate CSV files, so the base path is modified provide paths
74 * to the individual CSV files
75 */
76 HFractalDatabase::HFractalDatabase (std::string path) {
77     // Generate and assign the base path
78     int cutoff = path.find(".csv");

```

```

79     // Try to read the database, failing that write a new one, failing that,
80     // error out
81     if (!read()) if (!commit ()) {
82         throw new std::runtime_error ("unable to create database");
83     }
84
85 HFractalDatabase::HFractalDatabase () {}
86
87 /**
88  * @brief Get a list of config profile descriptions paired with their IDs.
89  * Allows the GUI to easily grab a profile summary without having to fetch all
90  * the data one-by-one
91  *
92  * @return std::vector of pairs of ID and string values
93  */
94 std::vector<std::pair<long, std::string>>
95 HFractalDatabase::getConfigDescriptions () {
96     std::vector<std::pair<long, std::string>> ret_val;
97     for (auto conf : configs) {
98         std::pair<long, std::string> desc_pair;
99         desc_pair.first = conf.first;
100        desc_pair.second = (
101            conf.second->name
102            + " ("
103            + conf.second->equation
104            + ")");
105        ret_val.push_back (desc_pair);
106    }
107    return ret_val;
108 }
109
110 /**
111  * @brief Function to get a configuration profile by its ID
112  *
113  * @param id The ID of the profile
114  * @return A pointer to the configuration profile
115  */
116 HFractalConfigProfile* HFractalDatabase::getConfig (long id) {
117     HFractalConfigProfile *ret = NULL;
118     if (configs.count(id) != 0) ret = configs[id];
119     return ret;
120 }
```

```

118
119 /**
120 * @brief Function to get a user profile by its ID
121 *
122 * @param id The ID of the profile
123 * @return A pointer to the user profile
124 */
125 HFractalUserProfile* HFractalDatabase::getUser (long id) {
126     HFractalUserProfile *ret = NULL;
127     if (users.count(id) != 0) ret = users[id];
128     return ret;
129 }
130
131 /**
132 * @brief Insert a configuration profile into the database. Automatically
133 * generates and assigns a unique ID
134 *
135 * @param c Pointer to the config profile being inserted
136 * @return Generated ID of the profile
137 */
138 long HFractalDatabase::insertConfig (HFractalConfigProfile* c) {
139     long max_id = -1;
140     for (pair<long, HFractalConfigProfile*> p : configs)
141         if (p.first > max_id) max_id = p.first;
142
143     c->profile_id = max_id+1;
144     configs.emplace (c->profile_id, c);
145     return c->profile_id;
146 }
147 /**
148 * @brief Insert a user profile into the database. Automatically generates
149 * and assigns a unique ID
150 *
151 * @param u Pointer to the user profile being inserted
152 * @return Generated ID of the profile
153 */
154 long HFractalDatabase::insertUser (HFractalUserProfile* u) {
155     long max_id = -1;
156     for (pair<long, HFractalUserProfile*> p : users)
157         if (p.first > max_id) max_id = p.first;

```

```

158     u->user_id = max_id+1;
159     users.emplace (u->user_id, u);
160     return u->user_id;
161 }
162
163 /**
164 * @brief Delete a config profile record from the database
165 *
166 * @param id ID of the profile to be deleted
167 * @return True if the delete succeeded, False if the record did not exist
168 */
169 bool HFractalDatabase::removeConfig (long id) {
170     return configs.erase (id);
171 }
172
173 /**
174 * @brief Delete a user profile record from the database
175 *
176 * @param id ID of the profile to be deleted
177 * @return True if the delete succeeded, False if the record did not exist
178 */
179 bool HFractalDatabase::removeUser (long id) {
180     return users.erase (id);
181 }
182
183 /**
184 * @brief Write out the contents of the cached database to CSV files.
185 *
186 * @return True if the write succeeded, False if it failed
187 */
188 bool HFractalDatabase::commit () {
189     // Create path strings and file streams
190     string db_path_configs = db_path + "_configs.csv";
191     string db_path_users = db_path + "_users.csv";
192     ofstream db_file_configs (db_path_configs.c_str());
193     ofstream db_file_users (db_path_users.c_str());
194
195     // If the file streams are open, write data
196     if (db_file_configs.is_open() && db_file_users.is_open()) {
197         // Iterate over config files
198         for (auto copair : configs) {
199             HFractalConfigProfile* config = copair.second;

```

```

200     string line = "";
201     line += forCSV (config->profile_id) + ",";
202     line += forCSV (config->x_offset) + ",";
203     line += forCSV (config->y_offset) + ",";
204     line += forCSV (config->zoom) + ",";
205     line += forCSV (config->iterations) + ",";
206     line += forCSV (config->equation) + ",";
207     line += forCSV (config->name) + ",";
208     line += forCSV (config->palette) + ",";
209     line += forCSV (config->user_id);
210     db_file_configs << line.c_str() << endl;
211 }
212
213 // Iterate over user files
214 for (auto upair : users) {
215     HFractalUserProfile* user = upair.second;
216     string line = "";
217     line += forCSV (user->user_id) + ",";
218     line += forCSV (user->user_name);
219     db_file_users << line.c_str() << endl;
220 }
221
222 // Close the files and return success
223 db_file_configs.close();
224 db_file_users.close();
225 return true;
226 } else return false; // Return failure
227 }
228
229 /**
230 * @brief Read the contents of CSV files into the cached database
231 *
232 * @return True if the read succeeded, False if it failed
233 */
234 bool HFRACTALDatabase::read () {
235     // Create paths and file streams
236     string db_path_configs = db_path + "_configs.csv";
237     string db_path_users = db_path + "_users.csv";
238     ifstream db_file_configs (db_path_configs.c_str());
239     ifstream db_file_users (db_path_users.c_str());
240
241     // If the file streams are open, read data

```

```

242 if (db_file_configs.is_open() && db_file_users.is_open()) {
243     string line;
244     int line_number = 0;
245
246     configs.clear();
247     // Read config profiles
248     HFractalConfigProfile* config;
249     while (getline (db_file_configs, line)) {
250         try {
251             // Attempt to convert the record to a config profile
252             vector<string> components = componentify (line);
253             config = new HFractalConfigProfile ();
254             config->profile_id = stol(components[0]);
255             config->x_offset = stold(components[1]);
256             config->y_offset = stold(components[2]);
257             config->zoom = stold(components[3]);
258             config->iterations = stoi(components[4]);
259             config->equation = components[5];
260             config->name = components[6];
261             config->palette = stoi(components[7]);
262             config->user_id = stol(components[8]);
263             configs.emplace (config->profile_id, config);
264         } catch (std::invalid_argument e) {
265             // Print a console error if a line could not be read
266             cout << "Failed to read config profile on line " <<
267             line_number << endl;
268         }
269     }
270     line_number = 0;
271
272     // Read user profiles
273     users.clear();
274     HFractalUserProfile* user;
275     while (getline (db_file_users, line)) {
276         try {
277             // Attempt to convert the record to a user profile
278             vector<string> components = componentify (line);
279             user = new HFractalUserProfile ();
280             user->user_id = stol(components[0]);
281             user->user_name = components[1];

```

```

283         users.emplace (user->user_id, user);
284     } catch (std::invalid_argument e) {
285         // Print a console error if a line could not be read
286         cout << "Failed to read user profile on line " <<
287         line_number << endl;
288     }
289 }
290
291     // Return success
292     return true;
293 }
294
295 /**
296 * @brief Generate a CSV-happy string from a given input
297 *
298 * @param s String input
299 * @return CSV-writeable string
300 */
301 std::string HFractalDatabase::forCSV (std::string s) {
302     return forCSVInner (s);
303 }
304
305 /**
306 * @brief Generate a CSV-happy string from a given input
307 *
308 * @param l Long integer input
309 * @return CSV-writeable string
310 */
311 std::string HFractalDatabase::forCSV (long l) {
312     return forCSVInner (to_string(l));
313 }
314
315 /**
316 * @brief Generate a CSV-happy string from a given input
317 *
318 * @param ld Long double input
319 * @return CSV-writeable string
320 */
321 std::string HFractalDatabase::forCSV (long double ld) {
322     return forCSVInner (to_string(ld));
323 }

```

```
324 |
325 /**
326 * @brief Generate a CSV-happy string from a given input
327 *
328 * @param i Integer input
329 * @return CSV-writeable string
330 */
331 std::string HFractalDatabase::forCSV (int i) {
332     return forCSVInner (to_string(i));
333 }
```

```

1 // src/database.hh
2
3 #ifndef DATABASE_H
4 #define DATABASE_H
5
6 #include <string>
7 #include <vector>
8 #include <unordered_map>
9 #include <cstring>
10
11 // Struct describing the Config Profile record type
12 struct HFractalConfigProfile {
13     long profile_id; // Primary key
14
15     long double x_offset;
16     long double y_offset;
17     long double zoom;
18     int iterations;
19     std::string equation;
20     std::string name;
21     int palette;
22     long user_id; // Foreign key of HFractalUserProfile
23
24     HFractalConfigProfile () { memset (this, 0,
25         sizeof(HFractalConfigProfile)); }
26 };
27
28 // Struct describing the User Profile record type
29 struct HFractalUserProfile {
30     long user_id; // Primary key
31
32     std::string user_name;
33
34     HFractalUserProfile () { memset (this, 0, sizeof(HFractalUserProfile)); }
35 };
36
37 // Class for managing the database of users and configurations and providing
38 // access to data for the GUI
39 class HFractalDatabase {
40 private:
41     std::string db_path; // Base path to the database
42     std::unordered_map<long, HFractalConfigProfile*> configs; // Map of

```

```

config profiles against their IDs
41     std::unordered_map<long, HFractalUserProfile*> users; // Map of user
profiles against their IDs
42     static std::string forCSVInner (std::string); // Static function to
convert a string into a form CSVs will read/write properly
43     std::vector<std::string> componentify (std::string); // Break a line of
CSV into fields
44     std::string fixDoubleQuote (std::string); // Remove double quotes in
strings read from CSV file
45
46     static std::string forCSV (std::string); // Generate a string which can
be written to a CSV file as a field of a record
47     static std::string forCSV (long); // Generate a string which can be
written to a CSV file as a field of a record
48     static std::string forCSV (int); // Generate a string which can be
written to a CSV file as a field of a record
49     static std::string forCSV (long double); // Generate a string which can
be written to a CSV file as a field of a record
50 public:
51     HFractalDatabase (std::string); // Initialise the database from a given
base path
52     HFractalDatabase (); // Dead initialiser for implicit instantiation
53
54     std::vector<std::pair<long, std::string>> getConfigDescriptions (); // //
Generate a list of ID and description pairs for the GUI to display
55     HFractalConfigProfile* getConfig (long); // Get a pointer to the config
profile with a given ID
56     HFractalUserProfile* getUser (long); // Get a pointer to the user profile
with a given ID
57
58     long insertConfig (HFractalConfigProfile*); // Insert a new config record
and return its ID
59     long insertUser (HFractalUserProfile*); // Insert a new user record and
return its ID
60
61     bool removeConfig (long); // Remove a config record by ID
62     bool removeUser (long); // Remove a user record by ID
63
64     bool commit (); // Write the contents of the cached database out to CSV
files
65     bool read (); // Read the contents of a CSV file into the database cache
66 };
67
68 #endif

```

```

1 // src/equationparser.cc
2
3 #include "equationparser.hh"
4
5 #include <vector>
6
7 using namespace std;
8
9 /**
10 * @brief Clean whitespace out of the input string
11 *
12 * @param s Input string
13 * @return Cleaned string
14 */
15 string HFractalEquationParser::epClean (string s) {
16     string ret_val = "";
17     for (char c : s) if (c != ' ') ret_val += c;
18     return ret_val;
19 }
20
21 /**
22 * @brief Check that the input string is valid for the HFractalEquation
23 * parser to analyse. Checks for the following and returns an enum value
24 * accordingly:
25 *
26 * 0 - No error found
27 * 1 - Bracket error: '()' , '(' not equal number to ')' , ')'...('
28 * 2 - Operation error: '**' , '*' or any other repetition of an operation
29 * 3 - Implicit multiplication error: 'z2' rather than correct synax '2z'
30 * 4 - Floating point error: '.46' , '34.'
31 * 5 - Unsupported character error: '$' , 'd' , or any other character not
32 * accounted for
33 *
34 * @param s Input string
35 * @return Either the reference of the first error detected or SUCCESS if no
36 * error is found
37 */
38 EP_CHECK_STATUS HFractalEquationParser::epCheck (string s) {
39     int bracket_depth = 0;
40     char c_last = '\0';
41     int index = 0;
42     for (char c : s) {
43         switch (c) {

```

```

40     case '(':
41         bracket_depth++;
42         if (c_last == '.') return FPOINT_ERROR;
43         break;
44     case ')':
45         bracket_depth--;
46         if (c_last == '(') return BRACKET_ERROR;
47         if (c_last == '.') return FPOINT_ERROR;
48         break;
49     case 'z':
50     case 'c':
51     case 'a':
52     case 'b':
53     case 'x':
54     case 'y':
55     case 'i':
56         if (c_last == '.') return FPOINT_ERROR;
57         break;
58     case '*':
59     case '/':
60     case '+':
61     case '^':
62         if (c_last == '*' || c_last == '/' || c_last == '-' || c_last ==
63 '+' || c_last == '^') return OPERATION_ERROR;
64         if (c_last == '.') return FPOINT_ERROR;
65         if (index == 0 || index == s.length()-1) return OPERATION_ERROR;
66         break;
67     case '-':
68         if (c_last == '-') return OPERATION_ERROR;
69         if (c_last == '.') return FPOINT_ERROR;
70         if (index == s.length()-1) return OPERATION_ERROR;
71         break;
72     case '0':
73     case '1':
74     case '2':
75     case '3':
76     case '4':
77     case '5':
78     case '6':
79     case '7':
80     case '8':
81     case '9':

```

```

81     if (c_last == 'z' || c_last == 'c' || c_last == 'i' || c_last ==
82         'a' || c_last == 'b' || c_last == 'x' || c_last == 'y') return IMULT_ERROR;
83         break;
84     case '.':
85         if (!(c_last == '0' || c_last == '1' || c_last == '2' || c_last ==
86             '3' || c_last == '4' || c_last == '5' || c_last == '6' || c_last == '7'
87             || c_last == '8' || c_last == '9')) return FPOINT_ERROR;
88         break;
89     default:
90         return UNSUPCHAR_ERROR;
91     break;
92 }
93
94     c_last = c;
95     index++;
96     if (bracket_depth < 0) return BRACKET_ERROR;
97 }
98 }
99
100 /**
101 * @brief Break string into tokens for processing. Assumes epCheck has been
102 *        called on `s` previously and that this has returned 0
103 *
104 * @param s Input string
105 * @return std::vector of tokens
106 */
107 vector<IntermediateToken> HFractalEquationParser::epTokenise (string s) {
108     vector<IntermediateToken> token_vec;
109     string current_token = "";
110     int current_token_type = -1;
111     bool is_last_run = false;
112     bool is_singular_token = false; // Informs the program whether the token
113     is a single-char token
114
115     for (int i = 0; i < s.length(); i++) {
116         char current_char = s[i];
117         int char_token_type = -1;
118
119         // Decide the type of the current character
120         switch (current_char) {

```

```

119     case '0':
120     case '1':
121     case '2':
122     case '3':
123     case '4':
124     case '5':
125     case '6':
126     case '7':
127     case '8':
128     case '9':
129     case '.':
130         char_token_type = 0;
131         break;
132     case 'z':
133         char_token_type = 1;
134         break;
135     case 'c':
136         char_token_type = 2;
137         break;
138     case 'a':
139         char_token_type = 6;
140         break;
141     case 'b':
142         char_token_type = 7;
143         break;
144     case 'x':
145         char_token_type = 8;
146         break;
147     case 'y':
148         char_token_type = 9;
149         break;
150     case 'i':
151         char_token_type = 3;
152         break;
153     case '*':
154     case '/':
155     case '-':
156     case '+':
157     case '^':
158         char_token_type = 4;
159         break;
160     case '(':

```

```

161         char_token_type = 5;
162         break;
163     default:
164         break;
165     }
166
167     // Save the current token the token vector
168     if (char_token_type != current_token_type || is_last_run || is_singular_token) {
169         is_singular_token = false;
170         if (current_token.length () > 0) {
171             IntermediateToken token;
172             switch (current_token_type) {
173                 case 0:
174                     token.type = INT_NUMBER;
175                     token.num_val = stod (current_token);
176                     break;
177                 case 1:
178                 case 2:
179                 case 3:
180                 case 6:
181                 case 7:
182                 case 8:
183                 case 9:
184                     token.type = INT_LETTER;
185                     token.let_val = current_token[0];
186                     break;
187                 case 4:
188                     token.type = INT_OPERATION;
189                     token.op_val = current_token[0];
190                     break;
191                 case 5:
192                     token.type = INT_BRACKET;
193                     token.bracket_val = epTokenise (current_token);
194             default:
195                 break;
196             }
197             token_vec.push_back (token);
198         }
199         current_token = "";
200         current_token_type = char_token_type;
201     }

```

```

202         if (is_last_run) break;
203     }
204
205     // Jump automatically to the end of the brackets, recursively
206     // processing their contents
207     if (char_token_type == 5) {
208         int bracket_depth = 1;
209         int end = -1;
210         for (int j = i+1; j < s.length(); j++) {
211             if (s[j] == '(') bracket_depth++;
212             if (s[j] == ')') bracket_depth--;
213             if (bracket_depth == 0) { end = j; break; }
214         }
215         current_token = s.substr (i+1, end-(i+1));
216         i = end;
217     } else { // Otherwise append the current character to the current
218     token
219         current_token += s[i];
220     }
221
222     // Mark a, b, c, x, y, z, and i as singular
223     if ((char_token_type >= 1 && char_token_type <= 3) ||
224     (char_token_type >= 6 && char_token_type <= 9)) {
225         is_singular_token = true;
226     }
227
228     // If we've reached the end of the string, jump back and mark it as
229     // a last pass in order to save the current token
230     if (i == s.length()-1) {
231         is_last_run = true;
232         i--;
233     }
234
235     // Check through and repair any `...` expressions to be `0...`
236     for (int i = 0; i < token_vec.size(); i++) {
237         if (token_vec[i].type == INT_OPERATION && token_vec[i].op_val == '-')
238     } {
239         if (i == 0 || (i > 0 && token_vec[i-1].type == INT_OPERATION)) {
240             IntermediateToken bracket;
241             bracket.type = INT_BRACKET;
242             int bracket_length = 1;

```

```

240         bracket.bracket_val.push_back ({
241             .type = INT_NUMBER,
242             .num_val = 0
243         });
244         bracket.bracket_val.push_back ({
245             .type = INT_OPERATION,
246             .op_val = '_'
247         });
248         while (i+bracket_length < token_vec.size() &&
249             token_vec[i+bracket_length].type != INT_OPERATION) {
250             bracket.bracket_val.push_back
251             (token_vec[i+bracket_length]);
252             bracket_length++;
253         }
254     }
255     for (int tmp = 0; tmp < bracket_length; tmp++)
256     token_vec.erase (next(token_vec.begin(), i));
257     token_vec.insert (next(token_vec.begin(), i), bracket);
258     i -= bracket_length-1;
259   }
260   return token_vec;
261 }
262
263 /**
264 * @brief Search for and replace implicit multiplication (adjacent non-
265 * operation tokens such as '5z' or '(...)(...)') with explicit multiplication
266 *
267 * @param token_vec Token vector to fix
268 * @return Token vector with no implicit multiplication
269 */
270 vector<IntermediateToken> HFractalEquationParser::epFixImplicitMul
271 (vector<IntermediateToken> token_vec) {
272     vector<IntermediateToken> result = token_vec;
273     for (int i = 0; i < result.size()-1; i++) {
274         IntermediateToken t1 = result[i];
275         IntermediateToken t2 = result[i+1];
276
277         // Fix implicit multiplication within brackets
278         if (t1.type == INT_BRACKET) {
279             result[i].bracket_val = epFixImplicitMul (t1.bracket_val);

```

```

278         t1 = result[i];
279     }
280
281     // Detect two adjacent tokens, where neither is an operation
282     if (!(t1.type == INT_OPERATION || t2.type == INT_OPERATION)) {
283         result.erase (next(result.begin(), i));
284         result.erase (next(result.begin(), i));
285         IntermediateToken explicit_mul;
286         explicit_mul.type = INT_BRACKET;
287         explicit_mul.bracket_val.push_back (t1);
288         explicit_mul.bracket_val.push_back ({
289             .type = INT_OPERATION,
290             .op_val = '*'
291         });
292         explicit_mul.bracket_val.push_back (t2);
293
294         result.insert (next(result.begin(), i), explicit_mul);
295         i--;
296     }
297 }
298
299 IntermediateToken last = result[result.size()-1];
300 if (last.type == INT_BRACKET) {
301     last.bracket_val = epFixImplicitMul (last.bracket_val);
302     result[result.size()-1] = last;
303 }
304
305 return result;
306 }
307
308 /**
309 * @brief Ensure BIDMAS (Brackets Indices Division Multiplication Addition
310 * Subtraction) order mathematical evaluation by search-and-replacing each with
311 * brackets
312 *
313 * @param token_vec Token vector to simplify
314 * @return Token vector which requires only sequential evaluation
315 */
316
317 vector<IntermediateToken> HFractalEquationParser::epSimplifyBidmas
318 (vector<IntermediateToken> token_vec, bool first_half) {
319     vector<IntermediateToken> result = token_vec;
320     // Recurse down brackets
321     for (int i = 0; i < result.size(); i++) {

```

```

318     if (result[i].type == INT_BRACKET) {
319         result[i].bracket_val = epSimplifyBidmas (result[i].bracket_val,
first_half);
320     }
321 }
322
323 // Order of operations:
324 //          IDMAS
325 string ops = "^/*+-";
326
327 // First half allows the parser to make indices and division explicit,
then process implicit multiplication, and then to process other operations
328 // This allows us to maintain BIDMAS even with explicit multiplication
(e.g. `5z^2` should be `5*(z^2)` and not `(5*z)^2`)
329 if (first_half) {
330     ops = "^/";
331 } else {
332     ops = "*+-";
333 }
334
335 if (result.size() < 5) return result;
336
337 // Search and replace each sequentially
338 for (char c : ops) {
339     for (int t_ind = 0; t_ind < result.size()-2; t_ind++) {
340         if (t_ind >= result.size()-2) {
341             break;
342         }
343         if (result[t_ind+1].type == INT_OPERATION &&
result[t_ind+1].op_val == c) {
344             IntermediateToken bracket;
345             bracket.type = INT_BRACKET;
346             bracket.bracket_val.push_back (result[t_ind]);
347             bracket.bracket_val.push_back (result[t_ind+1]);
348             bracket.bracket_val.push_back (result[t_ind+2]);
349
350             for (int tmp = 0; tmp < 3; tmp++) result.erase
(next(result.begin(), t_ind));
351             result.insert (next(result.begin()), t_ind), bracket);
352             t_ind -= 2;
353         }
354     }
355 }

```

