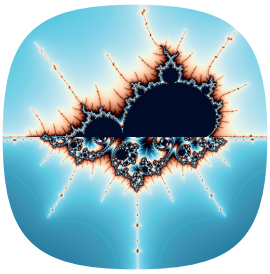


Mandelbrot Maps

Types and Huskies Are a Web App's Best Friends

João Filipe Maio

MInf Project (Part 2) Report

Master of Informatics
School of Informatics
University of Edinburgh

April 11, 2021

Abstract

Creating JavaScript web applications is easy. Maintaining them can be tedious and error-prone. We attempt to alleviate this by adding static typing to our existing open-source JavaScript web application *Mandelbrot Maps* [1] using TypeScript, while also creating a development workflow with industry-standard code-quality and automation tools like ESLint, Prettier, and `husky` to simplify and accelerate development.

To ensure that a stable production version of our application is always live, we use GitHub Actions to maintain various CI/CD pipelines which further automate the process of building, testing, and deploying our application.

We introduce visual changes and new functionality (like Deep Zoom) to the existing application, as well as a new mode for exploring Tan's Theorem [2] authored by Fraser Scott, potentially making this version of Mandelbrot Maps one of the most fully-featured fractal explorers currently available on the Internet.

The application is available at the following URL:

`jmaio.github.io/mandelbrot-maps`

Source code is available on GitHub, under the GNU GPL-3.0 License:

`github.com/JMaio/mandelbrot-maps`

Acknowledgements

I would like to thank:

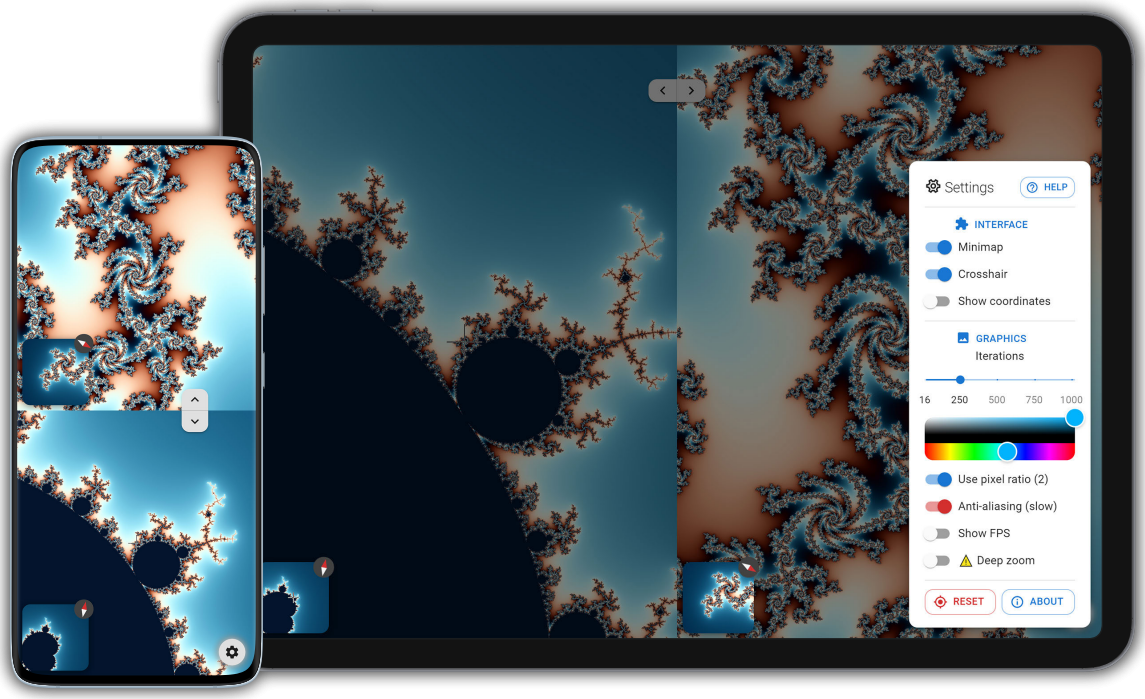
Philip Wadler, my project supervisor, for his guidance and feedback on my work.

My parents **Virginia and Filipe** for their constant love and support.

Table of Contents

Chapter 1

Introduction



1.1 Context – MInf 1

Mandelbrot Maps is an interactive fractal explorer. Last year's version of this Mandelbrot Maps project (MInf 1 [1]) was implemented as a web application, using React [3] and WebGL [4]. Its primary aim was to bring the functionality of the original project – a Java Applet created by Iain Parris in 2008 – to the modern web.

Early web technologies like Java Applets and Adobe Flash revolutionised interactive content on the internet, paving the way to the standards which exist today:

> *"Since Adobe no longer supports Flash Player after December 31, 2020 and blocked Flash content from running in Flash Player beginning January*

> *12, 2021, Adobe strongly recommends all users immediately uninstall Flash Player to help protect their systems."*
>
> *"Open standards such as HTML5, WebGL, and WebAssembly have continually matured over the years and serve as viable alternatives for Flash content."*
>
> — Adobe Flash Player EOL General Information Page [5]

The modern web is an exciting and powerful platform with exceptional native support for desktop and mobile devices, advanced gestures and controls, and hardware acceleration. Mandelbrot Maps combines some of these technologies to provide a great experience which is open to all, with zero installation required and independent of operating system, requiring only a modern web browser like Chrome, Safari, or Edge.

This first version of Mandelbrot Maps fulfilled its aims by successfully showcasing how modern web technologies could be used to create a fast, interactive fractal explorer as a cross-platform web application.

The core functionality included displaying a split view of the Mandelbrot set and the associated Julia set, good desktop and mobile compatibility including touch gestures, and providing options like selecting the maximum iteration count.

