Real-Time Fractal Rendering

Non-Examined Assessment
Optimised Fractal Rendering in C++

Toby Davis

Hampton School

March 17, 2023

Abstract

Write this bad boi when the thing is finished :)

Summary

1 Project Analysis

1.1 A Brief Introduction to Fractals

A fractal is "a curve or geometrical figure, each part of which has the same statistical character as the whole"[1].

Some fractals are defined by simple equations which exhibit chaotic behaviour. Arguably the most famous fractal, the Mandelbrot Set, is defined by the following iterative equation, where $Z_0 = 0 + 0i$ and C is the initial value in the complex plane.

$$Z_{n+1} = Z_n^2 + C \quad : \quad Z_n, c \in \mathbb{C} \tag{1}$$

For a given point C to be in the Mandelbrot Set, the value of Z_n must remain bounded (i.e. not diverge to infinity) after the iterative series is repeated infinitely many times. This approach is used in most iterative fractal equations.

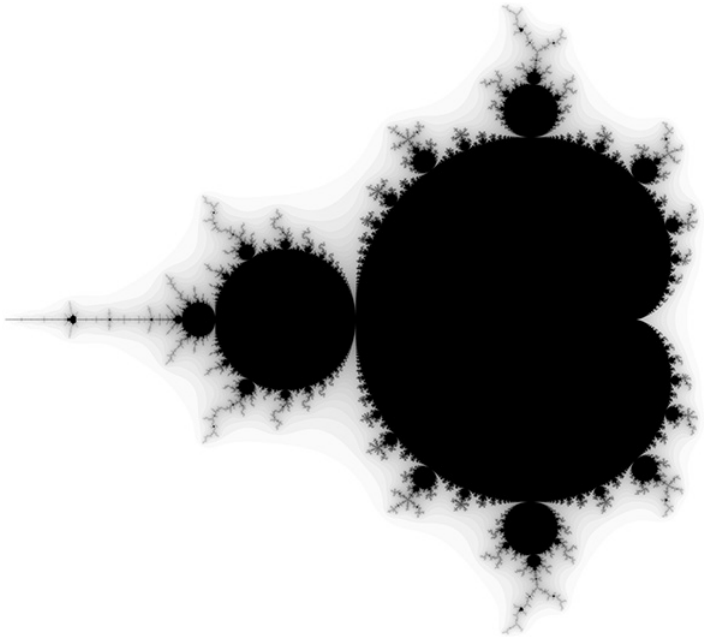


Figure 1: The standard Mandelbrot fractal[2]

Additionally, fractal variations can be created by changing the generating equation slightly. For example, changing the r in the Mandelbrot equation ($Z_{n+1} = Z_n^r + C$) yields the following fractals.

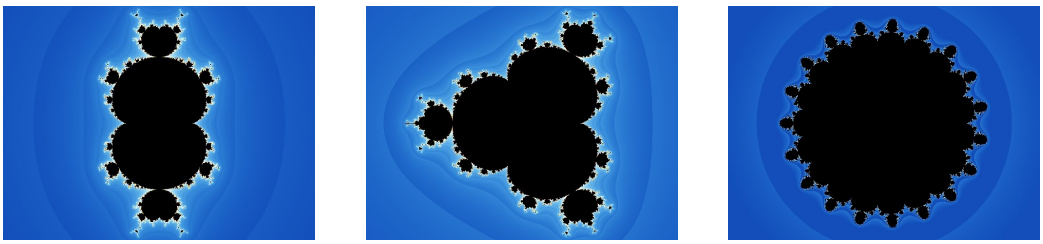


Figure 2: (left) $r = 3$, (centre) $r = 4$, (right) $r = 20$

Other famous fractals include the Sierpiński Triangle, the Julia Set, Hilbert Spirals, etc. All are defined either by infinitely-recursive self-similar patterns or repeated equations.

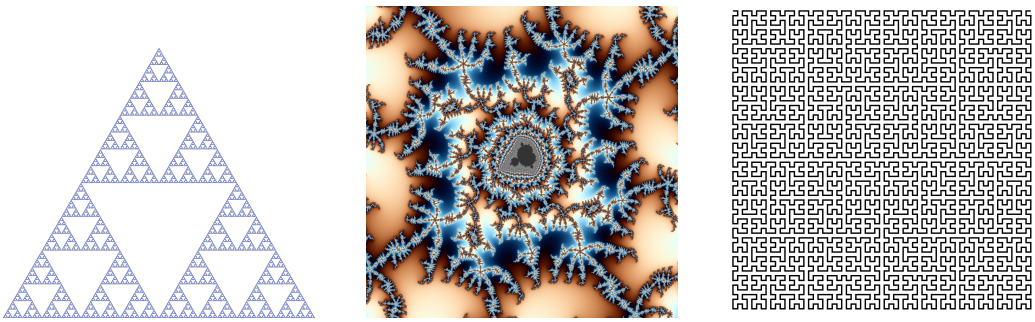


Figure 3: (left) Sierpiński-Triangle, (centre) Part of the Mandelbrot Set, (right) Hilbert Curve

1.2 Defining the Problem

Fractals have been the subject of much debate and curiosity throughout history. However, due to the computational requirements of generating them, research into them was minimal until the rise of the electronic computer.

The newfound processing power allowed increasingly detailed images to be generated, and mathematicians could better understand fractals' underlying equations and seemingly chaotic nature.

With the power of modern computers, it is possible to render some fractals in real-time and explore them to great depths, though there are still technical, physical and monetary hurdles to clear.

Many fractal rendering programs exist online; however, most are incomplete, inefficient applications not designed for high performance and increased zoom factors. While many high-quality applications exist, the best ones are often quite expensive, making them inaccessible to most potential users. For example, some extremely advanced software costs almost £80[3].

Technological Limitations The further you zoom into a fractal, the smaller the numbers you have to deal with. At lower zoom levels, this doesn't pose much of an issue, as 64 bit or even 32 bit floating point numbers often have the required precision to render an image accurately. At higher zoom levels, however, the precision of the numbers used affects the image quality.

When zoomed in far enough, floating point rounding errors start to cause certain pixel positions to merge into one, causing unattractive "blocky" patterns in the image. Eventually, these blocks will consume the entire image, and no more detail can be seen.

To get around this, it is possible to use high-precision floating point data types. However, since these are processed in software, not hardware, they are orders of magnitude slower than normal number types, which can make the rendering process impractically slow.

Some techniques can be taken to optimise the performance of high-performance number types. For example, it can be proven that Z_n will diverge to infinity if $|Z_n| > 2$ at any point. Additionally, advanced algorithms can mix fixed and multi-precision arithmetic to decrease the number of operations performed in software.

Program Limitations Many fractal renderers do just that; render fractals. They don't support any render export features and do not allow for saving, reloading or sharing render configurations.

Some programs have methods to save the rendered fractals as image files but often have limiting export settings and don't support many resolutions. Some programs allow the current position and zoom level, among other information, to be exported to a file, allowing interesting fractal locations to be shared easily.

Precision vs Performance To increase rendering performance, most implementations of fractal renderers use 32- or 64-bit floating point numbers. Since operations on these data types are performed directly by hardware, they are highly efficient. Unfortunately, 64-bit floating point values can only accurately represent around 15 decimal places, so zooming in far enough will exceed this precision and cause visual glitches.

To circumvent this issue, it is possible to use multi-precision floating point types capable of representing hundreds, thousands or even millions of bits, allowing for near-infinite zooms. These numbers, however, are implemented in software and are many orders of magnitude slower than standard floating point types. It is possible to use multi-precision floating point arithmetic for sufficiently optimised programs, though the performance will be abysmal.

Furthermore, some areas of different fractals require a considerable number of iterations before a reasonable amount of detail can be obtained. As a result, potentially millions of calculations must be done to determine the colour for a single pixel.

The two main issues above become even more extreme when combined with the goal of near-infinite zooming. Due to the nature of many fractals, the number of iterations required to get high levels of detail in areas close to the border of the fractal increases with zoom. Additionally, deeper zooms need higher precision numbers to represent all the points accurately. Combine these, and the result is a slow, inefficient program.

1.3 The End User

Mathematics Teachers and Professors could use the program to assist in their lessons and provide students with an interactive resource to help with homework and further their understanding. This could dramatically increase students' engagement in studies and inspire them to pursue further degrees in mathematics. Additionally, less well-off schools could afford and use the software if the program is free and open source, increasing its accessibility.

Researchers and General Acedmia could use the high precision, deep zooms and fast renders to further their studies on the properties of fractals. Furthermore, the more advanced fractal export tools could be used to share the exact configurations of the areas they explore to accelerate the peer-review process.

Anyone Interested in Mathematics could explore fractals' beauty and share the rendered images with their friends and family. If the program is suitably intuitive, even young children would be able to use it, potentially inspiring an interest in mathematics.

1.4 Analysis of Existing Programs

1.4.1 David J. Eck's Online Mandelbrot Renderer [4]

David Eck's online Mandelbrot rendering program implements many nice-to-have features, including the ability to export the current render settings as an XML file, intuitive controls and various configuration settings. The user can easily change the colour palette, image resolution, the number of threads to use, and more.

This implementation also supports a multi-precision floating point type, which allows the renderer to zoom in "infinitely". Unfortunately, this renderer is written in Javascript rather than a faster language like Java or, better yet, C++; as a result, it can take a long time to render an image, especially at higher resolutions and quality settings.

Interface The interface for this renderer is quite primitive and, while intuitive, is not pleasant to use. The box select to zoom in can be frustrating to use and often results in poor framing since it zooms in immediately after releasing the mouse.

The status indicator shows the current render progress and is extremely limited. It shows the current pass of the renderer, the precision it is using and the number of rows completed, but there is nothing showing the estimated time remaining, the speed at which it is rendering, or the time elapsed since the render started.

Configuration While the renderer allows for the major settings to be changed, such as the maximum number of iterations to perform, the colour palette and the number of threads to use, many of the more advanced settings cannot be changed. For advanced applications, it is often useful to know *exactly* where the frame is centred in fractal space. An arguably more important feature is the ability to specify a location and scaling factor directly, instead of zooming in manually.

On the other hand, the ability to revert to default settings, combined with the lists of predefined settings for each option, make the program much more accessible for the less experienced. This feature is a must-have for a program aimed at a wide audience.

Render Quality While the user can specify the resolution at which to render the image, it must fit on the screen you are using. You cannot render a higher-resolution image to a file, for example. Furthermore, there are no options to configure anti-aliasing with this renderer, which means the image quality may not be as high as required for some purposes.

1.4.2 XaoS.js Online Mandelbrot Renderer [5]

The Fractal Foundation *XaoS.js* Mandelbrot renderer is relatively primitive, using only machine-precision floating point arithmetic and no options to configure the fractal. On the other hand, the renderer is intuitive, with a simple click-to-zoom interface, making it ideal for less knowledgeable users who want to explore fractals.

However, the *XaoS.js* renderer does implement some complex rendering algorithms. It maintains existing pixel information from the previous frame and uses it to refine the next one, creating a smoother transition progressively. Unfortunately, these rendering techniques result in significant artefacts when sufficiently zoomed in (though the limited number of iterations performed means the fractal is unrecognisable at this point).

1.4.3 XaoS Offline Fractal Renderer [6]

The Fractal Foundation *XaoS* fractal renderer is a much more advanced, offline version of the previously examined program. Since it is free and open-source, many people have contributed code and developed its rendering algorithms. As a result, it can render fractals much faster than other programs. However, the advanced techniques used cause artefacts to appear in the final renders, making it unsuitable for academic or research purposes since it is not a true likeness of the fractal.

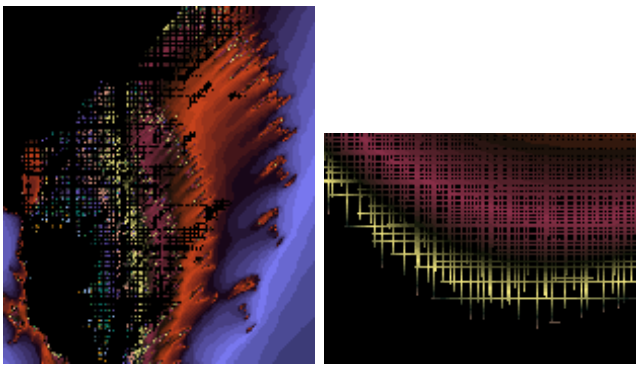


Figure 4: Artefacting in XaoS' renderer

Interface *XaoS* has the same intuitive controls as the online implementation, but all the configuration options are hidden in awkward menus at the top of the screen.

Configuration With enough searching, almost every parameter about the fractal can be changed, including filters such as edge detection and anti-aliasing. This is incredibly useful for advanced users, as it enables them to adjust the appearance of the fractal to their exact needs, emphasising the features they are investigating.

File Export *XaoS* allows the user to export a configuration file or save the current render as an image. The configuration file contains the information required to reconstruct the image in the renderer, simplifying the sharing of fractals between users. The image export saves the current pixels of the screen to a file, meaning the image's resolution is limited to the size of the window. This is unfortunate for those who may want to save a high-resolution image but cannot make the window large enough to support this.

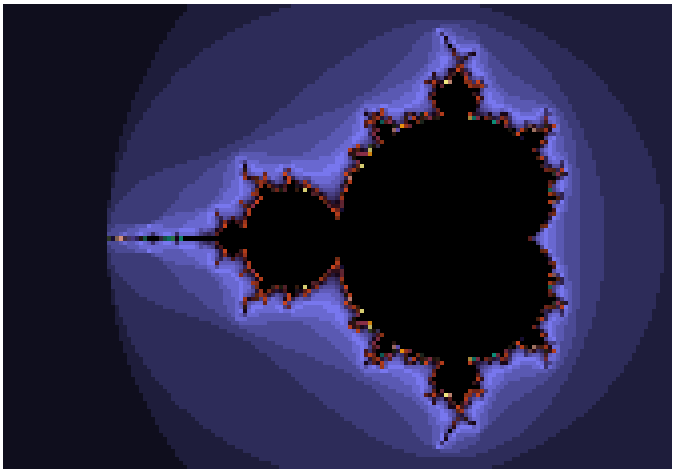


Figure 5: A low-quality image saved from XaoS

Supported Fractals The *XaoS* renderer supports 25 common fractals by default and can render simple user-defined fractals. While nice to have, this feature is unnecessary for a program to implement, assuming it supports at least two fractal types. Additionally, the user-defined fractals tend to render much slower than the built-in ones since optimised algorithms can be developed for them.

Precision *XaoS* uses fixed-precision arithmetic to perform calculations; hence, you cannot zoom into the fractals indefinitely. Given the performant nature of the program, it is unfortunate that this is not a feature since it could outcompete most other rendering programs.

1.5 Program Requirements

1. Rendering
2. Configuration
3. Interface and Movement
4. Import and Export
5. Installation

6. Performance

ID	Description	Importance
1.1	The program can render a fractal correctly	**HIGH**
1.2	Fractals can be rendered at high resolutions without artifacting	**HIGH**
1.3	The fractal can be coloured to bring out details	**HIGH**
1.4	Colouring algorithms can be isolated from the fractal rendering process, allowing different algorithms to be implemented more easily	**HIGH**
1.5	Anti-aliasing can be used to reduce noise and produce a cleaner image	**MEDIUM**
1.6	The background colour of the fractal can be changed	**LOW**
1.7	Fractal algorithms are optimised for the data type used	**MEDIUM**
1.8	Simple optimisations are made to accelerate the rate at which fractals are rendered	**MEDIUM**
1.9	Images can be rendered with high-precision floating point types, allowing for “infinite” zooms	**HIGH**
2.1	The area of the fractal currently being rendered, as well as the zoom factor, can be changed	**HIGH**
2.2	The number of threads used to render the fractal can be changed	**MEDIUM**
2.3	The maximum number of iterations allowed can be changed	**HIGH**
2.4	The bailout value can be changed	**LOW**
2.5	The anti-aliasing factor can be changed	**HIGH**
2.6	The image size can be changed independently of the image resolution	**HIGH**
2.7	The image resolution can be changed	**HIGH**
2.8	The colouring algorithm can be changed and customised	**MEDIUM**
2.9	Settings can be reset to default values	**HIGH**
2.10	The option to undo/redo changes to settings	**LOW**
2.11	Floating point precision can be customised	**HIGH**
2.12	Different fractals can be rendered	**MEDIUM**
2.13	Each fractal has predefined default settings which are loaded when a new fractal is selected	**LOW**

2.14	Settings are loaded from a JSON file at program startup	LOW
3.1	There is a graphical user interface (GUI)	HIGH
3.2	The GUI is fast and responsive	HIGH
3.3	Similar settings and options are contained in a single window which can be moved around the screen	MEDIUM
3.4	Input and numeric information fields should handle data to the current precision used by the program	HIGH
3.5	Not all menus are shown initially, and settings with different complexities can be shown or hidden	MEDIUM
3.6	Different workspaces can be selected from a menu, configuring the windows and settings shown for different levels of understanding – beginner, intermediate, advanced	LOW
3.7	The area to zoom into can be selected with the mouse	HIGH
3.8	The zoom box does not need to match the aspect ratio of the fractal	LOW
3.9	The zoom box can be moved and scaled after its creation, with the option to apply the zoom after the user is happy with it	MEDIUM
3.10	The current render progress, render time, render speed and estimated time remaining are displayed	HIGH
3.11	There is a history of previous frames rendered which can be reverted to	LOW
3.12	There is a way to zoom back out of the fractal	HIGH
3.13	The fractal should be rendered progressively, allowing the user to see roughly what is being rendered without having to wait for the full image	LOW
3.14	The current location can be copied to the clipboard easily	LOW
3.15	Any numeric input fields should accept scientific input formats	HIGH
4.1	The render configuration settings can be loaded from a JSON file at runtime	MEDIUM
4.2	The render configuration settings can be saved to a JSON file, allowing for easy sharing	MEDIUM
4.3	Images can be saved to a file	HIGH
4.4	The saved images can have a user-defined filetype, and are not limited to, for example, `*.png`	LOW

4.5	Images can be rendered separately from the main GUI, allowing incredibly high-resolution images to be saved	**POSSIBILTIY**
5.1	Program compiles on Windows	**HIGH**
5.2	Program compiles on MacOS	**HIGH**
5.3	Program compiles on Linux	**HIGH**
5.4	Program compiles with `msvc`	**HIGH**
5.5	Program compiles with `gcc/g++`	**HIGH**
5.6	Program compiles with `clang`	**HIGH**
5.7	Program is easy to compile from source with CMake (i.e. `cmake --build . --config Release`)	**HIGH**
5.8	Prebuilt executable is available for Windows	**MEDIUM**
5.9	Prebuilt executable is available for MacOS	**MEDIUM**
5.10	Prebuilt executable is available for Linux	**MEDIUM**
6.1	The program can render fractals in a reasonable time (under 2 seconds) on a single thread with machine word precision	**HIGH**
6.2	Multiple threads can be used to accelerate the rendering process	**HIGH**
6.3	The rendering algorithm runs on a separate thread to the GUI, ensuring the interface continues to refresh quickly	**HIGH**
6.4	Where possible, calculations are optimised to suit the data type being operated on	**LOW**
6.5	Some simple optimisations are implemented to accelerate the rendering of the fractals	**MEDIUM**
6.6	Low-quality images can be rendered quickly with multi-precision data types	**MEDIUM**

2 Design Phase

2.1 Third Party Libraries

Cinder [7] *Cinder* is a free, open-source graphics engine for C++. It provides a simple way to access OpenGL, ImGui and other tools, such as image loading and saving, optimised rendering in 2D and 3D, and more. I am using *Cinder* for this project instead of doing all the graphics processing with raw OpenGL because it dramatically simplifies the code and reduces the scope for hard-to-fix bugs.