```

356
357     return result;
358 }
359
360 /**
361 * @brief Convert intermediate token vector into usable Reverse Polish
362 Notation token queue
363 * Ensure that everything else is done before calling this function:
364 * epClean
365 * epTokenise
366 * epSimplifyBidmas first_half=true
367 * epSimplifyBidmas first_half=false
368 * These functions ensure the token vector is ready to be linearly parsed
369 into Reverse Polish
370 *
371 * @param intermediate Token vector to convert
372 * @return Vector of proper tokens, ready to use in the expression evaluator
373 */
374 vector<Token> HFractalEquationParser::epReversePolishConvert
375 (vector<IntermediateToken> intermediate) {
376     vector<Token> output;
377
378     IntermediateToken operation = { .op_val = '\0' };
379
380     for (int index = 0; index < intermediate.size(); index++) {
381         IntermediateToken current_intermediate_token = intermediate[index];
382         if (current_intermediate_token.type == INT_OPERATION) {
383             operation.op_val = current_intermediate_token.op_val;
384         } else {
385             // Append to token(s) rp notation
386             if (current_intermediate_token.type == INT_BRACKET) {
387                 vector<Token> inner_result = epReversePolishConvert
388 (current_intermediate_token.bracket_val);
389                 output.insert (output.end(), inner_result.begin(),
390 inner_result.end());
391             } else {
392                 output.push_back ({
393                     .type = (TOKEN_TYPE)current_intermediate_token.type,
394                     .num_val = current_intermediate_token.type == INT_NUMBER
395 ? current_intermediate_token.num_val : 0,
396                     .other_val = current_intermediate_token.type ==
397 INT_LETTER ? current_intermediate_token.let_val : '\0'
398                 });
399     }
400 }
```

```

393         }
394
395         // If relevant, append operation to rp notation
396         if (operation.op_val != '\0') {
397             output.push_back ({
398                 .type = OPERATION,
399                 .other_val = operation.op_val
400             });
401             operation.op_val = '\0';
402         }
403     }
404 }
405
406     return output;
407 }
408
409 /**
410 * @brief Convert a string mathematical expression into an HFractalEquation
411 * class instance using Reverse Polish Notation
412 *
413 * @param sequ String containing a mathematical expression to parse
414 * @return Pointer to an HFractalEquation instance representing the input
415 * string
416 */
417 HFractalEquation* HFractalEquationParser::extractEquation (string sequ) {
418     if (sequ.length() < 1) return NULL;
419     string cleaned = epClean (sequ);
420     EP_CHECK_STATUS check_result = epCheck (cleaned);
421     if (check_result != SUCCESS) {
422         return NULL;
423     }
424
425     vector<IntermediateToken> expression = epTokenise (cleaned);
426
427     expression = epSimplifyBidmas (expression, true);
428     expression = epFixImplicitMul (expression);
429     expression = epSimplifyBidmas (expression, false);
430
431     vector<Token> reverse_polish_expression = epReversePolishConvert
432     (expression);
433
434     return new HFractalEquation (reverse_polish_expression);
435 }
```

```

1 // src/equationparser.hh
2
3 #ifndef EQUATIONPARSER_H
4 #define EQUATIONPARSER_H
5
6 #include <string>
7 #include <vector>
8
9 #include "fractal.hh"
10
11 // Enum describing the token type for the intermediate parser
12 enum INTERMEDIATE_TOKEN_TYPE {
13     INT_NUMBER,
14     INT_LETTER,
15     INT_OPERATION,
16     INT_BRACKET
17 };
18
19 // Struct describing the token for the intermediate parser
20 struct IntermediateToken {
21     INTERMEDIATE_TOKEN_TYPE type;
22     double num_val;
23     char let_val;
24     char op_val;
25     std::vector<IntermediateToken> bracket_val;
26 };
27
28 // Enum describing the error types from the equation processor checking
29 enum EP_CHECK_STATUS {
30     SUCCESS,
31     BRACKET_ERROR,
32     OPERATION_ERROR,
33     IMULT_ERROR,
34     FPOINT_ERROR,
35     UNSUPCHAR_ERROR
36 };
37
38 // Class containing static methods used to parse a string into a postfix
39 // token vector
40 class HFractalEquationParser {
41 private:

```

```

41     static std::string epClean (std::string); // Preprocess the string to
remove whitespace
42     static EP_CHECK_STATUS epCheck (std::string); // Check for formatting
errors in the equation (such as mismatched brackets)
43     static std::vector<IntermediateToken> epTokenise (std::string); // Split
the string into intermediate tokens
44     static std::vector<IntermediateToken> epFixImplicitMul
(std::vector<IntermediateToken>); // Remove implicit multiplication
45     static std::vector<IntermediateToken> epSimplifyBidmas
(std::vector<IntermediateToken>, bool); // Convert BIDMAS rules into explicit
writing
46     static std::vector<Token> epReversePolishConvert
(std::vector<IntermediateToken>); // Convert intermediate tokens into a final
output postfix notation
47
48 public:
49     static HFractalEquation* extractEquation (std::string); // Extract an
equation containing postfix tokens from a string input
50 };
51
52
53 #endif

```

```

1 // src/fractal.cc
2
3 #include "fractal.hh"
4
5 #include <stack>
6 #include <complex>
7
8 #include "utils.hh"
9
10 using namespace std;
11
12 /**
13 * @brief Check if a complex number has tended to infinity. Allows methods
14 which use this check to be implementation independent
15 * Tending to infinity is typically defined as  $|z| > 2$ , which here is
16 expanded to maximise optimisation
17 *
18 */
19 bool HFractalEquation::isInfinity (complex<long double> comp) {
20     return (comp.real()*comp.real()) + (comp.imag()*comp.imag()) > (long
21 double)4;
22 }
23 /**
24 * @brief Set the equation preset value
25 *
26 * @param i Integer representing the preset ID, linked with EQ_PRESETS, or
27 -1 to disable preset mode in this instance
28 */
29 void HFractalEquation::setPreset (int i) {
30     is_preset = (i != -1);
31     preset = i;
32 }
33 /**
34 * @brief Parse the Reverse Polish notation Token vector and evaluate the
35 mathematical expression it represents
36 *
37 * @param z Current value of the z variable to feed in
38 * @param c Current value of the c variable to feed in
39 * @return Complex number with the value of the evaluated equation

```

```

39  /*
40 complex<long double> HFractalEquation::compute (complex<long double> z,
41   complex<long double> c) {
42     stack<complex<long double>> value_stack;
43
44     for (Token t : reverse_polish_vector) {
45       if (t.type == NUMBER) {
46         // Push number arguments onto the stack
47         value_stack.push (t.num_val);
48       } else if (t.type == LETTER) {
49         // Select based on letter and swap in the letter's value, before
50         // pushing it onto the stack
51         switch (t.other_val) {
52           case 'z':
53             value_stack.push (z);
54             break;
55           case 'c':
56             value_stack.push (c);
57             break;
58           case 'a':
59             value_stack.push (c.real());
60             break;
61           case 'b':
62             value_stack.push (c.imag());
63             break;
64           case 'x':
65             value_stack.push (z.real());
66             break;
67           case 'y':
68             value_stack.push (z.imag());
69             break;
70           case 'i':
71             value_stack.push (complex<long double> (0,1));
72             break;
73           default:
74             break;
75         }
76     } else if (t.type == OPERATION) {
77       // Perform an actual computation based on an operation token
78       complex<long double> v2 = value_stack.top(); value_stack.pop();
79       complex<long double> v1 = value_stack.top(); value_stack.pop();
80       switch (t.other_val) {

```

```

79         case '^':
80             value_stack.push (pow (v1, v2));
81             break;
82         case '/':
83             value_stack.push (v1/v2);
84             break;
85         case '*':
86             value_stack.push (v1*v2);
87             break;
88         case '+':
89             value_stack.push (v1+v2);
90             break;
91         case '-':
92             value_stack.push (v1-v2);
93             break;
94     default:
95         break;
96     }
97 }
98 }
99
100 // Return the final value
101 return value_stack.top();
102 }
103
104 /**
105 * @brief Evaluate a complex coordinate (i.e. a pixel) to find the point at
106 * which it tends to infinity, by iteratively applying the equation as a
107 * mathematical function
108 *
109 * @param c Coordinate in the complex plane to initialise with
110 * @param limit Limit for the number of iterations to compute before giving
111 * up, if the number does not tend to infinity
112 * @return Integer representing the number of iterations performed before
113 * the number tended to infinity, or the limit if this was reached first
114 */
115 int HFractalEquation::evaluate (complex<long double> c, int limit) {
116     complex<long double> last = c;
117     if (is_preset && preset == EQ_BURNINGSHIP_MODIFIED) {
118         last = complex<long double> (0, 0);
119     }
120     int depth = 0;

```

```

118     while (depth < limit) {
119         // Switch between custom parsing mode and preset mode for more
120         // efficient computing of presets
121         if (!is_preset) {
122             last = compute (last, c); // Slow custom compute
123         } else {
124             // Much faster hard coded computation
125             switch (preset) {
126                 case EQ_MANDELBROT:
127                     last = (last*last)+c;
128                     break;
129                 case EQ_JULIA_1:
130                     last = (last*last)+complex<long double>(0.285, 0.01);
131                     break;
132                 case EQ_JULIA_2:
133                     last = (last*last)-complex<long double>(0.70176, 0.3842);
134                     break;
135                 case EQ_RECIPROCAL:
136                     last = complex<long double>(1,0)/((last*last)+c);
137                     break;
138                 case EQ_ZPOWER:
139                     last = pow(last,last)+c-complex<long double>(0.5, 0);
140                     break;
141                 case EQ_BARS:
142                     last = pow(last, c*c);
143                     break;
144                 case EQ_BURNINGSHIP_MODIFIED:
145                     last = pow ((complex<long double>(abs(last.real()),0) -
146                     complex<long double>(0, abs(last.imag()))),2)+c;
147                     break;
148                 default:
149                     break;
150             }
151             depth++;
152             // Check if the value has tended to infinity, and escape the loop if
153             so
154             bool b = isInfinity (last);
155             if (b) break;
156         }
157     return depth;
158 }
```

```
158 /**
159  * @brief Initialise with the token sequence in postfix form which this
160  * class should use
161  * @param rp_vec Reverse Polish formatted vector of tokens
162  */
163 HFractalEquation::HFractalEquation (vector<Token> rp_vec) {
164     reverse_polish_vector = rp_vec;
165 }
166
167 /**
168  * @brief Base initialiser. Should only be used to construct presets, as the
169  * equation token vector cannot be assigned after initialisation
170  */
171 HFractalEquation::HFractalEquation () {}
```

```

1 // src/fractal.hh
2
3 #ifndef FRACTAL_H
4 #define FRACTAL_H
5
6 #include <complex>
7 #include <vector>
8
9 // Enum describing the token type
10 enum TOKEN_TYPE {
11     NUMBER,
12     LETTER,
13     OPERATION
14 };
15
16 // Struct describing the token
17 struct Token {
18     TOKEN_TYPE type;
19     double num_val;
20     char other_val;
21 };
22
23 // Class holding the equation and providing functions to evaluate it
24 class HFractalEquation {
25 private:
26     static bool isInfinity (std::complex<long double> comp); // Check if a
complex number has exceeded the 'infinity' threshold
27     std::vector<Token> reverse_polish_vector; // Sequence of equation tokens
in postfix form
28
29     bool is_preset = false; // Records whether this equation is using an
equation preset
30     int preset = -1; // Records the equation preset being used, if none, set
to -1
31
32 public:
33     void setPreset (int); // Set this equation to be a preset, identified
numerically
34
35     std::complex<long double> compute (std::complex<long double>,
std::complex<long double>); // Perform a single calculation using the
equation and the specified z and c values
36     int evaluate (std::complex<long double>, int); // Perform the fractal
calculation

```

```
37
38     HFractalEquation (std::vector<Token>); // Initialise with a sequence of
equation tokens
39     HFractalEquation (); // Base initialiser
40 };
41
42 #endif
```

```

1 // src/gui.cc
2
3 #include "gui.hh"
4 #include "guimain.hh"
5
6 #include <math.h>
7 #include <algorithm>
8 #include <thread>
9
10 #include "utils.hh"
11 #include "database.hh"
12
13 using namespace std;
14
15 /**
16 * @brief Automatically configure the styling for the GUI.
17 * Uses a stylesheet provided by raysan5, creator of the graphics library
18 * used in the project, raylib
19 */
20 void HFractalGui::configureStyling() {
21     // This function implements the 'cyber' interface style provided by
22     // raygui's documentation.
23     const char* stylesheet = R"(p 00 00 0x2f7486ff
24     DEFAULT_BORDER_COLOR_NORMAL
25     p 00 01 0x024658ff    DEFAULT_BASE_COLOR_NORMAL
26     p 00 02 0x51bfd3ff    DEFAULT_TEXT_COLOR_NORMAL
27     p 00 03 0x82cde0ff    DEFAULT_BORDER_COLOR_FOCUSED
28     p 00 04 0x3299b4ff    DEFAULT_BASE_COLOR_FOCUSED
29     p 00 05 0xb6e1eaff    DEFAULT_TEXT_COLOR_FOCUSED
30     p 00 06 0xeb7630ff    DEFAULT_BORDER_COLOR_PRESSED
31     p 00 07 0xffbc51ff    DEFAULT_BASE_COLOR_PRESSED
32     p 00 08 0xd86f36ff    DEFAULT_TEXT_COLOR_PRESSED
33     p 00 09 0x134b5aff    DEFAULT_BORDER_COLOR_DISABLED
34     p 00 10 0x02313dff    DEFAULT_BASE_COLOR_DISABLED
35     p 00 11 0x17505fff    DEFAULT_TEXT_COLOR_DISABLED
36     p 00 16 0x00000012    DEFAULT_TEXT_SIZE
37     p 00 17 0x00000001    DEFAULT_TEXT_SPACING
38     p 00 18 0x81c0d0ff    DEFAULT_LINE_COLOR
39     p 00 19 0x00222bff    DEFAULT_BACKGROUND_COLOR)";
40
41     // Iterate over string and extract styling properties
42     int offset = 0;

```

```

40     int stylePointIndex = 0;
41     string stylePointControl = "";
42     string stylePointProperty = "";
43     string stylePointValue = "";
44     while (offset <= strlen(stylesheets)) {
45         if (stylesheet[offset] == ' ') {
46             stylePointIndex++;
47         } else if (stylesheet[offset] == '\n' || stylesheet[offset] ==
48 '\0') {
49             GuiSetStyle (stoi(stylePointControl), stoi(stylePointProperty),
50 stoll(stylePointValue, nullptr, 16));
51             stylePointControl = "";
52             stylePointProperty = "";
53             stylePointValue = "";
54             stylePointIndex = 0;
55         } else {
56             switch (stylePointIndex) {
57                 case 1:
58                     stylePointControl += stylesheet[offset];
59                     break;
60                 case 2:
61                     stylePointProperty += stylesheet[offset];
62                     break;
63                 case 3:
64                     stylePointValue += stylesheet[offset];
65                     break;
66                 default:
67                     break;
68             }
69         }
70         offset++;
71     }
72 }
73
74 /**
75 * @brief Configure the GUI itself and all class properties
76 *
77 */
78 void HFractalGui::configureGUI(char* path) {
79     // Basic class initialisation

```

```

80     dialog_text = "";
81     console_text = "Ready.";
82     save_name_buffer = "Untitled";
83     for (int i = 0; i < BUTTON_NUM_TOTAL; i++) button_states[i] = false;
84     buffer_image = {};
85     buffer_texture = {};
86     is_rendering = false;
87     is_outdated_render = true;
88     render_percentage = 0;
89     showing_coordinates = false;
90     modal_view_state = MVS_NORMAL;
91     selected_palette = CP_RAINBOW;
92     database_load_dialog_scroll = 0;
93     textbox_focus = TEXT_FOCUS_STATE::TFS_NONE;
94
95     // Fetch configuration
96     unsigned int thread_count = std::thread::hardware_concurrency ();
97     long double start_zoom = 1;
98     long double start_x_offset = 0;
99     long double start_y_offset = 0;
100    image_dimension = WINDOW_INIT_HEIGHT;
101    control_panel_width = WINDOW_INIT_WIDTH - WINDOW_INIT_HEIGHT;
102    equation_buffer = equationPreset (EQ_MANDELBROT, false);
103
104    // GUI configuration
105    SetTraceLogLevel (LOG_WARNING | LOG_ERROR | LOG_DEBUG);
106    InitWindow(WINDOW_INIT_WIDTH, WINDOW_INIT_HEIGHT, "HyperFractal
Mathematical Visualiser");
107    SetWindowState (FLAG_WINDOW_RESIZABLE);
108    SetExitKey(-1);
109    int min_height = std::max (256, CONTROL_MIN_HEIGHT);
110    SetWindowMinSize(min_height+CONTROL_MIN_WIDTH, min_height);
111    SetTargetFPS(24);
112    configureStyling();
113
114    // Initialise database;
115    database = HFractalDatabase
(string(path)+string("FractalSavedStates.csv"));
116
117    // Initialise rendering environment
118    lowres_hm = new HFractalMain();
119    hm = new HFractalMain();

```

```

120
121     // Configure full resolution renderer
122     hm->setResolution (image_dimension);
123     hm->setEquation (equation_buffer);
124     hm->setEvalLimit (200);
125     hm->setWorkerThreads (thread_count);
126     hm->setZoom (start_zoom);
127     hm->setOffsetX (start_x_offset);
128     hm->setOffsetY (start_y_offset);
129
130     // Configure preivew renderer
131     lowres_hm->setResolution (128);
132     lowres_hm->setEquation (equation_buffer);
133     lowres_hm->setEvalLimit (200);
134     lowres_hm->setWorkerThreads (thread_count/2);
135     lowres_hm->setZoom (start_zoom);
136     lowres_hm->setOffsetX (start_x_offset);
137     lowres_hm->setOffsetY (start_y_offset);
138 }
139
140 /**
141 * @brief Called when a rendering parameter has been modified.
142 * Causes the rendered image to become 'out of date' and updates the
143 * preview render
144 */
145 void HFractalGui::parametersWereModified() {
146     is_outdated_render = true;
147     console_text = "Outdated render!";
148     updatePreviewRender();
149 }
150
151 /**
152 * @brief Generate and show an updated image from the preview renderer
153 *
154 * @return True if the update was successful, false if the equation was
155 * invalid
156 */
156 bool HFractalGui::updatePreviewRender() {
157     // Check if the equation is valid
158     if (!lowres_hm->isValidEquation()) return false;
159     lowres_hm->generateImage(true); // If it is, run a render

```

```

160     reloadImageFrom(lowres_hm); // And load it
161     return true;
162 }
163
164 /**
165 * @brief Trigger a full resolution render to start
166 *
167 * @return True if successful, false if the equation was invalid
168 */
169 bool HFractalGui::startFullRender() {
170     if (!hm->isValidEquation()) { // Check if this is a valid equation
171         console_text = "Invalid equation!";
172         return false;
173     }
174     // If it is, start the render
175     is_rendering = true;
176     console_text = "Rendering...";
177     hm->generateImage(false);
178     is_outdated_render = true;
179     render_percentage = 0;
180     return true;
181 }
182
183 /**
184 * @brief Check the status of the full resolution render, and produce an
185 * up-to-date display image showing the render progress.
186 *
187 * @return True if the image has finished rendering, false otherwise
188 */
189 bool HFractalGui::updateFullRender() {
190     if (!is_rendering) return true;
191     // Update completion percentage
192     render_percentage = round(hm->getImageCompletionPercentage());
193     if (hm->getIsRendering()) { // If still rendering, update the rendered
194         image and overlay it onto the preview
195         is_outdated_render = true;
196         Image overlay = getImage(hm);
197         ImageDraw(&buffer_image, overlay, (Rectangle){0,0,(float)hm-
198 >getResolution(),(float)hm->getResolution()}, (Rectangle){0,0,(float)hm-
199 >getResolution(),(float)hm->getResolution()}, WHITE);
200         UnloadImage(overlay);
201         tryUnloadTexture();

```

```

199         buffer_texture = LoadTextureFromImage(buffer_image);
200         return false;
201     } else { // Otherwise, set states to indicate completion and update the
202     image
203         is_outdated_render = false;
204         is_rendering = false;
205         reloadImageFrom (hm);
206         console_text = "Rendering done.";
207         return true;
208     }
209
210 /**
211 * @brief Reload the image and texture used for drawing to the screen from
212 * the specified render environment
213 *
214 * @param h Rendering environment to grab the image from
215 */
216 void HFractalGui::reloadImageFrom(HFractalMain* h) {
217     tryUnloadImage(); // Unload image and texture
218     tryUnloadTexture();
219     buffer_image = getImage(h);
220     if (buffer_image.height != image_dimension) // Resize it to fill the
frame
221         ImageResize(&buffer_image, image_dimension, image_dimension);
222     buffer_texture = LoadTextureFromImage(buffer_image);
223 }
224 /**
225 * @brief Check if the window has been resized, and handle it if so
226 *
227 */
228 void HFractalGui::checkWindowResize() {
229     if (is_rendering) { // If we're mid-render, snap back to previous
window dimensions
230         SetWindowSize(image_dimension+control_panel_width,
image_dimension);
231         return;
232     }
233     if (IsWindowResized()) { // Update render resolution and image
dimension based on new size
234         image_dimension = std::min(GetScreenWidth()-CONTROL_MIN_WIDTH,
GetScreenHeight());
235         control_panel_width = GetScreenWidth()-image_dimension;

```

```

236         hm->setResolution(image_dimension);
237         parametersWereModified(); // Notify that parameters have changed
238     }
239 }
240
241 /**
242 * @brief Draw the entire interface
243 *
244 */
245 void HFractalGui::drawInterface() {
246     BeginDrawing(); // Tell raylib we're about to start drawing
247
248     // Clear the background
249     Color bg_col = GetColor (GuiGetStyle(00, BACKGROUND_COLOR));
250     ClearBackground(bg_col);
251
252     // Draw the rendered HFractalImage
253     Vector2 v {0,0};
254     DrawTextureEx (buffer_texture, v, 0,
255     (float)image_dimension/(float)buffer_texture.height, WHITE);
255     // Draw Console
256     int button_offset = 0;
257     GuiTextBox((Rectangle){(float)image_dimension, BUTTON_HEIGHT*
258     (float)button_offset, (float)control_panel_width, BUTTON_HEIGHT},
259     (char*)console_text.c_str(), 1, false);
260
261     // Draw "Render Image" button
262     button_offset++;
263     button_states[BUTTON_ID::BUTTON_ID_RENDER] = GuiButton((Rectangle)
264     {(float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
265     (float)control_panel_width, BUTTON_HEIGHT}, "Render Image") &&
266     (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
267
268     // Draw render progress bar
269     button_offset++;
270     GuiProgressBar ((Rectangle){(float)image_dimension, BUTTON_HEIGHT*
271     (float)button_offset, (float)control_panel_width, BUTTON_HEIGHT}, "", "", 
272     render_percentage, 0, 100);
273
274     // Draw zoom buttons
275     button_offset++;
276     button_states[BUTTON_ID::BUTTON_ID_ZOOM_IN] = GuiButton((Rectangle)
277     {(float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
278     (float)control_panel_width/3, BUTTON_HEIGHT}, "Zoom In") &&
279     (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
280
281     button_states[BUTTON_ID::BUTTON_ID_ZOOM_RESET] = GuiButton((Rectangle)
282     {(float)image_dimension+(float)control_panel_width/3, BUTTON_HEIGHT*
283     (float)button_offset, (float)control_panel_width/3, BUTTON_HEIGHT}, "Reset
284     Zoom") && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);

```