1.2 Project Aim and Summary

Generally, as a software project increases in size and complexity, it inherently becomes harder to maintain, and this has been the case with Mandelbrot Maps. To address this common issue, the aims of this version of the project are to:

1. Define a software development and integration workflow to improve the maintainability of the application.
2. Improve the user experience and functionality of the application.
3. Provide information about the application and how to use it.
4. Evaluate the updated application's usability against last year's version.
5. Verify how effective the application's Search Engine Optimization was in improving its visibility on popular search engines.

The application was first converted from JavaScript to TypeScript [6] by assigning static types to functions and components. Types are checked by the TypeScript compiler, which provides enhanced type safety, and enables advanced code editing features like VS Code's IntelliSense [7] for faster development. Converting to TypeScript was completed over eight weeks, including miscellaneous changes. Following the conversion, a development workflow was designed using industry-standard code-quality tools (ESLint [8] and Prettier [9]) to automatically fix issues and format code consistently. Additional tools like `husky` [10], `pretty-quick` [11], `lint-staged` [12], `standard-version` [13], and `commitlint` [14] were combined to give a high degree of automation, reducing the need for contributors to manually perform some repeti-

tive tasks. A CI/CD pipeline implemented with GitHub Actions provides additional guarantees that new code is correctly tested once it reaches the remote repository.

The application's user interface (UI) has been updated to be more consistent and intuitive, with clearer UI elements, and making greater use of colour. New functionality includes *deep zoom* using perturbation theory (adapted from DeepMandelbrot, available at `github.com/munsocket/deep-mandelbrot`), the ability to rotate viewers, and support for *URL parameters*.

A new help menu has been added. Help topics are written using Markdown [15] and are loaded dynamically, making it simple to extend this menu with further information. It is currently possible to flip between the following three topics: a page with information about fractals and the layout of the viewers, a page explaining the controls and gestures for different devices, and a page explaining each setting where users can directly interact with it.

A survey was created to gather opinions on the application, with questions following last year's questions closely to allow for direct comparisons to be drawn between them. Questions created by Georgina and Fraser (section 1.5) were also included to maximize the amount of data collected while only requiring a single survey to be filled out. A total of 12 respondents completed the survey compared to 31 last year, giving the application an overall score of 4.08 out of 5, which is marginally lower than last year's. Responses were completely anonymous.

Lastly, the application's Search Engine Optimization (SEO) was tested using different search engines and queries to estimate how likely it is to appear in relevant searches, and if so, how far up it is displayed. When displaying search results, search engines tend to tailor those results to the user by building a profile based on factors like their geographical location, previous searches, and previous browsing history [16], so the evaluation has been designed to minimize those effects. This version of the project showcases the following achievements:

- Explicit declaration of the application architecture by converting the existing codebase from JavaScript into TypeScript, directly improving maintainability and reducing the barrier to entry for potential contributors.
- Definition of a development workflow which enforces consistent code quality and style, and automatically tests new code before it is pushed to a central repository.
- Creation of a continuous integration pipeline, enabling independent testing of new code after it has been pushed to a central repository.
- Implementation of new features, notably *deep zoom*, *viewer rotation*, *URL parameters*, and *a help menu*, giving the application a solid basis for further development in future.
- Improved search engine optimization, giving it first-page placement and even making it the top result in certain queries.

1.3 The Internet as a Self-Documenting Resource

This report details multiple concepts and technologies which are under active development, and as such are subject to constant change. Most of these technologies also see most of their use in commercial applications, and as such, they should be valued by their adoption and commercial success.

An Internet citing guide from Yale University explains how some professors discourage the use of "sources found or accessed over the Internet", and how these restrictions, even if sometimes justified, "may be excessive" [17].

Take Google's Material Design which is based on extensive Human-Computer Interaction research and design yet is published online through `material.io` as a living, breathing specification rather than a published paper. To refer to this and other technologies by pointing to a URL instead of a published journal resource may come across as carelessness, laziness, or both, but in cases where an online URL may legitimately be the primary, or perhaps even the only publishing medium, an Internet resource should not be frowned upon simply because it is an Internet resource.

In Mistry and Patel's "A Guide to Material Design, a Modern Software Design Language" [18], all references contained point to online URLs, which is completely justified because that is where the ground-truth specifications for Material Design and other resources are published. In "Understanding TypeScript" [19], Bierman, Abadi, and Torgersen also reference TypeScript (and Google's Dart programming language) by URL, once again citing primary sources which happen to be URLs.

The point to raise is that because the Internet is such a popular and easily accessible publishing medium, projects targeting an Internet audience will undoubtedly make use of those key strengths to reach said audience. Similarly, some articles cited within this report present recommendations on how to use and string together multiple technologies, and as such those are referenced as an opinion piece of how an end-goal may be achieved, not necessarily because they are a primary source, but because they help in reaching that end-goal by providing valuable insight which can only be gained by using said technology.

1.4 Report Overview

The rest of this chapter contains some more information about the logistics of the project, along with honourable mentions of those who have previously contributed to another version of Mandelbrot Maps as part of their projects.

Chapter 2 is intended as a quick recap of the topic of fractals, introducing some new concepts for this year's project update, and the basics of perturbation theory.

Chapter 3 explains the development and integration workflow proposed, and each of the technologies used in defining it.

Chapter 4 lists new updates to the application's looks and functionality.

Chapter 5 contains different metrics to evaluate the success of the development workflow and updates to the application, as well as the results from the user survey.

Chapter 6 gives a summary of the key features of the project and how it might be improved in future.