This project uses a modified version of *Cinder* with updated libraries and a few extra features. Most significantly, this modified version includes a much newer version of ImGui and an altered build configuration to fix common compile errors on some platforms.

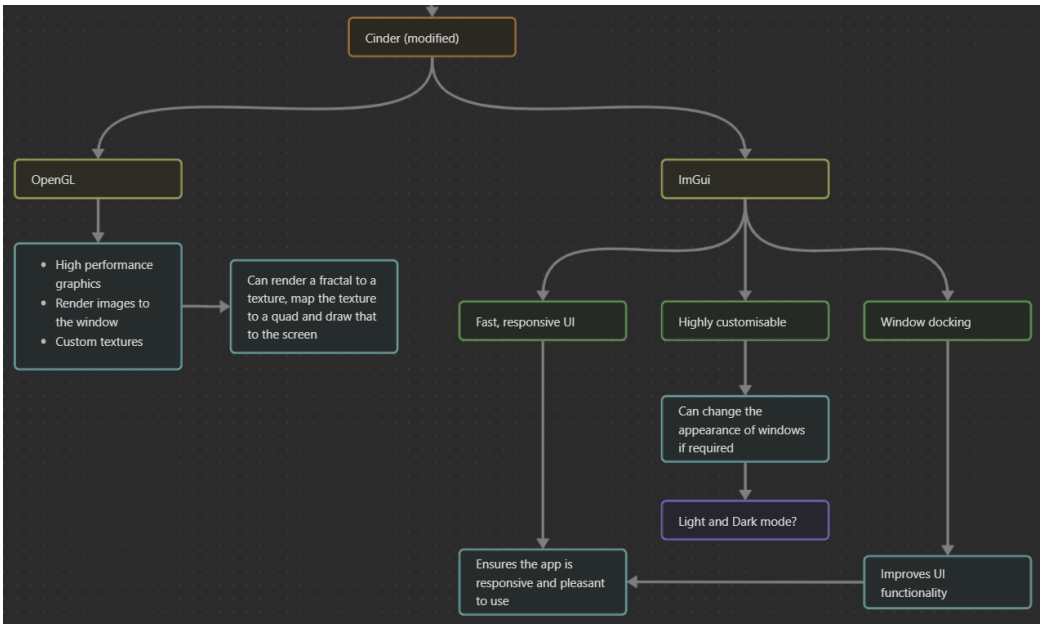
LibRapid [8] *LibRapid* is a high-performance library for mathematical applications, including optimised vector classes, complex number types and general mathematical functions. However, this library's most helpful feature is its support for MPIR and MPFR, which are highly-optimised multi-precision implementations. This will allow floating point calculations with more than 64 bits.

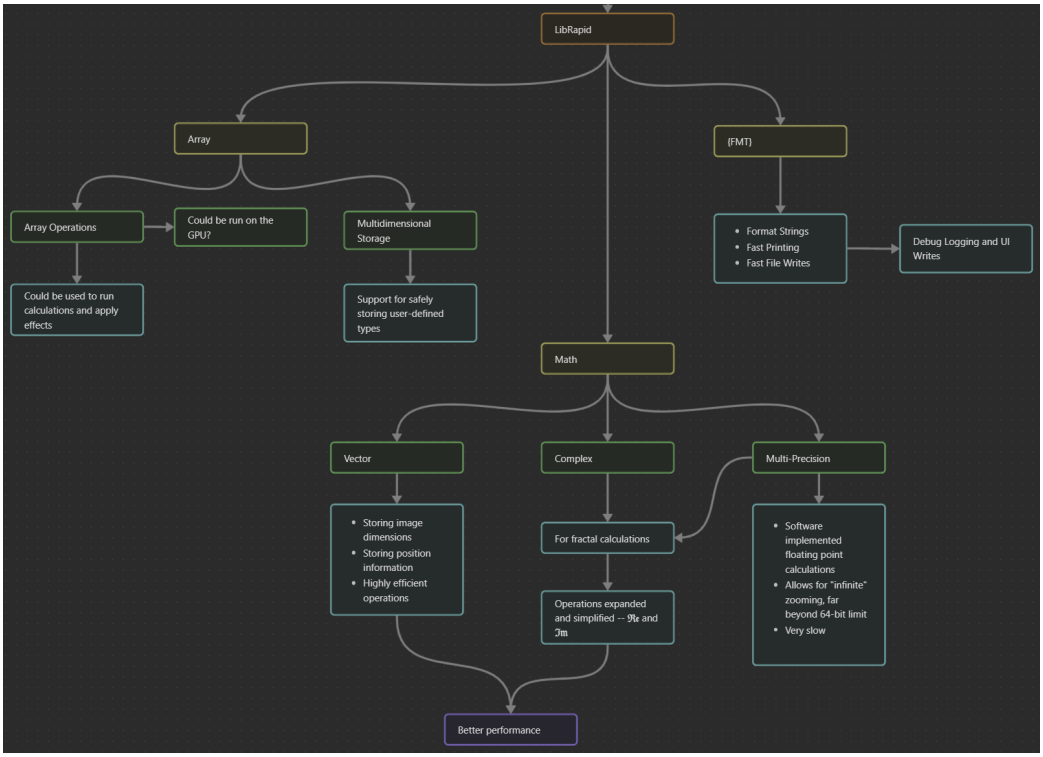
Incorporating an efficient multi-precision implementation into the project could allow for "infinite" fractal zooms since traditional floating-point limitations would no longer constrain the software.

Another feature of *LibRapid* used heavily in this project is the compiler- and system-agnostic macro definitions. Useful features like inlining of functions, no-discard specifiers and more are not implemented by all compilers and sometimes work differently on different operating systems. LibRapid implements macros which automatically detect the relevant information and define the most suitable replacement. This isn't strictly required for the project, but it might result in a slight performance improvement and can help reduce bugs.

Cinderbox Both of the afore mentioned libraries are packaged with *Cinderbox* for simple integration into *CMake* projects.

2.2 Library Heirarchy


Cinder (modified)
OpenGL
ImGui
High performance graphics
Render images to the window
Custom textures
Can render a fractal to a texture, map the texture to a quad and draw that to the screen
Fast, responsive UI
Highly customisable
Window docking
Can change the appearance of windows if required
Light and Dark mode?
Ensures the app is responsive and pleasant to use
Improves UI functionality


LibRapid
Array
{FMT}
Array Operations
Could be run on the GPU?
Multidimensional Storage
Format Strings
Fast Printing
Fast File Writes
Debug Logging and UI Writes
Could be used to run calculations and apply effects
Support for safely storing user-defined types
Math
Vector
Complex
Multi-Precision
Storing image dimensions
Storing position information
Highly efficient operations
For fractal calculations
Software implemented floating point calculations
Allows for "infinite" zooming, far beyond 64-bit limit
Very slow
Operations expanded and simplified -- ℜe and ℑm
Better performance

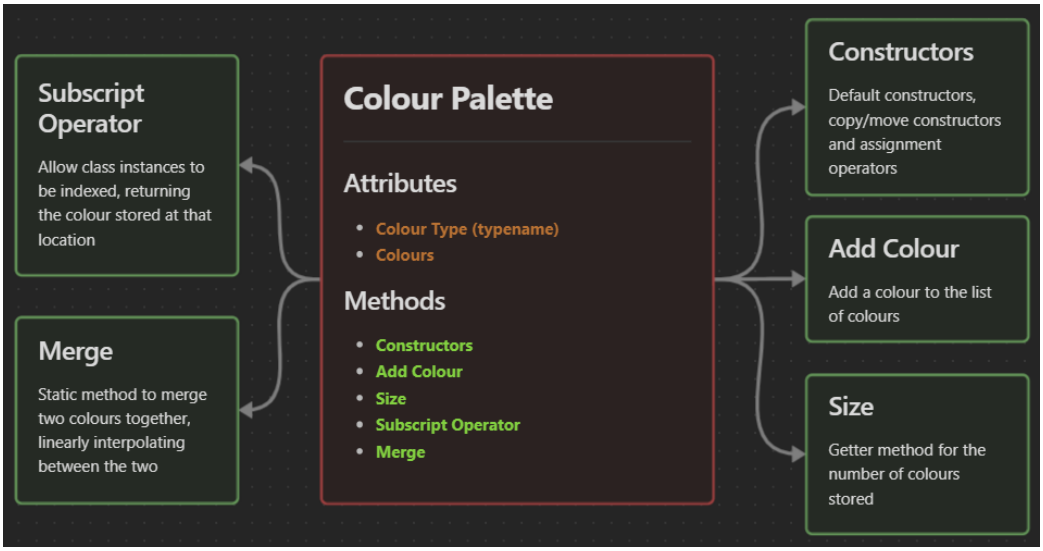
2.3 The Debug Logger

Since the program is written in C++and is a GUI application instead of a console application, there will not be a usable standard output to which debug information can be printed. To circumvent this issue, I will use a debug logger instance to write information to a file.

The logger's constructor will take a file path relative to the executable and attempt to open the specified file. If the document does not exist, it will be created for the user. The destructor will ensure that all buffers are flushed, and the file is closed. Without these checks, the program may terminate without saving the changes to be written to the file, and the debug log might be incomplete or corrupt. The logger will also have a priority level, optimising logging in release builds, as the user does not need all the information. For example, the logger could be configured to write only errors to the file in release mode. Finally, the logger has a function which enables the user to send data to be written to the file. New lines should be formatted appropriately, and logs should be timestamped. In addition to these functions, I will create a macro that captures the log statement's line number and filename, making tracebacks easier and faster during development.

2.4 Colour Palettes

A simple class containing a list of colours and a few helper methods is helpful for storing the colours and gradients used by the fractal rendering process. This class simplifies the act of colour palette generation and usage throughout the software, reducing bugs and improving the rate of development.



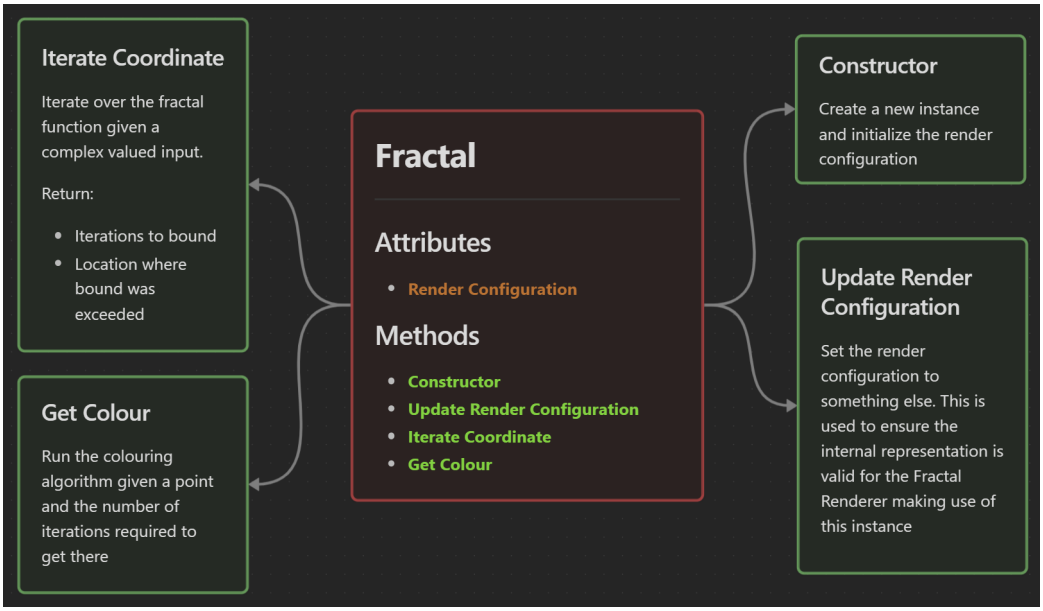
The colour merging function is a very simple static method of the class, which linearly interpolates between the colour's red, green and blue components.

$$\left\{ \quad t \times (R - r) + r, \quad t \times (G - g) + g, \quad t \times (B - b) + b \quad \right\} \tag{2}$$

Where t is the interpolation factor and $0 \leq t \leq 1$.

2.5 The Fractal Class

To support multiple fractal equations at runtime, each fractal will be implemented as a class inheriting from a main parent type. This is the fractal data type. It defines the functions required to iterate the fractal's equation from a given starting value, the logic to generate a colour from a starting point, an endpoint, and the number of iterations required to get there.



For example, the code below shows the definition of the Mandelbrot fractal class.

```
01  #pragma once
02
03  #include <fractal/genericFractal.hpp>
04
05  namespace frac {
06      class Mandelbrot : public Fractal {
07      public:
08          /// Constructor taking a RenderConfig object
09          /// \param config RenderConfig object
10          explicit Mandelbrot(const RenderConfig &config);
11          Mandelbrot(const Mandelbrot &)   = delete;
12          Mandelbrot(Mandelbrot &&)    = delete;
13          Mandelbrot &operator=(const Mandelbrot &) = delete;
14          Mandelbrot &operator=(Mandelbrot &&) = delete;
15
16          ~Mandelbrot() override = default;
17
18          LIBRAPID_NODISCARD std::pair<int64_t, lrc::Complex<LowPrecision>>
19          iterCoordLow(const lrc::Complex<LowPrecision> &coord) const
                override;
20
21          LIBRAPID_NODISCARD std::pair<int64_t,
```

```
            lrc::Complex<HighPrecision>>
22          iterCoordHigh(const lrc::Complex<HighPrecision> &coord) const
                override;
23      };
24  } // namespace frac
```

2.5.1 Render Box States

An enum of valid states is required to keep track of each render box's current state. This is drawn to the main window on top of the fractal as it renders, providing the user with information about which areas are rendered, which are actively rendering and which areas are yet to be processed.

```
1  /// Represents the state of a render box
2  enum class RenderBoxState {
3      None, // Not yet assigned a state
4      Queued, // Queued to be rendered
5      Rendering, // Currently being rendered
6      Rendered // Rendered and ready to be written to the image
7  };
```

2.5.2 Render Boxes

The position information required to render a small area of the main fractal is contained within a `RenderBox` struct. These can be passed to a function inside the `FractalRenderer` class to be processed.

```
1  /// Stores the pixel-space coordinates of a region to render
2  struct RenderBox {
3      lrc::Vec2i topLeft;
4      lrc::Vec2i dimensions;
5      RenderBoxState state = RenderBoxState::None;
6      double renderTime = 0;
7  };
```

2.5.3 Render Box Statistics

To calculate the remaining time of the render and the fastest and slowest render box times, the program needs to know precisely how long each box took to render. To improve efficiency, a separate struct is created to store this data.

```
struct RenderBoxTimeStats {
    double min = 0;
    double max = 0;
    double average = 0;
    double remainingTime = 0;
};
```

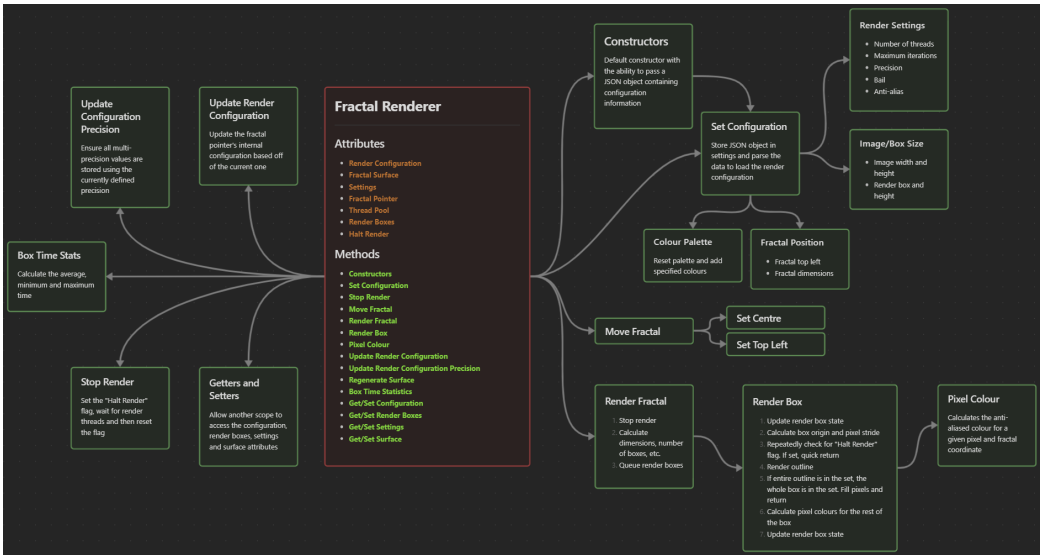
2.5.4 Render Configurations

Arguably the most important helper class, the `RenderConfig` struct contains all the information required for a `FractalRenderer` instance to render an image. The information in this struct can also be saved to a JSON file and shared, allowing people to send specific configurations between users easily.

```
struct RenderConfig {
    int64_t numThreads; // Number of threads to render on (max)
    int64_t maxIters; // Largest number of iterations to allow
    int64_t precision; // Precision (in bits) of floating point types
        used for arithmetic
    LowPrecision bail; // Bailout value
    int64_t antiAlias; // Anti-aliasing factor -- 1 = no anti-aliasing
    lrc::Vec2i imageSize; // Size of the image to render
    lrc::Vec2i boxSize; // Size of sub-regions to render (see RenderBox)
    lrc::Vec<HighPrecision, 2> fracTopLeft;  // The fractal-space center
        of the image
    lrc::Vec<HighPrecision, 2> fracSize;  // The width and height of the
        fractal space
    lrc::Vec<HighPrecision, 2> originalFracSize; // Original size for
        zoom factor calculation
    ColorPalette palette; // The palette to use for rendering the fractal
};
```