```

268     button_states[BUTTON_ID::BUTTON_ID_ZOOM_OUT] = GuiButton((Rectangle)
269     {((float)image_dimension+(float)control_panel_width/(3.0f/2.0f),
270     BUTTON_HEIGHT*(float)button_offset, (float)control_panel_width/3,
271     BUTTON_HEIGHT}, "Zoom Out") && (modal_view_state ==
272     MODAL_VIEW_STATE::MVS_NORMAL);
273
274     // Draw save image button
275     button_offset++;
276
277     button_states[BUTTON_ID::BUTTON_ID_SAVE_IMAGE] = GuiButton((Rectangle)
278     {((float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
279     (float)control_panel_width, BUTTON_HEIGHT}, "Save Image") &&
280     (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
281
282     // Draw render state load/save buttons
283     button_offset++;
284
285     button_states[BUTTON_ID::BUTTON_ID_SAVE_RSTATE] = GuiButton((Rectangle)
286     {((float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
287     (float)control_panel_width/2, BUTTON_HEIGHT}, "Save Render State") &&
288     (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
289
290     button_states[BUTTON_ID::BUTTON_ID_LOAD_RSTATE] = GuiButton((Rectangle)
291     {((float)image_dimension+(float)control_panel_width/2, BUTTON_HEIGHT*
292     (float)button_offset, (float)control_panel_width/2, BUTTON_HEIGHT}, "Load
293     Render State") && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
294
295     // Draw movement navigation buttons
296     button_offset++;
297
298     button_states[BUTTON_ID::BUTTON_ID_UP] = GuiButton((Rectangle)
299     {((float)image_dimension+((float)control_panel_width-40)/2, BUTTON_HEIGHT*
300     (float)button_offset, (float)40, BUTTON_HEIGHT}, "up") && (modal_view_state
301     == MODAL_VIEW_STATE::MVS_NORMAL);
302
303     button_offset++;
304
305     button_states[BUTTON_ID::BUTTON_ID_LEFT] = GuiButton((Rectangle)
306     {((float)image_dimension+((float)control_panel_width/2)-40, BUTTON_HEIGHT*
307     (float)button_offset, (float)40, BUTTON_HEIGHT}, "left") &&
308     (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
309
310     button_states[BUTTON_ID::BUTTON_ID_RIGHT] = GuiButton((Rectangle)
311     {((float)image_dimension+((float)control_panel_width/2), BUTTON_HEIGHT*
312     (float)button_offset, (float)40, BUTTON_HEIGHT}, "right") &&
313     (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
314
315     button_offset++;
316
317     button_states[BUTTON_ID::BUTTON_ID_DOWN] = GuiButton((Rectangle)
318     {((float)image_dimension+((float)control_panel_width-40)/2, BUTTON_HEIGHT*
319     (float)button_offset, (float)40, BUTTON_HEIGHT}, "down") &&
320     (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
321
322     button_offset++;
323
324     button_states[BUTTON_ID::BUTTON_ID_EQ_PRESETS] = GuiButton((Rectangle)
325     {((float)image_dimension+(float)control_panel_width/2, BUTTON_HEIGHT*
326     (float)button_offset, (float)control_panel_width/2, BUTTON_HEIGHT},
327     "Equation Presets") && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
328
329
330     // Draw equation input box
331
332     button_states[BUTTON_ID::BUTTON_ID_EQ_INPUTBOX] = GuiTextBox
333     ((Rectangle){((float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
334     (float)control_panel_width/2, BUTTON_HEIGHT}, equation_buffer.data(), 1,
335     false) && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);

```

```

289     button_offset++;
290
291     // Coordinate toggle button
292     string coord_button_text = "Hide coordinates";
293     if (!showing_coordinates) coord_button_text = "Show coordinates";
294     button_states[BUTTON_ID::BUTTON_ID_TOGGLE_COORDS] =
Guibutton((Rectangle){(float)image_dimension, BUTTON_HEIGHT*
(float)button_offset, (float)control_panel_width, BUTTON_HEIGHT},
coord_button_text.c_str()) && (modal_view_state ==
MODAL_VIEW_STATE::MVS_NORMAL);
295     button_offset++;
296
297     // Eval limit controls
298     button_states[BUTTON_ID::BUTTON_ID_EVAL_LIM_LESS] =
Guibutton((Rectangle){(float)image_dimension+(float)control_panel_width/2,
BUTTON_HEIGHT*(float)button_offset, (float)control_panel_width/4,
BUTTON_HEIGHT}, "<") && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
299     button_states[BUTTON_ID::BUTTON_ID_EVAL_LIM_MORE] =
Guibutton((Rectangle){(float)image_dimension+
((float)control_panel_width/4)*3, BUTTON_HEIGHT*(float)button_offset,
(float)control_panel_width/4, BUTTON_HEIGHT}, ">") && (modal_view_state ==
MODAL_VIEW_STATE::MVS_NORMAL);
300     char evalLimString[16];
301     sprintf (evalLimString, "%d (%d)", hm->getEvalLimit(), lowres_hm-
>getEvalLimit());
302     GuiTextBox ((Rectangle){(float)image_dimension, BUTTON_HEIGHT*
(float)button_offset, (float)control_panel_width/2, BUTTON_HEIGHT},
evalLimString, 1, false);
303     button_offset++;
304
305     // Colour palette preset selector button
306     button_states[BUTTON_ID::BUTTON_ID_CP_PRESETS] = Guibutton((Rectangle)
{(float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
(float)control_panel_width, BUTTON_HEIGHT}, "Colour Palettes") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
307
308     // Draw help button
309     button_states[BUTTON_ID::BUTTON_ID_HELP] = Guibutton((Rectangle)
{(float)image_dimension, (float)GetScreenHeight()-(2*BUTTON_HEIGHT),
(float)control_panel_width, BUTTON_HEIGHT*2}, "Help & Instructions") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
310
311     // Draw the equation preset dialog
312     if (modal_view_state == MODAL_VIEW_STATE::MVS_EQUATION_PRESET_SELECTOR)
{
313         float preset_dialog_x = (float)image_dimension+
(float)control_panel_width/2;
314         float preset_dialog_y = BUTTON_HEIGHT*10.0f;
315         for (int e = 0; e < NUM_EQUATION_PRESETS; e++) {

```

```

316         // Draw a button for each option
317         if (
318             GuiButton((Rectangle){preset_dialog_x, preset_dialog_y +
(BUTTON_HEIGHT*e), (float)control_panel_width/2, BUTTON_HEIGHT},
equationPreset((EQ_PRESETS)e, true).c_str()))
319             && !is_rendering
320         ) {
321             escapeEquationPresetDialog(e);
322         }
323     }
324     if (GetMouseX() < preset_dialog_x || GetMouseX() > preset_dialog_x +
(float)control_panel_width/2 || GetMouseY() < preset_dialog_y -
BUTTON_HEIGHT || GetMouseY() > preset_dialog_y +
(BUTTON_HEIGHT*NUM_EQUATION_PRESETS)) {
325         escapeEquationPresetDialog(-1);
326     }
327 }
328
329 // Draw the colour palette preset dialog
330 if (modal_view_state == MODAL_VIEW_STATE::MVS_COLOUR_PRESET_SELECTOR) {
331     float preset_dialog_x = (float)image_dimension;
332     float preset_dialog_y = BUTTON_HEIGHT*13.0f;
333     for (int c = 0; c < NUM_COLOUR_PRESETS; c++) {
334         // Draw a button for each option
335         if (
336             GuiButton((Rectangle){preset_dialog_x, preset_dialog_y +
(BUTTON_HEIGHT*c), (float)control_panel_width, BUTTON_HEIGHT},
colourPalettePreset((CP_PRESETS)c).c_str()))
337             && !is_rendering
338         ) {
339             escapeColourPalettePresetDialog(c);
340         }
341     }
342     if (GetMouseX() < preset_dialog_x || GetMouseX() > preset_dialog_x +
(float)control_panel_width || GetMouseY() < preset_dialog_y -
BUTTON_HEIGHT || GetMouseY() > preset_dialog_y +
(BUTTON_HEIGHT*NUM_COLOUR_PRESETS)) {
343         escapeColourPalettePresetDialog(-1);
344     }
345 }
346
347 // Draw the info dialog
348 if (modal_view_state == MODAL_VIEW_STATE::MVS_TEXT_DIALOG) {
349     float box_width = (2.0/3.0)*GetScreenWidth();
350     DrawRectangle (0, 0, GetScreenWidth(), GetScreenHeight(), (Color)

```

```

{200, 200, 200, 128});
351     Rectangle text_rec = (Rectangle){
352         ((float)GetScreenWidth()-box_width-10)/2,
353         ((float)GetScreenHeight()-DIALOG_TEXT_SIZE-10)/2,
354         box_width+10,
355         DIALOG_TEXT_SIZE
356     };
357     GuiDrawText (dialog_text.c_str(), text_rec, GUI_TEXT_ALIGN_CENTER,
358     BLACK);
358     button_states[BUTTON_ID::BUTTON_ID_TEXT_DIALOG_CLOSE] =
359     GuiButton((Rectangle){(float)(GetScreenWidth()-box_width-10)/2, (float)
360     (GetScreenHeight()*(3.0/4.0)+10), (float)(box_width+10), (float)
361     (DIALOG_TEXT_SIZE+10)}, "OK");
360 }
361
362 // Draw database dialog
363 if (modal_view_state == MODAL_VIEW_STATE::MVS_DATABASE_SAVE_DIALOG ||
364 modal_view_state == MODAL_VIEW_STATE::MVS_DATABASE_LOAD_DIALOG) {
365     DrawRectangle (0, 0, GetScreenWidth(), GetScreenHeight(), (Color)
366 {200, 200, 200, 128});
367     float box_width = (2.0/3.0)*GetScreenWidth();
368     button_states[BUTTON_ID::BUTTON_ID_DATABASE_CANCEL] = GuiButton(
369     (Rectangle){
370         (float)(GetScreenWidth()-box_width-10)/2,
371         (float)(GetScreenHeight()*(4.0/5.0)+10),
372         (float)((box_width+10)/2.0),
373         (float)(DIALOG_TEXT_SIZE+10)
374     },
375     "Cancel");
376
377     // Branch depending on whether the saving dialog or the loading
378     // dialog is open
379     if (modal_view_state == MODAL_VIEW_STATE::MVS_DATABASE_SAVE_DIALOG)
380     {
381         button_states[BUTTON_ID::BUTTON_ID_SAVE] = GuiButton(
382             (Rectangle){
383                 (float)(GetScreenWidth()-10)/2,
384                 (float)(GetScreenHeight()*(4.0/5.0)+10),
385                 (float)((box_width+10)/2.0),
386                 (float)(DIALOG_TEXT_SIZE+10)
387             },
388             "Save");
389
390         button_states[BUTTON_ID::BUTTON_ID_SAVE_NAME_INPUTBOX] =
391         GuiTextBox (

```

```

386         (Rectangle){
387             (float)((GetScreenWidth()-box_width)/2.0),
388             (float)(GetScreenHeight()*(1.0/5.0)),
389             (float)(box_width),
390             (float)(BUTTON_HEIGHT*2)
391         },
392         save_name_buffer.data(), 2, false);
393
394     } else if (modal_view_state ==
395     MODAL_VIEW_STATE::MVS_DATABASE_LOAD_DIALOG) {
396         button_states[BUTTON_ID::BUTTON_ID_LOAD] = GuiButton(
397             (Rectangle){
398                 (float)(GetScreenWidth()-10)/2,
399                 (float)(GetScreenHeight()*(4.0/5.0)+10),
400                 (float)((box_width+10)/2.0),
401                 (float)(DIALOG_TEXT_SIZE+10)
402             },
403             "Load (overwrites current config)");
404
405         button_states[BUTTON_ID::BUTTON_ID_SCROLL_UP] = GuiButton(
406             (Rectangle){
407                 (float)(GetScreenWidth()/2),
408                 (float)(GetScreenHeight()*(1.0/5.0))+9*BUTTON_HEIGHT,
409                 (float)120,
410                 (float)(BUTTON_HEIGHT)
411             },
412             "Scroll up");
413
414         button_states[BUTTON_ID::BUTTON_ID_SCROLL_DOWN] = GuiButton(
415             (Rectangle){
416                 (float)(GetScreenWidth()/2),
417                 (float)(GetScreenHeight()*(1.0/5.0))+10*BUTTON_HEIGHT,
418                 (float)120,
419                 (float)(BUTTON_HEIGHT)
420             },
421             "Scroll down");
422
423         auto descriptions = database.getConfigDescriptions();
424         int row_offset = 0;
425
426         for (auto item : descriptions) {
427             int draw_row = row_offset-database_load_dialog_scroll;

```

```

427         if (draw_row >= 0 && draw_row <= 8) {
428             if (
429                 GuiButton(
430                     (Rectangle){
431                         (float)(GetScreenWidth()-box_width)/2,
432                         (float)(GetScreenHeight()*(1.0/5.0) +
433                             BUTTON_HEIGHT*draw_row),
434                         (float)((box_width)-120),
435                         (float)(BUTTON_HEIGHT)
436                         },
437                         (((item.first == selected_profile_id) ? "(x)" : "( )")
438 + item.second).c_str())
439                     ) {
440                         selected_profile_id = item.first;
441                     }
442                     if (
443                         GuiButton(
444                             (Rectangle){
445                             (float)((GetScreenWidth()+box_width)/2)-120,
446                             (float)(GetScreenHeight()*(1.0/5.0) +
447                             BUTTON_HEIGHT*draw_row),
448                             (float)(120),
449                             (float)(BUTTON_HEIGHT)
450                             },
451                             "Delete? (!)")
452                         ) {
453                             database.removeConfig(item.first);
454                             database.commit();
455                         }
456                     }
457                 }
458
459 // Draw coordinates text next to cursor
460 if (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL &&
461 showing_coordinates && GetMouseX() <= image_dimension && GetMouseY() <=
462 image_dimension) {
463     float left = GetMouseX()+15;
464     float top = GetMouseY()+15;
465     Color col {250, 250, 250, 200};
466

```

```

465     long double location_x = hm->getOffsetX() + ((long double)((long
466     double)GetMouseX()/(image_dimension/2))-1))/hm->getZoom();
467
468     long double location_y = hm->getOffsetY() - ((long double)((long
469     double)GetMouseY()/(image_dimension/2))-1))/hm->getZoom();
470
471     char t[142];
472
473     sprintf (t, "%.10Lf\n%.10Lf", location_x, location_y);
474     DrawRectangle (left, top, 115, 40, col);
475     DrawText (t, left+5, top, 15, BLACK);
476 }
477
478 EndDrawing(); // Tell raylib we're done drawing
479 }
480
481 /**
482 * @brief Close the equation preset dialog, and switch to a given preset
483 *
484 * @param e The equation preset to switch to, or -1 if the dialog was
485 * cancelled
486 */
487 void HFractalGui::escapeEquationPresetDialog(int e) {
488
489     modal_view_state = MODAL_VIEW_STATE::MVS_NORMAL; // Switch back to
normal mode
490
491     if (is_rendering) return;
492
493     if (e != -1) { // If an option was selected, make it the current
equation and notify that parameters have changed
494         equation_buffer = equationPreset ((EQ_PRESETS)e, false);
495
496         hm->setEquation (equation_buffer);
497         lowres_hm->setEquation (equation_buffer);
498
499         // Check whether the equation is valid
500         if (!hm->isValidEquation()) console_text = "Invalid equation
input";
501
502         else {
503             parametersWereModified();
504         }
505     }
506 }
507
508 /**
509 * @brief Show the equation preset dialog
510 *
511 */
512 void HFractalGui::enterEquationPresetDialog() {
513
514     if (is_rendering) return;
515
516     // Switch to equation preset selector mode

```

```

503     modal_view_state = MODAL_VIEW_STATE::MVS_EQUATION_PRESET_SELECTOR;
504 }
505
506 /**
507 * @brief Show the colour palette preset dialog
508 *
509 */
510 void HFractalGui::enterColourPalettePresetDialog() {
511     if (is_rendering) return;
512     // Switch to colour preset selector mode
513     modal_view_state = MODAL_VIEW_STATE::MVS_COLOUR_PRESET_SELECTOR;
514 }
515
516 /**
517 * @brief Close the colour palette preset dialog and switch to a given
518 * palette
519 *
520 * @param c Palette to switch to, or -1 if the dialog was cancelled
521 */
522 void HFractalGui::escapeColourPalettePresetDialog(int c) {
523     modal_view_state = MODAL_VIEW_STATE::MVS_NORMAL; // Return to normal
524     GUI mode
525
526     if (is_rendering) return;
527     if (c != -1) {
528         // If an option was selected, reload the image with the selected
529         palette (no rerender necessary)
530         selected_palette = (CP_PRESETS)c;
531         if (is_outdated_render) {
532             reloadImageFrom(lowres_hm);
533         } else {
534             reloadImageFrom(hm);
535         }
536     }
537 }
538
539 /**
540 * @brief Get an image handleable by raylib from a given rendering
541 * environment
542 *
543 * @param h Rendering environment to extract from
544 * @return A raylib-style image for drawing into the GUI
545 */
546 Image HFractalGui::getImage(HFractalMain* h) {

```

```

542 // Fetch a 32 bit RGBA image in the selected colour palette
543 int size = h->getResolution();
544 uint32_t *data = h->getRGBAIImage(selected_palette);
545 Color *pixels = (Color *)malloc (size*size*sizeof(Color));
546 // Convert the image data to a format raylib will accept
547 for (int i = 0; i < size*size; i++) pixels[i] = GetColor(data[i]);
548 delete data;
549 // Construct a raylib image from the data
550 Image img = {
551     .data = pixels,
552     .width = size,
553     .height = size,
554     .mipmaps = 1,
555     .format = PIXELFORMAT_UNCOMPRESSED_R8G8B8A8
556 };
557 return img;
558 }
559
560 /**
561 * @brief Handle when the user clicks on the image.
562 * Automatically centres the area they clicked and notifies the GUI that
563 * rendering parameters have been modified, triggering a preview update
564 *
565 * @return True if a click was handled, otherwise false
566 */
567 bool HFractalGui::handleClickNavigation() {
568     if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON) && !is_rendering) {
569         Vector2 mpos = GetMousePosition();
570         // Check if the mouse click was inside the image
571         if (mpos.x <= image_dimension && mpos.y <= image_dimension) {
572             long double change_in_x = (long double)((mpos.x /
573             (image_dimension / 2)) - 1) / hm->getZoom();
574             long double change_in_y = (long double)((mpos.y /
575             (image_dimension / 2)) - 1) / hm->getZoom();
576             long double new_offset_x = hm->getOffsetX() + change_in_x;
577             long double new_offset_y = hm->getOffsetY() - change_in_y;
578             // Update parameters and notify of the modification
579             lowres_hm->setOffsetX(new_offset_x);
580             lowres_hm->setOffsetY(new_offset_y);
581             hm->setOffsetX(new_offset_x);
582             hm->setOffsetY(new_offset_y);
583             parametersWereModified();
584             return true;

```

```

582         }
583     }
584     return false;
585 }
586
587 /**
588 * @brief Show a text dialog with a given string as text
589 *
590 * @param text Text to display
591 */
592 void HFractalGui::launchTextDialog(std::string text) {
593     modal_view_state = MODAL_VIEW_STATE::MVS_TEXT_DIALOG;
594     dialog_text = text;
595 }
596
597 /**
598 * @brief Close the currently open text dialog and go back to normal GUI
599 mode
600 *
601 */
602 void HFractalGui::closeTextDialog() {
603     modal_view_state = MODAL_VIEW_STATE::MVS_NORMAL;
604     dialog_text = "";
605 }
606
607 /**
608 * @brief Handler for Zoom In button
609 *
610 */
611 void HFractalGui::zoomIn() {
612     if (hm->getZoom() <= SCALE_DEPTH_LIMIT) { // Check the zoom has not
613     exceeded the depth limit
614         long double new_zoom = hm->getZoom() * SCALE_STEP_FACTOR;
615         lowres_hm->setZoom (new_zoom);
616         hm->setZoom (new_zoom);
617         parametersWereModified();
618     } else launchTextDialog ("Zoom precision limit reached"); // Present a
619     text dialog to report the issue to the user
620 }
621 */

```

```

622 */
623 void HFractalGui::zoomOut() {
624     long double new_zoom = hm->getZoom() / SCALE_STEP_FACTOR;
625     lowres_hm->setZoom (new_zoom);
626     hm->setZoom (new_zoom);
627     parametersWereModified();
628 }
629
630 /**
631 * @brief Handler for Reset Zoom button
632 *
633 */
634 void HFractalGui::resetZoom() {
635     lowres_hm->setZoom(1);
636     hm->setZoom(1);
637     parametersWereModified();
638 }
639
640 /**
641 * @brief Handler for Save Image button
642 *
643 */
644 void HFractalGui::saveImage() {
645     bool result = false;
646     // Switch depending on whether there is a full render available to save
647     if (is_outdated_render) {
648         result = lowres_hm->autoWriteImage(IMAGE_TYPE::PGM);
649         console_text = "Saved preview render to desktop.";
650     } else {
651         result = hm->autoWriteImage (IMAGE_TYPE::PGM);
652         console_text = "Saved render to desktop.";
653     }
654     if (!result) {
655         console_text = "Image saving failed.";
656     }
657 }
658
659 /**
660 * @brief Make the save render state dialog visible
661 *
662 */

```

```

663 void HFractalGui::showSaveStateDialog() {
664     if (is_rendering) return;
665     modal_view_state = MODAL_VIEW_STATE::MVS_DATABASE_SAVE_DIALOG;
666 }
667 /**
668 * @brief Make the load render state dialog visible
669 *
670 */
671 */
672 void HFractalGui::showLoadStateDialog() {
673     if (is_rendering) return;
674     modal_view_state = MODAL_VIEW_STATE::MVS_DATABASE_LOAD_DIALOG;
675     database_load_dialog_scroll = 0;
676 }
677 /**
678 * @brief Save the current render state to the database and close the
679 dialog
680 *
681 */
682 void HFractalGui::saveStateToDatabase() {
683     // Create the new config profile and populate its fields
684     HFractalConfigProfile *cp = new HFractalConfigProfile();
685     cp->equation = hm->getEquation();
686     cp->iterations = hm->getEvalLimit();
687     cp->name = save_name_buffer;
688     cp->palette = selected_palette;
689     cp->x_offset = hm->getOffsetX();
690     cp->y_offset = hm->getOffsetY();
691     cp->zoom = hm->getZoom();
692
693     // Fetch or create the default user if necessary
694     HFractalUserProfile *default_user = database.getUser(0);
695     if (default_user == NULL) {
696         default_user = new HFractalUserProfile();
697         default_user->user_name = "default";
698         database.insertUser (default_user);
699     }
700
701     cp->user_id = default_user->user_id;
702
703     // Insert the new profile into the database

```

```

704     database.insertConfig(cp);
705     database.commit();
706     console_text = "Profile saved to database!";
707     closeDatabaseDialog(); // Escape from the dialog
708 }
709
710 /**
711 * @brief Load the selected render state from the database and close the
712 * dialog
713 */
714 void HFractalGui::loadStateFromDatabase() {
715     // Try to fetch the config profile
716     HFractalConfigProfile *cp = database.getConfig (selected_profile_id);
717     if (cp == NULL) {
718         console_text = "No profile selected to load.";
719     } else { // On success, load all properties into the rendering
720         environments
721         hm->setEquation(cp->equation);
722         lowres_hm->setEquation(cp->equation);
723
724         hm->setEvalLimit(cp->iterations);
725         lowres_hm->setEvalLimit(cp->iterations);
726
727         hm->setOffsetX(cp->x_offset);
728         lowres_hm->setOffsetX(cp->x_offset);
729
730         hm->setOffsetY(cp->y_offset);
731         lowres_hm->setOffsetY(cp->y_offset);
732
733         hm->setZoom(cp->zoom);
734         lowres_hm->setZoom(cp->zoom);
735
736         selected_palette = (CP_PRESETS)cp->palette;
737         save_name_buffer = cp->name;
738
739         updatePreviewRender();
740         console_text = "Profile '" + save_name_buffer + "' loaded from
741 database.";
742     }
743     closeDatabaseDialog(); // Close the dialog
744 }
745

```