1.5 Project Coordination

This year, two others were assigned this project: Georgina Medd and Fraser Scott. While my focus was set on defining a workflow and making general improvements, Georgina chose to explore the educational side of fractals, and Fraser researched and implemented a visualisation of Tan's theorem on top of this application.

Fraser's contribution has been tested and fully integrated into the main branch of the application, which is now live. By setting up a development workflow as proposed in chapter 3, it was easy to validate these new additions to the project, since they would have automatically been formatted, checked, and tested for compatibility both on the developer's local machine and separately on GitHub as a sanity check before being merged.

Chapter 2

Fractals: A Refresher

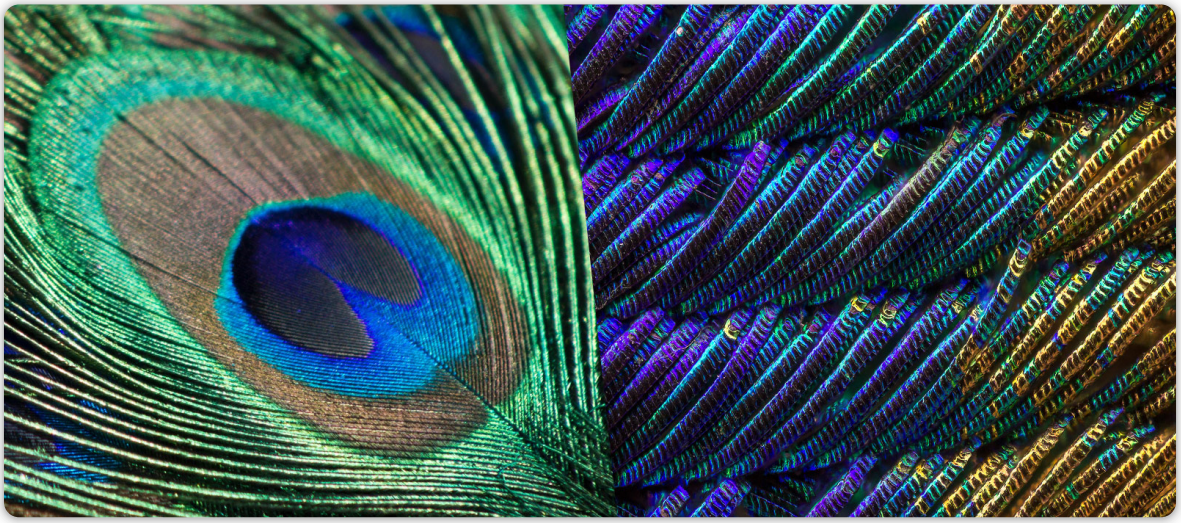


Figure 2.1: Peacock feathers exhibit fractal properties.
(L) The centre portion of the feather somewhat resembles the main bulb of the Mandelbrot set. © Caleb Minear (@calebminear) – unsplash.com/photos/BwMtUpBYLIs
(R) A beautiful close-up view of a peacock feather: its alternating, repeating series of strands made of smaller strands clearly visible. © Waldo Nell (pwnell) – flickr.com/photos/pwnell/25549906864

In Part 1 of the project report, the concept of fractals was introduced: fractals are "*structures which exhibit some kind of self-similarity at different scales*". Some real-life examples include Romanesco broccoli, mountains, lightning, and peacock feathers. As stunning as these look, they only exhibit approximate fractal properties – they are "*approximate fractals*" – since the real world has physical limitations dictating that self-similarity can only be observed over a finite range [20].

The domain of mathematics, however, has at its disposal a powerful instrument: the continuous real number line, which can be infinitely zoomed into further and further, only to reveal ever increasing detail.

2.1 The Complex Plane

As a quick reminder, complex numbers are defined by a *Real* component and an *Imaginary* component, and can be represented in the form:

$$z = a + b\mathbf{i} \quad \text{where } a, b \in \mathbb{R} \text{ and } \mathbf{i} = \sqrt{-1} \tag{2.1}$$

Plotting a and b on a two-dimensional plane with a as the horizontal axis and b as the vertical axis yields a visual representation of a complex number on the *complex plane*, shown in figure 2.2.

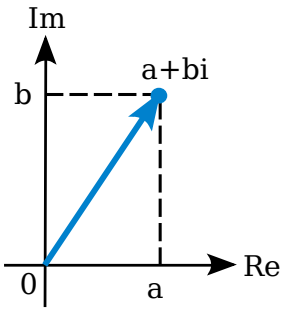


Figure 2.2: A plot of a complex number (© Wolfkeeper at English Wikipedia, CC BY-SA 3.0)

The magnitude of a complex number is the square root of the sum of the squares of its a and b components:

$$|z| = \sqrt{a^2 + b^2} \tag{2.2}$$

Complex numbers support addition and multiplication:

$$z + w = (a + b\mathbf{i}) + (c + d\mathbf{i}) = (a + c) + (b + d)\mathbf{i} \tag{2.3}$$

$$z \times w = (a + b\mathbf{i}) \times (c + d\mathbf{i}) = ac + ad\mathbf{i} + bc\mathbf{i} + bd\mathbf{i}^2 = (ac - bd) + (ad + bc)\mathbf{i} \tag{2.4}$$