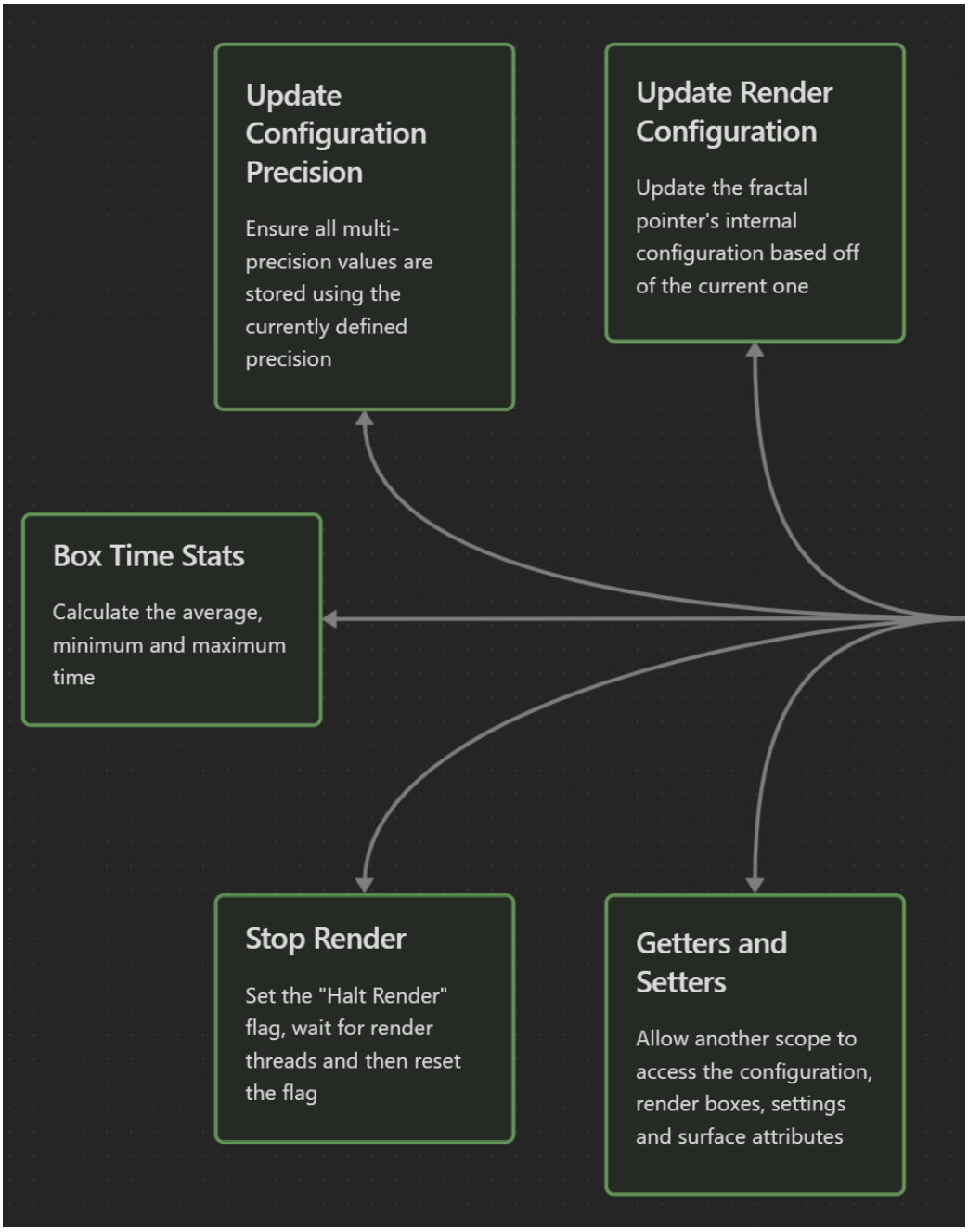
2.6 The Fractal Renderer Class

While it is essential to have a method of calculating the colour of a given point on the fractal (from the fractal class), it doesn't support rendering an entire image. This is the role of the fractal renderer class.



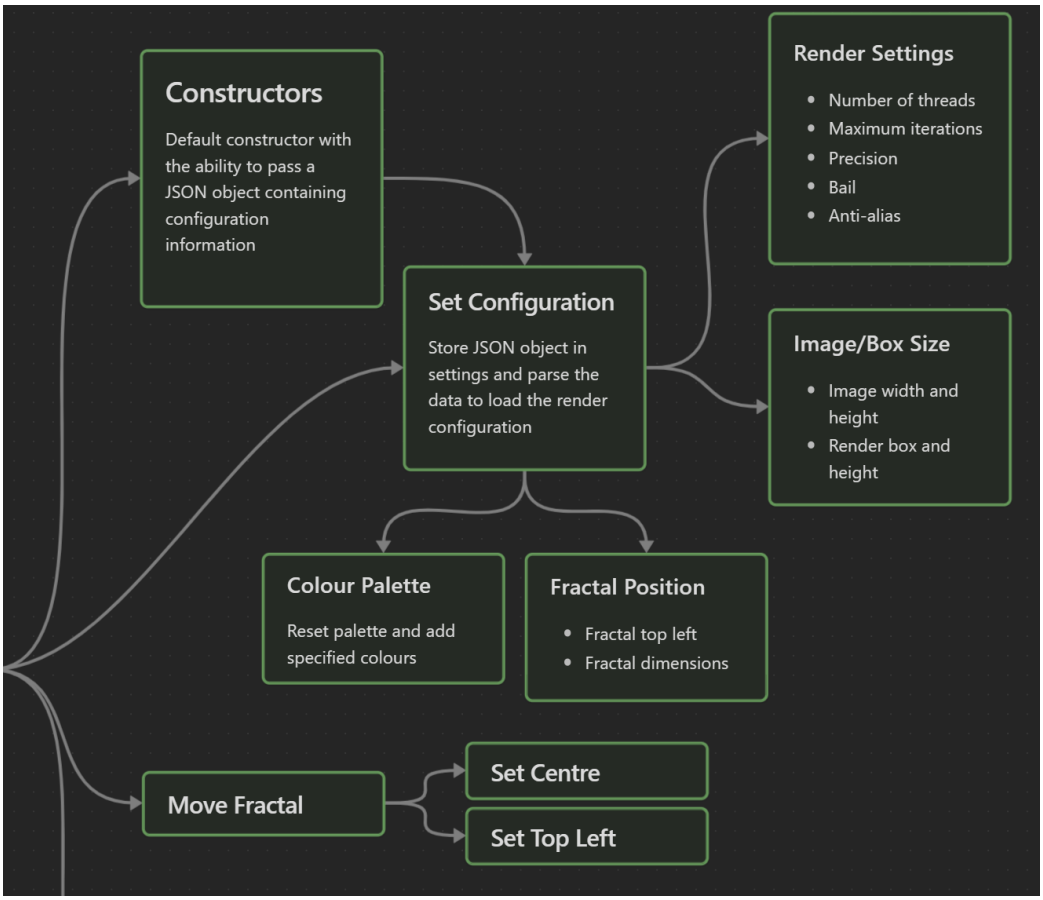
This class implements many functions at various levels of abstraction, allowing performance-critical sections of the code to be run with efficient, parallelised algorithms. At the same time, the high-level interfaces are easy to interact with and use.

2.6.1 Configuration, Getters, Setters and Statistics


Update Configuration Precision
Ensure all multi-precision values are stored using the currently defined precision
Update Render Configuration
Update the fractal pointer's internal configuration based off of the current one
Box Time Stats
Calculate the average, minimum and maximum time
Stop Render
Set the "Halt Render" flag, wait for render threads and then reset the flag
Getters and Setters
Allow another scope to access the configuration, render boxes, settings and surface attributes

These functions operate at a high level, allowing basic access to the information stored by the class. While simple, they are essential for the main program to function correctly, as there would be no way of accessing the rendered image, for example.

2.6.2 Constructors and Configurations



To render a fractal, much information is required about the dimensions of the image, the dimensions of the fractal, the origin in the complex plane, and more.

The fractal renderer class can parse a JSON object and load its configuration. By storing some data as a string instead of a number, it is possible to save and load high-precision numbers as well – this could be used to enable

fractal locations with extremely high zoom factors to be saved and shared easily.

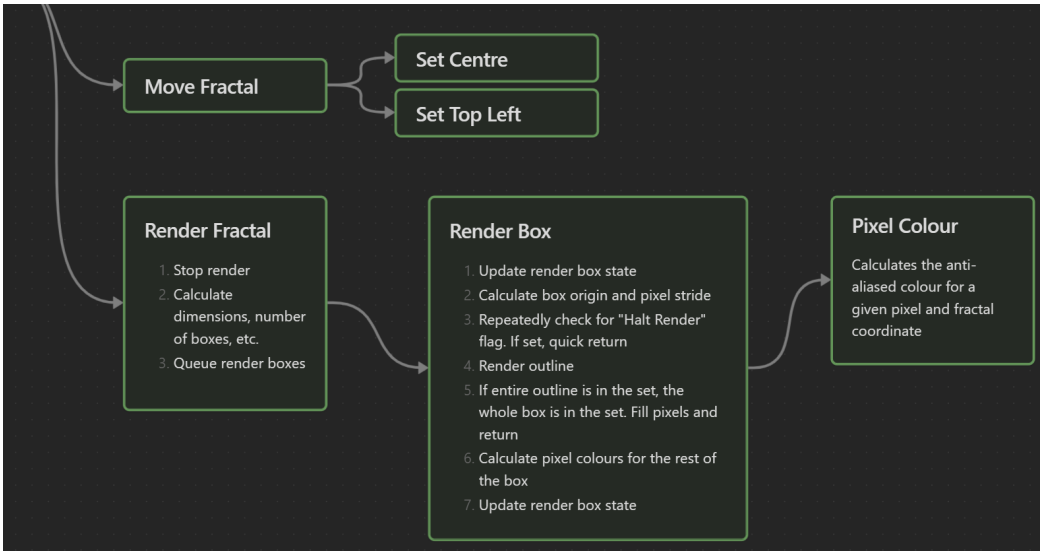
The JSON snippet below shows a highly simplified version of the default configuration used within the software (for a full version, see the code-listing at the end of the document).

```
{
    "renderConfig": {
        "numThreads": 8,
        "maxIters": 500,
        "precision": 64,
        "bail": 65536,
        "antiAlias": 2,
        "imageSize": {
            "width": 800,
            "height": 700
        },
        "colorPalette": [
            {
                "red": 0.5568628,
                "green": 0.23137255,
                "blue": 0.27450982,
                "alpha": 1.0
            },
            {
                "red": 0.88235295,
                "green": 0.8666667,
                "blue": 0.56078434,
                "alpha": 1.0
            }
        ]
    }
}
```

A vast number of configuration options can be configured in the file, including the threading options, render quality settings and even the size of the boxes to render in parallel.

The `loadConfiguration` method also enables the configuration to be changed or re-parsed at runtime, allowing quick and easy updates to the fractal settings.

2.6.3 Rendering Algorithms



The fractal renderer is responsible for creating an image buffer and setting the colour of each pixel in that image to represent the fractal at that location. Doing this efficiently is difficult, so the process is split into many small functions, providing fine-grained control over the algorithm.

Upon requesting the renderer to re-render the image, any existing render threads are halted, dependent image settings are recalculated, and the render-box queue is cleared.

Next, the render boxes are recalculated and pushed back to the render queue. Each box is dequeued by a thread in a thread pool and runs in parallel.

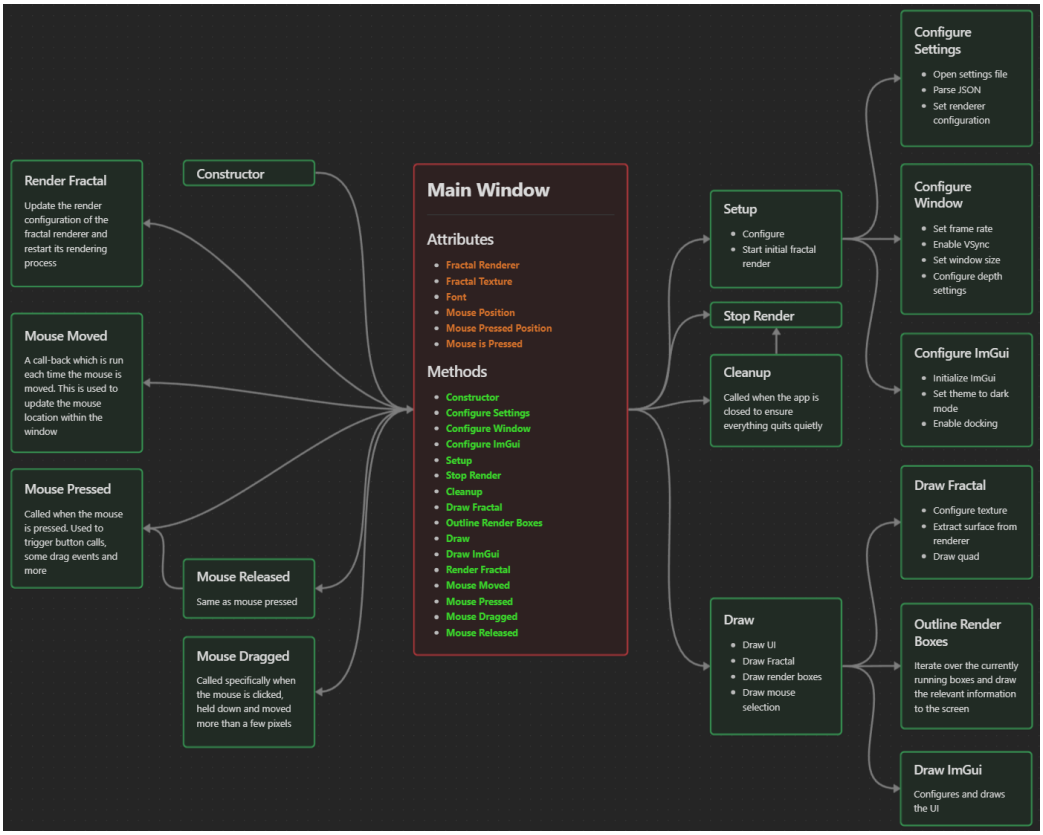
To optimise the rendering of each box, we can use the fact that if an outline can be drawn where every point is in the set, every point contained within that outline must also be within the set. Outlining each box before calculating the inner area makes it possible to check whether all points were in the set. If they were, we can quickly fill the rest of the box without calculating the colour of each pixel.

To calculate the colour of each pixel, we first call the `iterCoord` method of the fractal pointer stored to get the number of iterations required to exceed the bailout value (if it is exceeded at all), as well as the first point at which this occurs. This information is then passed to the fractal's colouring algorithm to generate a colour for the pixel. If anti-aliasing is enabled, this process is repeated for multiple points within the pixel, and the resulting colours are averaged. Anti-aliasing produces smoother images that appear to be higher

resolution, allowing for faster render times with high-quality results.

The fractal renderer class also implements routines to change the fractal-space coordinates to render. This is used to move the fractal when zooming in or out. For different use cases, there are methods to set the top left coordinate or the image's centre.

2.7 The Main Window



This class is responsible for creating, managing and drawing information to the window and controlling all the other classes mentioned previously.

When the window is created by *Cinder*, the `setup` routine is called, initializing the window's attributes. First, the settings JSON file is loaded, parsed and passed to the fractal renderer member. Next, the window itself is constructed and configured, including the framerate, enabling or disabling vertical syncing

(V-Sync), the window size and OpenGL depth buffer settings. Finally, *ImGui* is configured.

The `MainWindow` class is also responsible for drawing to the window. After setting the background colour, the *ImGui* windows are created and drawn. This produces most of the UI, but some extra parts must be drawn in later. Next, the fractal itself is drawn to the screen. An image texture is created and assigned the fractal image buffer, and a rectangle is drawn with the aforementioned texture. The rest of the UI is now drawn, including the render-box status indicators and the mouse selection.

When the application is requested to close, the `cleanup` routine is called. This signals any existing render threads to halt, ensures all members are correctly destructed and then destroys the window.

The window also has a variety of callback functions which are called when specific triggers occur. For example, functions are called every time the mouse is moved, dragged or pressed.

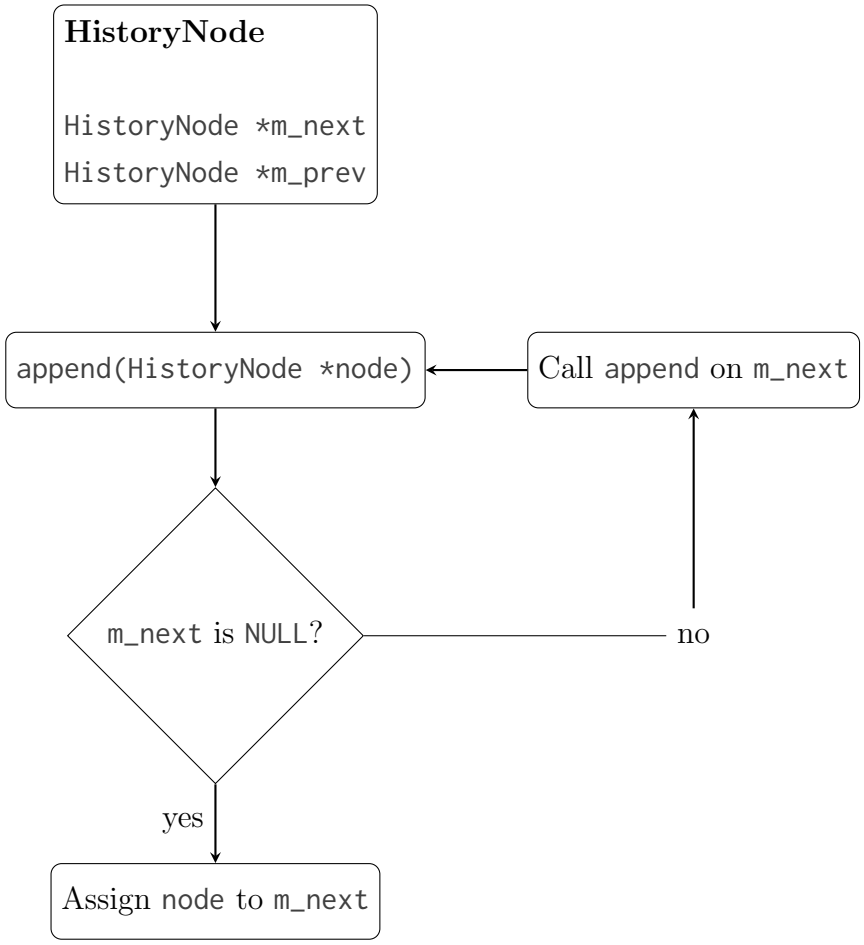
2.8 Additional Features

The features outlined previously comprise the vast majority of the program, but some additional features must also be designed and implemented. These are often the less critical features, though they still impact the user's interaction with the program.

2.8.1 Movement History

While we often take the "undo" and "redo" buttons for granted, they give us a powerful means of reverting unwanted changes. Furthermore, they allow the user to see what adjustments have been made to the fractal between movements, supporting verbal descriptions of how to arrive at fascinating points within the fractal.

Using a linked list, where each node stores a `RenderConfig` object and a `Surface`, it is possible to implement a move history into the program. Whenever the user requests to change a setting or move the origin of the fractal, the current surface and render configuration are appended to the history before the changes are applied. The algorithm to append to the linked list can be seen below.



The `MainWindow` class also stores a pointer to the current node in the history buffer. Using this, it is possible to detect whether there are changes before or after the current node by checking the validity of the `previous` and `next` pointers respectively.

When the user makes a change, the current node is checked against the end of the history buffer. If there are no changes after the current state, the new state is appended directly to the end of the buffer. However, if changes exist, it is necessary to destroy the rest of the tree with the `killChildren` method before appending the new configuration. To undo moves, the current

node is moved backwards down the list.