```

744 /**
745 * @brief Hide the database dialog and return to normal GUI mode
746 *
747 */
748 void HFractalGui::closeDatabaseDialog() {
749     modal_view_state = MODAL_VIEW_STATE::MVS_NORMAL;
750 }
751
752 /**
753 * @brief Scroll down inside the load render state dialog
754 *
755 */
756 void HFractalGui::databaseLoadScrollDown() {
757     database_load_dialog_scroll++;
758 }
759
760 /**
761 * @brief Scroll up inside the load render state dialog
762 *
763 */
764 void HFractalGui::databaseLoadScrollUp() {
765     database_load_dialog_scroll--;
766     if (database_load_dialog_scroll < 0) database_load_dialog_scroll = 0;
767 }
768
769 /**
770 * @brief Handler for Move Up button
771 *
772 */
773 void HFractalGui::moveUp() {
774     long double new_offset = hm->getOffsetY() + (MOVE_STEP_FACTOR/hm-
>getZoom());
775     hm->setOffsetY (new_offset);
776     lowres_hm->setOffsetY (new_offset);
777     parametersWereModified();
778 }
779
780 /**
781 * @brief Handler for Move Left button
782 *
783 */
784 void HFractalGui::moveLeft() {

```

```

785     long double new_offset = hm->getOffsetX() - (MOVE_STEP_FACTOR/hm-
786     >getZoom());
787     hm->setOffsetX (new_offset);
788     lowres_hm->setOffsetX (new_offset);
789     parametersWereModified();
790 }
791 /**
792 * @brief Handler for Move Right button
793 *
794 */
795 void HFractalGui::moveRight() {
796     long double new_offset = hm->getOffsetX() + (MOVE_STEP_FACTOR/hm-
797     >getZoom());
798     hm->setOffsetX (new_offset);
799     lowres_hm->setOffsetX (new_offset);
800     parametersWereModified();
801 }
802 /**
803 * @brief Handler for Move Down button
804 *
805 */
806 void HFractalGui::moveDown() {
807     long double new_offset = hm->getOffsetY() - (MOVE_STEP_FACTOR/hm-
808     >getZoom());
809     hm->setOffsetY (new_offset);
810     lowres_hm->setOffsetY (new_offset);
811     parametersWereModified();
812 }
813 /**
814 * @brief Handler for Show/Hide Coordinates button
815 *
816 */
817 void HFractalGui::toggleCoords() {
818     showing_coordinates = !showing_coordinates;
819 }
820
821 /**
822 * @brief Handlder for '<' button
823 *
824 */

```

```

825 void HFractalGui::evalLimitLess() {
826     int new_el = hm->getEvalLimit();
827     // Allow faster jumping if shift is held
828     if (IsKeyDown(KEY_LEFT_SHIFT) || IsKeyDown (KEY_RIGHT_SHIFT)) {
829         new_el -= 10;
830     } else {
831         new_el--;
832     }
833     hm->setEvalLimit (new_el);
834     lowres_hm->setEvalLimit (new_el);
835     parametersWereModified();
836 }
837
838 /**
839 * @brief Handler for '>' button
840 *
841 */
842 void HFractalGui::evalLimitMore() {
843     int new_el = hm->getEvalLimit();
844     // Allow faster jumping if shift is held
845     if (IsKeyDown(KEY_LEFT_SHIFT) || IsKeyDown (KEY_RIGHT_SHIFT)) {
846         new_el += 10;
847     } else {
848         new_el++;
849     }
850     hm->setEvalLimit (new_el);
851     lowres_hm->setEvalLimit (new_el);
852     parametersWereModified();
853 }
854
855 /**
856 * @brief Handler for Help & Instructions button
857 *
858 */
859 void HFractalGui::showHelp() {
860     // Open the help page in the repository, cross-platform
861     #ifdef _WIN32
862         system("explorer
https://github.com/JkyProgrammer/HyperFractal/blob/main/README.md#help--instructions");
863     #else
864         system("open
https://github.com/JkyProgrammer/HyperFractal/blob/main/README.md#help--instructions");

```

```

    instructions");
865     #endif
866 }
867
868 /**
869 * @brief Handle the user pressing a GUI button
870 *
871 * @return True if a button press was handled, otherwise false
872 */
873 bool HFractalGui::handleButtonPresses() {
874     if (is_rendering) return false;
875     // Branch to different handling modes depending on the dialog state,
876     // allowing certain sets of buttons to be disabled when dialogs are open
877     if (modal_view_state == MODAL_VIEW_STATE::MVS_TEXT_DIALOG) {
878         if (button_states[BUTTON_ID::BUTTON_ID_TEXT_DIALOG_CLOSE]) {
879             closeTextDialog(); return true; }
880         } else if (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL) {
881             if (button_states[BUTTON_ID::BUTTON_ID_RENDER]) {
882                 startFullRender(); return true; }
883             if (button_states[BUTTON_ID::BUTTON_ID_ZOOM_IN]) { zoomIn(); return
884                 true; }
885             if (button_states[BUTTON_ID::BUTTON_ID_ZOOM_OUT]) { zoomOut();
886                 return true; }
887             if (button_states[BUTTON_ID::BUTTON_ID_SAVE_IMAGE]) { saveImage();
888                 return true; }
889             if (button_states[BUTTON_ID::BUTTON_ID_SAVE_RSTATE]) {
890                 showSaveStateDialog(); return true; }
891             if (button_states[BUTTON_ID::BUTTON_ID_LOAD_RSTATE]) {
892                 showLoadStateDialog(); return true; }
893             if (button_states[BUTTON_ID::BUTTON_ID_UP]) { moveUp(); return
894                 true; }
895             if (button_states[BUTTON_ID::BUTTON_ID_LEFT]) { moveLeft(); return
896                 true; }
897             if (button_states[BUTTON_ID::BUTTON_ID_RIGHT]) { moveRight();
898                 return true; }
899             if (button_states[BUTTON_ID::BUTTON_ID_DOWN]) { moveDown(); return
900                 true; }
901             if (button_states[BUTTON_ID::BUTTON_ID_EQ_PRESETS]) {
902                 enterEquationPresetDialog(); return true; }
903             if (button_states[BUTTON_ID::BUTTON_ID_ZOOM_RESET]) { resetZoom();
904                 return true; }
905             if (button_states[BUTTON_ID::BUTTON_ID_TOGGLE_COORDS]) {
906                 toggleCoords(); return true; }
907             if (button_states[BUTTON_ID::BUTTON_ID_EVAL_LIM_LESS]) {
908                 evalLimitLess(); return true; }
909             if (button_states[BUTTON_ID::BUTTON_ID_EVAL_LIM_MORE]) {
910                 evalLimitMore(); return true; }

```

```

894     if (button_states[BUTTON_ID::BUTTON_ID_HELP]) { showHelp(); return true; }
895
896     if (button_states[BUTTON_ID::BUTTON_ID_EQ_INPUTBOX]) {
897         textbox_focus = TEXT_FOCUS_STATE::TFS_EQUATION; return true; }
898
899     if (button_states[BUTTON_ID::BUTTON_ID_CP_PRESETS]) {
900         enterColourPalettePresetDialog(); return true; }
901
902     } else if (modal_view_state ==
903                 MODAL_VIEW_STATE::MVS_DATABASE_SAVE_DIALOG) {
904
905         if (button_states[BUTTON_ID::BUTTON_ID_SAVE_NAME_INPUTBOX]) {
906             textbox_focus = TEXT_FOCUS_STATE::TFS_SAVE_NAME; return true; }
907
908         if (button_states[BUTTON_ID::BUTTON_ID_SAVE]) {
909             saveStateToDatabase(); return true; }
910
911         if (button_states[BUTTON_ID::BUTTON_ID_DATABASE_CANCEL]) {
912             closeDatabaseDialog(); return true; }
913
914     } else if (modal_view_state ==
915                 MODAL_VIEW_STATE::MVS_DATABASE_LOAD_DIALOG) {
916
917         if (button_states[BUTTON_ID::BUTTON_ID_LOAD]) {
918             loadStateFromDatabase(); return true; }
919
920     } else if (button_states[BUTTON_ID::BUTTON_ID_SCROLL_DOWN]) {
921         databaseLoadScrollDown(); return true; }
922
923     if (button_states[BUTTON_ID::BUTTON_ID_SCROLL_UP]) {
924         databaseLoadScrollUp(); return true; }
925
926     if (button_states[BUTTON_ID::BUTTON_ID_DATABASE_CANCEL]) {
927             closeDatabaseDialog(); return true; }
928
929     }
930
931     return false;
932 }
933
934 /**
935 * @brief Clear the contents of the button states array to prevent
936 unhandled button presses hanging over to the next update
937 *
938 */
939
940 void HFractalGui::clearButtonStates() {
941     for (int i = 0; i < BUTTON_NUM_TOTAL; i++) {
942         button_states[i] = false;
943     }
944 }
945
946 /**
947 * @brief Unload the image buffer to prevent memory leaks
948 *
949 */
950
951 void HFractalGui::tryUnloadImage() {
952     UnloadImage (buffer_image);

```

```

927 }
928
929 /**
930 * @brief Unload the texture buffer to prevent memory leaks
931 *
932 */
933 void HFractalGui::tryUnloadTexture() {
934     UnloadTexture (buffer_texture);
935 }
936
937 /**
938 * @brief Handle when the user presses a key
939 *
940 * @return True if a key press was handled, false otherwise
941 */
942 bool HFractalGui::handleKeyPresses() {
943     // Escape currently editing text box when escape is pressed
944     if (IsKeyDown(KEY_ESCAPE)) { textbox_focus =
TEXT_FOCUS_STATE::TFS_NONE; return true; }
945
946     // Handle keys depending on which text box is focussed (if none, use
them for navigation)
947     if (textbox_focus == TEXT_FOCUS_STATE::TFS_NONE) {
948         for (auto key : key_map) {
949             if (IsKeyDown (key.first)) {
950                 button_states[key.second] = true;
951                 return true;
952             }
953         }
954     } else if (textbox_focus == TEXT_FOCUS_STATE::TFS_EQUATION) {
955         if (IsKeyDown(KEY_ENTER)) {
button_states[BUTTON_ID::BUTTON_ID_RENDER] = true; return true; }
956         int key = GetCharPressed();
957         if (((int)'a' <= key && key <= (int)'c') || ((int)'x' <= key &&
key <= (int)'z') || key == 122 || (key >= 48 && key <= 57) || key == 94 ||
(key >= 40 && key <= 43) || key == 45 || key == 46 || key == 47 || key ==
'i') && !is_rendering) {
958             equation_buffer += (char)key;
959             hm->setEquation (equation_buffer);
960             lowres_hm->setEquation (equation_buffer);
961             if (!hm->isValidEquation()) console_text = "Invalid equation
input";
962             else parametersWereModified();
963         } else if (GetKeyPressed () == KEY_BACKSPACE && !is_rendering &&

```

```

equation_buffer.length() > 0) {
964         equation_buffer.pop_back();
965         hm->setEquation(equation_buffer);
966         lowres_hm->setEquation(equation_buffer);
967         if (!hm->isValidEquation()) console_text = "Invalid equation
input";
968         else parametersWereModified();
969     }
970     else if (textbox_focus == TEXT_FOCUS_STATE::TFS_SAVE_NAME) {
971         int key = GetCharPressed();
972         if (((int)'a' <= key && key <= (int)'z') || ((int)'A' <= key && key
<= (int)'Z')) {
973             save_name_buffer += (char)key;
974         } else if (GetKeyPressed() == KEY_BACKSPACE) {
975             if (save_name_buffer.length() > 0) save_name_buffer.pop_back();
976         }
977     }
978     return false;
979 }
980
981 /**
982 * @brief Start the GUI and run the mainloop.
983 * Blocks on current thread
984 *
985 * @return Integer showing exit status
986 */
987 int HFractalGui::guiMain(char* path) {
988     // Run the setup code
989     configureGUI(path);
990     parametersWereModified();
991     while(!WindowShouldClose()) { // Loop until the application closes
992         checkWindowResize();
993         if (!is_rendering && modal_view_state == MVS_NORMAL) {
994             bool click_handled = handleClickNavigation();
995             // Defocus the textbox if a click is handled somewhere
996             if (click_handled) { textbox_focus =
TEXT_FOCUS_STATE::TFS_NONE; }
997         }
998         handleKeyPresses();
999         bool button_pressed = handleButtonPresses();
1000        // Defocus the textbox if a button press is handled
1001        if (button_pressed &&
!button_states[BUTTON_ID::BUTTON_ID_EQ_INPUTBOX] &&

```

```

1001     !button_states[BUTTON_ID::BUTTON_ID_SAVE_NAME_INPUTBOX]) { textbox_focus =
1002         TEXT_FOCUS_STATE::TFS_NONE; }
1003         clearButtonStates();
1004         // If a render is in progress, update the status of it
1005         if (is_rendering) updateFullRender();
1006         // Finally, draw everything
1007         drawInterface();
1008     }
1009     // Release resources and close
1010     tryUnloadImage();
1011     tryUnloadTexture();
1012     CloseWindow();
1013     return 0;
1014 }
1015
1016 /**
1017 * @brief Construct a new GUI object
1018 *
1019 */
1020 HFractalGui::HFractalGui() {}
1021
1022 /**
1023 * @brief Method to start the GUI, isolates the GUI module from the main
1024 * module to prevent linker conflicts with raylib
1025 *
1026 */
1027 int guiMain (char* path) {
1028     HFractalGui gui = HFractalGui ();
1029     int res = gui.guiMain(path);
1030     return res;
1031 }

```

```

1 // src/gui.hh
2
3 #ifndef GUI_H
4 #define GUI_H
5
6 #include <map>
7
8 #define RAYGUI_IMPLEMENTATION
9 #define RAYGUI_SUPPORT_ICONS
10 #include "../lib/raygui.h"
11 #include "../lib/ricons.h"
12
13 #include "hyperfractal.hh"
14 #include "utils.hh"
15 #include "database.hh"
16
17 #define SCALE_STEP_FACTOR 1.5           // Factor by which scaling changes
18 #define SCALE_DEPTH_LIMIT 1.0e15       // Limit to prevent user from going too
                                         deep due to limited precision
19 #define MOVE_STEP_FACTOR 0.1           // Factor by which position changes
20 #define WINDOW_INIT_WIDTH 900          // Initial window - width
21 #define WINDOW_INIT_HEIGHT 550         //           - height
22 #define BUTTON_HEIGHT 30              // Height of a single button in the
                                         interface
23 #define ELEMENT_NUM_VERTICAL 15        // Number of vertical elements
24 #define BUTTON_NUM_TOTAL 25           // Total number of buttons in the
                                         interface
25 #define CONTROL_MIN_WIDTH 400          // Minimum width of the control panel
26 #define CONTROL_MIN_HEIGHT BUTTON_HEIGHT*ELEMENT_NUM_VERTICAL // Minimum
                                         height of the panel
27 #define DIALOG_TEXT_SIZE 25            // Size of text in dialog windows
28
29 // Enum listing button IDs to abstract and make code clearer
30 enum BUTTON_ID {
31     BUTTON_ID_RENDER = 0,
32     BUTTON_ID_ZOOM_IN,
33     BUTTON_ID_ZOOM_OUT,
34     BUTTON_ID_SAVE_IMAGE,
35     BUTTON_ID_SAVE_RSTATE,
36     BUTTON_ID_LOAD_RSTATE,
37     BUTTON_ID_UP,
38     BUTTON_ID_LEFT,
39     BUTTON_ID_RIGHT,

```

```

40     BUTTON_ID_DOWN,
41     BUTTON_ID_EQ_PRESETS,
42     BUTTON_ID_ZOOM_RESET,
43     BUTTON_ID_TOGGLE_COORDS,
44     BUTTON_ID_EVAL_LIM_LESS,
45     BUTTON_ID_EVAL_LIM_MORE,
46     BUTTON_ID_HELP,
47     BUTTON_ID_TEXT_DIALOG_CLOSE,
48     BUTTON_ID_EQ_INPUTBOX,
49     BUTTON_ID_CP_PRESETS,
50     BUTTON_ID_SAVE_NAME_INPUTBOX,
51     BUTTON_ID_SAVE,
52     BUTTON_ID_LOAD,
53     BUTTON_ID_SCROLL_DOWN,
54     BUTTON_ID_SCROLL_UP,
55     BUTTON_ID_DATABASE_CANCEL
56 };
57
58 // Enum listing GUI states for cases when a dialog or modal is open (i.e. to
59 // disable certain interface elements)
60 enum MODAL_VIEW_STATE {
61     MVS_NORMAL,
62     MVS_TEXT_DIALOG,
63     MVS_DATABASE_SAVE_DIALOG,
64     MVS_DATABASE_LOAD_DIALOG,
65     MVS_EQUATION_PRESET_SELECTOR,
66     MVS_COLOUR_PRESET_SELECTOR
67 };
68
69 // Enum listing text focus states to enable/disable input to specific fields
70 enum TEXT_FOCUS_STATE {
71     TFS_NONE,
72     TFS_EQUATION,
73     TFS_SAVE_NAME
74 };
75
76 // Class managing the GUI environment
77 class HFractalGui {
78     private:
79         std::map<KeyboardKey, BUTTON_ID> key_map = {
80             {KEY_ENTER, BUTTON_ID::BUTTON_ID_RENDER},

```

```

81         {KEY_EQUAL, BUTTON_ID::BUTTON_ID_ZOOM_IN},
82         {KEY_MINUS, BUTTON_ID::BUTTON_ID_ZOOM_OUT},
83         {KEY_UP, BUTTON_ID::BUTTON_ID_UP},
84         {KEY_DOWN, BUTTON_ID::BUTTON_ID_DOWN},
85         {KEY_LEFT, BUTTON_ID::BUTTON_ID_LEFT},
86         {KEY_RIGHT, BUTTON_ID::BUTTON_ID_RIGHT},
87         {KEY_LEFT_BRACKET, BUTTON_ID::BUTTON_ID_EVAL_LIM_LESS},
88         {KEY_RIGHT_BRACKET, BUTTON_ID::BUTTON_ID_EVAL_LIM_MORE}
89     };
90
91     HFRACTALMAIN* hm; // Pointer to main rendering environment
92     HFRACTALMAIN* lowres_hm; // Pointer to an identical rendering
environment, but with a lower resolution for preview renders
93
94     std::string dialog_text; // Text to show in the text dialog widget
95     std::string console_text; // Text to show in the application console
96     std::string equation_buffer; // Contains the equation being used by both
renderer classes
97     std::string save_name_buffer; // Contains the text shown/edited in the
name field in the save render state dialog
98     bool button_states[BUTTON_NUM_TOTAL]; // Contains the current states of
every button in the GUI (true for pressed, false for not pressed)
99     Image buffer_image; // Image being used by raygui for displaying the
render result
100    Texture2D buffer_texture; // Texture being used by raygui for displaying
the render result
101    bool is_rendering; // Stores whether the GUI is currently waiting on a
full-resolution render (and thus should freeze controls)
102    bool is_outdated_render; // Stores whether the GUI is showing a preview
render (i.e. needs a full-resolution render to be run by the user)
103    TEXT_FOCUS_STATE textbox_focus; // Stores the currently focussed text
box
104    int render_percentage; // Stores the percentage completion of the
current render
105    bool showing_coordinates; // Stores whether coordinates are currently
being shown on the mouse cursor
106    MODAL_VIEW_STATE modal_view_state; // Stores the current modal state of
the GUI, allowing certain controls to be enabled and disabled in different
modes
107    int image_dimension; // Stores the size of the image, used for sizing
the window, scaling and rendering images, and positioning elements
108    int control_panel_width; // Stores the width of the control panel
109    CP_PRESETS selected_palette; // Determines the colour palette in which
the GUI is currently displaying the rendered image
110    HFRACTALDATABASE database; // Database which manages saved profile
states
111    long selected_profile_id; // Records the ID of the profile currently

```

```

selected in the load render state dialog
112     int database_load_dialog_scroll; // Records the current amount of scroll
in the load render state dialog
113
114     void configureStyling(); // Configures the GUI styling from a stylesheet
provided by raylib's creator as part of the library
115     void configureGUI(char*); // Configures the GUI and initialises all
class variables ready for the first GUI mainloop update
116
117     void parametersWereModified(); // Marks the GUI as using an outdated
render and triggers a preview render update
118     bool updatePreviewRender(); // Rerenders the preview image
119     bool startFullRender(); // Triggers a full resolution render
120     bool updateFullRender(); // Updates the image and texture buffers from
the partially-finished rendering environment image, and finalises if the
render has completed
121     void reloadImageFrom(HFractalMain*); // Automatically fetch and reload
the image and texture buffers from a given rendering environment
122
123     void checkWindowResize(); // Check to see if the window has been
resized, and handle it
124
125     bool handleClickNavigation(); // Check to see if the user has clicked
somewhere on the image, and jump to focus that location if so
126     bool handleButtonPresses(); // Handle any interface button presses the
user has made since the last update
127     bool handleKeyPresses(); // Handle any keyboard key presses the user has
made since the last update
128     void drawInterface(); // Draw the entire interface, called each update
129
130     Image getImage(HFractalMain*); // Extract image data from a rendering
environment
131
132     void enterEquationPresetDialog(); // Show the equation preset selector
and disable other GUI controls
133     void escapeEquationPresetDialog(int); // Close the equation preset
selector and return to normal GUI mode
134     void enterColourPalettePresetDialog(); // Show the colour palette preset
selector and disable other GUI controls
135     void escapeColourPalettePresetDialog(int); // Close the colour palette
preset selector and return to normal GUI mode
136     void launchTextDialog(std::string); // Show a text dialog over the
window with a given string as text
137     void closeTextDialog(); // Close the text dialog currently being shown
138
139     void zoomIn(); // Handler for Zoom In button
140     void zoomOut(); // Handler for Zoom Out button

```

```

141     void resetZoom(); // Handler for Reset Zoom button
142     void saveImage(); // Handler for Save Image button
143
144     void moveUp(); // Handler for Move Up button
145     void moveLeft(); // Handler for Move Left button
146     void moveRight(); // Handler for Move Right button
147     void moveDown(); // Handler for Move Down button
148
149     void toggleCoords(); // Handler for Show/Hide Coordinates button
150
151     void evalLimitLess(); // Handler for '<' button
152     void evalLimitMore(); // Handler for '>' button
153
154     void showHelp(); // Handler for Help & Instructions button
155     void clearButtonStates(); // Clears current button states to ignore
unhandled button presses
156
157     void tryUnloadImage(); // Unload the image buffer, prevents memory leaks
158     void tryUnloadTexture(); // Unload the texture buffer, prevents memory
leaks
159
160     void showSaveStateDialog(); // Make the save render state dialog visible
161     void showLoadStateDialog(); // Make the load render state dialog visible
162     void saveStateToDatabase(); // Save the current render state to the
database
163     void loadStateFromDatabase(); // Load the selected config profile from
the database to be the current render state
164     void closeDatabaseDialog(); // Hide the save/load render state dialog
165     void databaseLoadScrollDown(); // Scroll down in the load render state
dialog
166     void databaseLoadScrollUp(); // Scroll up in the load render state
dialog
167 public:
168     int guiMain(char*); // Start and run the entire GUI. Blocks on current
thread
169
170     HFractalGui(); // Basic constructor
171 };
172
173 #endif

```

```
1 // src/guimain.hh
2
3 #ifndef GUIMAIN_H
4 #define GUIMAIN_H
5
6 // GUI Main function to isolate the GUI module from the main module to prevent
7 // linker conflicts
7 int guiMain(char* );
8
9 #endif
```