2.2 Mandelbrot set

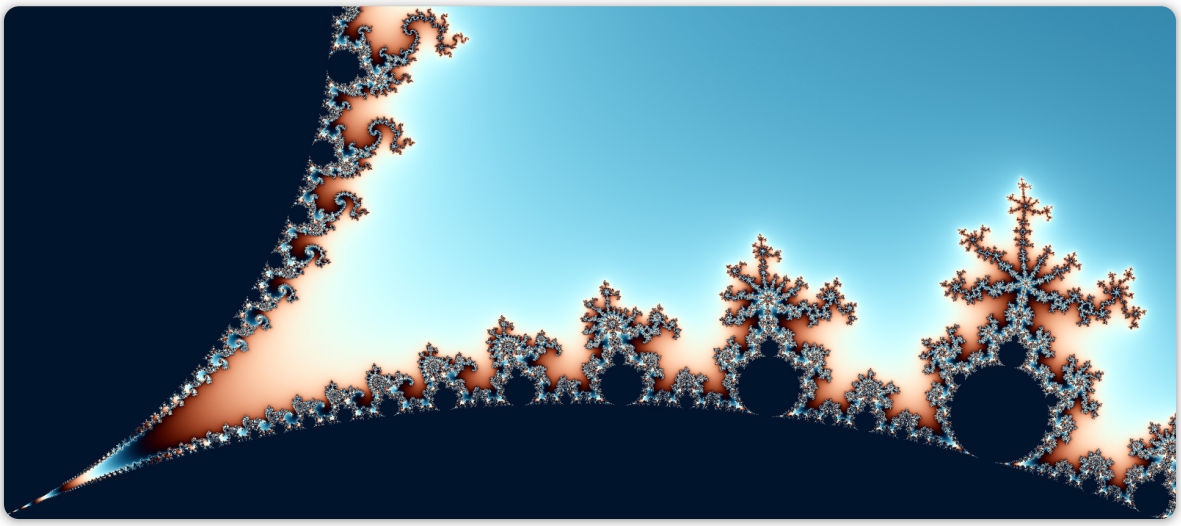


Figure 2.3: An image of the Mandelbrot set rendered by this application: jmaio.github.io/mandelbrot-maps/#/m@-0.751842,0.287638,9,1.17

The Mandelbrot set is a fractal defined by the quadratic recurrence in equation 2.5:

$$z_{n+1} = z_n^2 + z_0 \quad \text{where } z \in \mathbb{C} \tag{2.5}$$

Iterating according to this equation yields the following:

$$
\begin{aligned}
z_0 & & & = z_0 \\
z_1 = (z_0)^2 + z_0 & & = (z_0)^2 + z_0 & = z_0^2 + z_0 \\
z_2 = (z_1)^2 + z_0 & & = (z_0^2 + z_0)^2 + z_0 & = z_0^4 + 2z_0^3 + z_0^2 + z_0 \\
z_3 = (z_2)^2 + z_0 & & = (z_0^4 + 2z_0^3 + z_0^2 + z_0)^2 + z_0 & = \dots
\end{aligned}
$$

Instinctively, this recurrence in z_n might already indicate a form of self-similarity since each iteration is dependent on the previous value in the series. For point z_0 to be in the Mandelbrot set, it must remain bounded (not diverge) under iteration [21]. Divergence is guaranteed if the magnitude of z_n is ever greater than 2, such that it satisfies $|z_n| > 2$ [22]. *Note that starting with $z_0 = 0$ or with an initial point $z_0 = c$ is roughly equivalent, with the only difference being that starting with $z_0 = 0$ produces equivalent results with a delay of one iteration compared to the other method.*

The number of iterations n until divergence (or up to a limit $I_{\max}$ if the point does not diverge) is then used to colour the point - this constitutes the *escape-time algorithm* [21]. In figure 2.3, the dark blue-black colouring along the left and bottom edges denotes regions where the points contained do not diverge, whereas the orange and light blue colouring denotes regions which diverge at different speeds, with regions in lighter blue diverging within fewer iterations than those in orange.

On a digital display, which can be thought of as a finite rectangular grid of square pixels, the centre of each pixel is mapped to a complex number. All the pixels will then follow this iterative process, being assigned a different colour which represents how quickly they diverge, and will eventually form the patterns above.

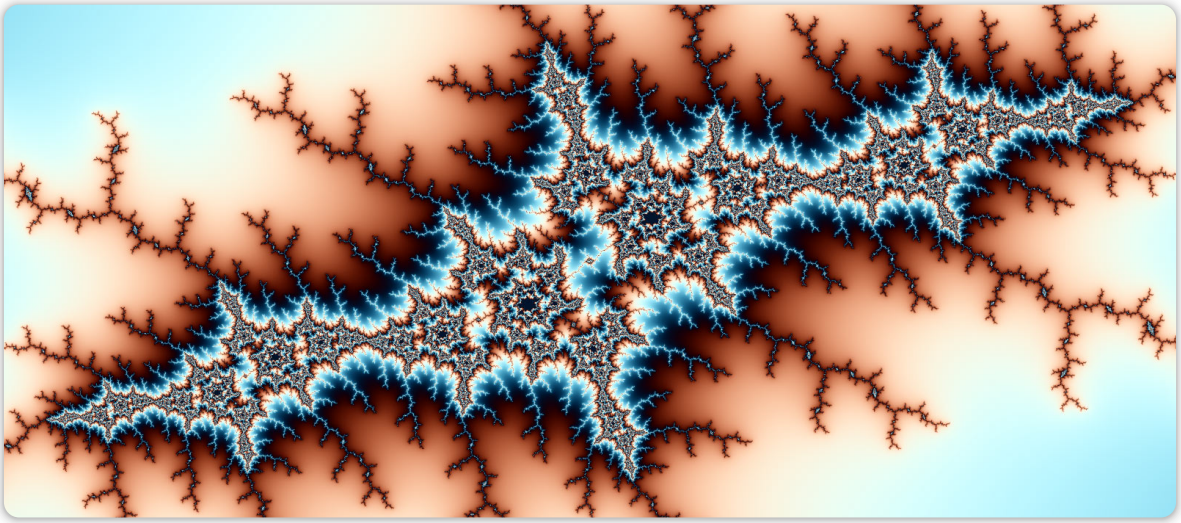


Figure 2.4: An image of a Julia set rendered by this application:
jmaio.github.io/mandelbrot-maps/#/m@-0.03997,0.98612,6500,0/j@0,0,23,1