To make these commands as intuitive as possible, the undo and redo commands are triggered by pressing `ctrl+z` and `ctrl+shift+z` respectively. These are very common commands and will be easy to remember.

To show the user the current history, a small tab is rendered on the right side of the window. Each item in the history buffer is shown by drawing the saved surface, with the current item outlined to make it more obvious where in the history tree you are.

Mouse events within this tab are also supported, such as scrolling and clicking on frames to select items in the history.

Scrolling is implemented with a single offset which is increased or decreased as the mouse wheel is moved. The rate at which the window scrolls is defined by the `settings[''menus''][''history''][''scrollSpeed'']` value in the settings JSON file.

2.8.2 Improved Selection Area

To improve the user-interface, the zoom selection box should not zoom in immediately after being drawn. Once the box is specified, it should be possible to drag the box around to more precisely specify where the zoom will occur. Additionally, keyboard inputs (for example, the arrow keys) should move the selected region.

To support dragging the box with the mouse, adding the mouse position delta between each frame to the current box's position maintains the mouse's location within the box and provides intuitive movement. The code below shows a simple implementation.

```
01  void MainWindow::mouseDrag(ci::app::MouseEvent event) {
02      if (ImGui::GetIO().WantCaptureMouse) return;
03
04      lrc::Vec2i delta = lrc::Vec2i(event.getPos()) - m_mousePos;
05      m_mousePos  = event.getPos();
06
07      if (m_moveZoomBox) {
08          m_zoomBoxStart += delta;
09          m_zoomBoxEnd += delta;
10      }
11  }
```

3 Technical Solution

3.1 Color Palette (`colorPalette.hpp`)

```
01 #pragma once
02
03 namespace frac {
04     class ColorPalette {
05     public:
06         using ColorType = lrc::Vec<float, 4>;
07
08         ColorPalette()        = default;
09         ColorPalette(const ColorPalette &)   = default;
10         ColorPalette(ColorPalette &&)    = default;
11         ColorPalette &operator=(const ColorPalette &) = default;
12         ColorPalette &operator=(ColorPalette &&) = default;
13
14         /// Append a new colour to the palette
15         /// \param color The colour to add
16         void addColor(const ColorType &color);
17
18         /// Return the number of colours in the palette
19         /// \return The number of colours in the palette
20         LIBRAPID_NODISCARD size_t size() const;
21
22         /// Indexing operator (const)
23         /// \param index The index of the colour to return
24         /// \return The colour at the given index
25         const ColorType &operator[](size_t index) const;
26
27         /// Indexing operator (non-const)
28         /// \param index The index of the colour to return
29         /// \return The colour at the given index
30         ColorType &operator[](size_t index);
31
32         /// Linearly interpolate between two colours
33         /// \param a First colour
34         /// \param b Second colour
35         /// \param t Interpolation factor
36         /// \return The interpolated colour
37         static ColorType merge(const ColorType &a, const ColorType &b,
               float t);
38
39     private:
40         std::vector<ColorType> m_colors;
```

```
    };
} // namespace frac
```

3.2 Color Palette (`colorPalette.cpp`)

```
#include <fractal/fractal.hpp>

namespace frac {
    void ColorPalette::addColor(const ColorType &color) {
        m_colors.push_back(color);
        FRAC_LOG(fmt::format("Adding Color: {} {} {} {}", color.x(),
            color.y(), color.z(), color.w()));
    }

    size_t ColorPalette::size() const { return m_colors.size(); }

    const ColorPalette::ColorType &ColorPalette::operator[](size_t
        index) const {
        if (index > m_colors.size())
            FRAC_ERROR(fmt::format(
              "Index {} out of bounds for ColorPalette with size {}",
                  index, m_colors.size()));
        return m_colors[index];
    }

    ColorPalette::ColorType &ColorPalette::operator[](size_t index) {
        if (index > m_colors.size())
            FRAC_ERROR(fmt::format(
              "Index {} out of bounds for ColorPalette with size {}",
                  index, m_colors.size()));
        return m_colors[index];
    }

    ColorPalette::ColorType ColorPalette::merge(const ColorType &a,
        const ColorType &b, float t) {
        return a + (b - a) * t;
    }
} // namespace frac
```

3.3 Debug Logger (debug.hpp)

```
01 #pragma once
02
03 #define FRAC_LOG(message) \
04     ::frac::debugLogger.write(message, ::frac::Priority::Info, FILENAME,
       __LINE__)
05 #define FRAC_WARN(message) \
06     ::frac::debugLogger.write(message, ::frac::Priority::Warning,
       FILENAME, __LINE__)
07 #define FRAC_ERROR(message) \
08     ::frac::debugLogger.write(message, ::frac::Priority::Error,
       FILENAME, __LINE__)
09
10 namespace frac {
11     // A way to specify the verbosity of the debug logger
12     enum class Priority { Info = 0, Warning = 5, Error = 10 };
13
14     /// A logger type that can write debug information to a file
15     class DebugLogger {
16     public:
17         /*
18          * Since there should only ever be a single DebugLogger
               instance, delete the
19          * majority of the constructors and assignment operators as they
               are not needed
20          */
21
22         DebugLogger()        = delete;
23         DebugLogger(const DebugLogger &)   = delete;
24         DebugLogger(DebugLogger &&)     = delete;
25         DebugLogger &operator=(const DebugLogger &) = delete;
26         DebugLogger &operator=(DebugLogger &&)  = delete;
27
28         /// Create a new DebugLogger instance from a filename
29         /// \param filename
30         explicit DebugLogger(const std::string &filename,
31                              Priority priority = Priority::Info);
32
33         /// Close the file stream on destruction
34         ~DebugLogger();
35
36         /// Set the priority level of the debugger's logs. Higher
               priorities will be
37         /// logged in release mode, while lower priorities will only be
```

```
            logged in debug
38          /// mode
39          /// \param newPriority The new priority level
40          void setPriorityLevel(Priority newPriority) { m_priority =
                newPriority; }
41
42          /// Write a message to the log file
43          /// \param message The message to write
44          /// \param priority The priority of the message (see Priority)
45          /// \param filename The filename of the file that called this
                function
46          /// \param line The line number of the file that called this
                function
47          void write(const std::string &message, Priority priority,
48                     const std::string &filename, int64_t line);
49
50      private:
51          std::fstream m_log;     // File stream
52          double m_startTime;     // Time at which the logger was created
53          Priority m_priority = Priority::Info; // Priority level of the
                logger
54      };
55
56      extern DebugLogger debugLogger;
57  } // namespace frac
```

3.4 Debug Logger (debug.cpp)

```
01  #include <fractal/fractal.hpp>
02
03  namespace frac {
04      DebugLogger::DebugLogger(const std::string &filename, Priority
            priority) {
05          m_log.open(filename, std::fstream::out);
06          m_startTime = lrc::now();
07
08          m_log << "============[ FRACTAL RENDERER DEBUG LOG
                ]============\n" << std::endl;
09      }
10
11      DebugLogger::~DebugLogger() {
12          m_log << "\n============[ FRACTAL RENDERER DEBUG LOG
                ]============";
13          m_log.flush();
14          m_log.close();
```

```
15      }
16
17      void DebugLogger::write(const std::string &message, Priority
            priority,
18                              const std::string &filename, int64_t line) {
19          if (static_cast<size_t>(priority) <
                static_cast<size_t>(m_priority)) return;
20
21          double time      = lrc::now();
22          constexpr size_t maxFilenameLength = 40;
23          std::string truncatedFilename;
24          std::string priorityString;
25          std::string cleanedMessage;
26
27          if (filename.size() > maxFilenameLength)
28              truncatedFilename =
29                "..." + filename.substr(filename.size() - maxFilenameLength
                      + 3, maxFilenameLength);
30          else
31              truncatedFilename = filename;
32
33          switch (priority) {
34              case Priority::Info: priorityString = "INFO"; break;
35              case Priority::Warning: priorityString = "WARNING"; break;
36              case Priority::Error: priorityString = "ERROR"; break;
37          }
38
39          // Pad new lines with 26 spaces to align with the beginning of
                the message
40          // in the log file.
41          int64_t preambleLength = 26 + maxFilenameLength;
42          for (char c : message) {
43              cleanedMessage += c;
44              if (c == '\n') cleanedMessage += std::string(preambleLength,
                    ' ');
45          }
46
47          // Use std::endl here to force-flush the buffer -- this is the
                only way
48          // to ensure that the log is written to disk in the event of a
                crash
49          m_log << fmt::format("[ {:.5f} ] {:>7} {:>{}}:{:0>4} {}",
50                               time - m_startTime,
51                               priorityString,
52                               truncatedFilename,
```

```
53                          maxFilenameLength,
54                          line,
55                          cleanedMessage)
56              << std::endl;
57      }
58
59  #if defined(LIBRAPID_DEBUG) // More reliable than NDEBUG
60      DebugLogger debugLogger("./log.txt", Priority::Info); // Log all
            messages
61  #else
62      DebugLogger debugLogger("./log.txt", Priority::Warning); // Only
            warnings and above
63  #endif
64  } // namespace frac
```

3.5 Fractal Base Class Implementation (`fractal.hpp`)

```
01 #pragma once
02
03 #include <cinderbox/cinderbox.hh>
04 #include <librapid>
05 #include <fstream>
06 #include <nlohmann/json.hpp>
07 #include <BS_thread_pool.hpp>
08
09 #ifndef FRACTAL_SETTINGS_PATH
10 # define FRACTAL_UI_SETTINGS_PATH FRACTAL_RENDERER_ROOT_DIR
       "/settings/settings.json"
11 #endif
12
13 namespace lrc = librapid;
14
15 using ThreadPool = BS::thread_pool;
16 using json  = nlohmann::json;
17
18 namespace frac {
19     using HighPrecision = lrc::mpf;
20     using LowPrecision = double;
21
22     using HighVec2 = lrc::Vec<HighPrecision, 2>;
23     using LowVec2 = lrc::Vec<LowPrecision, 2>;
24 } // namespace frac
25
26 #include <fractal/debug.hpp>
27 #include <fractal/colorPalette.hpp>
28 #include <fractal/openglUtils.hpp>
29 #include <fractal/renderConfig.hpp>
30 #include <fractal/genericFractal.hpp>
31 #include <fractal/mandelbrot.hpp>
32 #include <fractal/fractalRenderer.hpp>
33 #include <fractal/history.hpp>
34 #include <fractal/mainWindow.hpp>
```

3.6 Fractal Renderer Definition (`fractalRenderer.hpp`)

```
001 #pragma once
002
003 namespace frac {
004     class FractalRenderer {
005     public:
006         FractalRenderer() = default;
007
008         /// Construct a new renderer object from a JSON config object
009         /// \param config The JSON config object
010         explicit FractalRenderer(const json &config);
011
012         ~FractalRenderer();
013
014         /// Set the fractal renderer config
015         /// \param config JSON object
016         void setConfig(const json &config);
017
018         /// Stop the renderer gracefully and wait for all threads to
                rejoin main
019         void stopRender();
020
021         /// Set the complex-valued coordinate of the top-left corner of
                the fractal and
022         /// its size
023         /// \param topLeft Top-left corner
024         /// \param size Size of the fractal
025         /// \see moveFractalCenter
026         void moveFractalCorner(const lrc::Vec<HighPrecision, 2> &topLeft,
027                                const lrc::Vec<HighPrecision, 2> &size);
028
029         /// Set the complex-valued coordinate of the center of the
                fractal and its size
030         /// \param center Center of the fractal
031         /// \param size Size of the fractal
032         /// \see moveFractalCorner
033         void moveFractalCenter(const lrc::Vec<HighPrecision, 2> &center,
034                                const lrc::Vec<HighPrecision, 2> &size);
035
036         /// Render the fractal into the fractal surface, and copy that
                to the
037         /// fractal surface to be drawn. This will be executed on a
                separate thread
038         /// in order to keep the UI updating
```

```
039         void renderFractal();
040
041         /// Render a sub-section of the fractal, defined by the \p box
                variable. This is
042         /// intended to be used within the call queue to render multiple
                sections in
043         /// parallel
044         /// \param box The box configuration
045         /// \param boxIndex Box ID (for updating states)
046         void renderBox(const RenderBox &box, int64_t boxIndex = -1);
047
048         /// Calculate the colour of a pixel at standard-precision. This
                implements
049         /// anti-aliasing as well
050         /// \param pixPos Pixel-space coordinate
051         /// \param aliasFactor Anti-aliasing factor
052         /// \param step Step size
053         /// \param aliasStepCorrect Anti-aliasing step correction
054         /// \return Color of the pixel
055         ci::ColorA pixelColorLow(const LowVec2 &pixPos, int64_t
                aliasFactor,
056                                  const LowVec2 &step, const LowVec2
                                       &aliasStepCorrect);
057
058         /// Calculate the colour of a pixel at high-precision. See
                pixelColorLow
059         /// \param pixPos Pixel-space coordinate
060         /// \param aliasFactor Anti-aliasing factor
061         /// \param step Step size
062         /// \param aliasStepCorrect Anti-aliasing step correction
063         /// \return Color of the pixel
064         /// \see pixelColorLow
065         ci::ColorA pixelColorHigh(const HighVec2 &pixPos, int64_t
                aliasFactor,
066                                   const HighVec2 &step, const HighVec2
                                        &aliasStepCorrect);
067
068         /// Update the render configuration of the internal fractal
                pointer
069         void updateRenderConfig();
070
071         /// Ensure all values are using the highest precision possible
072         void updateConfigPrecision();
073
074         /// Regenerate the surfaces and resize them to fit the image size
```

```
075         void regenerateSurface();
076
077         /// Getter method for the render box time statistics
078         /// \return Statistics
079         LIBRAPID_NODISCARD RenderBoxTimeStats boxTimeStats() const;
080
081         /// Constant getter method for the render configuration
082         /// \return Render configuration
083         LIBRAPID_NODISCARD const RenderConfig &config() const;
084
085         /// Non-const getter method for the render configuration
086         /// \return Render configuration
087         LIBRAPID_NODISCARD RenderConfig &config();
088
089         /// Constant getter method for the internal render box vector
090         /// \return Render box vector
091         LIBRAPID_NODISCARD const std::vector<RenderBox> &renderBoxes()
                const;
092
093         /// Non-const getter method for the internal render box vector
094         /// \return Render box vector
095         LIBRAPID_NODISCARD std::vector<RenderBox> &renderBoxes();
096
097         /// Constant getter method for the internal settings object
098         /// \return Settings object
099         LIBRAPID_NODISCARD const json &settings() const;
100
101         /// Non-const getter method for the internal settings object
102         /// \return Settings object
103         LIBRAPID_NODISCARD json &settings();
104
105         /// Constant getter method for the internal surface
106         /// \return Surface
107         LIBRAPID_NODISCARD const ci::Surface &surface() const;
108
109         /// Non-const getter method for the internal surface
110         /// \return Surface
111         LIBRAPID_NODISCARD ci::Surface &surface();
112
113     private:
114         RenderConfig m_renderConfig;  // The settings for the fractal
                renderer
115         ci::Surface m_fractalSurface;  // The surface that the fractal
                is rendered to
116         json m_settings;     // The settings for the fractal
```

```
117         std::unique_ptr<Fractal> m_fractal; // The fractal to render
118         ThreadPool m_threadPool;   // Pool for render threads
119 
120         std::vector<RenderBox> m_renderBoxes; // The state of each
                render box
121 
122         bool m_haltRender = false; // Used to gracefully stop the render
                threads
123     };
124 } // namespace frac
```