```

1 // src/hyperfractal.cc
2
3 #include "hyperfractal.hh"
4
5 #include <iostream>
6 #include <iomanip>
7 #include <chrono>
8 #include <thread>
9
10 #include "utils.hh"
11
12 using namespace std;
13 using namespace std::chrono;
14
15 /**
16 * @brief Main function called when each worker thread starts. Contains code
17 * to actually fetch and render pixels
18 */
19 void HFractalMain::threadMain () {
20     // Pre-compute constants to increase performance
21     long double p = 2/(zoom*resolution);
22     long double q = (1/zoom)-offset_x;
23     long double r = (1/zoom)+offset_y;
24
25     // Get the next unrendered pixel
26     int next = img->getUncompleted();
27     while (next != -1) {
28         // Find the x and y coordinates based on the pixel index
29         int x = next%resolution;
30         int y = next/resolution;
31         // Apply the mathematical transformation of offsets and zoom to find
32         // a and b, which form a coordinate pair representing this pixel in the complex
33         // plane
34         long double a = (p*x) - q;
35         long double b = r - (p*y);
36         // Construct the initial coordinate value, and perform the
37         // evaluation on the main equation
38         complex<long double> c = complex<long double> (a,b);
39         int res = (main_equation->evaluate (c, eval_limit));
40         // Set the result back into the image class, and get the next
41         // available unrendered pixel
42         img->set (x, y, res);

```

```

39         next = img->getUncompleted();
40     }
41
42     // When there appear to be no more pixels to compute, mark this thread
43     // as completed
44     thread_completion[std::this_thread::get_id()] = true;
45
46     // Check to see if any other threads are still rendering, if not then
47     // set the flag to mark the environment as no longer rendering
48     bool is_incomplete = false;
49     for (auto p : thread_completion) is_incomplete |= !p.second;
50     if (!is_incomplete) is_rendering = false;
51 }
52 /**
53 * @brief Generate a fractal image based on all the environment parameters
54 *
55 * @param wait Whether to wait and block the current thread until the image
56 * has been fully computed, useful if you want to avoid concurrency somewhere
57 * else (functionality hiding)
58 * @return Integer representing status code, 0 for success, else for failure
59 */
60 int HFractalMain::generateImage (bool wait=true) {
61     if (getIsRendering()) { std::cout << "Aborting!" << std::endl; return 2;
62 } // Prevent overlapping renders from starting
63     // Output a summary of the rendering parameters
64     std::setprecision (100);
65     std::cout << "Rendering with parameters: " << std::endl;
66     std::cout << "Resolution=" << resolution << std::endl;
67     std::cout << "EvaluationLimit=" << eval_limit << std::endl;
68     std::cout << "Threads=" << worker_threads << std::endl;
69     std::cout << "Zoom="; printf ("%Le", zoom); std::cout << std::endl;
70     std::cout << "OffsetX="; printf ("% .70Lf", offset_x); std::cout <<
71     std::endl;
72     std::cout << "OffsetY="; printf ("% .70Lf", offset_y); std::cout <<
73     std::endl;
74
75     // Abort rendering if the equation is invalid
76     if (!isValidEquation()) { std::cout << "Aborting!" << std::endl; return
77     1; }
78
79     // Mark the environment as now rendering, locking resources/parameters
80     is_rendering = true;
81 }
```

```

75 // Clear and reinitialise the image class with the requested resolution
76 if (img != NULL) img->~HFractalImage();
77 img = new HFractalImage (resolution, resolution);
78
79 // Clear the thread pool, and populate it with fresh worker threads
80 thread_pool.clear();
81 thread_completion.clear();
82 for (int i = 0; i < worker_threads; i++) {
83     std::thread *t = new std::thread(&HFractalMain::threadMain, this);
84     thread_completion[t->get_id()] = false;
85     thread_pool.push_back(t);
86 }
87
88 // Optionally, wait for the render to complete before returning
89 if (wait) {
90     while (true) {
91         // If enabled at compile time, show a progress bar in the
terminal
92         #ifdef TERMINAL_UPDATES
93             float percent = getImageCompletionPercentage();
94             std::cout << "\r";
95             std::cout << "Working: ";
96             for (int k = 2; k <= 100; k+=2) { if (k <= percent) std::cout <<
"█"; else std::cout << " "; }
97             std::cout << " | ";
98             std::cout << round(percent) << "%";
99             #endif
100            // Break out when the image has been fully completed (all pixels
computed)
101            if (img->isDone()) break;
102            crossPlatformDelay (10);
103        }
104        // Wait for all the threads to join, then finish up
105        for (auto th : thread_pool) th->join();
106        is_rendering = false;
107    }
108    std::cout << std::endl << "Rendering done." << std::endl;
109    return 0;
110 }
111
112 /**
113 * @brief Construct a new rendering environment, with blank parameters
114 *

```

```

115 */
116 HFRACTALMAIN::HFRACTALMAIN () {
117     resolution = 1;
118     offset_x = 0;
119     offset_y = 0;
120     zoom = 1;
121     img = NULL;
122 }
123
124 /**
125 * @brief Convert the raw data stored in the image class into a coloured
126 *        RGBA 32 bit image using a particular colour scheme preset
127 *
128 * @param colour_preset The colour scheme to use
129 * @return uint32_t* Pointer to the image stored in memory as an array of 4-
130 *         byte chunks
131 */
132
133 uint32_t* HFRACTALMAIN::getRGBAIIMAGE (int colour_preset) {
134     // Return a blank result if the image is uninitialised, other conditions
135     // should ensure this never occurs
136
137     if (img == NULL) {
138         return (uint32_t *)malloc(0);
139     }
140
141     // Copy parameters to local
142     int size = resolution;
143     int limit = eval_limit;
144
145     // Construct a pixel buffer with RGBA channels
146     uint32_t *pixels = (uint32_t *)malloc(size*size*sizeof(uint32_t));
147     for (int x = 0; x < size; x++) {
148         for (int y = 0; y < size; y++) {
149             int v = img->get(x,y);
150             pixels[(y*size)+x] = (v == limit) ? 0x000000ff :
151             HFRACTALIMAGE::colourFromValue(v, colour_preset);
152
153             // If the pixel has not been computed, make it transparent
154             if (img->completed[(y*size)+x] != 2) pixels[(y*size)+x] = 0;
155         }
156     }
157
158     // Return the pointer to the pixel buffer
159     return pixels;

```

```

154 }
155
156 /**
157 * @brief Get the percentage of pixels in the image which have been computed
158 *
159 * @return Unrounded percentage
160 */
161 float HFractalMain::getImageCompletionPercentage () {
162     if (img == NULL) return 100;
163     return ((float)(img->getInd())/(float)(resolution*resolution))*100;
164 }
165
166 /**
167 * @brief Automatically write an image to a generated file address.
168 * File path will be dynamically constructed so that the file name is
169 * unique, timestamped, and placed on the user's desktop
170 *
171 * @param type Image format to write image out to
172 * @return True for success, false for failure
173 */
174 bool HFractalMain::autoWriteImage (IMAGE_TYPE type) {
175     string image_name = "Fractal render from ";
176
177     // Get current system time
178     auto time = system_clock::to_time_t (system_clock::now());
179     string c_time = string (ctime (&time));
180
181     // Separate ctime result into components
182     vector<string> time_components;
183     string current_component = "";
184     for (char c : c_time) {
185         if (c == ' ') {
186             time_components.push_back (current_component);
187             current_component = "";
188         } else if (c != '\n') current_component.push_back (c != ':' ? c :
189         '.');
190     }
191     time_components.push_back (current_component);
192
193     // Get milliseconds, not part of ctime
194     system_clock::duration dur = system_clock::now().time_since_epoch();
195     seconds s = duration_cast<seconds> (dur);

```

```

194     dur -= s;
195     milliseconds ms = duration_cast<milliseconds> (dur);
196
197     // Components are in the form: dayofweek month day hour:minute:second
198     image_name += time_components[2] + " ";
199     image_name += time_components[1] + " ";
200     image_name += time_components[4] + " ";
201     image_name += "at ";
202     image_name += time_components[3];
203     image_name += ".";
204     image_name += to_string(ms.count());
205
206     cout << image_name << endl;
207
208     string image_path = "";
209
210     image_path += getDesktopPath();
211     image_path += image_name;
212
213     // Call into the image's writer to write out data
214     switch (type) {
215     case PGM:
216         image_path += ".pgm";
217         return img->writePGM (image_path);
218     default:
219         return false;
220     }
221 }
```

```

1 // src/hyperfractal.hh
2
3 #ifndef HYPERFRACTAL_H
4 #define HYPERFRACTAL_H
5
6 #include <string>
7 #include <thread>
8 #include <vector>
9 #include <map>
10
11 #include "image.hh"
12 #include "fractal.hh"
13 #include "utils.hh"
14 #include "equationparser.hh"
15
16 // When defined, progress updates will be written to terminal.
17 #define TERMINAL_UPDATES
18
19 // Class defining a fractal rendering environment, fully encapsulated
20 class HFractalMain {
21 private:
22     int resolution; // Horizontal and vertical dimension of the desired image
23     long double offset_x; // Horizontal offset in the complex plane
24     long double offset_y; // Vertical offset in the complex plane
25     long double zoom; // Scaling value for the image (i.e. zooming in)
26
27     std::string eq; // String equation being used
28     HFractalEquation *main_equation; // Actual pointer to the equation
29     manager class being used for computation
30
31     int worker_threads; // Number of worker threads to be used for
32     computation
33     int eval_limit; // Evaluation limit for the rendering environment
34
35     HFractalImage *img = new HFractalImage(0,0); // Pointer to the image
36     class containing data for the rendered image
37
38     std::vector<std::thread*> thread_pool; // Thread pool containing
39     currently active threads
40     std::map<std::thread::id, bool> thread_completion; // Map of which
41     threads have finished computing pixels
42     bool is_rendering = false; // Marks whether there is currently a render
43     ongoing (locking resources to prevent concurrent modification e.g. changing
44     resolution mid-render)

```

```

38
39     void threadMain (); // Method called on each thread when it starts,
contains the worker/rendering code
40
41 public:
42     int generateImage (bool); // Perform the render, and optionally block the
current thread until it is done
43
44     HFractalMain (); // Base initialiser
45
46     int getResolution () { return resolution; } // Inline methods to get/set
the resolution
47     void setResolution (int resolution_) { if (!getIsRendering()) resolution =
resolution_; }
48
49     long double getOffsetX () { return offset_x; } // Inline methods to
get/set the x offset
50     void setOffsetX (long double offset_x_) { if (!getIsRendering()) offset_x =
offset_x_; }
51
52     long double getOffsetY () { return offset_y; } // Inline methods to
get/set the y offset
53     void setOffsetY (long double offset_y_) { if (!getIsRendering()) offset_y =
offset_y_; }
54
55     long double getZoom () { return zoom; } // Inline methods to get/set the
zoom
56     void setZoom (long double zoom_) { if (!getIsRendering()) zoom = zoom_; }
57
58     std::string getEquation () { return eq; } // Inline methods to get/set
the equation
59     void setEquation (std::string eq_) {
60         if (!getIsRendering()) {
61             eq = eq_;
62             main_equation = HFractalEquationParser::extractEquation (eq);
63             if (main_equation == NULL) return;
64             // Detect if the equation matches the blueprint of a preset
65             int preset = -1;
66             for (int i = 0; i < NUM_EQUATION_PRESETS; i++) {
67                 if (eq == equationPreset ((EQ_PRESETS)i, false)) {
68                     preset = i;
69                     break;
70                 main_equation->setPreset (preset);
71             }
72         }

```

```

73     main_equation->setPreset (preset);
74 }
75 }
76
77     int getWorkerThreads () { return worker_threads; } // Inline methods to
get/set the number of worker threads
78     void setWorkerThreads (int wt_) { if (!getIsRendering()) worker_threads =
wt_; }
79
80     int getEvalLimit () { return eval_limit; } // Inline methods to get/set
the evaluation limit
81     void setEvalLimit (int el_) { if (!getIsRendering()) eval_limit = el_; }
82
83     bool isValidEquation () { return main_equation != NULL; } // Check if the
equation the user entered was parsed correctly last time it was set
84
85     bool getIsRendering() { return is_rendering; } // Get if there is
currently a render happening in this environment
86
87     uint32_t* getRGBAImage (int); // Return a pointer to a 32 bit RGBA
formatted image, produced using a particular colour scheme preset, from the
generated image
88
89     float getImageCompletionPercentage (); // Get the current percentage of
pixels that have been actually computed
90
91     bool autoWriteImage (IMAGE_TYPE); // Automatically write out the render
to desktop using a particular image type
92 };
93 #endif

```

```

1 // src/image.cc
2
3 #include "image.hh"
4
5 #include <iostream>
6 #include <math.h>
7
8 /**
9  * @brief Set the value of a pixel, and automatically mark it as complete
10 *
11 * @param x Horizontal coordinate
12 * @param y Vertical coordinate
13 * @param p Value of the pixel to assign
14 */
15 void HFractalImage::set(int x, int y, uint16_t p) {
16     int offset = ((y*width)+x);
17     data_image[offset] = p;
18     completed[offset] = 2;
19 }
20
21 /**
22 * @brief Get the value of the pixel at the specified coordinates, as
23 * measured from top-left
24 *
25 * @param x Horizontal coordinate
26 * @param y Vertical coordinate
27 * @return The value of the pixel at the coordinates
28 */
29 uint16_t HFractalImage::get(int x, int y) {
30     return data_image[(y*width)+x];
31 }
32 /**
33 * @brief Initialise a new image with a specified width and height
34 *
35 * @param w Horizontal size
36 * @param h Vertical size
37 */
38 HFractalImage::HFractalImage(int w, int h) {
39     width = w;
40     height = h;
41     c_ind = 0;

```

```

42     data_image = new uint16_t[width*height];
43     completed = new uint8_t[width*height];
44     // Clear both buffers
45     for (int i = 0; i < width*height; i++) { data_image[i] = 0xffff;
46     completed[i] = 0; }
47
48 /**
49  * @brief Write the contents of the image buffer out to a PGM file (a
50  * minimal image format using grayscale linear colour space)
51  *
52  * @param path Path to the output file
53  * @return True for success, false for failure
54  */
55 bool HFractalImage::writePGM (std::string path) {
56     // Abort if the image is incomplete
57     if (!isDone()) return false;
58     FILE *img_file;
59     img_file = fopen(path.c_str(), "wb");
60
61     // Write the header
62     fprintf(img_file, "P5\n");
63     fprintf(img_file, "%d %d\n", width, height);
64     fprintf(img_file, "65535\n");
65
66     // Write each pixel
67     for(int y = 0; y < height; y++){
68         for(int x = 0; x < width; x++){
69             uint16_t p = data_image[(y*width)+x];
70
71             fputc (p & 0x00ff, img_file);
72             fputc ((p & 0xff00) >> 8, img_file);
73         }
74     }
75
76     // Close and return success
77     fclose(img_file);
78     return true;
79 }
80 /**
81  * @brief Create an RGBA 32 bit colour from hue, saturation, value
82  * components

```

```

82 *
83 * @param h Hue value
84 * @param s Saturation value
85 * @param v Value (brightness) value
86 * @return A 32 bit colour in RGBA form
87 */
88 uint32_t HFractalImage::HSVToRGB (float h, float s, float v) { // TODO: Test
this
89     float c = v * s;
90     float h_ = fmod(h,1)*6;
91     float x = c * (1 - fabsf(fmodf(h_, 2)-1));
92     float m = v - c;
93
94     float r;
95     float g;
96     float b;
97
98     if (h_ >= 0 && h_ < 1) {
99         r = c; g = x; b = 0;
100    } else if (h_ < 2) {
101        r = x; g = c; b = 0;
102    } else if (h_ < 3) {
103        r = 0; g = c; b = x;
104    } else if (h_ < 4) {
105        r = 0; g = x; b = c;
106    } else if (h_ < 5) {
107        r = x; g = 0; b = c;
108    } else if (h_ < 6) {
109        r = c; g = 0; b = x;
110    }
111
112    r = r + m;
113    g = g + m;
114    b = b + m;
115
116    uint32_t final_value = 0x000000ff;
117    final_value |= (int)(r*255) << (8*3);
118    final_value |= (int)(g*255) << (8*2);
119    final_value |= (int)(b*255) << (8*1);
120
121    return final_value;
122}

```

```

123
124 /**
125 * @brief Convert a computed value (i.e. from the image_data buffer) into a
126 renderable RGBA 32 bit colour
127 *
128 * @param value The value to convert
129 * @param colour_preset Colour palette preset to map colour onto
130 * @return The converted colour as a 32 bit integer
131 */
132 uint32_t HFractalImage::colourFromValue (uint16_t value, int colour_preset)
{
133     uint32_t col = 0x000000ff;
134     if (colour_preset == 0) {
135         col |= 0x3311ff00;
136         col |= ((value % 256) << (8*3)) + 0x33000000;
137     } else if (colour_preset == 1) {
138         uint8_t looped = 255-(uint8_t)(value % 256);
139         col |= looped << (8*3);
140         col |= (2*looped) << (8*2);
141         col |= 0x00007000;
142     } else if (colour_preset == 2) {
143         float hue = (float)value/(float)0xffff;
144         hue = fmod(512*hue, 1);
145         col = HFractalImage::HSVToRGB (hue, 0.45, 0.8);
146     } else if (colour_preset == 3) {
147         uint8_t looped = 255-(uint8_t)(value % 256);
148         col |= looped << (8*1); // B
149         col |= looped << (8*2); // G
150         col |= looped << (8*3); // R
151     } else if (colour_preset == 4) {
152         uint8_t looped = (uint8_t)(value % 256);
153         col |= looped << (8*1); // B
154         col |= looped << (8*2); // G
155         col |= looped << (8*3); // R
156     }
157
158     return col;
159 }
160
161 /**
162 * @brief Destroy the image class, freeing the buffers

```

```

163 */
164 HFractalImage::~HFractalImage () {
165     free (data_image);
166     free (completed);
167 }
168
169 /**
170 * @brief Fetch the index (i.e. (y*width)+x) of the next pixel which needs
171 * to be computed
172 *
173 * @return The index of the next pixel to compute, -1 if there is no
174 * available pixel
175 */
176 int HFractalImage::getUncompleted () {
177     // Lock resources to prevent collisions
178     mut.lock();
179     int i = -1;
180     // Find the next available pixel index and increment c_ind
181     if (c_ind < height*width) {
182         i = c_ind;
183         c_ind++;
184     }
185     // Unlock before returning
186     mut.unlock();
187     return i;
188 }
189 /**
190 * @brief Check every pixel to see if the image is fully computed. Use with
191 * caution, especially with large images
192 */
193 bool HFractalImage::isDone () {
194     // Iterate over every pixel to check its status
195     for (int i = 0; i < height*width; i++) {
196         if (completed[i] != 2) {
197             // Fail if the pixel is not complete
198             return false;
199         }
200     }
201     // Succeed if every pixel is fully computed
202     return true;

```

```
203 }
204
205 /**
206 * @brief Get the current completion index of the image
207 *
208 * @return The current completion index
209 */
210 int HFractalImage::getInd () { return c_ind; }
```

```

1 // src/image.hh
2
3 #ifndef IMAGE_H
4 #define IMAGE_H
5
6 #include <mutex>
7
8 // Class containing information about an image currently being generated
9 class HFractalImage {
10 private:
11     int width; // Width of the image
12     int height; // Height of the image
13     uint16_t * data_image; // Computed data values of the image
14     int c_ind = 0; // Index of the next pixel to be sent out to a rendering
15     thread
16     std::mutex mut; // Mutex object used to lock class resources during
17     multi-threading events
18
19 public:
20     HFractalImage (int, int); // Constructor, creates a new image buffer of
21     the specified size
22     ~HFractalImage (); // Destructor, destroys and deallocates resources used
23     in the current image
24
25     void set (int, int, uint16_t); // Set the value of a pixel
26     uint16_t get (int, int); // Get the value of a pixel
27     uint8_t * completed; // Stores the completion status of each pixel, 0 =
28     not computed, 1 = in progress, 2 = computed
29     int getUncompleted (); // Get the index of an uncomputed pixel, to be
30     sent to a rendering thread, and update completion data
31     bool isDone (); // Check if the image has been completed or not
32     int getInd (); // Get the current completion index
33     bool writePGM (std::string); // Write out the contents of the data buffer
34     to a simple image file, PGM format, with the given path
35
36     static uint32_t HSVToRGB (float h, float s, float v); // Create a 32 bit
37     RGB colour from hue, saturation, value components
38     static uint32_t colourFromValue (uint16_t, int); // Convert a computed
39     value into a 32 bit RGBA colour value, using the specified palette
40 };
41
42
43 #endif

```

```

1 // src/main.cc
2
3 #include <iostream>
4
5 #include "hyperfractal.hh"
6 #include "guimain.hh"
7 #include "utils.hh"
8
9 using namespace std;
10
11 /**
12 * Naming Convention:
13 * Classes & Structs - CapitalisedCamelCase
14 * Variables - snake_case
15 * Functions - uncapitalisedCamelCase
16 * Constants - SCREAMING_SNAKE_CASE
17 *
18 */
19
20 int main (int argc, char *argv[]) {
21     if (argc == 8) {
22         // If we have the required arguments, run a console-only render
23         HFractalMain hm;
24         int argument_error = 0;
25         try {
26             hm.setResolution (stoi (argv[1]));
27             if (hm.getResolution() <= 0) throw runtime_error("Specified
resolution too low.");
28             argument_error++;
29             hm.setOffsetX (stod (argv[2]));
30             argument_error++;
31             hm.setOffsetY (stod (argv[3]));
32             argument_error++;
33             hm.setZoom (stod (argv[4]));
34             argument_error++;
35             hm.setEquation (string (argv[5]));
36             if (!hm.isValidEquation()) throw runtime_error("Specified
equation is invalid.");
37             argument_error++;
38             hm.setWorkerThreads (stoi (argv[6]));
39             if (hm.getWorkerThreads() <= 0) throw runtime_error("Must use at
least one worker thread.");

```

```

40         argument_error++;
41         hm.setEvalLimit (stoi (argv[7]));
42         if (hm.getEvalLimit() <= 0) throw runtime_error("Must use at
least one evaluation iteration.");
43         argument_error++;
44         hm.generateImage(true);
45         return !hm.autoWriteImage (IMAGE_TYPE::PGM);
46     } catch (runtime_error e) {
47         cout << "Parameter error on argument number " << argument_error
<< ":" << endl;
48         cout << "    " << e.what() << endl;
49         return 1;
50     }
51 } else if (argc != 1) {
52     // If we have only some arguments, show the user what arguments they
need to provide
53     cout << "Provide all the correct arguments please:" << endl;
54     cout << "int resolution, long double offset_x, long double offset_y,
long double zoom, string equation, int worker_threads, int eval_limit" <<
endl;
55     return 1;
56 } else {
57     // Otherwise, start the GUI
58     cout << trimExecutableFromPath(argv[0]) << endl;
59     return guiMain(trimExecutableFromPath(argv[0]));
60 }
61 }

```

```

1 // src/utils.cc
2
3 #include "utils.hh"
4
5 #ifdef _WIN32
6 #include <windows.h>
7 #include <shlobj.h>
8 #else
9     #include <unistd.h>
10 #endif
11
12 using namespace std;
13
14 /**
15 * @brief Get the details of an equation preset
16 *
17 * @param p The preset to return
18 * @param t True - return the name of the preset, False - return the string
19 *          equation of it instead
20 */
21 string equationPreset (EQ_PRESETS p, bool t) {
22     switch (p) {
23     case EQ_MANDELBROT:
24         return t ? "Mandelbrot" : "(z^2)+c";
25     case EQ_JULIA_1:
26         return t ? "Juila 1" : "(z^2)+(0.285+0.01i)";
27     case EQ_JULIA_2:
28         return t ? "Julia 2" : "(z^2)+(-0.70176-0.3842i)";
29     case EQ_RECIPROCAL:
30         return t ? "Reciprocal" : "1/((z^2)+c)";
31     case EQ_ZPOWER:
32         return t ? "Z Power" : "(z^z)+(c-0.5)";
33     case EQ_BARS:
34         return t ? "Bars" : "z^(c^2)";
35     case EQ_BURNINGSHIP_MODIFIED:
36         return t ? "Burning Ship Modified" : "((x^2)^0.5-((y^2)^0.5)i)^2+c";
37     default:
38         return "NONE";
39     }
40     return "";
41 }

```