2.3 Julia sets

Julia sets – note, plural – are similar to the Mandelbrot set. They follow roughly the same recurrence:

$$z_{n+1} = z_n^2 + c \quad \text{where } c \in \mathbb{C} \tag{2.6}$$

The constant c is fixed for a single Julia set – or, inversely, there is a single Julia set associated with every complex number $c \in \mathbb{C}$. By extension, if every point c is associated with a Julia set, then every point on the Mandelbrot set will have an associated Julia set. This can be shown visually by "choosing" a point from the Mandelbrot set to represent the fixed constant c.

This set of points which do not diverge under iteration is called a "filled-in" Julia set J_r, where the *true* Julia set J is the *boundary* of the filled-in set J_r [23]. Note that for simplicity, we use the term "Julia set" to refer to the "filled-in" Julia set, as that is the only one which we refer to in this context.

2.4 Quick Maths - Graphics Processing Units

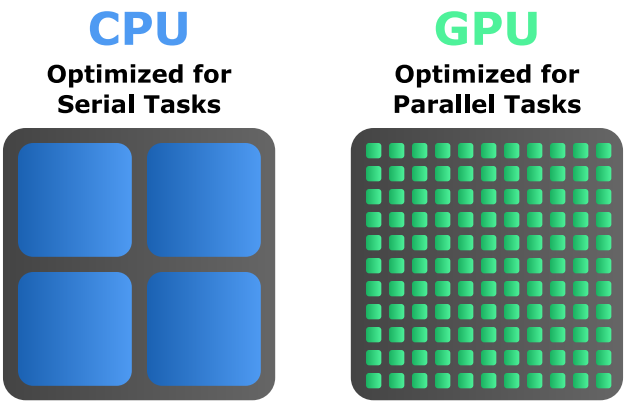


Figure 2.5: *"While a modern CPU might be good at running four dissimilar tasks at once, a GPU is good at running many hundreds of tasks at once (as long as those tasks are very similar)."* – Chapter 1, Graphics Programming Compendium, Ian Dunn and Zoë Wood (graphicscompendium.com). In Mandelbrot Maps, the "*task*" is to iterate different z according to equation 2.5.

Iteratively computing whether a point is a member of the Mandelbrot set using the escape-time algorithm is computationally intensive, primarily due to having to iterate multiple times on a previous result. Despite this limitation, it is possible to speed up overall execution by performing multiple of these computations simultaneously, using parallelism.

Unlike regular general-purpose processors (*Central Processing Units*, or *CPU*s) found in desktop and laptop computers, *Graphics Processing Units* (*GPU*s) are specialized hardware accelerators designed for highly parallel workloads. A GPU will generally perform better than a CPU in this specific task, since it can spread the work of

iterating multiple points over hundreds to thousands of individual processing cores simultaneously.

2.5 Perturbation Theory and "Arbitrary" Precision

One of the limitations presented in last year's version of the project was the reduced amount of zoom offered by that version of Mandelbrot Maps when compared to traditional CPU-based implementations, which tend to use 64-bit floating-point numbers. To get around this limitation, CPU-based implementations usually switch to arbitrary-precision numbers after a certain zoom level, which reduces performance significantly. Due to WebGL's comparatively lower precision 32-bit floating-point numbers, zooming into a region of the Mandelbrot set viewer quickly reaches this precision limit, after which the viewer loses detail and becomes pixelated (figure 2.6).

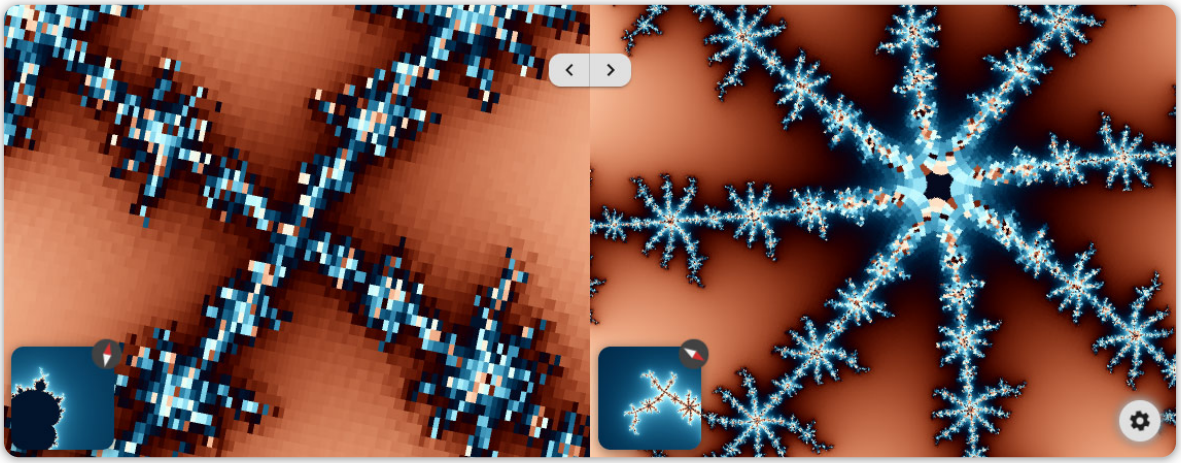


Figure 2.6: WebGL has limited, 32-bit precision compared to traditional, 64-bit precision CPU renderers. In Mandelbrot Maps, blocky, pixelated structures begin to appear at large factors of magnification. The centre of the structure on the right appears hyperbolic, likely due to floating-point precision which transitions abruptly around certain points. jmaio.github.io/mandelbrot-maps/#/m@0.366363,0.5915338,870293.2,0.2/j@-0.0294855,-0.7480627,1353.36,2.12