3.7 Fractal Renderer Implementation (`fractalRenderer.cpp`)

```
001 #include <fractal/fractal.hpp>
002 
003 namespace frac {
004     FractalRenderer::FractalRenderer(const json &config) {
            setConfig(config); }
005     FractalRenderer::~FractalRenderer() { stopRender(); }
006 
007     void FractalRenderer::setConfig(const json &config) {
008         m_settings = config;
009 
010         try {
011             // Set the default precision
012             lrc::prec2(m_settings["renderConfig"]["precision"].get<int64_t>());
013 
014             // Load settings from settings JSON object
015             m_renderConfig = RenderConfig {
016               m_settings["renderConfig"]["numThreads"].get<int64_t>(),
017               m_settings["renderConfig"]["maxIters"].get<int64_t>(),
018               m_settings["renderConfig"]["precision"].get<int64_t>(),
019               m_settings["renderConfig"]["bail"].get<LowPrecision>(),
020               m_settings["renderConfig"]["antiAlias"].get<int>(),
021 
022               lrc::Vec2i(
023                 m_settings["renderConfig"]["imageSize"]["width"].get<int64_t>(),
024                 m_settings["renderConfig"]["imageSize"]["height"].get<int64_t>()),
025               lrc::Vec2i(m_settings["renderConfig"]["boxSize"]["width"].get<int64_t>(),
026                          m_settings["renderConfig"]["boxSize"]["height"].get<int64_t>()),
027 
028               lrc::Vec<HighPrecision, 2>(
029                 m_settings["renderConfig"]["fracTopLeft"]["Re"].get<float>(),
030                 m_settings["renderConfig"]["fracTopLeft"]["Im"].get<float>()),
031               lrc::Vec<HighPrecision, 2>(
```

```
032                 m_settings["renderConfig"]["fracSize"]["Re"].get<float>(),
033                 m_settings["renderConfig"]["fracSize"]["Im"].get<float>()),
034               lrc::Vec<HighPrecision, 2>(0, 0),
035
036               ColorPalette(), // Default for now -- colors added later
037
038               m_settings["renderConfig"]["draftRender"].get<bool>(),
039               m_settings["renderConfig"]["draftInc"].get<int64_t>()};
040
041           m_renderConfig.originalFracSize = m_renderConfig.fracSize;
042
043           // Load the colour palette from the JSON object
044           for (const auto &color :
                   m_settings["renderConfig"]["colorPalette"]) {
045               m_renderConfig.palette.addColor(
046                 ColorPalette::ColorType(color["red"].get<float>(),
047                                         color["green"].get<float>(),
048                                         color["blue"].get<float>(),
049                                         color["alpha"].get<float>()));
050           }
051
052           m_fractal = std::make_unique<Mandelbrot>(m_renderConfig);
053       } catch (std::exception &e) {
054           FRAC_LOG(fmt::format("Failed to load settings: {}",
                   e.what()));
055           stopRender();
056       }
057   }
058
059   void FractalRenderer::stopRender() {
060       m_haltRender = true;
061       m_threadPool.wait_for_tasks();
062       m_haltRender = false;
063   }
064
065   void FractalRenderer::moveFractalCorner(const
          lrc::Vec<HighPrecision, 2> &topLeft,
066                                           const lrc::Vec<HighPrecision, 2>
                                                &size) {
067       m_renderConfig.fracTopLeft = topLeft;
068       m_renderConfig.fracSize = size;
069       m_fractal->updateRenderConfig(m_renderConfig);
070   }
071
072   void FractalRenderer::moveFractalCenter(const
```

```
        lrc::Vec<HighPrecision, 2> &center,
073                                     const lrc::Vec<HighPrecision, 2>
                                            &size) {
074         moveFractalCorner(center - size / lrc::Vec<HighPrecision, 2>(2,
                2), size);
075     }
076
077     void FractalRenderer::renderFractal() {
078         if (m_threadPool.get_tasks_queued() > 0) {
079             FRAC_WARN("Render already in progress. Halting...");
080             m_haltRender = true;
081             m_threadPool.wait_for_tasks();
082             m_haltRender = false;
083             FRAC_LOG("Render halted");
084         }
085
086         FRAC_LOG("Rendering Fractal...");
087
088         m_renderBoxes.clear();
089         m_threadPool.reset(m_renderConfig.numThreads);
090
091         // Split the render into boxes to be rendered in parallel
092         auto imageSize = m_renderConfig.imageSize;
093         auto boxSize = m_renderConfig.boxSize;
094
095         // Round number of boxes up so the full image is covered
096         auto numBoxes =
097           lrc::Vec2i(lrc::ceil(lrc::Vec2f(imageSize) /
                    lrc::Vec2f(boxSize)));
098
099         m_renderBoxes.reserve(numBoxes.x() * numBoxes.y());
100
101         // Iterate over all boxes
102         for (int64_t i = 0; i < numBoxes.y(); ++i) {
103             for (int64_t j = 0; j < numBoxes.x(); ++j) {
104                 lrc::Vec2i adjustedBoxSize(
105                   lrc::min(boxSize.x(), imageSize.x() - j * boxSize.x()),
106                   lrc::min(boxSize.y(), imageSize.y() - i * boxSize.y()));
107                 RenderBox box {lrc::Vec2i(j, i) * boxSize,
108                                adjustedBoxSize,
109                                m_renderConfig.draftRender,
110                                m_renderConfig.draftInc,
111                                RenderBoxState::Queued};
112
113                 auto prevSize = (int64_t)m_renderBoxes.size();
```

```
114
115                 // Must happen before pushing to render queue
116                 m_renderBoxes.emplace_back(box);
117
118                 m_threadPool.push_task(
119                   [this, box, prevSize]() { renderBox(box, prevSize); });
120             }
121         }
122
123         FRAC_LOG("Fractal Complete...");
124     }
125
126     void FractalRenderer::renderBox(const RenderBox &box, int64_t
        boxIndex) {
127         // Update the render box state
128         m_renderBoxes[boxIndex].state = RenderBoxState::Rendering;
129         const double start   = lrc::now();
130
131         const int64_t inc = box.draftRender ? box.draftInc : 1;
132
133         HighVec2 fractalOrigin = lrc::map(
134           static_cast<HighVec2>(box.topLeft),
135           HighVec2({0, 0}),
136           static_cast<HighVec2>(m_renderConfig.imageSize),
137           m_renderConfig.fracTopLeft,
138           m_renderConfig.fracTopLeft +
                static_cast<HighVec2>(m_renderConfig.fracSize));
139
140         HighVec2 step =
141           m_renderConfig.fracSize /
                static_cast<HighVec2>(m_renderConfig.imageSize);
142
143         int64_t aliasFactor = m_renderConfig.antiAlias;
144         if (box.draftRender) aliasFactor = 1; // No anti-aliasing for
                drafts
145
146         HighPrecision scaleFactor =
147           HighPrecision(1) / static_cast<HighPrecision>(aliasFactor);
148         HighVec2 aliasStepCorrect(scaleFactor, scaleFactor);
149
150         bool blackEdges = true; // Assume edges are black to begin with
151
152         if (m_haltRender) return;
153
154         if (box.draftRender) {
```

```
155             for (int64_t py = box.topLeft.y(); py < box.topLeft.y() +
                    box.dimensions.y();
156                  ++py) {
157                 for (int64_t px = box.topLeft.x();
158                      px < box.topLeft.x() + box.dimensions.x();
159                      ++px) {
160                     m_fractalSurface.setPixel(lrc::Vec2i(px, py),
161                                               ci::ColorA {0.2, 0, 0.2, 0.5});
162                 }
163             }
164         }
165
166         // Render the top edge
167         for (int64_t px = box.topLeft.x(); px < box.topLeft.x() +
                box.dimensions.x();
168              px += inc) {
169             // Anti-aliasing
170             auto pixPos = fractalOrigin + step * HighVec2(px -
                    box.topLeft.x(), 0);
171
172             ci::ColorA pix;
173
174             if (m_renderConfig.precision <= 64) {
175                 pix = pixelColorLow(pixPos, aliasFactor, step,
                        aliasStepCorrect);
176             } else {
177                 pix = pixelColorHigh(pixPos, aliasFactor, step,
                        aliasStepCorrect);
178             }
179
180             if (blackEdges && (pix.r != 0 && pix.g != 0 && pix.b != 0)) {
181                 blackEdges = false;
182             }
183
184             m_fractalSurface.setPixel(lrc::Vec2i(px, box.topLeft.y()),
                    pix);
185         }
186
187         if (m_haltRender) return;
188
189         // Render the right edge
190         for (int64_t py = box.topLeft.y(); py < box.topLeft.y() +
                box.dimensions.y();
191              py += inc) {
192             // Anti-aliasing
```

```
193             auto pixPos =
194               fractalOrigin + step * HighVec2(box.dimensions.x(), py -
                      box.topLeft.y());
195
196             ci::ColorA pix;
197
198             if (m_renderConfig.precision <= 64) {
199                 pix = pixelColorLow(pixPos, aliasFactor, step,
                          aliasStepCorrect);
200             } else {
201                 pix = pixelColorHigh(pixPos, aliasFactor, step,
                          aliasStepCorrect);
202             }
203
204             if (blackEdges && (pix.r != 0 && pix.g != 0 && pix.b != 0)) {
205                 blackEdges = false;
206             }
207
208             m_fractalSurface.setPixel(
209               lrc::Vec2i(box.topLeft.x() + box.dimensions.x() - 1, py),
                      pix);
210         }
211
212         if (m_haltRender) return;
213
214         // Render the bottom edge
215         for (int64_t px = box.topLeft.x(); px < box.topLeft.x() +
                box.dimensions.x();
216              px += inc) {
217             // Anti-aliasing
218             auto pixPos = fractalOrigin +
219                           step * HighVec2(px - box.topLeft.x(),
                                  box.dimensions.y() - 1);
220
221             ci::ColorA pix;
222
223             if (m_renderConfig.precision <= 64) {
224                 pix = pixelColorLow(pixPos, aliasFactor, step,
                          aliasStepCorrect);
225             } else {
226                 pix = pixelColorHigh(pixPos, aliasFactor, step,
                          aliasStepCorrect);
227             }
228
229             if (blackEdges && (pix.r != 0 && pix.g != 0 && pix.b != 0)) {
```

```
230                 blackEdges = false;
231             }
232 
233             m_fractalSurface.setPixel(
234               lrc::Vec2i(px, box.topLeft.y() + box.dimensions.y() - 1),
                      pix);
235         }
236 
237         if (m_haltRender) return;
238 
239         // Render the left edge
240         for (int64_t py = box.topLeft.y(); py < box.topLeft.y() +
                box.dimensions.y();
241              py += inc) {
242             // Anti-aliasing
243             auto pixPos =
244               fractalOrigin +
245               step * HighVec2(box.topLeft.x() - box.topLeft.x(), py -
                      box.topLeft.y());
246 
247             ci::ColorA pix;
248 
249             if (m_renderConfig.precision <= 64) {
250                 pix = pixelColorLow(pixPos, aliasFactor, step,
                          aliasStepCorrect);
251             } else {
252                 pix = pixelColorHigh(pixPos, aliasFactor, step,
                          aliasStepCorrect);
253             }
254 
255             if (blackEdges && (pix.r != 0 && pix.g != 0 && pix.b != 0)) {
256                 blackEdges = false;
257             }
258 
259             m_fractalSurface.setPixel(lrc::Vec2i(box.topLeft.x(), py),
                      pix);
260         }
261 
262         if (blackEdges) {
263             for (int64_t py = box.topLeft.y() + 1;
264                  py < box.topLeft.y() + box.dimensions.y() - 1;
265                  py += inc) {
266                 for (int64_t px = box.topLeft.x() + 1;
267                      px < box.topLeft.x() + box.dimensions.x() - 1;
268                      px += inc) {
```

```
269                     m_fractalSurface.setPixel(lrc::Vec2i(px, py),
270                                               ci::ColorA {0, 0, 0, 1});
271                 }
272             }
273         } else {
274             // Make the primary axis of iteration the x-axis to improve
                    cache
275             // efficiency and increase performance.
276             for (int64_t py = box.topLeft.y() + 1;
277                  py < box.topLeft.y() + box.dimensions.y() - 1;
278                  py += inc) {
279                 // Quick return if required. Without this, the
280                 // render threads will continue running after the
281                 // application is closed, leading to weird behaviour.
282                 if (m_haltRender) return;
283 
284                 for (int64_t px = box.topLeft.x() + 1;
285                      px < box.topLeft.x() + box.dimensions.x() - 1;
286                      px += inc) {
287                     // Anti-aliasing
288                     auto pixPos = fractalOrigin + step * HighVec2(px -
                            box.topLeft.x(),
289                                                                   py -
                                                                       box.topLeft.y());
290 
291                     ci::ColorA pix;
292 
293                     if (m_renderConfig.precision <= 64) {
294                         pix = pixelColorLow(pixPos, aliasFactor, step,
                                aliasStepCorrect);
295                     } else {
296                         pix = pixelColorHigh(pixPos, aliasFactor, step,
                                aliasStepCorrect);
297                     }
298 
299                     m_fractalSurface.setPixel(lrc::Vec2i(px, py), pix);
300                 }
301             }
302         }
303 
304         // Update the render box state
305         m_renderBoxes[boxIndex].state = RenderBoxState::Rendered;
306         m_renderBoxes[boxIndex].renderTime = lrc::now() - start;
307     }
308 
```

```
309     ci::ColorA FractalRenderer::pixelColorLow(const LowVec2 &pixPos,
            int64_t aliasFactor,
310                                               const LowVec2 &step,
311                                               const LowVec2
                                                      &aliasStepCorrect) {
312         ci::ColorA pix(0, 0, 0, 1);
313 
314         for (int64_t aliasY = 0; aliasY < aliasFactor; ++aliasY) {
315             for (int64_t aliasX = 0; aliasX < aliasFactor; ++aliasX) {
316                 auto pos = pixPos + step * LowVec2(aliasX, aliasY) *
                            aliasStepCorrect;
317 
318                 auto [iters, endPoint] =
319                   m_fractal->iterCoordLow(lrc::Complex<LowPrecision>(pos.x(),
                             pos.y()));
320                 if (endPoint.real() * endPoint.real() +
321                       endPoint.imag() * endPoint.imag() <
322                     4) {
323                     pix += ci::ColorA(0, 0, 0, 1); // Probably in the set
324                 } else {
325                     pix +=
326                       m_fractal->getColorLow(endPoint, iters); // Probably
                                not in the set
327                 }
328             }
329         }
330 
331         return pix / static_cast<float>(aliasFactor * aliasFactor);
332     }
333 
334     ci::ColorA FractalRenderer::pixelColorHigh(const HighVec2 &pixPos,
335                                                int64_t aliasFactor, const
                                                       HighVec2 &step,
336                                               const HighVec2
                                                      &aliasStepCorrect) {
337         ci::ColorA pix(0, 0, 0, 1);
338 
339         for (int64_t aliasY = 0; aliasY < aliasFactor; ++aliasY) {
340             for (int64_t aliasX = 0; aliasX < aliasFactor; ++aliasX) {
341                 auto pos = pixPos + step * HighVec2(aliasX, aliasY) *
                            aliasStepCorrect;
342 
343                 auto [iters, endPoint] =
344                   m_fractal->iterCoordHigh(lrc::Complex<HighPrecision>(pos.x(),
                             pos.y()));
```

```
345                 if (endPoint.real() * endPoint.real() +
346                       endPoint.imag() * endPoint.imag() <
347                     4) {
348                     pix += ci::ColorA(0, 0, 0, 1); // Probably in the set
349                 } else {
350                     pix +=
351                       m_fractal->getColorHigh(endPoint, iters); //
                              Probably not in the set
352                 }
353             }
354         }
355 
356         return pix / static_cast<float>(aliasFactor * aliasFactor);
357     }
358 
359     void FractalRenderer::updateRenderConfig() {
360         m_fractal->updateRenderConfig(m_renderConfig);
361     }
362 
363     void FractalRenderer::regenerateSurface() {
364         FRAC_LOG("Regenerating Surface...");
365         int64_t w  = m_renderConfig.imageSize.x();
366         int64_t h  = m_renderConfig.imageSize.y();
367         m_fractalSurface = ci::Surface((int32_t)w, (int32_t)h, true);
368         FRAC_LOG("Surface regenerated");
369     }
370 
371     void FractalRenderer::updateConfigPrecision() {
372         int64_t prec = m_renderConfig.precision;
373         HighPrecision highPrecTopLeftX(m_renderConfig.fracTopLeft.x(),
              prec);
374         HighPrecision highPrecTopLeftY(m_renderConfig.fracTopLeft.y(),
              prec);
375         HighPrecision highPrecFracSizeX(m_renderConfig.fracSize.x(),
              prec);
376         HighPrecision highPrecFracSizeY(m_renderConfig.fracSize.y(),
              prec);
377         HighPrecision
              highPrecOriginalFracSizeX(m_renderConfig.originalFracSize.x(),
378                                       prec);
379         HighPrecision
              highPrecOriginalFracSizeY(m_renderConfig.originalFracSize.y(),
380                                       prec);
381 
382         m_renderConfig.fracTopLeft  = {highPrecTopLeftX,
```

```
            highPrecTopLeftY};
383         m_renderConfig.fracSize    = {highPrecFracSizeX,
                highPrecFracSizeY};
384         m_renderConfig.originalFracSize = {highPrecOriginalFracSizeX,
385                                            highPrecOriginalFracSizeY};
386     }
387
388     RenderBoxTimeStats FractalRenderer::boxTimeStats() const {
389         double min = 1e10;
390         double max = -1e10;
391         double total = 0;
392         size_t count = 0;
393
394         for (const auto &box : m_renderBoxes) {
395             if (box.renderTime == 0) continue;
396
397             if (box.renderTime < min) min = box.renderTime;
398             if (box.renderTime > max) max = box.renderTime;
399             total += box.renderTime;
400             count += 1;
401         }
402
403         double average  = total / (double)count;
404         size_t remainingBoxes = m_renderBoxes.size() - count;
405         double remainingTime =
406           ((double)remainingBoxes * average) /
                (double)m_renderConfig.numThreads;
407
408         return {min, max, average, remainingTime};
409     }
410
411     const RenderConfig &FractalRenderer::config() const { return
            m_renderConfig; }
412     RenderConfig &FractalRenderer::config() { return m_renderConfig; }
413
414     const std::vector<RenderBox> &FractalRenderer::renderBoxes() const {
415         return m_renderBoxes;
416     }
417     std::vector<RenderBox> &FractalRenderer::renderBoxes() { return
            m_renderBoxes; }
418
419     const json &FractalRenderer::settings() const { return m_settings; }
420     json &FractalRenderer::settings() { return m_settings; }
421
422     const ci::Surface &FractalRenderer::surface() const { return
```

```
            m_fractalSurface; }
423     ci::Surface &FractalRenderer::surface() { return m_fractalSurface; }
424 } // namespace frac
```

3.8 Generic Fractal Definition (`genericFractal.hpp`)

```
01 #pragma once
02
03 namespace frac {
04     class Fractal {
05     public:
06         /// Constructor taking a RenderConfig object
07         /// \param config
08         explicit Fractal(const RenderConfig &config);
09         Fractal(const Fractal &)   = delete;
10         Fractal(Fractal &&)     = delete;
11         Fractal &operator=(const Fractal &) = delete;
12         Fractal &operator=(Fractal &&)  = delete;
13         virtual ~Fractal()     = default;
14
15         /// Configure a new set of options for the fractal
16         /// \param config The new RenderConfig to use
17         virtual void updateRenderConfig(const RenderConfig &config);
18
19         /// Iterate over a complex-valued coordinate and return the
               value at which the
20         /// coordinate exceeds the given threshold or reaches the
               desired number of
21         /// iterations. The return value also includes the number of
               iterations it took
22         /// to reach the return coordinate.
23         /// \param coord The initial complex-valued coordinate
24         /// \return <iterations, resulting coordinate (low precision)>
25         virtual std::pair<int64_t, lrc::Complex<LowPrecision>>
26         iterCoordLow(const lrc::Complex<LowPrecision> &coord) const = 0;
27
28         /// See iterCoordLow(const lrc::Complex<LowPrecision> &coord)
               const
29         /// \param coord The initial complex-valued coordinate
30         /// \return <iterations, resulting coordinate (high precision)>
31         /// \see iterCoordLow(const lrc::Complex<LowPrecision> &coord)
               const
32         virtual std::pair<int64_t, lrc::Complex<HighPrecision>>
33         iterCoordHigh(const lrc::Complex<HighPrecision> &coord) const =
               0;
34
35         /// Apply the colouring algorithm given the result of an
               `iterCoord` call
36         /// \param coord Resulting coordinate
```

```
37          /// \param iters Number of iterations
38          /// \return Colour of the point
39          virtual ci::ColorA getColorLow(const lrc::Complex<LowPrecision>
                &coord,
40                                         int64_t iters) const;
41
42          /// See getColorLow(const lrc::Complex<LowPrecision> &coord,
                int64_t iters) const
43          /// \param coord Resulting coordinate
44          /// \param iters Number of iterations
45          /// \return Colour of the point
46          /// \see getColorLow(const lrc::Complex<LowPrecision> &coord,
                int64_t iters) const
47          virtual ci::ColorA getColorHigh(const
                lrc::Complex<HighPrecision> &coord,
48                                          int64_t iters) const;
49
50      protected:
51          RenderConfig m_renderConfig;
52      };
53  } // namespace frac
```