```

42
43 /**
44 * @brief Return the name of a colour palette preset
45 *
46 * @param p Preset to return
47 * @return The string name of the colour palette
48 */
49 string colourPalettePreset (CP_PRESETS p) {
50     switch (p) {
51         case CP_VAPORWAVE:
52             return "Vaporwave";
53         case CP_YELLOWGREEN:
54             return "Yellow-Green";
55         case CP_RAINBOW:
56             return "Rainbow";
57         case CP_GREYSCALE_BRIGHT:
58             return "Greyscale Bright";
59         case CP_GREYSCALE_DARK:
60             return "Greyscale Dark";
61         default:
62             return "NONE";
63     }
64     return "";
65 }
66
67 /**
68 * @brief Wrap text given a certain line length
69 *
70 * @param s String to wrap
71 * @param line_length Number of characters which limit the length of the
72 * line
73 * @return string
74 */
74 string textWrap (string s, int line_length) {
75     string output = "";
76     int line_offset = 0;
77     for (char c : s) {
78         if (c == '\n') line_offset = -1;
79         if (line_offset == line_length-1) { if (c != ' ') output += "-";
80         output += "\n"; line_offset = 0; }
81         if (!(line_offset == 0 && c == ' ')) {
82             output += c;

```

```

82         line_offset++;
83     }
84 }
85 return output;
86 }
87
88 /**
89 * @brief Get the desktop path of the user
90 *
91 * @return The user's desktop path
92 */
93 string getDesktopPath () {
94     #ifdef _WIN32
95         static char path[MAX_PATH+1];
96         if (SHGetSpecialFolderPathA(HWND_DESKTOP, path, CSIDL_DESKTOP, FALSE))
97             return string(path) + string("\\");
98         else
99             return "";
100    #else
101        return string(getenv ("HOME")) + string("/Desktop/");
102    #endif
103 }
104
105 /**
106 * @brief Delay for a number of millisecs, across any platform
107 *
108 * @param milliseconds Time to delay
109 */
110 void crossPlatformDelay(int milliseconds) {
111     #ifdef _WIN32
112         Sleep(milliseconds);
113     #else
114         usleep(milliseconds * 1000);
115     #endif
116 }
117
118 /**
119 * @brief Trim the executable name from the end of the path, returning just
120 * the working directory
121 *
122 * @param path Input path from command line arguments
123 * @return Trimmed path

```

```

123 */
124 char* trimExecutableFromPath(char* path) {
125     int str_len = 0;
126     while (true) {
127         if (path[str_len] == '\0') break;
128         str_len++;
129     }
130
131     uint32_t slash_index = -1;
132     for (int i = str_len-1; i >= 0; i--) {
133         if (path[i] == '\\' || path[i] == '/') {
134             slash_index = i;
135             break;
136         }
137     }
138
139     char* result = (char*)malloc(slash_index+2);
140
141     for (int i = 0; i < slash_index+1; i++) {
142         result[i] = path[i];
143     }
144     result[slash_index+1] = '\0';
145
146     return result;
147 }
```

```

1 // src/utils.hh
2
3 #ifndef UTILS_H
4 #define UTILS_H
5
6 #include <string>
7
8 #define NUM_EQUATION_PRESETS 7
9 #define NUM_COLOUR_PRESETS 5
10
11 // Wrap text along a line length
12 std::string textWrap (std::string, int);
13
14 // Enum describing possible equation presets
15 enum EQ_PRESETS {
16     EQ_MANDELBROT = 0, // "(z^2)+c"
17     EQ_JULIA_1, // "(z^2)+(0.285+0.01i)"
18     EQ_JULIA_2, // "(z^2)+(-0.70176-0.3842i)"
19     EQ_RECIPROCAL, // "1/((z^2)+c)"
20     EQ_ZPOWER, // "(z^z)+(c-0.5)"
21     EQ_BARS, // "z^(c^2)"
22     EQ_BURNINGSHIP_MODIFIED // "((x^2)^0.5-((y^2)^0.5)i)^2+c"
23 };
24
25 // Enum describing possible colour palette presets
26 enum CP_PRESETS {
27     CP_VAPORWAVE = 0,
28     CP_YELLOWGREEN,
29     CP_RAINBOW,
30     CP_GREYSCALE_BRIGHT,
31     CP_GREYSCALE_DARK
32 };
33
34 // Enum describing available image types which can be saved to disk
35 enum IMAGE_TYPE {
36     PGM
37 };
38
39 // Delay for a given number of milliseconds
40 void crossPlatformDelay (int);
41

```

```
42 // Get the user's desktop path
43 std::string getDesktopPath ();
44
45 // Get information about an equation preset
46 std::string equationPreset (EQ_PRESETS, bool);
47 // Get information about a colour palette preset
48 std::string colourPalettePreset (CP_PRESETS);
49
50 // Get the current working directory from the path argument
51 char* trimExecutableFromPath (char*);
52
53 #endif
```

Testing

In the following test description, some technical elements have been used. Setup statements which begin with ‘./HyperFractal’ signify that the test will consist of running the console version of the application as a command using the arguments specified after the ‘./HyperFractal’.

Expected and Actual Output statements may begin with ‘> ...’ which indicates console output occurring. Some tests link to images as evidence, which are shown in the Test Appendices below the testing table.

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
1	1-01	Test available resolution using console interface	./HyperFractal -32 0 0 1 "z^2+c" 4 32	Program exits reporting parameter error.	> Parameter error on argument number 0: > Specified resolution too low.	Pass
2			./HyperFractal 0 0 0 1 "z^2+c" 4 32	Program exits reporting parameter error.	> Parameter error on argument number 0: > Specified resolution too low.	Pass
3			./HyperFractal 32 0 0 1 "z^2+c" 4 32	Program renders the fractal, reports success, and saves a 32x32 image file.	> Rendering with parameters: > Resolution=32 > EvaluationLimit=32 > Threads=4 > Zoom=1.000000e+00 > OffsetX=0.000000000000 00000000000000000000 00000000000000000000 00000000000000000000 > OffsetY=0.000000000000 00000000000000000000 00000000000000000000 00000000000000000000 > Rendering done. > Fractal render from 14 Mar 2022 at 10.13.23.507	Pass

Test	Req.	Test Description	Test Data/ Setup	Expected Output/ Result	Actual Output/Result	Test Result
4			./HyperFractal 320 0 0 1 "z^2+c" 4 32	Program renders the fractal, reports success, and saves a 320x320 image file.	> Rendering with parameters: > Resolution=320 > EvaluationLimit=32 > Threads=4 > Zoom=1.000000e+00 > OffsetX=0.00000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000 > OffsetY=0.00000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000 > Rendering done. > Fractal render from 14 Mar 2022 at 10.21.43.488	Pass
5			./HyperFractal 3200 0 0 1 "z^2+c" 4 32	Program renders the fractal, reports success, and saves a 3200x3200 image file.	> Rendering with parameters: > Resolution=3200 > EvaluationLimit=32 > Threads=4 > Zoom=1.000000e+00 > OffsetX=0.00000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000 > OffsetY=0.00000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000 > Rendering done. > Fractal render from 14 Mar 2022 at 10.22.23.807	Pass

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
12			Run in GUI mode and click zoom in button.	Viewport on rendered image is scaled up.	Viewport on rendered image is scaled up.	Pass
13			Run in GUI mode and click zoom out button.	Viewport on rendered image is scaled down.	Viewport on rendered image is scaled down.	Pass
14	1-03 2-02	Demonstrate navigating around the interface using keys	Run in GUI mode and press left arrow key.	Viewport on rendered image shifts to the left.	Viewport on rendered image shifts to the left.	Pass
15			Run in GUI mode and press right arrow key.	Viewport on rendered image shifts to the right.	Viewport on rendered image shifts to the right.	Pass
16			Run in GUI mode and press up arrow key.	Viewport on rendered image shifts up.	Viewport on rendered image shifts up.	Pass
17			Run in GUI mode and press down arrow key.	Viewport on rendered image shifts down.	Viewport on rendered image shifts down.	Pass
18			Run in GUI mode and press '-' key.	Viewport on rendered image is scaled up.	Viewport on rendered image is scaled up.	Pass
19			Run in GUI mode and press '+' key.	Viewport on rendered image is scaled down.	Viewport on rendered image is scaled down.	Pass

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
20	1-04	Demonstrate equation customisation	Run in GUI mode, click 'Equation Presets' and select each one sequentially.	Equation is changed to the selected equation, image is updated accordingly for each one.	Equation is changed to the selected equation, image is updated accordingly for each one.	Pass
21			Run in GUI mode, click on equation input box and type to change equation to '1/(zz+ccc)'	Equation input box displays new equation, image is updated to show entered equation.	Equation input box displays new equation, image is updated to show entered equation.	Pass
22	1-04 2-04	Demonstrate equation parsing at runtime	Inputs to HFractalEquationParser::extract_equation(): Valid data: "z^2+c" "(x-y)/z" "((a-x)^2)+((b-y)^2)" "z^c-z" "z^(c-z)" Invalid data: "z^/c" ")z-c)" "iy-c(" "w" ")(+a" "z-.5" "z-3."	"z 2 ^ c +" "x y - z /" "a x - 2 ^ b y - 2 ^ +" "z c ^ z -" "z c z - ^" Operation Error Bracket Error Bracket Error Unsupported Character Error Bracket Error Floating Point Error Floating Point Error	"z 2 ^ c +" "x y - z /" "a x - 2 ^ b y - 2 ^ +" "z c ^ z -" "z c z - ^" Operation Error Bracket Error Bracket Error Unsupported Character Error Bracket Error Floating Point Error Floating Point Error	Pass

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
23	1-05	Demonstrate image saving	Run in GUI mode, press 'Save Image'	Rendered image is saved to a bitmap image on the desktop with a generated name based on the date and time.	Rendered image is saved to a bitmap image on the desktop with a generated name based on the date and time.	Pass
24	1-06 2-06	Demonstrate database saving system	Run in GUI mode, press 'Save Render State', type a name and press 'Save'	Console displays 'Profile saved to database'. Database files created/ have the record appended to them.	Console displays 'Profile saved to database'. Database has record appended, see Test Appendix 1.	Pass
25		Demonstrate database reading system	Run in GUI mode, press 'Load Render State', select profile from the displayed list and press 'Load'. For contents of configuration profile table, see Test Appendix 2.	Console displays 'Profile loaded from database'. Records are visible in list, see Test Appendix 3. When load is pressed, configuration parameters are copied into rendering environment and the image is rerendered.	Console displays 'Profile loaded from database'. Records are visible in list, see Test Appendix 3. When load is pressed, configuration parameters are copied into rendering environment and the image is rerendered.	Pass

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
26	1-07	Timing the rendering process on various different render parameters .	Run in GUI mode, fullscreen application. For each of the following, configure and press 'Render Image' and start a timer. Stop the timer when the render reaches 100% completion: Equation and iteration limit: "(z^2)+c", 200 "(z^2)+c", 500 "1/((z^2)+c)", 200 "1/((z^2)+c)", 500 "z^(c^2)", 300 "(z^z)+(c-0.5)", 300 "z/(zz+ccc)", 300	For first set of tests, render time should be less than 30 seconds on an average laptop. For the second set, they should be less than 1 minute.	01.35 seconds 02.51 seconds 03.61 seconds 08.00 seconds 19.40 seconds 21.73 seconds 24.48 seconds	Pass
27	1-09 2-08	Demonstrate help and instructions	Run in GUI mode, click 'Help & Instructions'.	Application displays a help document by opening the project's README in the native system web browser. See Appendix 1 for this help document.	Application displays a help document by opening the project's README in the native system web browser. See Appendix 1 for this help document.	Pass
28	1-10	Demonstrate changing render parameters using interface buttons	Run in GUI mode, click '>' button to increase iteration limit.	Iteration limit is increased by one, iteration limit display box is updated.	Iteration limit is increased by one, iteration limit display box is updated.	Pass

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
29			Run in GUI mode, click '<' button to decrease iteration limit.	Iteration limit is decreased by one, iteration limit display box is updated.	Iteration limit is decreased by one, iteration limit display box is updated.	Pass
30			Run in GUI mode, click 'Colour Palettes'. Select each one sequentially.	Colour palette of displayed image changes when a colour palette is selected, for each one.	Colour palette of displayed image changes to the selected palette when a colour palette is selected, for each one.	Pass
31		Demonstrate changing render parameters using keyboard keys	Run in GUI mode, press ']' key to. Repeat, holding shift.	Iteration limit is increased by one, iteration limit display box is updated. Iteration limit is increased by ten, iteration limit display box is updated.	Iteration limit is increased by one, iteration limit display box is updated. Iteration limit is increased by ten, iteration limit display box is updated.	Pass

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
32			Run in GUI mode, press 'l' key to. Repeat, holding shift.	Iteration limit is decreased by one, iteration limit display box is updated. Iteration limit is decreased by ten, iteration limit display box is updated.	Iteration limit is decreased by one, iteration limit display box is updated. Iteration limit is decreased by ten, iteration limit display box is updated.	Pass
33	1-11 2-03	Demonstrate the application running on both macOS and Windows.	On macOS: Double-click the packaged application. Open a terminal, navigate to the executable, run "./Hyperfractal" Do the same, running "./Hyperfractal 256 0.0 0.0 1.0 "z^2+c" 4 200"	The application will open in GUI mode. The application will open in GUI mode. Program runs in terminal, performs the render, saves the image to the desktop.	Application opens in GUI mode. Application opens in GUI mode. Program runs in terminal, performs the render, saves the image to the desktop.	Pass
34			On Windows: Double-click the packaged application.	The application will open in GUI mode.	Application opens in GUI mode.	Pass

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
35	2-01	Demonstrate preview render	<p>Run in GUI mode.</p> <p>Press ‘Render Image’.</p> <p>Perform any navigation event.</p> <p>Press ‘Render Image’.</p>	<p>Preview render shown.</p> <p>Full render occurs and is shown</p> <p>Preview render switched to.</p> <p>Full render occurs and is shown.</p>	<p>Preview render shown.</p> <p>Full render occurs and is shown</p> <p>Preview render switched to.</p> <p>Full render occurs and is shown.</p> <p>See Test Appendix 4 for a screenshot after step 3, and Test Appendix 5 for a screenshot after step 4.</p>	Pass
36	2-05	Demonstrate detection of available threads.	<p>Run in CLI mode, specifying the number of threads to be used (here 4):</p> <pre>./HyperFractal 1024 0.0 0.0 0.75 "(z^2)+c" 4 150</pre>	<p>Program displays details of render environment before rendering.</p> <p>Activity monitor shows process using 5 threads (4 workers + 1 main).</p>	<p>> Rendering with parameters:</p> <p>> Resolution=1024 > EvaluationLimit=150 > Threads=4 > Zoom=7.500000e-01 > > OffsetX=0.0000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 > > OffsetY=0.0000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 > Rendering done.</p> <p>Activity monitor shows executable process is using 5 threads.</p>	Pass

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
37		Demonstrate performance boost using multiple threads.	Run in CLI mode with only 1 worker: ./HyperFractal 2048 0.0 0.0 0.75 "(z^2)+c" 1 500 ./HyperFractal 2048 0.0 0.0 0.75 "(z^2)+c" 2 500 ./HyperFractal 2048 0.0 0.0 0.75 "(z^2)+c" 4 500 ./HyperFractal 2048 0.0 0.0 0.75 "(z^2)+c" 8 500	First run will be the slowest, the third the fastest, the second and fourth somewhere between. Test host machine has 4 cores, hence the prediction.	First run: 10.86 seconds Second run: 7.08 seconds Third run: 5.56 seconds Fourth run: 5.79 seconds As expected, the optimal number of threads for best performance is the number of CPU cores in the host machine.	Pass
38	2-07	Demonstrate resizing ability	Run in GUI mode, drag corner to resize window. Press 'Render Image' and resize the window while the render occurs.	Render image frame resizes with window. Window snaps back to original size without image resizing.	Render image frame resizes with window. Window snaps back to original size without image resizing.	Pass
39	2-09	Demonstrate memory requirement	Run in GUI mode, enter fullscreen. Run 20 renders. Leave application open overnight.	Memory requirement is constant over time, and remains less than 500MiB.	Initial requirement: ~30MiB After 20 renders: ~30MiB After being left: ~30MiB Memory requirement is minimal and consistent.	Pass

Test	Req.	Test Description	Test Data/Setup	Expected Output/Result	Actual Output/Result	Test Result
40	2-10	Demonstrate render progress	Run in GUI mode, press ‘Render Image’.	Console text changes to “Rendering...”. Progress bar shows render progress. Upon completion, console text changes to “Rendering done.”.	Console text changes to “Rendering...”. Progress bar shows render progress. Upon completion, console text changes to “Rendering done.”.	Pass
41	2-11	Demonstrate parameter locking during render	Run in GUI mode and press ‘Render Image’. While render is happening, click zoom in, zoom out, increase/ decrease iteration limit, and navigate up/down/left/right. Press all navigation keys on the keyboard. Click inside the render area.	GUI should not respond to any inputs (navigation should not occur, iteration limit and zoom should be kept the same).	GUI does not respond to any inputs, all parameters are kept the same.	Pass
42	Misc	Test custom HSV to RGB conversion function	Feed in sets of three inputs to the function. Test: H = from 0 to 1 in steps of 1/6 S = from 0 to 1 in steps of 0.1 V = from 0 to 1 in steps of 0.1 In every combination.	Function should output a 32bit integer representing the RGBA colour generated.	Function outputs a fully correct test set (compared with a commercially available HSV to RGB converter). See Test Appendix 5 for results.	Pass

Summary

The tests above have targeted all of the requirements for this project, except those which require user feedback (which are discussed below). Every test has successfully passed, proving that all of my high-level and low-level objectives (except for 1-08) have been met by the technical solution I developed.

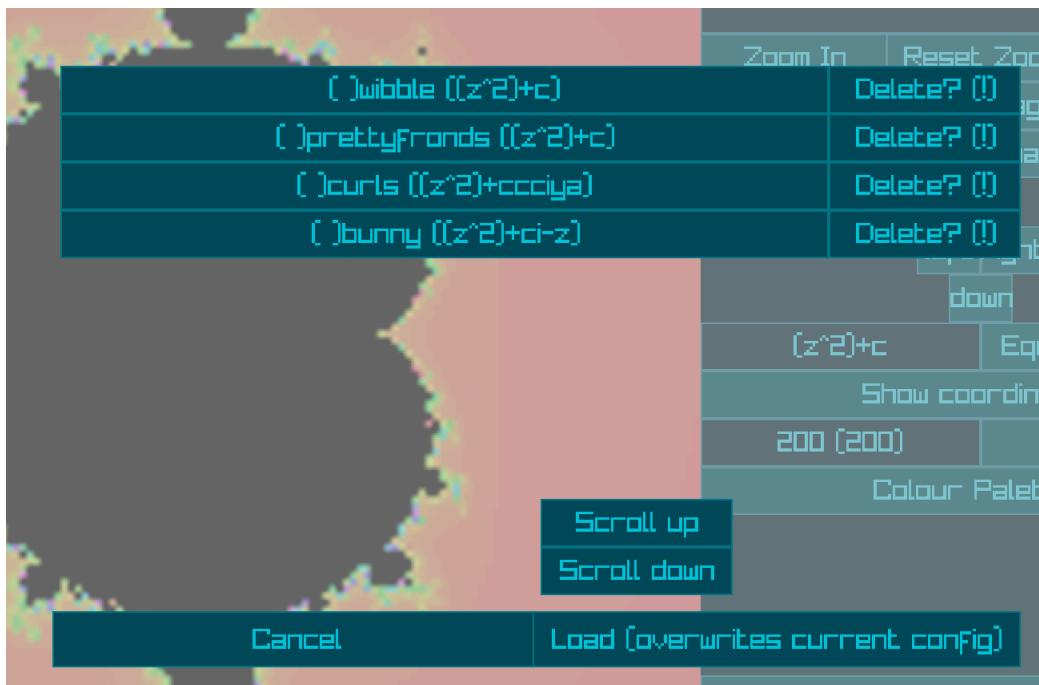
Test Appendices

4	0.000000	0.000000	1.000000	200	$(z^2)+c$	deletethis	2	0
0	-0.105455	0.909091	86.497559	200	$(z^2)+c$	wibble	1	0
2	0.439259	0.333015	3325.256730	200	$(z^2)+c$	prettyfronds	1	0
1	0.708664	-1.054763	5.062500	100	$(z^2)+ccciya$	curls	1	0
3	0.140720	-0.308651	1.000000	160	$(z^2)+ci-z$	bunny	2	0

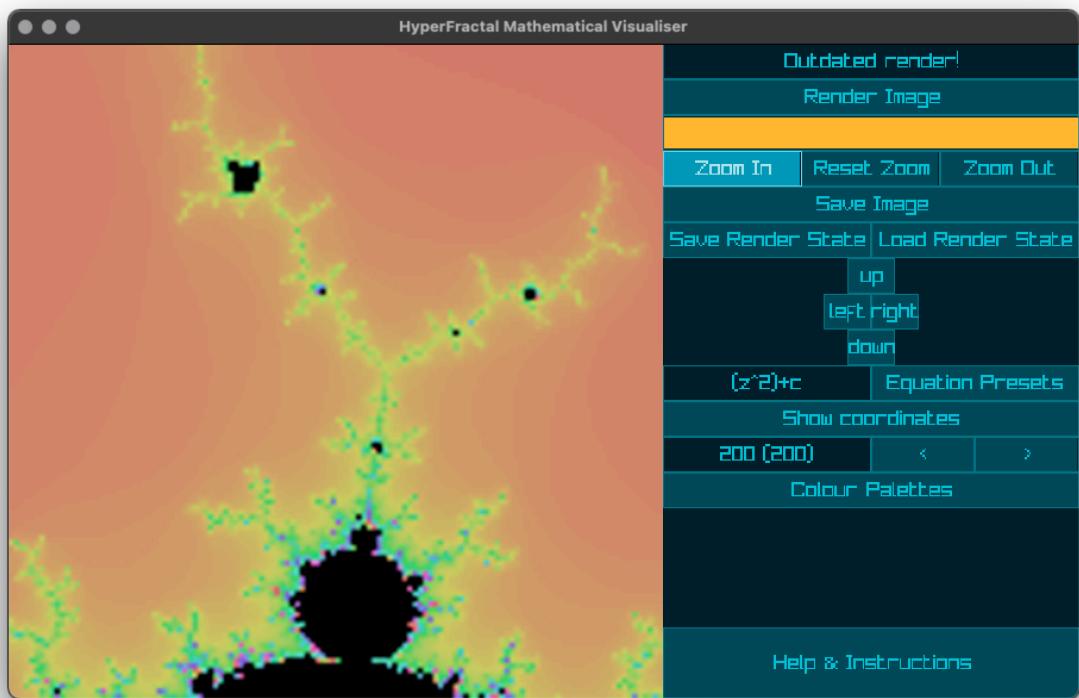
Test Appendix 1

0	-0.105455	0.909091	86.497559	200	$(z^2)+c$	wibble	1	0
2	0.439259	0.333015	3325.256730	200	$(z^2)+c$	prettyfronds	1	0
1	0.708664	-1.054763	5.062500	100	$(z^2)+ccciya$	curls	1	0
3	0.140720	-0.308651	1.000000	160	$(z^2)+ci-z$	bunny	2	0

Test Appendix 2



Test Appendix 3



Test Appendix 4

Test Appendix 5

The code used for this test is below, while the results are on the following pages.