DeepFractal was a WebGL-based Mandelbrot set explorer reviewed in the previous MInf 1 report which was able to achieve a much deeper level of zoom than other WebGL applications by using "perturbation theory", a technique proposed by K. I. Martin which uses series approximations to accelerate fractal rendering in SuperFractalThing. [24] This technique leverages the fact that the standard implementation of floating-point numbers – the *IEEE Standard for Floating-Point Arithmetic* (IEEE 754) – is not evenly distributed across the full range of numbers, but rather is "denser" around zero, which can be seen in figure 2.7. Hardware floating-point numbers trade accuracy for speed while still being sufficiently accurate for most purposes but tend to require workarounds for those purposes where high-precision arithmetic is essential. [25]

The mathematics behind this technique is summarised and adapted to our naming convention in definition 1.

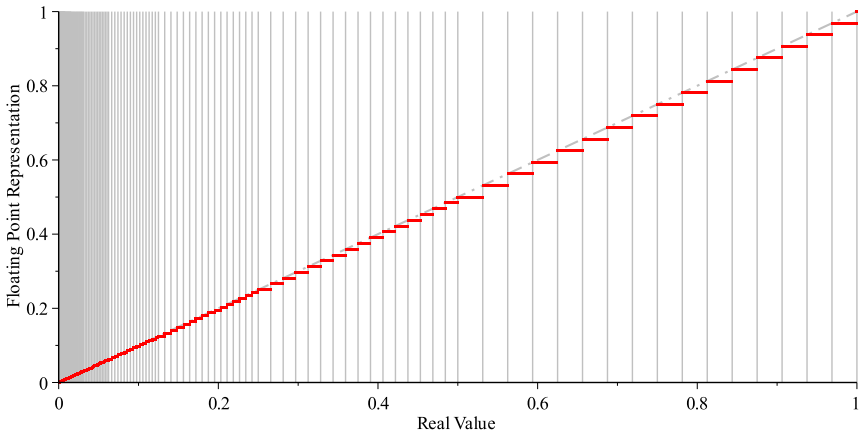


Figure 2.7: Non-uniform Distribution of (64-bit) Floating-Point Numbers. "*Floating-point numbers are non-uniformly distributed over the real line. The spacing of the floating-point numbers is magnified by a factor 2 at each power of 2.*" (ElShaarawy and Gomaa) [26]

Definition 1 (from SuperFractalThing Maths by K. I. Martin, [24] adapted to follow this project's convention, and supplemented by Neal Lawton's answer on Mathematics Stack Exchange: `math.stackexchange.com/a/1071945/757765`)

We consider two nearby points z_0 and w_0 in the complex plane, and attempt to compute whether each is in the Mandelbrot set. The Mandelbrot set equation can be written as in equation 2.5:

$$z_{n+1} = z_n^2 + z_0$$

where the complex number z_0 is in the Mandelbrot set if z_n stays finite as n tends to infinity. Consider another point given by w_n:

$$w_{n+1} = w_n^2 + w_0$$

The difference between these two points at a given iteration is given by Δ_n such that:

$$\Delta_n = w_n - z_n \tag{2.7}$$

$$(w_n = z_n + \Delta_n \quad or \quad z_n = w_n - \Delta_n)$$

Iterating the difference Δ_n yields:

$$\begin{aligned}
\Delta_{n+1} &= w_{n+1} - z_{n+1} \\
\Delta_{n+1} &= (w_n^2 + w_0) - (z_n^2 + z_0) \\
\Delta_{n+1} &= ((z_n + \Delta_n)^2 + (z_0 + \Delta_0)) - (z_n^2 + z_0) && \text{[by } w_n = z_n + \Delta_n \text{, equation 2.7]} \\
\Delta_{n+1} &= (z_n^2 + 2z_n\Delta_n + \Delta_n^2 + z_0 + \Delta_0) - (z_n^2 + z_0) \\
\Delta_{n+1} &= 2z_n\Delta_n + \Delta_n^2 + \Delta_0 && (2.8)
\end{aligned}$$

All the numbers in equation 2.8 are "small", allowing it to be calculated with hardware floating point numbers. The values z_n can then be used to calculate w_n without having to use arbitrary precision calculations.

Let $\delta = \Delta_0$; by equation 2.8, collecting by δ^n terms:

$$
\begin{aligned}
(n=0) \quad \Delta_1 &= 2z_0\Delta_0 + \Delta_0^2 + \delta \\
&= 2z_0\delta + \delta^2 + \delta \\
&= (2z_0+1)\delta + \delta^2
\end{aligned}
$$

$$
\begin{aligned}
(n=1) \quad \Delta_2 &= 2z_1\Delta_1 + \Delta_1^2 + \delta \\
&= 2z_1((2z_0+1)\delta + \delta^2) + ((2z_0+1)\delta + \delta^2)^2 + \delta \\
&= 2z_1(2z_0+1)\delta + 2z_1\delta^2 + ((2z_0+1)\delta)^2 + 2(2z_0+1)\delta\delta^2 + (\delta^2)^2 + \delta \\
&= (4z_1z_0 + 2z_1 + 1)\delta + 2z_1\delta^2 + (2z_0+1)^2\delta^2 + (4z_0+2)\delta^3 + \delta^4 \\
&= (4z_1z_0 + 2z_1 + 1)\delta + 2z_1\delta^2 + (2z_0+1)^2\delta^2 + (4z_0+2)\delta^3 + \delta^4 \\
&= (4z_1z_0 + 2z_1 + 1)\delta + (2z_1 + (2z_0+1)^2)\delta^2 + (4z_0+2)\delta^3 + \delta^4
\end{aligned}
$$