3.9 Generic Fractal Implementation (`genericFractal.cpp`)

```
01  #include <fractal/fractal.hpp>
02
03  namespace frac {
04      Fractal::Fractal(const RenderConfig &config) :
            m_renderConfig(config) {}
05
06      void Fractal::updateRenderConfig(const RenderConfig &config) {
            m_renderConfig = config; }
07
08      ci::ColorA Fractal::getColorLow(const lrc::Complex<LowPrecision>
            &coord, int64_t iters) const {
09          using Col = ColorPalette::ColorType;
10
11          // float logZN  =
                lrc::log(lrc::abs(lrc::Complex<float>(coord.real(),
                coord.imag()))) / 2;
12          // float nu  = lrc::log(logZN / lrc::LN2) / lrc::LN2;
13          // float iteration = static_cast<float>(iters) + 1 - nu;
14          // const auto &palette = m_renderConfig.palette;
15          // Col color1   = palette[static_cast<size_t>(iteration) %
                palette.size()];
```

```
16          // Col color2   = palette[(static_cast<size_t>(iteration) + 1) %
                palette.size()];
17          // Col merged   = ColorPalette::merge(color1, color2,
                lrc::mod(iteration, 1.0f));
18          // return {merged.x(), merged.y(), merged.z(), 1};
19
20          // Nice gradient
21          double s1 = (double)iters -
22                      lrc::log2(lrc::log2((float)coord.real() *
                            (float)coord.real() +
23                                          (float)coord.imag() *
                                                (float)coord.imag())) +
24                      4;
25          Col color = 0.5 + 0.5 * lrc::cos(3.0 + s1 * 0.15 +
                lrc::Vec3d(0.0, 0.6, 1.0));
26          return {(float)color.x(), (float)color.y(), (float)color.z(), 1};
27
28          // Cool stepped gradients
29          // return {(iters % 10) / 10.f, 0, 0, 1};
30          // return {(iters % 11) / 11.f, (iters % 23) / 23.f, (iters %
                31) / 31.f, 1};
31          // return {(iters % 2) / 2.f, (iters % 3) / 3.f, (iters % 7) /
                7.f, 1};
32      }
33
34      ci::ColorA Fractal::getColorHigh(const lrc::Complex<HighPrecision>
            &coord,
35                                       int64_t iters) const {
36          using Col = ColorPalette::ColorType;
37
38          // float logZN  =
                lrc::log(lrc::abs(lrc::Complex<float>(coord.real(),
                coord.imag()))) / 2;
39          // float nu  = lrc::log(logZN / lrc::LN2) / lrc::LN2;
40          // float iteration = static_cast<float>(iters) + 1 - nu;
41          // const auto &palette = m_renderConfig.palette;
42          // Col color1   = palette[static_cast<size_t>(iteration) %
                palette.size()];
43          // Col color2   = palette[(static_cast<size_t>(iteration) + 1) %
                palette.size()];
44          // Col merged   = ColorPalette::merge(color1, color2,
                lrc::mod(iteration, 1.0f));
45          // return {merged.x(), merged.y(), merged.z(), 1};
46
47          // Nice gradient
```

```
48          double s1 = (double)iters -
49                      lrc::log2(lrc::log2((float)coord.real() *
                            (float)coord.real() +
50                                          (float)coord.imag() *
                                                (float)coord.imag())) +
51                      4;
52          Col color = 0.5 + 0.5 * lrc::cos(3.0 + s1 * 0.15 +
                lrc::Vec3d(0.0, 0.6, 1.0));
53          return {(float)color.x(), (float)color.y(), (float)color.z(), 1};
54
55          // Cool stepped gradients
56          // return {(iters % 10) / 10.f, 0, 0, 1};
57          // return {(iters % 11) / 11.f, (iters % 23) / 23.f, (iters %
                31) / 31.f, 1};
58          // return {(iters % 2) / 2.f, (iters % 3) / 3.f, (iters % 7) /
                7.f, 1};
59      }
60  } // namespace frac
```

3.10 History Definition(`history.hpp`)

```
001 #pragma once
002
003 namespace frac {
004     class HistoryNode {
005     public:
006         HistoryNode()        = default;
007         HistoryNode(const HistoryNode &)   = delete;
008         HistoryNode(HistoryNode &&)     = delete;
009         HistoryNode &operator=(const HistoryNode &) = delete;
010         HistoryNode &operator=(HistoryNode &&)  = delete;
011         ~HistoryNode()        = default;
012
013         /// Append a new node to the end of the linked list
014         /// \param node The node to append (can contain links to other
                nodes)
015         void append(HistoryNode *node);
016
017         /// Free all child nodes following this one
018         void killChildren();
019
020         /// The number of nodes in the list, iterating forwards from
                this node
021         /// \param prevSize The number of nodes in the list before this
                one (default to
022         /// zero) \return The number of nodes in the list
023         LIBRAPID_NODISCARD size_t sizeForward(size_t prevSize = 0) const;
024
025         /// The number of nodes in the list, iterating backwards from
                this node
026         /// \param prevSize
027         /// \return The number of nodes in the list
028         /// \see sizeForward
029         LIBRAPID_NODISCARD size_t sizeBackward(size_t prevSize = 0)
                const;
030
031         /// The next node in the linked list. This may be `nullptr`, so
                always check
032         /// the value is valid
033         /// \return The next node in the linked list
034         LIBRAPID_NODISCARD HistoryNode *next() const;
035
036         /// The previous node in the linked list. See `next()` for more
                information
```

```
037         /// \return The previous node in the linked list
038         /// \see next
039         LIBRAPID_NODISCARD HistoryNode *prev() const;
040
041         /// Iterate backwards until a node with an invalid parent is
                found (i.e. the
042         /// first node in the linked list)
043         /// \return The first node in the linked list
044         LIBRAPID_NODISCARD HistoryNode *first();
045
046         /// return the last node in the linked list
047         /// \return
048         /// \see first
049         LIBRAPID_NODISCARD HistoryNode *last();
050
051         /// Update the configuration and surface members of this node
052         /// \param config New configuration
053         /// \param surface New surface
054         void set(const RenderConfig &config, const ci::Surface &surface);
055
056         /// See `set()`
057         /// \param config New configuration
058         /// \see set
059         void setConfig(const RenderConfig &config);
060
061         /// See `set()`
062         /// \param surface New surface
063         /// \see set
064         void setSurface(const ci::Surface &surface);
065
066         /// Getter method for the configuration instance stored
067         /// \return RenderConfig
068         LIBRAPID_NODISCARD const RenderConfig &config() const;
069
070         /// Getter method for the surface instance stored
071         /// \return ci::Surface
072         LIBRAPID_NODISCARD const ci::Surface &surface() const;
073
074         /// Non-const getter method for the configuration instance stored
075         /// \return RenderConfig
076         LIBRAPID_NODISCARD RenderConfig &config();
077
078         /// Non-const getter method for the surface instance stored
079         /// \return ci::Surface
080         LIBRAPID_NODISCARD ci::Surface &surface();
```

```
081
082     private:
083         HistoryNode *m_next = nullptr;
084         HistoryNode *m_prev = nullptr;
085
086         RenderConfig m_config;
087         ci::Surface m_surface;
088     };
089
090     class HistoryBuffer {
091     public:
092         HistoryBuffer()         = default;
093         HistoryBuffer(const HistoryBuffer &)   = delete;
094         HistoryBuffer(HistoryBuffer &&)     = delete;
095         HistoryBuffer &operator=(const HistoryBuffer &) = delete;
096         HistoryBuffer &operator=(HistoryBuffer &&)  = delete;
097
098         ~HistoryBuffer();
099
100         /// Append a new point to the history buffer
101         /// \param config The settings for the fractal renderer
102         /// \param surface A saved copy of the fractal surface
103         void append(const RenderConfig &config, const ci::Surface
                &surface);
104
105         /// Undo the last operation
106         bool undo();
107
108         /// If possible, redo the last operation
109         bool redo();
110
111         /// Return the number of elements in the history buffer
112         /// \return Number of elements
113         LIBRAPID_NODISCARD size_t size() const;
114
115         /// Return the first buffer item (a HistoryNode pointer)
116         /// \return First item in the buffer
117         LIBRAPID_NODISCARD HistoryNode *first() const;
118
119         /// Return the last buffer item (a HistoryNode pointer)
120         /// \return Last item in the buffer
121         LIBRAPID_NODISCARD HistoryNode *last() const;
122
123     private:
124         HistoryNode *m_listHead = nullptr;
```

```
125         HistoryNode *m_currentNode = nullptr;
126     };
127 } // namespace frac
```

3.11 History Implementation (`history.cpp`)

```
001 #include <fractal/fractal.hpp>
002
003 namespace frac {
004     void HistoryNode::append(HistoryNode *list) {
005         if (m_next) {
006             m_next->append(list);
007         } else {
008             m_next  = list;
009             m_next->m_prev = this;
010         }
011     }
012
013     void HistoryNode::killChildren() {
014         if (m_next) m_next->killChildren();
015         delete m_next;
016         m_next = nullptr;
017     }
018
019     size_t HistoryNode::sizeForward(size_t prevSize) const {
020         if (m_next)
021             return m_next->sizeForward(prevSize + 1);
022         else
023             return prevSize;
024     }
025
026     size_t HistoryNode::sizeBackward(size_t prevSize) const {
027         if (m_prev)
028             return m_prev->sizeBackward(prevSize + 1);
029         else
030             return prevSize;
031     }
032
033     HistoryNode *HistoryNode::next() const { return m_next; }
034     HistoryNode *HistoryNode::prev() const { return m_prev; }
035
036     HistoryNode *HistoryNode::first() {
037         if (m_prev)
038             return m_prev->first();
039         else
```

```
040             return this;
041     }
042
043     HistoryNode *HistoryNode::last() {
044         if (m_next)
045             return m_next->last();
046         else
047             return this;
048     }
049
050     void HistoryNode::set(const RenderConfig &config, const ci::Surface
            &surface) {
051         m_config = config;
052         m_surface = surface;
053     }
054
055     void HistoryNode::setConfig(const RenderConfig &config) { m_config =
            config; }
056
057     void HistoryNode::setSurface(const ci::Surface &surface) { m_surface
            = surface; }
058
059     const RenderConfig &HistoryNode::config() const { return m_config; }
060     const ci::Surface &HistoryNode::surface() const { return m_surface; }
061     RenderConfig &HistoryNode::config() { return m_config; }
062     ci::Surface &HistoryNode::surface() { return m_surface; }
063
064     HistoryBuffer::~HistoryBuffer() {
065         m_listHead->killChildren();
066         // No need to delete m_currentNode, since it will be killed
                recursively
067         LIBRAPID_ASSERT(!m_listHead->next() && !m_listHead->prev(),
068                         "HistoryBuffer is not empty");
069         delete m_listHead;
070     }
071
072     void HistoryBuffer::append(const RenderConfig &config, const
            ci::Surface &surface) {
073         auto list = new HistoryNode;
074         list->set(config, surface);
075         if (m_listHead) {
076             m_listHead->append(list);
077             m_currentNode = m_listHead->last();
078         } else {
079             m_listHead = list;
```

```
080             m_currentNode = list;
081         }
082     }
083
084     bool HistoryBuffer::undo() {
085         if (m_currentNode->prev()) {
086             m_currentNode = m_currentNode->prev();
087             return true;
088         }
089         return false;
090     }
091
092     bool HistoryBuffer::redo() {
093         if (m_currentNode->next()) {
094             m_currentNode = m_currentNode->next();
095             return true;
096         }
097         return false;
098     }
099
100     size_t HistoryBuffer::size() const {
101         if (!m_listHead) return 0;
102         return m_listHead->sizeForward();
103     }
104
105     HistoryNode *HistoryBuffer::first() const { return
            m_listHead->first(); }
106     HistoryNode *HistoryBuffer::last() const { return
            m_listHead->last(); }
107 } // namespace frac
```

3.12 Main Window Definition (`mainWindow.hpp`)

```
001 #pragma once
002
003 namespace frac {
004     class MainWindow : public ci::app::App {
005     public:
006         /// Nothing passed to the constructor
007         MainWindow() = default;
008
009         void configureSettings();
010
011         /// Set up the main window, making sure it's the right size and
                that the frame
012         /// rates are set correctly
013         void configureWindow();
014
015         /// Configure ImGui, setting up the style and enabling docking
016         void configureImGui();
017
018         /// Set up the window, configure ImGui and initialize the
                fractal rendering
019         /// surfaces
020         void setup() override;
021
022         /// Halt all render threads and wait for them to join main
023         void stopRender();
024
025         void appendConfigToHistory();
026
027         std::vector<std::tuple<lrc::Vec2f, lrc::Vec2f, HistoryNode *>>
028         getHistoryFrameLocations() const;
029
030         void undoLastMove();
031         void redoLastMove();
032
033         // Run on shutdown
034         void cleanup() override;
035
036         /// Render the fractal to the screen (from the FractalRenderer
                surface)
037         void drawFractal();
038
039         /// Outline each render box (if active) to show their current
                states
```

```
040         void outlineRenderBoxes();
041
042         /// Called every frame
043         void draw() override;
044
045         /// Draw the UI
046         void drawImGui();
047
048         /// Draw the history window
049         void drawHistory();
050
051         /// Update the most recent history item with the current fractal
                configuration
052         /// and surface
053         void updateHistoryItem();
054
055         /// Move the top left corner of the fractal and set a new size
056         /// \param topLeft Top-left corner (complex coordinate)
057         /// \param size Size of the fractal (Re, Im)
058         void moveFractalCorner(const lrc::Vec<HighPrecision, 2> &topLeft,
059                                const lrc::Vec<HighPrecision, 2> &size);
060
061         /// Set the center of the fractal and the dimensions -- see
                moveFractalCorner
062         /// \param center Top-left corner
063         /// \param size Size of fractal
064         /// \see moveFractalCorner
065         void moveFractalCenter(const lrc::Vec<HighPrecision, 2> &center,
066                                const lrc::Vec<HighPrecision, 2> &size);
067
068         /// Advanced zooming method -- given pixel coordinates for the
                top left and
069         /// bottom right of the new area, perform the following:
070         /// 1. Copy existing pixels in the specified region to a buffer
071         /// 2. Regenerate the surface
072         /// 3. Copy the buffer to the fractal surface
073         /// 4. Reconfigure the fractal renderer
074         /// 5. Trigger another fractal render
075         /// \param pixTopLeft
076         /// \param pixBottomRight
077         void zoomFractal(const lrc::Vec2i &pixTopLeft, const lrc::Vec2i
                &pixBottomRight);
078
079         /// Render the fractal into the fractal surface, and copy that
                to the
```

```
080         /// fractal surface to be drawn. This will be executed on a
                separate
081         /// thread in order to keep the UI updating
082         void renderFractal(bool amendHistory = true);
083
084         /// Callback for mouse movement (this does not include mouse
                clicks or
085         /// drags) \param event The mouse event
086         void mouseMove(ci::app::MouseEvent event) override;
087
088         /// Callback for mouse clicks
089         /// \param event The mouse event
090         void mouseDown(ci::app::MouseEvent event) override;
091
092         /// Callback for mouse drags
093         /// \param event The mouse event
094         void mouseDrag(ci::app::MouseEvent event) override;
095
096         /// Callback for mouse releases
097         /// \param event The mouse event
098         void mouseUp(ci::app::MouseEvent event) override;
099
100         /// Callback for mouse wheel events
101         /// \param event The mouse event
102         void mouseWheel(ci::app::MouseEvent event) override;
103
104         /// Callback for when a key is pressed, including the modifiers
                (shift, ctrl,
105         /// etc) \param event The key event
106         void keyDown(ci::app::KeyEvent event) override;
107
108         template<typename T>
109         static lrc::Vec<T, 2> aspectCorrectedBox(const lrc::Vec<T, 2>
                &p1,
110                                                  const lrc::Vec<T, 2> &p2,
111                                                  float aspectRatio) {
112             lrc::Vec<T, 2> correctedBox;
113             lrc::Vec<T, 2> delta = p2 - p1;
114             if (delta.y() < delta.x() / aspectRatio)
115                 correctedBox = {delta.x(), delta.x() / aspectRatio};
116             else
117                 correctedBox = {delta.y() * aspectRatio, delta.y()};
118             return correctedBox;
119         }
120
```

```
121         /// Draw a zoom box at a given point. This includes a
                transparent box
122         /// surrounded by a solid rectangle with a cross in the middle.
                \param start
123         /// The top left corner of the box \param end The bottom right
                corner of the
124         /// box
125         void drawZoomBox(const lrc::Vec2f &start, const lrc::Vec2f &end)
                const;
126
127     private:
128         FractalRenderer m_renderer;    // The fractal renderer
129         ci::gl::Texture2dRef m_fractalTexture; // The fractal texture
130         ci::Font m_font = ci::Font("Arial", 24); // The font to use for
                rendering text
131         lrc::Vec2i m_mousePos; // The current position of the mouse in
                the window
132         lrc::Vec2i m_mouseDownPos; // The position of the mouse when it
                was clicked
133         bool m_mouseDown = false; // Whether the mouse is currently down
134
135         HistoryBuffer m_history;
136         HistoryNode *m_historyNode = nullptr;
137         float m_historyScrollTarget = 0.0f;
138
139         bool m_drawingZoomBox = false;
140         bool m_showZoomBox = false;
141         bool m_moveZoomBox = false;
142         lrc::Vec2i m_zoomBoxStart;
143         lrc::Vec2i m_zoomBoxEnd;
144
145         // Values used for ImGui input fields
146         std::string m_fineMovementRe;
147         std::string m_fineMovementIm;
148         std::string m_fineMovementZoom;
149     };
150 } // namespace frac
```