```
bool success = true;
cout << hex;
for (float f = 0; f < 1; f+=(1.0/6.0)) {
    for (float s = 0; s < 1; s+=0.1) {
        for (float v = 0; v < 1; v+=0.1) {
            int my_colour = HFractalImage::HSVToRGB (f, s, 1);
            int raylib_colour = ColorToInt(ColorFromHSV(f*360, s, 1));
            if (my_colour != raylib_colour) success = false;
            cout << my_colour << " " << raylib_colour << ((my_colour != raylib_colour) ? " FAIL" : " PASS") << endl;
        }
    }
}

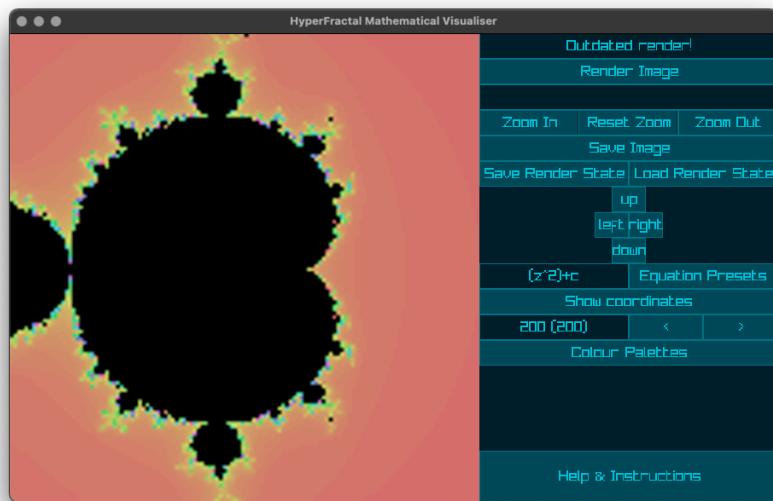
if (success) {
    cout << "Test succeeded :D" << endl;
} else {
    cout << "Test failed :( " << endl;
}
```

ff ff PASS	991e1eff 991e1eff PASS	33330fff 33330fff PASS	66cc66ff 66cc66ff PASS
191919ff 191919ff PASS	b22323ff b22323ff PASS	4c4c16ff 4c4c16ff PASS	72e572ff 72e572ff PASS
333333ff 333333ff PASS	cc2828ff cc2828ff PASS	66661eff 66661eff PASS	ff ff PASS
4c4c4cff 4c4c4cff PASS	e52d2dff e52d2dff PASS	7f7f26ff 7f7f26ff PASS	a190aff a190aff PASS
666666ff 666666ff PASS	ff ff PASS	99992dff 99992dff PASS	143314ff 143314ff PASS
7f7f7fff 7f7f7fff PASS	190202ff 190202ff PASS	b2b235ff b2b235ff PASS	1e4c1eff 1e4c1eff PASS
999999ff 999999ff PASS	330505ff 330505ff PASS	cccc3dff cccc3dff PASS	286628ff 286628ff PASS
b2b2b2ff b2b2b2ff PASS	4c0707ff 4c0707ff PASS	e5e544ff e5e544ff PASS	327f32ff 327f32ff PASS
ccccccff ccccccff PASS	660a0aff 660a0aff PASS	ff ff PASS	3d993dff 3d993dff PASS
e5e5e5ff e5e5e5ff PASS	7f0c0cff 7f0c0cff PASS	191905ff 191905ff PASS	47b247ff 47b247ff PASS
ff ff PASS	990f0fff 990f0fff PASS	33330aff 33330aff PASS	51cc51ff 51cc51ff PASS
191616ff 191616ff PASS	b21111ff b21111ff PASS	4c40cff 4c40cff PASS	5be55bff 5be55bff PASS
332d2dff 332d2dff PASS	cc1414ff cc1414ff PASS	666614ff 666614ff PASS	ff ff PASS
4c4444ff 4c4444ff PASS	e51616ff e51616ff PASS	7f7f19ff 7f7f19ff PASS	71907ff 71907ff PASS
665b5bff 665b5bff PASS	ff ff PASS	99991eff 99991eff PASS	f330ffff f330ffff PASS
7f7272ff 7f7272ff PASS	191919ff 191919ff PASS	b2b223ff b2b223ff PASS	164c16ff 164c16ff PASS
998989ff 998989ff PASS	333333ff 333333ff PASS	cccc28ff cccc28ff PASS	1e661eff 1e661eff PASS
b2a0a0ff b2a0a0ff PASS	4c4c4cff 4c4c4cff PASS	e5e52dff e5e52dff PASS	267f26ff 267f26ff PASS
ccb7b7ff ccb7b7ff PASS	666666ff 666666ff PASS	ff ff PASS	2d992dff 2d992dff PASS
e5ceceff e5ceceff PASS	7f7f7fff 7f7f7fff PASS	191902ff 191902ff PASS	35b235ff 35b235ff PASS
ff ff PASS	999999ff 999999ff PASS	333305ff 333305ff PASS	3dcc3dff 3dcc3dff PASS
191414ff 191414ff PASS	b2b2b2ff b2b2b2ff PASS	4c4c07ff 4c4c07ff PASS	44e544ff 44e544ff PASS
332828ff 332828ff PASS	ccccccff ccccccff PASS	66660aff 66660aff PASS	ff ff PASS
4c3d3dff 4c3d3dff PASS	e5e5e5ff e5e5e5ff PASS	7f7f0cff 7f7f0cff PASS	51905ff 51905ff PASS
665151ff 665151ff PASS	ff ff PASS	99990ff 99990ff PASS	a330aff a330aff PASS
7f6666ff 7f6666ff PASS	191916ff 191916ff PASS	b2b211ff b2b211ff PASS	f4c0fff f4c0fff PASS
997a7aff 997a7aff PASS	33332dff 33332dff PASS	cccc14ff cccc14ff PASS	146614ff 146614ff PASS
b28e8eff b28e8eff PASS	4c4c44ff 4c4c44ff PASS	e5e516ff e5e516ff PASS	197f19ff 197f19ff PASS
cca3a3ff cca3a3ff PASS	66665bff 66665bff PASS	ff ff PASS	1e991eff 1e991eff PASS
e5b7b7ff e5b7b7ff PASS	7f7f72ff 7f7f72ff PASS	191919ff 191919ff PASS	23b223ff 23b223ff PASS
ff ff PASS	999989ff 999989ff PASS	333333ff 333333ff PASS	28cc28ff 28cc28ff PASS
191111ff 191111ff PASS	b2b2a0ff b2b2a0ff PASS	4c4c4cff 4c4c4cff PASS	2de52dff 2de52dff PASS
332323ff 332323ff PASS	ccccb7ff ccccb7ff PASS	666666ff 666666ff PASS	ff ff PASS
4c3535ff 4c3535ff PASS	e5e5ceff e5e5ceff PASS	7f7f7fff 7f7f7fff PASS	21902ff 21902ff PASS
664747ff 664747ff PASS	ff ff PASS	999999ff 999999ff PASS	53305ff 53305ff PASS
7f5959ff 7f5959ff PASS	191914ff 191914ff PASS	b2b2b2ff b2b2b2ff PASS	74c07ff 74c07ff PASS
996b6bff 996b6bff PASS	333328ff 333328ff PASS	ccccccff ccccccff PASS	a660aff a660aff PASS
b27c7cff b27c7cff PASS	4c4c3dff 4c4c3dff PASS	e5e5eff e5e5eff PASS	c7f0cff c7f0cff PASS
cc8e8eff cc8e8eff PASS	666651ff 666651ff PASS	ff ff PASS	f990ffff f990ffff PASS
e5a0a0ff e5a0a0ff PASS	7f7f66ff 7f7f66ff PASS	161916ff 161916ff PASS	11b211ff 11b211ff PASS
ff ff PASS	99997aff 99997aff PASS	2d332dff 2d332dff PASS	14cc14ff 14cc14ff PASS
190f0fff 190f0fff PASS	b2b28eff b2b28eff PASS	444c44ff 444c44ff PASS	16e516ff 16e516ff PASS
331e1eff 331e1eff PASS	ccccca3ff cccca3ff PASS	5b665bff 5b665bff PASS	ff ff PASS
4c2d2dff 4c2d2dff PASS	e5e5b7ff e5e5b7ff PASS	727f72ff 727f72ff PASS	191919ff 191919ff PASS
663d3dff 663d3dff PASS	ff ff PASS	899989ff 899989ff PASS	333333ff 333333ff PASS
7f4c4cff 7f4c4cff PASS	191911ff 191911ff PASS	a0b2a0ff a0b2a0ff PASS	4c4c4cff 4c4c4cff PASS
995b5bff 995b5bff PASS	333323ff 333323ff PASS	b7ccb7ff b7ccb7ff PASS	666666ff 666666ff PASS
b26b6bff b26b6bff PASS	4c4c35ff 4c4c35ff PASS	cee5ceff cee5ceff PASS	7f7f7fff 7f7f7fff PASS
cc7a7aff cc7a7aff PASS	666647ff 666647ff PASS	ff ff PASS	999999ff 999999ff PASS
e58989ff e58989ff PASS	7f7f59ff 7f7f59ff PASS	141914ff 141914ff PASS	b2b2b2ff b2b2b2ff PASS
ff ff PASS	99996bfff 99996bfff PASS	283328ff 283328ff PASS	ccccccff ccccccff PASS
190c0cff 190c0cff PASS	b2b27cff b2b27cff PASS	3d4c3dff 3d4c3dff PASS	e5e5eff e5e5eff PASS
331919ff 331919ff PASS	cccc8eff cccccc8eff PASS	516651ff 516651ff PASS	ff ff PASS
4c2626ff 4c2626ff PASS	e5e5a0ff e5e5a0ff PASS	667f66ff 667f66ff PASS	161919ff 161919ff PASS
663333ff 663333ff PASS	ff ff PASS	7a997aff 7a997aff PASS	2d3333ff 2d3333ff PASS
7f3f3fff 7f3f3fff PASS	19190fff 19190fff PASS	8eb28eff 8eb28eff PASS	444c4cff 444c4cff PASS
994c4cff 994c4cff PASS	33331eff 33331eff PASS	a3cca3ff a3cca3ff PASS	5b6666ff 5b6666ff PASS
b25959ff b25959ff PASS	4c4c2dff 4c4c2dff PASS	b7e5b7ff b7e5b7ff PASS	727f7fff 727f7fff PASS
cc6666ff cc6666ff PASS	66663dff 66663dff PASS	ff ff PASS	899999ff 899999ff PASS
e57272ff e57272ff PASS	7f7f4cff 7f7f4cff PASS	111911ff 111911ff PASS	a0b2b2ff a0b2b2ff PASS
ff ff PASS	99995bfff 99995bfff PASS	233323ff 233323ff PASS	b7ccccff b7ccccff PASS
190a0aff 190a0aff PASS	b2b26bff b2b26bff PASS	354c35ff 354c35ff PASS	cee5eff cee5eff PASS
331414ff 331414ff PASS	cccc7aff ccccc7aff PASS	476647ff 476647ff PASS	ff ff PASS
4c1e1eff 4c1e1eff PASS	e5e589ff e5e589ff PASS	597f59ff 597f59ff PASS	141919ff 141919ff PASS
662828ff 662828ff PASS	ff ff PASS	6b996bff 6b996bff PASS	283333ff 283333ff PASS
7f3232ff 7f3232ff PASS	19190cff 19190cff PASS	7cb27cff 7cb27cff PASS	3d4c4cff 3d4c4cff PASS
993d3dff 993d3dff PASS	333319ff 333319ff PASS	8ecc8eff 8ecc8eff PASS	516666ff 516666ff PASS
b24747ff b24747ff PASS	4c4c26ff 4c4c26ff PASS	a0e5a0ff a0e5a0ff PASS	667f7fff 667f7fff PASS
cc5151ff cc5151ff PASS	666633ff 666633ff PASS	ff ff PASS	7a9999ff 7a9999ff PASS
e55b5bff e55b5bff PASS	7f7f3ffff 7f7f3ffff PASS	f190ffff f190ffff PASS	8eb2b2ff 8eb2b2ff PASS
ff ff PASS	99994cff 99994cff PASS	1e331eff 1e331eff PASS	a3ccccff a3ccccff PASS
190707ff 190707ff PASS	b2b259ff b2b259ff PASS	2d4c2dff 2d4c2dff PASS	b7e5eff b7e5eff PASS
330f0fff 330f0fff PASS	cccc66ff ccccc66ff PASS	3d663dff 3d663dff PASS	ff ff PASS
4c1616ff 4c1616ff PASS	e5e572ff e5e572ff PASS	4c7f4cff 4c7f4cff PASS	111919ff 111919ff PASS
661e1eff 661e1eff PASS	ff ff PASS	5b995bff 5b995bff PASS	233333ff 233333ff PASS
7f2626ff 7f2626ff PASS	19190aff 19190aff PASS	6bb26bff 6bb26bff PASS	354c4cff 354c4cff PASS
992d2dff 992d2dff PASS	333314ff 333314ff PASS	7acc7aff 7acc7aff PASS	476666ff 476666ff PASS
b23535ff b23535ff PASS	4c4c1eff 4c4c1eff PASS	89e589ff 89e589ff PASS	597f7fff 597f7fff PASS
cc3d3dff cc3d3dff PASS	666628ff 666628ff PASS	ff ff PASS	6b9999ff 6b9999ff PASS
e54444ff e54444ff PASS	7f7f32ff 7f7f32ff PASS	c190cff c190cff PASS	7cb2b2ff 7cb2b2ff PASS
ff ff PASS	99993dff 99993dff PASS	193319ff 193319ff PASS	8ecccccff 8ecccccff PASS
190505ff 190505ff PASS	b2b247ff b2b247ff PASS	264c26ff 264c26ff PASS	a0e5eff a0e5eff PASS
330a0aff 330a0aff PASS	cccc51ff ccccc51ff PASS	336633ff 336633ff PASS	
4c0f0fff 4c0f0fff PASS	e5e55bff e5e55bff PASS	3f7f3ffff 3f7f3ffff PASS	
661414ff 661414ff PASS	ff ff PASS	4c994cff 4c994cff PASS	
7f1919ff 7f1919ff PASS	191907ff 191907ff PASS	59b259ff 59b259ff PASS	

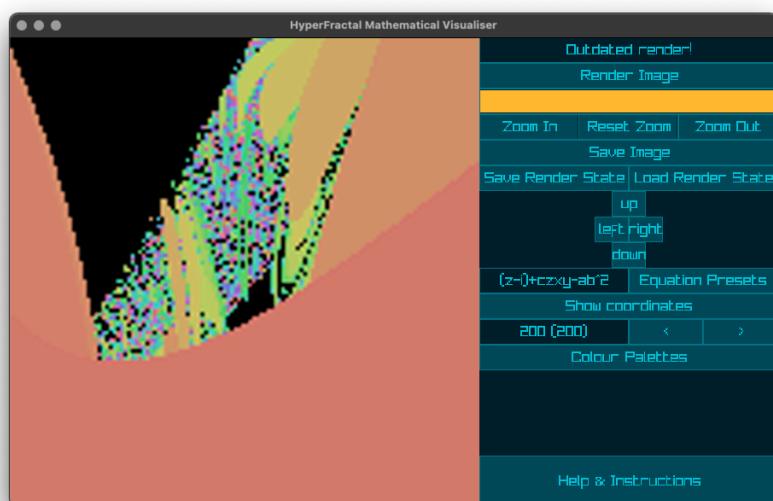
ff ff PASS	6b6bb2ff 6b6bb2ff PASS	664766ff 664766ff PASS
a1919ff a1919ff PASS	7a7acccf 7a7acccf PASS	7f597fff 7f597fff PASS
143333ff 143333ff PASS	8989e5ff 8989e5ff PASS	996b99ff 996b99ff PASS
1e4c4cff 1e4c4cff PASS	ff ff PASS	b27cb2ff b27cb2ff PASS
286666ff 286666ff PASS	c0c19ff c0c19ff PASS	cc8eccff cc8eccff PASS
327f7fff 327f7fff PASS	191933ff 191933ff PASS	e5a0e5ff e5a0e5ff PASS
3d9999ff 3d9999ff PASS	26264cff 26264cff PASS	ff ff PASS
47b2b2ff 47b2b2ff PASS	333366ff 333366ff PASS	190f19ff 190f19ff PASS
51ccccff 51ccccff PASS	3f3f7fff 3f3f7fff PASS	331e33ff 331e33ff PASS
5be5e5ff 5be5e5ff PASS	4c4c99ff 4c4c99ff PASS	4c2d4cff 4c2d4cff PASS
ff ff PASS	5959b2ff 5959b2ff PASS	663d66ff 663d66ff PASS
71919ff 71919ff PASS	6666ccff 6666ccff PASS	7f4c7fff 7f4c7fff PASS
f33333ff f33333ff PASS	7272e5ff 7272e5ff PASS	995b99ff 995b99ff PASS
164c4cff 164c4cff PASS	ff ff PASS	b26bb2ff b26bb2ff PASS
1e6666ff 1e6666ff PASS	a0a19ff a0a19ff PASS	cc7accff cc7accff PASS
267f7fff 267f7fff PASS	141433ff 141433ff PASS	e589e5ff e589e5ff PASS
2d9999ff 2d9999ff PASS	1e1e4cff 1e1e4cff PASS	ff ff PASS
35b2b2ff 35b2b2ff PASS	282866ff 282866ff PASS	190c19ff 190c19ff PASS
3dcccccff 3dcccccff PASS	32327fff 32327fff PASS	331933ff 331933ff PASS
44e5e5ff 44e5e5ff PASS	3d3d99ff 3d3d99ff PASS	4c264cff 4c264cff PASS
ff ff PASS	4747b2ff 4747b2ff PASS	663366ff 663366ff PASS
51919ff 51919ff PASS	5151ccff 5151ccff PASS	7f3f7fff 7f3f7fff PASS
a3333ff a3333ff PASS	5b5be5ff 5b5be5ff PASS	994c99ff 994c99ff PASS
f4c4cff f4c4cff PASS	ff ff PASS	b259b2ff b259b2ff PASS
146666ff 146666ff PASS	70719ff 70719ff PASS	cc66ccff cc66ccff PASS
197f7fff 197f7fff PASS	f0f33ff f0f33ff PASS	e572e5ff e572e5ff PASS
1e9999ff 1e9999ff PASS	16164cff 16164cff PASS	ff ff PASS
23b2b2ff 23b2b2ff PASS	1e1e66ff 1e1e66ff PASS	190a19ff 190a19ff PASS
28ccccff 28ccccff PASS	26267fff 26267fff PASS	331433ff 331433ff PASS
2de5e5ff 2de5e5ff PASS	2d2d99ff 2d2d99ff PASS	4c1e4cff 4c1e4cff PASS
ff ff PASS	3535b2ff 3535b2ff PASS	662866ff 662866ff PASS
21919ff 21919ff PASS	3d3dccff 3d3dccff PASS	7f327fff 7f327fff PASS
53333ff 53333ff PASS	4444e5ff 4444e5ff PASS	993d99ff 993d99ff PASS
74c4cff 74c4cff PASS	ff ff PASS	b247b2ff b247b2ff PASS
a6666ff a6666ff PASS	50519ff 50519ff PASS	cc51ccff cc51ccff PASS
c7f7fff c7f7fff PASS	a0a33ff a0a33ff PASS	e55be5ff e55be5ff PASS
f9999ff f9999ff PASS	f0f4cff f0f4cff PASS	ff ff PASS
11b2b2ff 11b2b2ff PASS	141466ff 141466ff PASS	190719ff 190719ff PASS
14ccccff 14ccccff PASS	19197fff 19197fff PASS	330f33ff 330f33ff PASS
16e5e5ff 16e5e5ff PASS	1e1e99ff 1e1e99ff PASS	4c164cff 4c164cff PASS
ff ff PASS	2323b2ff 2323b2ff PASS	661e66ff 661e66ff PASS
191919ff 191919ff PASS	2828ccff 2828ccff PASS	7f267fff 7f267fff PASS
333333ff 333333ff PASS	2d2d5eff 2d2d5eff PASS	992d99ff 992d99ff PASS
4c4c4cff 4c4c4cff PASS	ff ff PASS	b235b2ff b235b2ff PASS
666666ff 666666ff PASS	20219ff 20219ff PASS	cc3dccff cc3dccff PASS
7f7f7fff 7f7f7fff PASS	50533ff 50533ff PASS	e544e5ff e544e5ff PASS
999999ff 999999ff PASS	7074cff 7074cff PASS	ff ff PASS
b2b2b2ff b2b2b2ff PASS	a0a66ff a0a66ff PASS	190519ff 190519ff PASS
ccccccff cccccccff PASS	c0c7ffff c0c7ffff PASS	330a33ff 330a33ff PASS
e5e5e5ff e5e5e5ff PASS	f0f99ff f0f99ff PASS	4c0f4cff 4c0f4cff PASS
ff ff PASS	1111b2ff 1111b2ff PASS	661466ff 661466ff PASS
161619ff 161619ff PASS	1414ccff 1414ccff PASS	7f197fff 7f197fff PASS
2d2d33ff 2d2d33ff PASS	1616e5ff 1616e5ff PASS	991e99ff 991e99ff PASS
44444cff 44444cff PASS	ff ff PASS	b223b2ff b223b2ff PASS
5b5b66ff 5b5b66ff PASS	191919ff 191919ff PASS	cc28ccff cc28ccff PASS
72727fff 72727fff PASS	333333ff 333333ff PASS	e52de5ff e52de5ff PASS
888999ff 888999ff PASS	4c4c4cff 4c4c4cff PASS	ff ff PASS
a0a0b2ff a0a0b2ff PASS	666666ff 666666ff PASS	190219ff 190219ff PASS
b7b7ccff b7b7ccff PASS	7f7f7fff 7f7f7fff PASS	330533ff 330533ff PASS
cecee5ff cecee5ff PASS	999999ff 999999ff PASS	4c074cff 4c074cff PASS