(N.B. the final δ^3 coefficient above is "$(4z_0+2)$", where the original paper contains "$(4z_0-2)$" in its derivation, which is incorrect as pointed out by Neal Lawton on Stack Exchange [27])

$$\text{Let } \Delta_n = A_n\delta + B_n\delta^2 + C_n\delta^3 + O(\delta^4) \tag{2.9}$$

Then:

$$
\begin{aligned}
\Delta_0 = \delta &= 1\delta + 0\delta^2 + 0\delta^3 \\
\therefore A_0 &= 1,\ B_0 = 0,\ C_0 = 0
\end{aligned}
$$

$$
\begin{aligned}
\Delta_{n+1} &= 2z_n\Delta_n + \Delta_n^2 + \Delta_0 \\
&= 2z_n(A_n\delta + B_n\delta^2 + C_n\delta^3 + O(\delta^4)) + (A_n\delta + B_n\delta^2 + C_n\delta^3 + O(\delta^4))^2 + \delta \\
&= 2z_nA_n\delta + 2z_nB_n\delta^2 + 2z_nC_n\delta^3 + 2z_nO(\delta^4) + (A_n\delta + B_n\delta^2 + C_n\delta^3 + O(\delta^4))^2 + \delta \\
&\quad \text{(terms of order } \delta^4 \text{ or higher are small)} \\
&= (2z_nA_n + 1)\delta + (2z_nB_n)\delta^2 + (2z_nC_n)\delta^3 + A_n^2\delta^2 + 2A_nB_n\delta^3 + O(\delta^4) \\
&= (2z_nA_n + 1)\delta + (2z_nB_n + A_n^2)\delta^2 + (2z_nC_n + 2A_nB_n)\delta^3 + O(\delta^4)
\end{aligned}
$$

Finally:

$$A_{n+1} = 2z_nA_n + 1 \qquad (A_0 = 1) \tag{2.10}$$

$$B_{n+1} = 2z_nB_n + A_n^2 \qquad (B_0 = 0) \tag{2.11}$$

$$C_{n+1} = 2z_nC_n + 2A_nB_n \qquad (C_0 = 0) \tag{2.12}$$

Equations 2.10 to 2.12 can be applied iteratively to calculate the coefficients for equation 2.9, which can then be used to calculate the value for the nth iteration for all the points around z_0. The approximation should be good as long as the δ^3 term has a magnitude significantly smaller than the δ^2 term.

Chapter 3

Architecture Definition and Workflow

Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments.

— Atlassian Agile Coach [28]

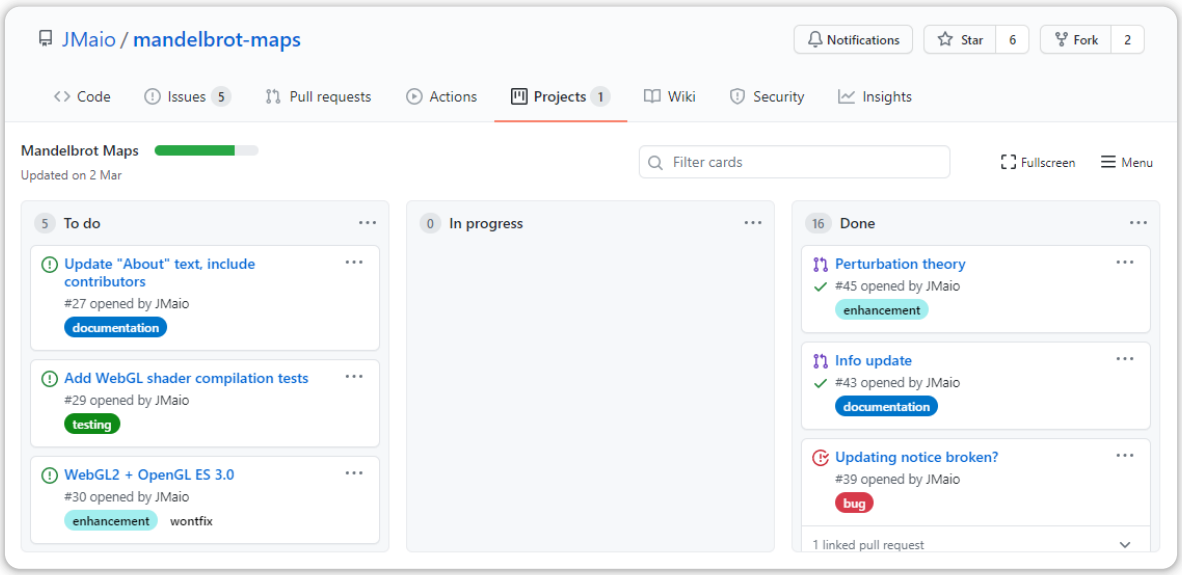


Figure 3.1: GitHub project board for Mandelbrot Maps, like a Kanban board [29]. Columns correspond to a status, cards within columns indicate tasks, and tasks move towards the right as they progress.

3.1 Motivation

The primary aims of Mandelbrot Maps are well-defined, but its development process has seen the codebase undergo constant change as new design and functionality is trialled. Given the constrained development timeframe of an honours project, iterating quickly is a necessity. As explained by Atlassian, adopting an Agile methodology is generally a

good way to start managing the complexity of a software engineering project, and as Mandelbrot Maps has grown and become more complex, it reached the point where it was necessary to acknowledge and address this complexity by fully embracing Agile. While last year's version of the project loosely followed an Agile methodology, this year's version does so more rigorously by having it more explicitly defined.