3.13 Main Window Implementation (`mainWindow.cpp`)

```
001 #include <fractal/fractal.hpp>
002
003 namespace frac {
004     void MainWindow::configureSettings() {
005         // Load the settings file
```

```
006         FRAC_LOG(fmt::format("Loading settings from {}",
                FRACTAL_UI_SETTINGS_PATH));
007
008         std::fstream settingsFile(FRACTAL_UI_SETTINGS_PATH,
                std::ios::in);
009         if (settingsFile.is_open()) {
010             m_renderer.setConfig(json::parse(settingsFile));
011             m_history.append(m_renderer.config(), m_renderer.surface());
012             m_historyNode = m_history.first();
013         } else {
014             FRAC_ERROR("Failed to open settings file");
015             quit();
016         }
017
018         FRAC_LOG("Settings Configured");
019     }
020
021     void MainWindow::configureWindow() {
022         setFrameRate(-1);    // Unlimited framerate
023         ci::gl::enableVerticalSync(true); // Enable vertical sync to
                avoid tearing
024         setWindowSize(1364, 700);  // Set the initial window size
025
026         // Set up rendering settings
027         ci::gl::enableDepthWrite();
028         ci::gl::enableDepthRead();
029         ci::gl::enableDepth();
030         glDepthFunc(GL_ALWAYS);
031
032         FRAC_LOG("Window Configured");
033     }
034
035     void MainWindow::configureImGui() {
036         ImGui::Initialize();
037         ImGui::StyleColorsDark();
038         ImGui::GetIO().ConfigFlags |= ImGuiConfigFlags_DockingEnable;
039         ImGui::GetIO().FontGlobalScale = 1.0f;
040
041         FRAC_LOG("ImGui Configured");
042     }
043
044     void MainWindow::setup() {
045         FRAC_LOG("Setup Called");
046
047         configureSettings();
```

```
048         configureWindow();
049         configureImGui();
050 
051         m_renderer.regenerateSurface();
052         renderFractal();
053 
054         FRAC_LOG("Setup Complete");
055     }
056 
057     void MainWindow::stopRender() { m_renderer.stopRender(); }
058 
059     void MainWindow::appendConfigToHistory() {
060         FRAC_LOG("Appending to history");
061 
062         if (m_historyNode != m_history.last()) {
063             // There is a pre-existing history, so it must be overwritten
064             m_historyNode->killChildren();
065         }
066 
067         m_history.append(m_renderer.config(), m_renderer.surface());
068         m_historyNode = m_history.last();
069     }
070 
071     std::vector<std::tuple<lrc::Vec2f, lrc::Vec2f, HistoryNode *>>
072     MainWindow::getHistoryFrameLocations() const {
073         std::vector<std::tuple<lrc::Vec2f, lrc::Vec2f, HistoryNode *>>
                ret;
074 
075         const json &settings  = m_renderer.settings();
076         const float historyFrameWidth =
                settings["menus"]["history"]["frameWidth"];
077         const float historyFrameSep =
                settings["menus"]["history"]["frameSep"];
078 
079         const auto windowWidth = (float)getWindowWidth();
080         const RenderConfig &config = m_renderer.config();
081         size_t historySize  = m_history.size();
082         HistoryNode *node  = m_history.first();
083         int64_t index   = 0;
084         int64_t totalHeight  = 0;
085 
086         while (node) {
087             float aspect = (float)config.imageSize.x() /
                    (float)config.imageSize.y();
088             lrc::Vec2f renderSize(historyFrameWidth, historyFrameWidth /
```

```
                aspect);
089             lrc::Vec2f drawPos(windowWidth - historyFrameWidth -
                    historyFrameSep,
090                                (renderSize.y() + historyFrameSep) *
091                                    (float)(historySize - index - 1) +
092                                historyFrameSep);
093
094             totalHeight += (int64_t)(renderSize.y() + historyFrameSep);
095             drawPos.y(drawPos.y() + m_historyScrollTarget);
096
097             ret.emplace_back(std::tuple<lrc::Vec2f, lrc::Vec2f,
                    HistoryNode *>(
098               drawPos, renderSize, node));
099
100             node = node->next();
101             index++;
102         }
103
104         return ret;
105     }
106
107     void MainWindow::undoLastMove() {
108         FRAC_LOG("Attempting to Undo...");
109         // Ensure a previous configuration actually exists
110         if (m_historyNode->prev()) {
111             FRAC_LOG("Undo successful");
112             m_historyNode = m_historyNode->prev();
113
114             // Update the render configuration
115             m_renderer.config() = m_historyNode->config();
116             renderFractal(false);
117         }
118     }
119
120     void MainWindow::redoLastMove() {
121         FRAC_LOG("Attempting to Redo...");
122         if (m_historyNode->next()) {
123             FRAC_LOG("Redo successful");
124             m_historyNode = m_historyNode->next();
125
126             // Update the render configuration
127             m_renderer.config() = m_historyNode->config();
128             renderFractal(false);
129         }
130     }
```

```
131
132     void MainWindow::cleanup() {
133         stopRender();
134         FRAC_LOG("Cleaned Up");
135     }
136
137     void MainWindow::drawFractal() {
138         m_fractalTexture =
                ci::gl::Texture2d::create(m_renderer.surface());
139
140         const RenderConfig &config = m_renderer.config();
141         double aspect = (double)config.imageSize.x() /
                (double)config.imageSize.y();
142         double height = getWindowHeight();
143         lrc::Vec2f renderSize(height * aspect, height);
144
145         ci::gl::color(ci::ColorA(1, 1, 1, 1));
146         ci::gl::draw(m_fractalTexture, ci::Rectf({0, 0}, renderSize));
147     }
148
149     void MainWindow::outlineRenderBoxes() {
150         const RenderConfig &config    = m_renderer.config();
151         const std::vector<RenderBox> &renderBoxes =
                m_renderer.renderBoxes();
152         for (const auto &box : renderBoxes) {
153             switch (box.state) {
154                 case RenderBoxState::None:
155                 case RenderBoxState::Rendered: continue;
156                 case RenderBoxState::Queued:
157                     ci::gl::color(ci::ColorA(0, 1, 0, 0.2));
158                     break;
159                 case RenderBoxState::Rendering:
160                     ci::gl::color(ci::ColorA(1, 1, 0, 0.2));
161                     break;
162             }
163
164             ci::ivec2 boxPos = box.topLeft;
165             ci::ivec2 boxSize = box.dimensions;
166             boxPos.y += getWindowHeight() - config.imageSize.y();
167             ci::gl::drawStrokedRect(ci::Rectf(boxPos, boxPos + boxSize),
                    1);
168         }
169     }
170
171     void MainWindow::draw() {
```

```
172         ci::gl::clear(ci::Color(0.2f, 0.2f, 0.2f));
173
174         drawImGui();
175         drawFractal();
176         outlineRenderBoxes();
177         drawHistory();
178
179         if (m_drawingZoomBox) {
180             // Draw an aspect-ratio corrected box
181             RenderConfig config = m_renderer.config();
182             float aspectRatio = (float)config.imageSize.x() /
                    (float)config.imageSize.y();
183             lrc::Vec2i correctedBox =
184               aspectCorrectedBox(m_mouseDownPos, m_mousePos, aspectRatio);
185             auto correctedEnd = m_mouseDownPos + correctedBox;
186             drawZoomBox(m_mouseDownPos, correctedEnd);
187         }
188
189         if (m_showZoomBox) { drawZoomBox(m_zoomBoxStart, m_zoomBoxEnd); }
190     }
191
192     void MainWindow::drawImGui() {
193         // Ensure there is space for the labels in the ImGui windows
194         constexpr int64_t labelledItemWidth = -120;
195
196         RenderConfig &config = m_renderer.config();
197         const json &settings = m_renderer.settings();
198
199         // Fractal Information Window
200         json fractalInfo = settings["menus"]["fractalInfo"];
201         ImGui::SetNextWindowPos({(float)fractalInfo["posX"],
                (float)fractalInfo["posY"]},
202                                 ImGuiCond_Once);
203         ImGui::SetNextWindowSize(
204           {(float)fractalInfo["width"], (float)fractalInfo["height"]},
                ImGuiCond_Once);
205         ImGui::Begin("Fractal Info", nullptr);
206         {
207             ImGui::Text("Fractal Type: Mandelbrot");
208
209             HighPrecision re = config.fracTopLeft.x() +
                    config.fracSize.x() / 2;
210             HighPrecision im = config.fracTopLeft.y() +
                    config.fracSize.y() / 2;
211             HighPrecision zoom = config.originalFracSize.x() /
```

```
                config.fracSize.x();
212
213             ImGui::TextWrapped("%s", fmt::format("Re: {}", re).c_str());
214             ImGui::TextWrapped("%s", fmt::format("Im: {}", im).c_str());
215
216             std::ostringstream os;
217             os << std::fixed << std::setprecision(6) << std::scientific
                    << zoom;
218             ImGui::TextWrapped("%s", fmt::format("Zoom: {}x",
                    os.str()).c_str());
219
220             double maxZoomExponent = (double)config.precision /
                    lrc::log2(10);
221             ImGui::TextWrapped(
222               "%s", fmt::format("Max Zoom: e+{:.3f}",
                      maxZoomExponent).c_str());
223         }
224         ImGui::End();
225
226         // Fine movement window
227         json fineMovement = settings["menus"]["fineMovement"];
228         ImGui::SetNextWindowPos(
229           {(float)fineMovement["posX"], (float)fineMovement["posY"]},
                ImGuiCond_Once);
230         ImGui::SetNextWindowSize(
231           {(float)fineMovement["width"], (float)fineMovement["height"]},
                ImGuiCond_Once);
232         ImGui::Begin("Fine Movement", nullptr);
233         {
234             ImGui::InputText("Re", &m_fineMovementRe);
235             ImGui::InputText("Im", &m_fineMovementIm);
236             ImGui::InputText("Zoom", &m_fineMovementZoom);
237
238             if (ImGui::Button("Apply")) {
239                 HighPrecision re, im, zoom, sizeRe, sizeIm;
240                 scn::scan(m_fineMovementRe, "{}", re);
241                 scn::scan(m_fineMovementIm, "{}", im);
242                 scn::scan(m_fineMovementZoom, "{}", zoom);
243
244                 FRAC_LOG(fmt::format("Received Real Part: {}", re));
245
246                 sizeRe = config.originalFracSize.x() / zoom;
247                 sizeIm = config.originalFracSize.y() / zoom;
248                 moveFractalCenter(lrc::Vec<HighPrecision, 2>(re, im),
249                                   lrc::Vec<HighPrecision, 2>(sizeRe,
```

```
                                    sizeIm));
250                 renderFractal();
251             }
252         }
253         ImGui::End();
254
255         // Render configuration
256         json renderConfigMenu = settings["menus"]["renderConfig"];
257         ImGui::SetNextWindowPos(
258           {(float)renderConfigMenu["posX"],
                (float)renderConfigMenu["posY"]},
259           ImGuiCond_Once);
260         ImGui::SetNextWindowSize(
261           {(float)renderConfigMenu["width"],
                (float)renderConfigMenu["height"]},
262           ImGuiCond_Once);
263         ImGui::Begin("Render Configuration", nullptr);
264         {
265             static int64_t minThreads = 1;
266             static int64_t maxThreads =
                    std::thread::hardware_concurrency();
267             static int64_t minIters  = 1;
268             static int64_t maxIters  = 100000;
269             static int64_t minPrecision = 64;
270             static int64_t maxPrecision = 1024;
271             static int64_t minAntiAlias = 1;
272             static int64_t maxAntiAlias = 16;
273             static int64_t minDraftInc = 1;
274             static int64_t maxDraftInc = 4;
275
276             static int64_t newThreads = config.numThreads;
277             static int64_t newIters  = config.maxIters;
278             static int64_t newPrecision = config.precision;
279             static int64_t newAntiAlias = config.antiAlias;
280             static bool newDraftRender = config.draftRender;
281             static int64_t newDraftInc = config.draftInc;
282
283             ImGui::PushItemWidth(labelledItemWidth);
284             ImGui::SliderScalar(
285               "Threads", ImGuiDataType_S64, &newThreads, &minThreads,
                    &maxThreads);
286
287             ImGui::PushItemWidth(labelledItemWidth);
288             ImGui::SliderScalar("Anti Aliasing",
289                                 ImGuiDataType_S64,
```

```
290                             &newAntiAlias,
291                             &minAntiAlias,
292                             &maxAntiAlias);
293
294             ImGui::PushItemWidth(labelledItemWidth);
295             ImGui::DragScalarN(
296               "Iterations", ImGuiDataType_S64, &newIters, 1, 5,
                      &minIters, &maxIters);
297
298             ImGui::PushItemWidth(labelledItemWidth);
299             ImGui::DragScalarN("Precision",
300                                ImGuiDataType_S64,
301                                &newPrecision,
302                                1,
303                                0.1,
304                                &minPrecision,
305                                &maxPrecision);
306
307             if (ImGui::Button("Apply")) {
308                 stopRender();
309                 config.numThreads = newThreads;
310                 config.maxIters = newIters;
311                 config.precision = newPrecision;
312                 config.antiAlias = newAntiAlias;
313                 config.draftRender = newDraftRender;
314                 config.draftInc = newDraftInc;
315                 lrc::prec2(newPrecision);
316                 m_renderer.updateRenderConfig();
317                 m_renderer.updateConfigPrecision();
318                 renderFractal();
319             }
320
321             ImGui::SameLine();
322             ImGui::Checkbox("Draft Mode", &newDraftRender);
323
324             if (newDraftRender) {
325                 ImGui::SameLine();
326                 ImGui::PushItemWidth(labelledItemWidth);
327                 ImGui::SliderScalar("Draft Increment",
328                                     ImGuiDataType_S64,
329                                     &newDraftInc,
330                                     &minDraftInc,
331                                     &maxDraftInc);
332             }
333         }
```

```
334         ImGui::End();
335
336         // Render Statistics
337         json renderStatistics = settings["menus"]["renderStatistics"];
338         ImGui::SetNextWindowPos(
339           {(float)renderStatistics["posX"],
                (float)renderStatistics["posY"]},
340           ImGuiCond_Once);
341         ImGui::SetNextWindowSize(
342           {(float)renderStatistics["width"],
                (float)renderStatistics["height"]},
343           ImGuiCond_Once);
344
345         RenderBoxTimeStats stats = m_renderer.boxTimeStats();
346         ImGui::Begin("Render Statistics", nullptr);
347         {
348             ImGui::Text("Pixels/s (min): %s", fmt::format("{:.3f}",
                    stats.min).c_str());
349             ImGui::Text("Pixels/s (max): %s", fmt::format("{:.3f}",
                    stats.max).c_str());
350             ImGui::Text("Pixels/s (avg): %s",
351                         fmt::format("{:.3f}", stats.average).c_str());
352             ImGui::Text("Estimated Time Remaining: %s",
353                         lrc::formatTime(stats.remainingTime).c_str());
354         }
355         ImGui::End();
356     }
357
358     void MainWindow::drawHistory() {
359         const json &settings  = m_renderer.settings();
360         const float historyFrameWidth =
                settings["menus"]["history"]["frameWidth"];
361         const float historyFrameSep =
                settings["menus"]["history"]["frameSep"];
362
363         const auto windowWidth = (float)getWindowWidth();
364         const auto windowHeight = (float)getWindowHeight();
365         const RenderConfig &config = m_renderer.config();
366         const std::vector<std::tuple<lrc::Vec2f, lrc::Vec2f, HistoryNode
                *>> frames =
367           getHistoryFrameLocations();
368
369         int64_t totalHeight = 0;
370
371         // Draw a bounding box for the frames to sit within
```

```
372         float boxLeft = windowWidth - historyFrameWidth -
                historyFrameSep * 2;
373         ci::gl::color(ci::ColorA(0.15, 0.15, 0.3, 1));
374         ci::gl::drawSolidRect(ci::Rectf(boxLeft, 0, windowWidth,
                windowHeight));
375
376         for (const auto &frame : frames) {
377             const lrc::Vec2f framePos = std::get<0>(frame);
378             const lrc::Vec2f frameSize = std::get<1>(frame);
379             const HistoryNode *node = std::get<2>(frame);
380
381             auto texture = ci::gl::Texture2d::create(node->surface());
382             totalHeight += (int64_t)(frameSize.y() + historyFrameSep);
383
384             // Don't draw the frame if it's not visible on the screen
385             if (framePos.y() > windowHeight || framePos.y() +
                    frameSize.y() < 0) continue;
386
387             ci::gl::color(ci::ColorA(1, 1, 1, 1));
388             ci::gl::draw(texture, ci::Rectf(framePos, framePos +
                    frameSize));
389             ci::gl::color(ci::ColorA(0, 0, 0, 1));
390             glu::drawStrokedRectangle(framePos, framePos + frameSize, 3);
391
392             if (node == m_historyNode) {
393                 // If this frame is selected, outline it gold
394                 ci::gl::color(ci::ColorA(1, 1, 0, 1));
395                 glu::drawStrokedRectangle(framePos, framePos + frameSize,
                        4);
396             }
397         }
398
399         if (windowHeight < (float)totalHeight) {
400             m_historyScrollTarget =
401               lrc::clamp(m_historyScrollTarget, windowHeight -
                      (float)totalHeight, 0);
402         } else {
403             m_historyScrollTarget = 0;
404         }
405     }
406
407     void MainWindow::moveFractalCorner(const lrc::Vec<HighPrecision, 2>
            &topLeft,
408                                        const lrc::Vec<HighPrecision, 2>
                                               &size) {
```

```
409         m_renderer.moveFractalCorner(topLeft, size);
410     }
411
412     void MainWindow::moveFractalCenter(const lrc::Vec<HighPrecision, 2>
            &center,
413                                        const lrc::Vec<HighPrecision, 2>
                                               &size) {
414         m_renderer.moveFractalCenter(center, size);
415     }
416
417     void MainWindow::zoomFractal(const lrc::Vec2i &pixTopLeft,
418                                  const lrc::Vec2i &pixBottomRight) {
419         // The vast majority of the calculations must be done at the
                highest
420         // precision possible. Without this, the zooming will not
                function
421         // correctly when the zoom factor exceeds the range of 64-bit
                precision.
422
423         FRAC_LOG("Moving Fractal...");
424         FRAC_LOG(fmt::format("Pixel Coordinates: {} -> {}", pixTopLeft,
                pixBottomRight));
425         updateHistoryItem();
426
427         RenderConfig &config = m_renderer.config();
428         ci::Surface &surface = m_renderer.surface();
429
430         // Resize the fractal area
431         HighVec2 imageSize = config.imageSize;
432         HighVec2 pixelDelta = pixBottomRight - pixTopLeft;
433
434         HighVec2 newFracPos = lrc::map(HighVec2(pixTopLeft),
435                                        HighVec2(0, 0),
436                                        imageSize,
437                                        config.fracTopLeft,
438                                        config.fracTopLeft +
                                               config.fracSize);
439
440         HighVec2 newFracSize = lrc::map(
441           pixelDelta, HighVec2(0, 0), imageSize, HighVec2(0, 0),
                config.fracSize);
442
443         // Copy the pixels from the selected region to the new region
444         // A temporary buffer is required here because, at some point,
                the loop
```

```
445         // would be writing to the same pixels it is reading from,
                resulting in
446         // strange visual glitches.
447         int64_t imgWidth = config.imageSize.x();
448         int64_t imgHeight = config.imageSize.y();
449         int64_t index = 0;
450         std::vector<ci::ColorA> newPixels(imgWidth * imgHeight);
451         auto mouseStartInImageLow =
452           pixTopLeft - lrc::Vec2i {0, getWindowHeight() -
                config.imageSize.y()};
453         for (int64_t y = 0; y < imgHeight; ++y) {
454             for (int64_t x = 0; x < imgWidth; ++x) {
455                 int64_t pixX = lrc::map(x,
456                                         0.f,
457                                         imgWidth,
458                                         mouseStartInImageLow.x(),
459                                         mouseStartInImageLow.x() +
                                            (float)pixelDelta.x());
460                 int64_t pixY = lrc::map(y,
461                                         0.f,
462                                         imgHeight,
463                                         mouseStartInImageLow.y(),
464                                         mouseStartInImageLow.y() +
                                            (float)pixelDelta.y());
465
466                 newPixels[index++] = surface.getPixel({pixX, pixY});
467             }
468         }
469
470         // Write the new pixels to the surface
471         index = 0;
472         for (int64_t y = 0; y < imgHeight; ++y) {
473             for (int64_t x = 0; x < imgWidth; ++x) {
474                 surface.setPixel({x, y}, newPixels[index++]);
475             }
476         }
477
478         config.fracTopLeft = newFracPos;
479         config.fracSize = newFracSize;
480         m_renderer.updateRenderConfig();
481         renderFractal();
482     }
483
484     void MainWindow::renderFractal(bool amendHistory) {
485         m_renderer.renderFractal();
```

```
486
487         // m_history.append(m_renderer.config(), m_renderer.surface());
488         if (amendHistory) appendConfigToHistory();
489     }
490
491     void MainWindow::updateHistoryItem() {
492         // Update history buffer surface before re-rendering the fractal
493         if (m_history.size() > 0) {
494             m_history.last()->setSurface(m_renderer.surface());
495             FRAC_LOG("Writing to surface");
496         }
497     }
498
499     void MainWindow::mouseMove(ci::app::MouseEvent event) { m_mousePos =
        event.getPos(); }
500
501     void MainWindow::mouseDown(ci::app::MouseEvent event) {
502         // Don't capture mouse events if ImGui wants them
503         if (ImGui::GetIO().WantCaptureMouse) return;
504
505         m_mouseDown = true;
506         m_mouseDownPos = event.getPos();
507
508         // Ensure mouse is within the image
509         if (m_mouseDownPos.x() >= 0 &&
510             m_mouseDownPos.x() < m_renderer.config().imageSize.x() &&
511             m_mouseDownPos.y() >= 0 &&
512             m_mouseDownPos.y() < m_renderer.config().imageSize.y()) {
513             // Check if mouse is inside the box (and it is being shown)
514             if (m_showZoomBox && m_mouseDownPos.x() > m_zoomBoxStart.x()
                    &&
515                 m_mouseDownPos.x() < m_zoomBoxEnd.x() &&
516                 m_mouseDownPos.y() > m_zoomBoxStart.y() &&
517                 m_mouseDownPos.y() < m_zoomBoxEnd.y()) {
518                 m_moveZoomBox = true;
519             } else {
520                 m_drawingZoomBox = true;
521                 m_showZoomBox = false;
522             }
523         }
524
525         // See if mouse is in the history buffer section
526         const json &settings  = m_renderer.settings();
527         const float historyFrameWidth =
                settings["menus"]["history"]["frameWidth"];
```

```
528         const float historyFrameSep =
                settings["menus"]["history"]["frameSep"];
529         const std::vector<std::tuple<lrc::Vec2f, lrc::Vec2f, HistoryNode
                *>> frames =
530           getHistoryFrameLocations();
531         const auto windowWidth = (float)getWindowWidth();
532         const auto windowHeight = (float)getWindowHeight();
533         float boxLeft   = windowWidth - historyFrameWidth -
                historyFrameSep * 2;
534         if (m_mouseDownPos.x() > boxLeft && m_mouseDownPos.x() <
                windowWidth) {
535             // Iterate over frames to see if any contain the mouse
536             for (const auto &frame : frames) {
537                 const lrc::Vec2i framePos = std::get<0>(frame);
538                 const lrc::Vec2i frameSize = std::get<1>(frame);
539                 HistoryNode *node  = std::get<2>(frame);
540
541                 if (m_mouseDownPos.x() >= framePos.x() &&
542                     m_mouseDownPos.x() < framePos.x() + frameSize.x() &&
543                     m_mouseDownPos.y() >= framePos.y() &&
544                     m_mouseDownPos.y() < framePos.y() + frameSize.y()) {
545                     // Mouse was within this frame, so set it as current
546                     m_historyNode  = node;
547                     m_renderer.config() = node->config();
548                     renderFractal(false); // Re-render the fractal
549                 }
550             }
551         }
552     }
553
554     void MainWindow::mouseDrag(ci::app::MouseEvent event) {
555         if (ImGui::GetIO().WantCaptureMouse) return;
556
557         lrc::Vec2i delta = lrc::Vec2i(event.getPos()) - m_mousePos;
558         m_mousePos  = event.getPos();
559
560         if (m_moveZoomBox) {
561             m_zoomBoxStart += delta;
562             m_zoomBoxEnd += delta;
563         }
564     }
565
566     void MainWindow::mouseUp(ci::app::MouseEvent event) {
567         if (ImGui::GetIO().WantCaptureMouse) return;
568         RenderConfig &config = m_renderer.config();
```

```
569         m_mouseDown   = false;
570
571         // Resize the fractal area
572         lrc::Vec2i imageSize = config.imageSize;
573         float aspectRatio = (float)imageSize.x() / (float)imageSize.y();
574
575         lrc::Vec2f aspectCorrected =
576           aspectCorrectedBox(m_mouseDownPos, m_mousePos, aspectRatio);
577
578         // Persistent zoom area
579         if (m_drawingZoomBox) {
580             m_zoomBoxStart = m_mouseDownPos;
581             m_zoomBoxEnd = m_mouseDownPos + lrc::Vec2i(aspectCorrected);
582             m_showZoomBox = true;
583             m_drawingZoomBox = false;
584         }
585     }
586
587     void MainWindow::mouseWheel(ci::app::MouseEvent event) {
588         if (ImGui::GetIO().WantCaptureMouse) return;
589
590         const json &settings = m_renderer.settings();
591         if (m_mousePos.x() > settings["menus"]["history"]["frameWidth"])
                {
592             m_historyScrollTarget +=
593               event.getWheelIncrement() *
594               settings["menus"]["history"]["scrollSpeed"].get<float>();
595         }
596     }
597
598     void MainWindow::keyDown(ci::app::KeyEvent event) {
599         const json &settings = m_renderer.settings();
600
601         if (m_showZoomBox) {
602             const int64_t scrollSpeed =
                    settings["menus"]["history"]["scrollSpeed"];
603
604             switch (event.getCode()) {
605                 case ci::app::KeyEvent::KEY_RETURN: {
606                     zoomFractal(m_zoomBoxStart, m_zoomBoxEnd);
607                 }
608                 case ci::app::KeyEvent::KEY_ESCAPE: {
609                     m_showZoomBox = false;
610                     break;
611                 }
```

```
612                 case ci::app::KeyEvent::KEY_RIGHT: {
613                     m_zoomBoxStart.x(m_zoomBoxStart.x() + scrollSpeed);
614                     m_zoomBoxEnd.x(m_zoomBoxEnd.x() + scrollSpeed);
615                     break;
616                 }
617                 case ci::app::KeyEvent::KEY_LEFT: {
618                     m_zoomBoxStart.x(m_zoomBoxStart.x() - scrollSpeed);
619                     m_zoomBoxEnd.x(m_zoomBoxEnd.x() - scrollSpeed);
620                     break;
621                 }
622                 case ci::app::KeyEvent::KEY_UP: {
623                     m_zoomBoxStart.y(m_zoomBoxStart.y() - scrollSpeed);
624                     m_zoomBoxEnd.y(m_zoomBoxEnd.y() - scrollSpeed);
625                     break;
626                 }
627                 case ci::app::KeyEvent::KEY_DOWN: {
628                     m_zoomBoxStart.y(m_zoomBoxStart.y() + scrollSpeed);
629                     m_zoomBoxEnd.y(m_zoomBoxEnd.y() + scrollSpeed);
630                     break;
631                 }
632             }
633         }
634 
635         if (event.getCode() == ci::app::KeyEvent::KEY_z &&
636             event.getModifiers() & ci::app::KeyEvent::CTRL_DOWN) {
637             if (event.getModifiers() & ci::app::KeyEvent::SHIFT_DOWN) {
638                 redoLastMove();
639             } else {
640                 undoLastMove();
641             }
642         }
643     }
644 
645     void MainWindow::drawZoomBox(const lrc::Vec2f &start, const
            lrc::Vec2f &end) const {
646         // Translucent inner box
647         ci::gl::color(ci::ColorA(1, 0, 0, 0.333)); // Red with alpha
648         ci::gl::drawSolidRect(ci::Rectf(start.x(), start.y(), end.x(),
                end.y()));
649         // Small cross in the middle
650         ci::gl::color(ci::ColorA(0, 0, 1, 0.333)); // Blue with alpha
651         glu::drawCross((start + end) * 0.5f, 5.f, 2.f);
652         // Bounding box
653         ci::gl::color(ci::ColorA(1, 0, 0, 1)); // Red
654         glu::drawStrokedRectangle(start, end, 5);
```

```
655     }
656 } // namespace frac
```