ff ff PASS	b2b2b2ff b2b2b2ff PASS	660a66ff 660a66ff PASS
141419ff 141419ff PASS	ccccccff cccccccff PASS	7f0c7fff 7f0c7fff PASS
282833ff 282833ff PASS	e5e5e5ff e5e5e5ff PASS	990f99ff 990f99ff PASS
3d3d4cff 3d3d4cff PASS	ff ff PASS	b211b2ff b211b2ff PASS
515166ff 515166ff PASS	191619ff 191619ff PASS	cc14ccff cc14ccff PASS
66667fff 66667fff PASS	332d33ff 332d33ff PASS	e516e5ff e516e5ff PASS
7a7a99ff 7a7a99ff PASS	4c444cff 4c444cff PASS	ff ff PASS
8e8eb2ff 8e8eb2ff PASS	665b66ff 665b66ff PASS	f1919ff f1919ff PASS
a3a3ccff a3a3ccff PASS	7f727fff 7f727fff PASS	1e3333ff 1e3333ff PASS
b7b7e5ff b7b7e5ff PASS	998999ff 998999ff PASS	2d4c4cff 2d4c4cff PASS
ff ff PASS	b2a0b2ff b2a0b2ff PASS	3d6666ff 3d6666ff PASS
111119ff 111119ff PASS	ccb7ccff ccb7ccff PASS	4c7f7fff 4c7f7fff PASS
232333ff 232333ff PASS	e5cee5ff e5cee5ff PASS	5b9999ff 5b9999ff PASS
35354cff 35354cff PASS	ff ff PASS	6bb2b2ff 6bb2b2ff PASS
474766ff 474766ff PASS	191419ff 191419ff PASS	7acccfff 7acccfff PASS
59597fff 59597fff PASS	332833ff 332833ff PASS	89e5e5ff 89e5e5ff PASS
6b6b99ff 6b6b99ff PASS	4c3d4cff 4c3d4cff PASS	ff ff PASS
7c7cb2ff 7c7cb2ff PASS	665166ff 665166ff PASS	c1919ff c1919ff PASS
8e8eccff 8e8eccff PASS	7f667fff 7f667fff PASS	193333ff 193333ff PASS
a0a0e5ff a0a0e5ff PASS	997a99ff 997a99ff PASS	264c4cff 264c4cff PASS
ff ff PASS	b28eb2ff b28eb2ff PASS	336666ff 336666ff PASS
f0f19ff f0f19ff PASS	cca3ccff cca3ccff PASS	3f7f7fff 3f7f7fff PASS
1e1e33ff 1e1e33ff PASS	e5b7e5ff e5b7e5ff PASS	4c9999ff 4c9999ff PASS
2d2d4cff 2d2d4cff PASS	ff ff PASS	59b2b2ff 59b2b2ff PASS
3d3d66ff 3d3d66ff PASS	191119ff 191119ff PASS	66ccccff 66ccccff PASS
4c4c7fff 4c4c7fff PASS	332333ff 332333ff PASS	72e5e5ff 72e5e5ff PASS
5b5b99ff 5b5b99ff PASS	4c354cff 4c354cff PASS	

Evaluation

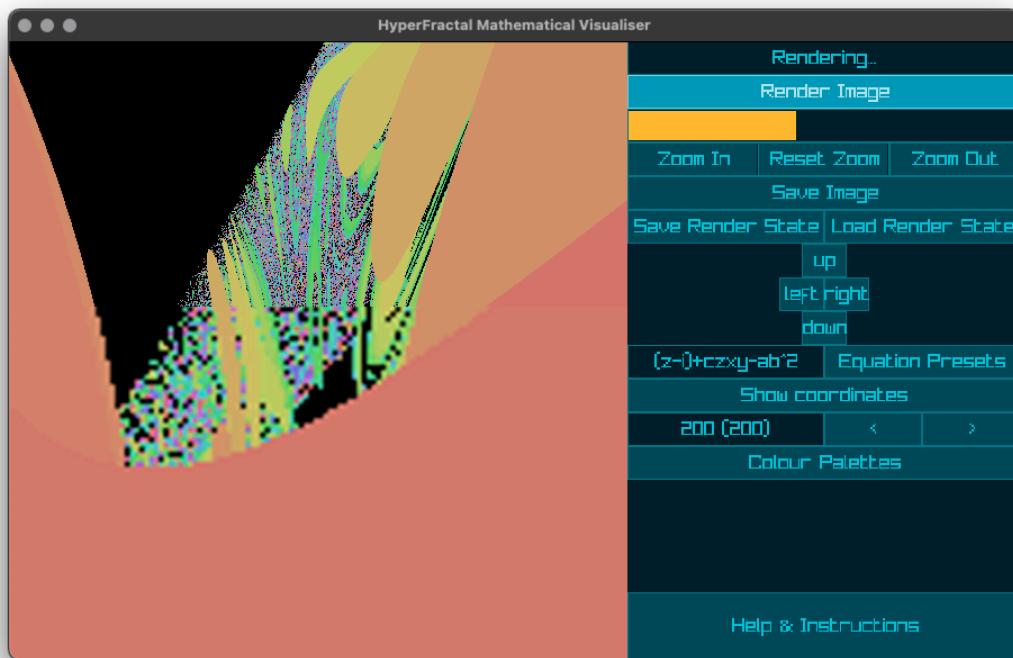
Having targeted each of my high-level requirements already during the testing process, the primary focus of my evaluation of this project will be demonstrating and reviewing the usability of it by interviewing my user and demonstrating the final application working. Below are screenshots showing the application working, using most of its features including saving and loading render states, rendering images, zooming in and out, navigating, and changing the equation.



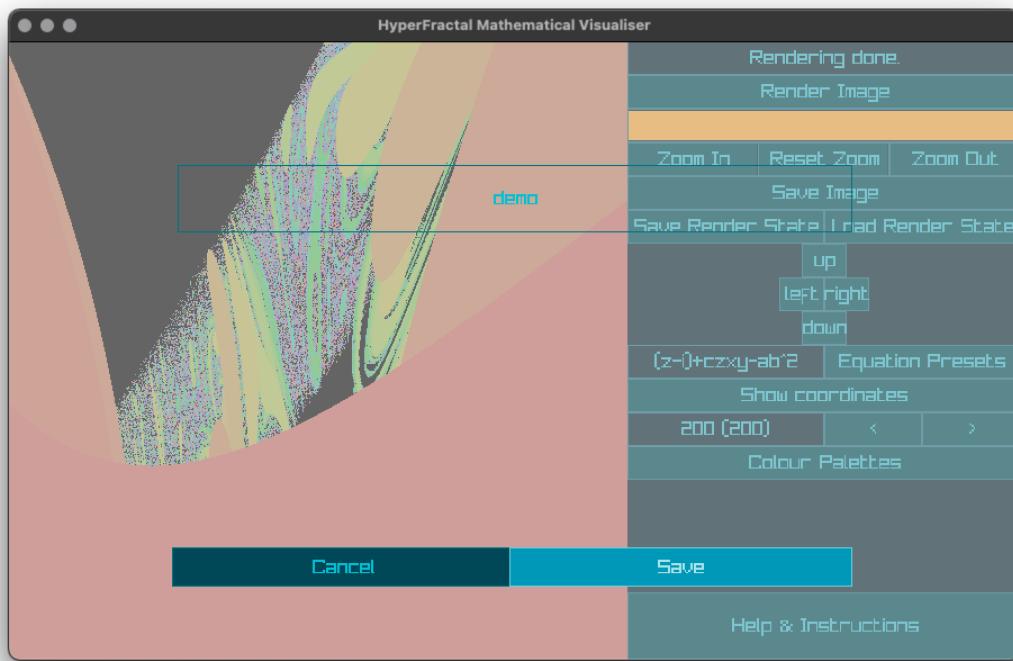
1. Immediately after starting the application, the preview render is shown using default parameters (the Mandelbrot Set, 200 iterations, zoom=0.75, x offset=0, y offset=0).



2. Typing in a custom equation to the application, the preview render is shown until the 'Render Image' button is pressed



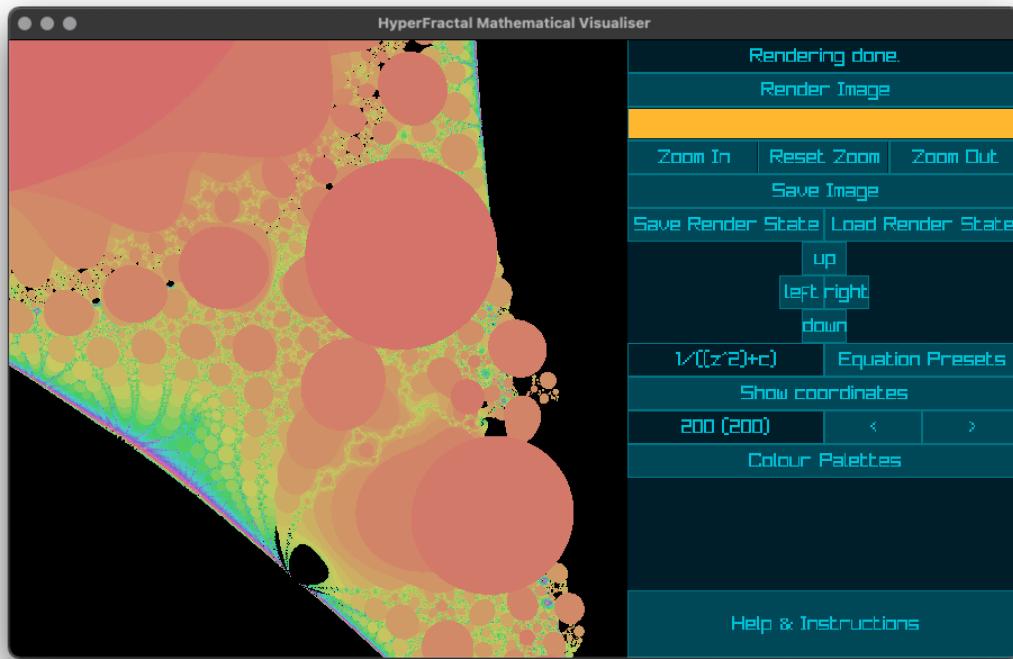
3. The application mid-render, after pressing the ‘Render Image’ button



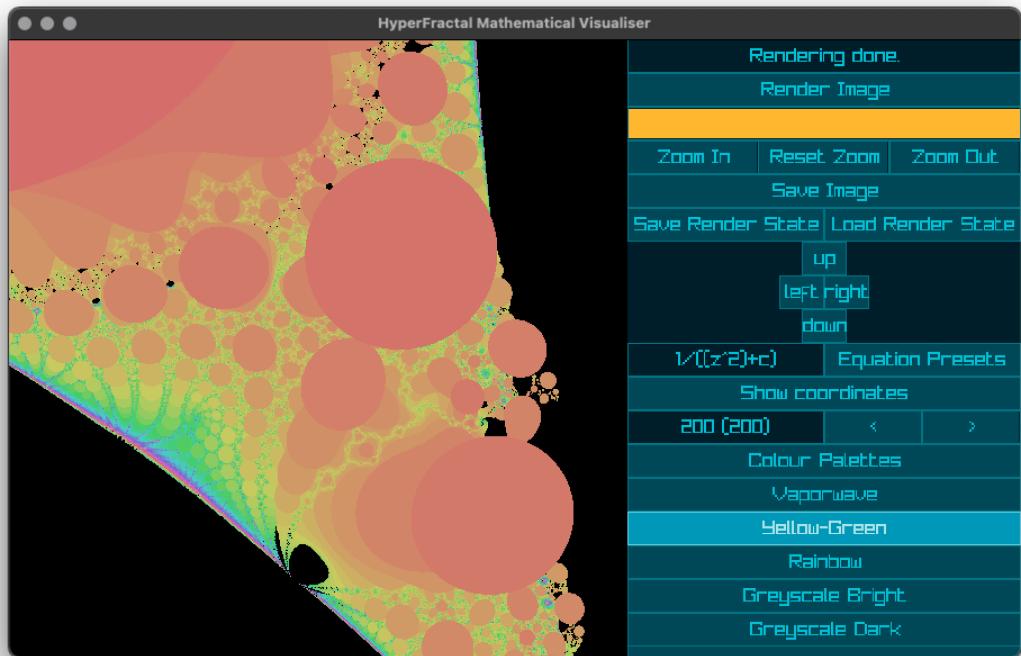
4. After clicking the ‘Save Render State’ button, we save the current render parameters, including the custom equation, under the name ‘demo’



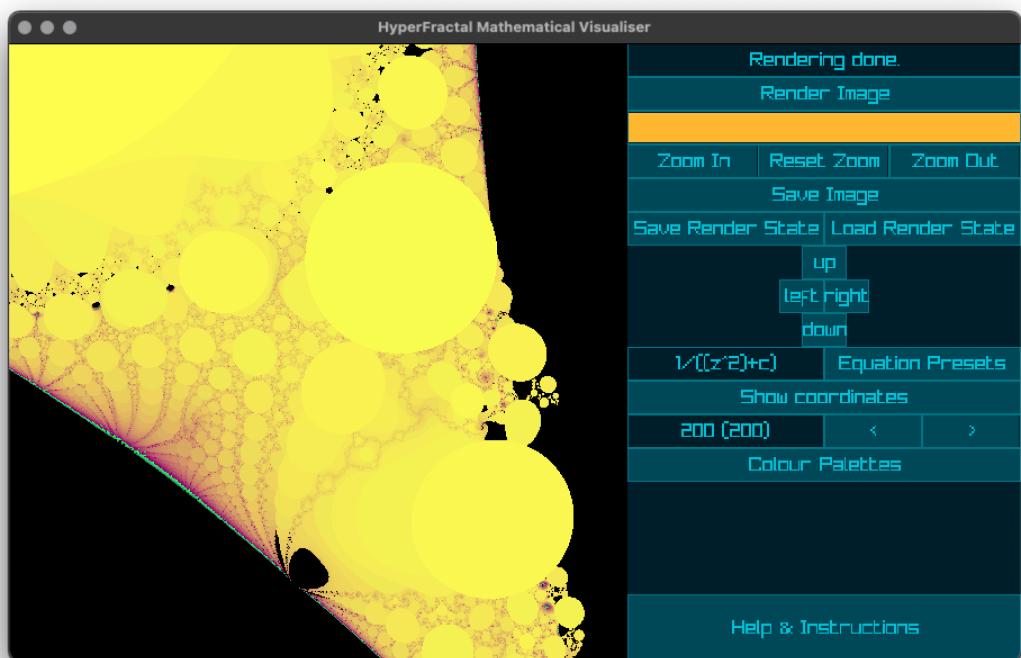
5. The equation changes after selecting the ‘Reciprocal’ equation from the ‘Equation Presets’ dropdown



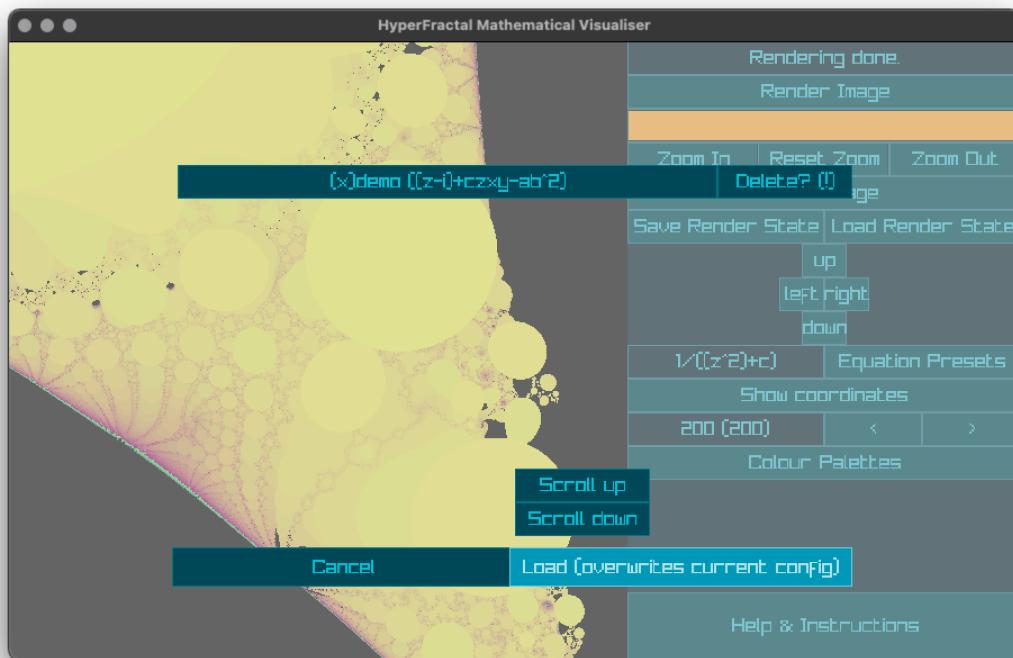
6. Pressing ‘Render Image’ produces a full-resolution render in under 5 seconds



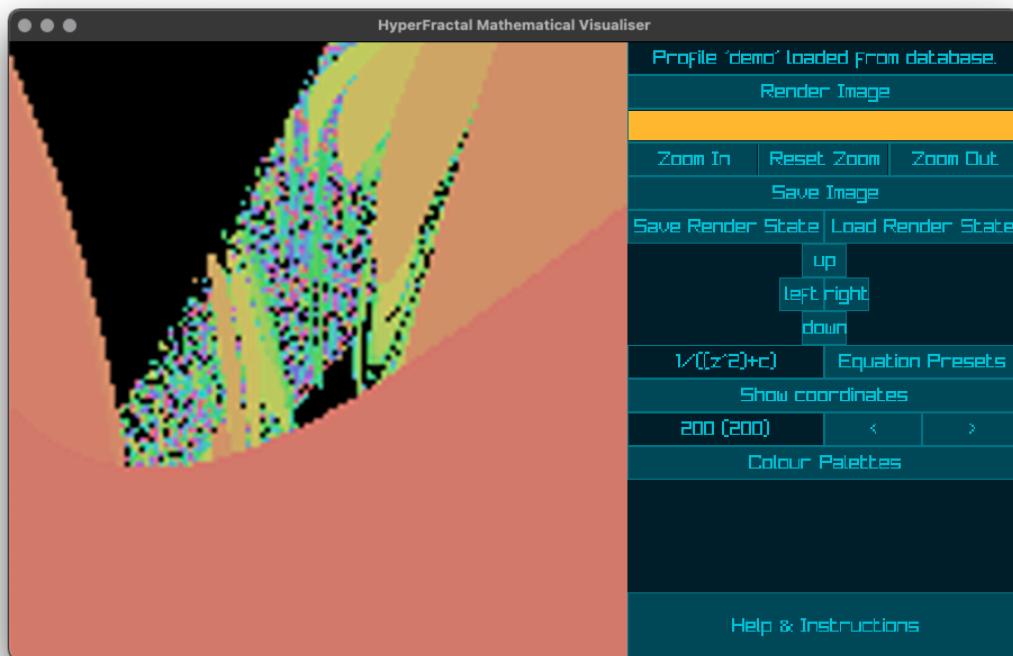
7. Clicking the 'Colour Palettes' button shows a list of available colour palettes



8. The colour palette has changed to be the one selected



9. Opening the configuration loading dialog in order to reload the 'demo' profile



10. Successfully reloading the 'demo' profile from the database

User Feedback

I gave my finished application to my maths teacher (my primary client) to use. After a brief introduction and them reading the instructions which come with the application, they very quickly got the hang of navigating using the interface, learning to use both the interface buttons and keyboard shortcuts with almost no direction from me. They found it easy and intuitive to switch between equation presets and colour palettes, and even started using the ability to save and load render states from the database. This clearly demonstrates that my interface design is easily usable for a teacher. It also demonstrated the interface's robustness when faced with a completely new user. For instance, the interface locking during a render feature was used at least once as my user tried to navigate the fractal during a render a few times. Thus, my program clearly satisfies high-level objective 1-08, as the interface is intuitive to my user.

One point that my maths teacher mentioned was how quickly images were rendered out. This is very good, as it indicates that the render times possible with the application are appropriate for an average user to tolerate (most users will not want to sit around waiting for a render to appear), and that my initial requirement on the render time for a fractal, under 30 seconds (high-level objective 1-07), was appropriate.

My maths teacher particularly liked the ability to select different colour schemes for renders, and quickly saw the usefulness of being able to save and load render configurations for demonstrations during lessons. As such, I'm very pleased with my solution as it clearly satisfies the requirements and expectations of my client.

Improvements and Further Evaluation

Overall, I'm very pleased with the completeness of this project. One thing which particularly surprised me was how efficient the rendering code is, despite it being relatively simple. As shown in my tests, standard renders using built-in equations happen in as little as 3 seconds, which I count as a major success.

However, there are a number of improvements I would have liked to have made. First, I would like to have expanded on the database management system, potentially adding the ability to export and import configuration profiles to share them between users. On top of this, I would have liked to be able to save a low-resolution preview of each profile along with the database, in order to make it easier for the user to find the profile they want. I would also have liked to add the ability to search the database by name or by other parameters (such as the equation being used). Here I would also have added the ability to create multiple user profiles and manage the database in a more central location (e.g. 'Program Files' on Windows or 'Application Support' under macOS).

Second, I would have liked to add support for saving PNGs. The application only saves images to a standard bitmap format, which is readable but does not show the colourful patterns shown in the interface. I was disappointed that I did not have time to implement this more complex method, but I felt that given the primary use of this application is in the GUI, the usage of this feature would be limited anyway.

Finally, during the development of my technical solution, I came up with an idea for a more efficient way of assigning pixels to computation threads, which was to assign them 'blocks' from the image, which they would gradually fill in. This would eliminate any blocking when threads were attempting to get new pixels from the image to be computed, potentially having a significant performance improvement. On top of this, I could have taken the time to write a GPU kernel in order to utilise host machine GPU capabilities. However, bearing in mind that I would have had to learn this from scratch, and that the majority of the target computers (school-issued teacher laptops) would have extremely limited, if not non-existent GPUs, I decided to stay with standard CPU based computation.

Appendices

Appendix 1 - Help Document

The following pages show the 'Help & Instructions' document which explains how to use the application.

Help & Instructions

Feeling lost? Below is an explanation of how to use the GUI mode of the application.

Rendering

When you start the application or make changes to parameters such as zoom, equation, centering, etc, you will be shown a low-resolution preview render in the viewport on the left side of the window.

By clicking on the 'Render Image' button in the control panel, the application will then generate a screen-resolution image in the viewport, during which time parameter controls are locked. Rendering progress is shown below the 'Render Image' button.

Parameters

Parameters determine the image which is output. These include the following:

- Zoom - allows you to look closer at particular areas by stretching the mathematical space on both axes - can be altered using the 'Zoom In', 'Reset Zoom', 'Zoom Out' buttons, or the '+' and '-' buttons on the keyboard
- Offset - allows you to move the viewport around the fractal by offsetting the mathematical space on one or more axes - can be altered using the 'up', 'down', 'left', 'right' buttons, or the arrow keys
- Equation - the iterative mathematical expression which is used to generate values for each pixel - clicking in the equation input box and typing allows you to alter this: **see dedicated help section**
- Iteration limit - limits the number of times the equation is iterated before the program assumes it does not tend to infinity - can be altered using the '<' and '>' buttons, or using the left and right square bracket keys (accelerate by holding Shift)
- Palette - can be selected from presets using the 'Colour Palettes' button

Saving and Loading Render States

The application allows you to load or save render states. A render state is a configuration profile containing all the rendering parameters used, including zoom

offsets, iteration limit, palette, etc.

In order to save a render state, click the 'Save Render State' button, click the input box in the middle of the screen, enter a descriptive name, and press 'Save'.

To load a render state, click the 'Load Render State' button, select an available saved profile from the list, and click 'Load'. This will overwrite current settings, so if you have a nice configuration currently showing, make sure to save it first before loading another. The loading dialog also gives you the option to delete saved profiles if you want, but be careful because this is permanent.

Equations

Fractals are generated by iteratively applying a formula to a complex number and counting the number of times it can be iterated before tending toward infinity. This count is then used to compute a colour for a given pixel.

Formulas (also referred to as expressions or equations) are in the format $Z = f(Z, C)$, where Z is referring to the iterated value, and C is constant. On the first iteration, Z and C are both initialised to be equal to a complex number, with components $a+bi$. a and b are the horizontal and vertical coordinates (respectively) of the pixel being computed, meaning each pixel has different initial conditions for computation.

The equation field in the application can handle brackets ((. . .)), indices (^), division (/), multiplication (*), addition (+) and subtraction (-), and processes them in that order. It supports the use of z and c as basic variables, as well as x and y , which behave as the real and imaginary part of z , and a and b , which represent the real and imaginary parts of c . It also supports the use of numerical (decimal) constants which can be in terms of i .

There is no limit to equation length or complexity, but more complex equations are likely to be more computationally expensive and increase render time significantly.

The application also contains a number of equation presets, which can be fun to explore and are good starting points if you want to come up with your own equation. Presets run much faster than custom entered equations due to them being hard-coded.

When writing your own equations, it is advised to replace z^2 with $z*z$, as the latter evaluates much faster in a computational environment. The same can be done for any integer power of a function. In general, *you are advised to simplify your expression to a reasonable extent before entering it into the application, in order to reduce render time.*