3.14 Mandelbrot Fractal Definition (`mandelbrot.hpp`)

```
01 #pragma once
02
03 #include <fractal/genericFractal.hpp>
04
05 namespace frac {
06     /*
07      * No need to document this file, since the class inherits from a
            generic fractal
08      * class and does not implement any novel functions.
09      */
10
11     class Mandelbrot : public Fractal {
12     public:
13         /// Constructor taking a RenderConfig object
14         /// \param config RenderConfig object
15         explicit Mandelbrot(const RenderConfig &config);
16         Mandelbrot(const Mandelbrot &)   = delete;
17         Mandelbrot(Mandelbrot &&)    = delete;
18         Mandelbrot &operator=(const Mandelbrot &) = delete;
19         Mandelbrot &operator=(Mandelbrot &&) = delete;
20
21         ~Mandelbrot() override = default;
22
23         LIBRAPID_NODISCARD std::pair<int64_t, lrc::Complex<LowPrecision>>
24         iterCoordLow(const lrc::Complex<LowPrecision> &coord) const
               override;
25
26         LIBRAPID_NODISCARD std::pair<int64_t,
               lrc::Complex<HighPrecision>>
27         iterCoordHigh(const lrc::Complex<HighPrecision> &coord) const
               override;
28     };
29 } // namespace frac
```

3.15 Mandelbrot Fractal Implementation (`mandelbrot.cpp`)

```
01 #include <fractal/fractal.hpp>
02
03 namespace frac {
04     Mandelbrot::Mandelbrot(const RenderConfig &config) : Fractal(config)
           {}
05
06     /*
```

```
07      * Note that, since this class will be used polymorphically with
            other classes,
08      * these two functions must be implemented separately and cannot be
            templated, as the compiler
09      * would error when trying to identify which function to call.
            Additionally, splitting the
10      * functions in this way allows for more targeted optimisations to
            be made in some cases.
11      */
12
13     std::pair<int64_t, lrc::Complex<LowPrecision>>
14     Mandelbrot::iterCoordLow(const lrc::Complex<LowPrecision> &coord)
           const {
15         LowPrecision re_0 = lrc::real(coord); // Real component (initial)
16         LowPrecision im_0 = lrc::imag(coord); // Imaginary component
               (initial)
17         LowPrecision re = 0, im = 0;
18         LowPrecision tmp; // Temporary variable for use in the
               calculation
19         int64_t iteration = 0;
20
21         // Bail when larger than this
22         double bailout = Fractal::m_renderConfig.bail;
23
24         while (re * re + im * im <= bailout && iteration <
               Fractal::m_renderConfig.maxIters) {
25             tmp = re * re - im * im + re_0;
26             im = 2 * re * im + im_0;
27             re = tmp;
28             ++iteration;
29         }
30
31         return {iteration, lrc::Complex<LowPrecision>(re, im)};
32
33         // lrc::Complex<LowPrecision> tempVarThing = coord;
34         // while (lrc::abs(tempVarThing) <= bailout && iteration <
               Fractal::m_renderConfig.maxIters) {
35         //  tempVarThing = lrc::pow(tempVarThing, LowPrecision(4)) +
               coord;
36         //  ++iteration;
37         // }
38
39         // return {iteration, tempVarThing};
40     }
41
```

```
42      std::pair<int64_t, lrc::Complex<HighPrecision>>
43      Mandelbrot::iterCoordHigh(const lrc::Complex<HighPrecision> &coord)
            const {
44          HighPrecision re_0 = lrc::real(coord); // Real component
                (initial)
45          HighPrecision im_0 = lrc::imag(coord); // Imaginary component
                (initial)
46          HighPrecision re = 0, im = 0;
47          HighPrecision tmp; // Temporary variable for use in the
                calculation
48          int64_t iteration = 0;
49
50          // Bail when larger than this
51          double bailout = 1 << 16;
52
53          while (re * re + im * im <= bailout && iteration <
                Fractal::m_renderConfig.maxIters) {
54              tmp = re * re - im * im + re_0;
55              im = 2 * re * im + im_0;
56              re = tmp;
57              ++iteration;
58          }
59
60          return {iteration, lrc::Complex<HighPrecision>(re, im)};
61      }
62  } // namespace frac
```

3.16 OpenGL Utilities Definition (`openglUtils.hpp`)

```
01 #pragma once
02
03 namespace frac::glu {
04     /// Draw a line from p1 to p2 with a given thickness. The line has a
           circle drawn at each end
05     /// to make it look nice.
06     /// \param p1 Line start
07     /// \param p2 Line end \param
08     /// thickness Line thickness
09     void drawLine(const lrc::Vec2f &p1, const lrc::Vec2f &p2, float
           thickness = 1);
10
11     /// Draw a stroked rectangle with a given thickness -- note this
           draws the EDGES of
12     /// the rectangle, and does not fill the inside \param topLeft Top
           left corner of the
13     /// rectangle \param bottomRight Bottom right corner of the
           rectangle \param thickness
14     /// Line thickness
15     void drawStrokedRectangle(const lrc::Vec2f &topLeft, const
           lrc::Vec2f &bottomRight,
16                               float thickness = 1);
17
18     /// Draw a cross at \p center, where each "arm" has length \p radius
           and thickness
19     /// \p thickness
20     /// \param center Where to draw the cross
21     /// \param radius Radius of the cross
22     /// \param thickness Thickness of the cross
23     void drawCross(const lrc::Vec2f &center, float radius, float
           thickness = 1);
24 } // namespace frac::glu
```

3.17 OpenGL Utilities Implementation (`openglUtils.cpp`)

```
01 #include <fractal/fractal.hpp>
02
03 namespace frac::glu {
04     void drawLine(const lrc::Vec2f &p1, const lrc::Vec2f &p2, float
           thickness) {
05         ci::vec3 translation({p1.x(), p1.y(), 0});
06         float rotation = lrc::atan2(p2.y() - p1.y(), p2.x() - p1.x());
07         float lineLength = static_cast<float>((p2 - p1).mag());
```

```
08
09          ci::gl::pushMatrices();
10          ci::gl::translate(translation);
11          ci::gl::rotate(rotation);
12
13          // Line segment
14          ci::gl::drawSolidRect(ci::Rectf({0.f, -thickness / 2.f},
                {lineLength, thickness / 2}));
15
16          // Draw two circles to make it look nice :)
17          ci::gl::drawSolidCircle({0, 0}, thickness / 2);
18          ci::gl::drawSolidCircle({lineLength, 0}, thickness / 2);
19
20          ci::gl::popMatrices();
21      }
22
23      void drawStrokedRectangle(const lrc::Vec2f &topLeft, const
            lrc::Vec2f &bottomRight,
24                                float thickness) {
25          // Draw each edge using drawLine, as it allows a thickness to be
                specified
26          drawLine(topLeft, {bottomRight.x(), topLeft.y()}, thickness);
27          drawLine({bottomRight.x(), topLeft.y()}, bottomRight, thickness);
28          drawLine(bottomRight, {topLeft.x(), bottomRight.y()}, thickness);
29          drawLine({topLeft.x(), bottomRight.y()}, topLeft, thickness);
30      }
31
32      void drawCross(const lrc::Vec2f &center, float radius, float
            thickness) {
33          ci::vec3 translation({center.x(), center.y(), 0});
34          ci::gl::pushMatrices();
35          ci::gl::translate(translation);
36          ci::gl::lineWidth(thickness);
37          ci::gl::drawLine(ci::vec2(0, -radius), ci::vec2(0, radius));
38          ci::gl::drawLine(ci::vec2(-radius, 0), ci::vec2(radius, 0));
39          ci::gl::popMatrices();
40      }
41  } // namespace frac::glu
```

3.18 Render Configuration Definition (`renderConfig.hpp`)

```
01 #pragma once
02
03 namespace frac {
04     /// Represents the state of a render box
05     enum class RenderBoxState {
06         None, // Not yet assigned a state
07         Queued, // Queued to be rendered
08         Rendering, // Currently being rendered
09         Rendered // Rendered and ready to be written to the image
10     };
11
12     /// Stores the pixel-space coordinates of a region to render
13     struct RenderBox {
14         lrc::Vec2i topLeft;
15         lrc::Vec2i dimensions;
16         bool draftRender;
17         int64_t draftInc;
18         RenderBoxState state = RenderBoxState::None;
19         double renderTime = 0;
20     };
21
22     /// Information about the time taken to render a box
23     struct RenderBoxTimeStats {
24         double min = 0;
25         double max = 0;
26         double average = 0;
27         double remainingTime = 0;
28     };
29
30     /// Contains all the information required to define an image (not
           including the
31     /// fractal type and colouring algorithm)
32     struct RenderConfig {
33         int64_t numThreads; // Number of threads to render on (max)
34         int64_t maxIters; // Largest number of iterations to allow
35         int64_t precision; // Precision of floating point types used for
               arithmetic
36         LowPrecision bail; // Bailout value
37         int64_t antiAlias; // Anti-aliasing factor -- 1 = no
               anti-aliasing
38
39         lrc::Vec2i imageSize; // Size of the image to render
40         lrc::Vec2i boxSize; // Size of sub-regions to render (see
```

```
            RenderBox)
41
42          lrc::Vec<HighPrecision, 2> fracTopLeft; // The fractal-space
                center of the image
43          lrc::Vec<HighPrecision, 2> fracSize; // The width and height of
                the fractal space
44          lrc::Vec<HighPrecision, 2> originalFracSize; // Original size
45
46          ColorPalette palette; // The palette to use for rendering the
                fractal
47
48          bool draftRender; // Whether to render the fractal in draft mode
49          int64_t draftInc; // Increment for draft rendering
50      };
51  } // namespace frac
```

3.19 Render Configuration Implementation (`renderConfig.cpp`)

```
1   #include <fractal/fractal.hpp>
2
3   namespace frac {
4
5   } // namespace frac
```

3.20 CMake File (`CMakeLists.txt`)

```
cmake_minimum_required(VERSION 3.10)
project(FractalRenderer)

set(CMAKE_CXX_STANDARD 17)

file(GLOB_RECURSE FRACTAL_SOURCES
    "${CMAKE_CURRENT_SOURCE_DIR}/src/*.cpp")
add_executable(FractalRenderer main.cpp ${FRACTAL_SOURCES})

# Include multiprecision floating point
set(LIBRAPID_USE_MULTIPREC ON)
set(LIBRAPID_FAST_MATH ON)

add_subdirectory(cinderbox)
add_subdirectory(json)
target_link_libraries(FractalRenderer PUBLIC cinderbox
    nlohmann_json::nlohmann_json)

target_include_directories(FractalRenderer PUBLIC
    ${CMAKE_CURRENT_SOURCE_DIR}/include)
target_include_directories(FractalRenderer PUBLIC
    ${CMAKE_CURRENT_SOURCE_DIR}/thread-pool)

target_compile_definitions(FractalRenderer PUBLIC
    -DFRACTAL_RENDERER_ROOT_DIR="${CMAKE_CURRENT_SOURCE_DIR}")
```

3.21 Main File (`main.cpp`)

```
#include <fractal/fractal.hpp>
CINDER_APP(frac::MainWindow,
    ci::app::RendererGl(ci::app::RendererGl::Options().msaa(4)))
```

References

[1] Catherine Soanes and Sara Hawker. *Compact Oxford English Dictionary of current English*. Oxford University Press, 2013.

[2] Paul Bourke, Nov 2002. URL http://paulbourke.net/fractals/mandelbrot/.

[3] Frederik Slijkerman. Ultra fractal: Advanced fractal software for windows and macos, Jul 2022. URL https://www.ultrafractal.com/.

[4] David J Eck. URL https://math.hws.edu/eck/index.html.

[5] URL https://xaos-project.github.io/XaoSjs/.

[6] Fractal software. URL https://fractalfoundation.org/resources/fractal-software/.

[7] Cinder Developers and Toby Davis. Pencilcaseman/cinderbox: A modified version of cinder with extra libraries for improved functionality. URL https://github.com/Pencilcaseman/cinderbox.

[8] Toby Davis. LibRapid: Optimised Mathematics for C++, 1 2023. URL https://github.com/LibRapid/librapid.