After the first part of the project was completed last year, the pause between then and the start of the next semester was dedicated to researching possible approaches that would minimise the complexity of maintaining or contributing to the project. Approaches like Scrum [30] were considered but not pursued since they mostly consider teams of developers as opposed to a single developer. As an iterative process in itself, defining and refining the workflow for this part of the project has seen it make heavy use of core GitHub features [31] like *Issues* and *Pull Requests*, *Boards*, and *Actions* (for CI/CD, section 3.4), and it has evolved to resemble a Kanban workflow [29].

Another consideration is that the application has been open-source and live on the internet for a year, so every effort has been made to ensure that the live "production" version of the application remains online and stable during development, even as new functionality is added. If the production version of the application breaks, the page will be unavailable: this might be an inconvenience to users, but it could also hurt the application's rank on search engines since broken pages are less likely to be indexed correctly. Breaking the application in a production environment must be avoided at all costs. Reverting back to an older working version is the current safeguard in case this ever happens.

3.2 TypeScript

> *Writing types helps programmers be more confident about their code because types catch mistakes.*
>
> — TypeScript Docs: Why does TypeScript exist? [6]

Last year, the project was implemented using plain JavaScript, which is *dynamically typed* [32], and can make it difficult to detect potential errors during an application's development phase.

TypeScript is "*a superset of JavaScript that compiles to clean JavaScript output*" [33] which is developed and maintained by Microsoft. Using TypeScript makes it possible to explicitly declare static type definitions for existing JavaScript code to clearly "*describe the shape of objects and functions*" [6]. These static types are then verified by the TypeScript compiler which ensures that type errors are identified during compilation, and subsequently transforms that code into plain JavaScript code. TypeScript enables detection of potential issues during development rather than after they are already affecting users.

Converting Mandelbrot Maps to TypeScript was therefore considered a valuable way to spend the initial part of the project. Migrating the project to TypeScript could have proved time-consuming at first, but the potential for faster future development would

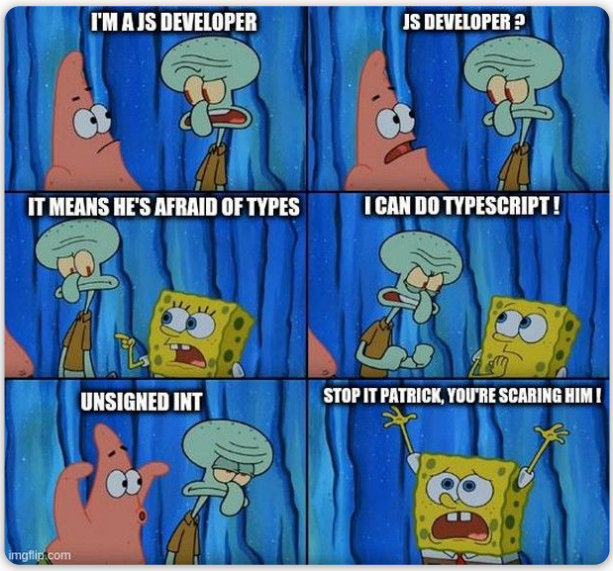


Figure 3.2: An internet comic depicting the common sentiment towards JavaScript's lack of static typing. TypeScript alleviates this by performing static type checking at compile-time, making it far more robust than was previously possible. From: javascript.plainenglish.io/webgl-frameworks-three-js-vs-babylon-js-36975d915694

offset the initial time investment, so this migration was given a higher priority over implementing new functionality.

The trade-off equated to either spending time at the start of the semester converting to TypeScript, or, alternatively, not converting to TypeScript, but then potentially having to deal with obscure issues that could arise later in development, inevitably spending time looking for and fixing errors which could have easily been prevented if static typing had been in place from the start.

Such a commitment could have carried considerable risk if the conversion process proved unsuccessful. However, since TypeScript is designed for gradual adoption, it is possible to migrate gradually and still enjoy its benefits, as opposed to being forced into an all-or-nothing situation where either none of the code is converted or all of it is for the application to function. Initial testing of TypeScript conversion was carried out on simpler components to become familiar with the TypeScript syntax and tooling, which required establishing basic static types such as those used to represent viewer locations:

- `XYType = [number, number]`
- `ZoomType = number`
- `ThetaType = number`

Once some basic types had been established, more complex types and interfaces could be composed using those types and others from existing libraries (like `react-spring`'s `SpringConfig` [34]):

- `interface ViewerLocation {xy: XYType; z: ZoomType; theta: ThetaType}`
- `interface SpringControl {config?: SpringConfig}`
- `interface ViewerXYControl extends SpringControl {xy: XYType}`

- `interface ViewerControls {xy: ViewerXYControl; ...}`

Declaring types has given each part of the existing application a set of constraints which it must adhere to, extracting the underlying architecture which had already been designed last year and formalising it in the process. For Mandelbrot Maps, being a small project, this process was completed, part-time, in about eight weeks.

```
function MandelbrotRenderer(props) {
  ...
}
```

```
interface RendererProps extends React.StyleHTMLAttributes<HTMLDivElement> {
  controls: ViewerControlSprings;
  maxI: number;
  useDPR: boolean;
  DPR: number;
  useAA: boolean;
  colour: RgbColor;
  precision: precisionSpecifier;
}

interface MandelbrotRendererProps extends RendererProps {
  showCrosshair: boolean;
}

function MandelbrotRenderer({
  precision,
  ...props
}: MandelbrotRendererProps): JSX.Element {
  ...
}
```

Figure 3.3: Converting to TypeScript requires declaring multiple types, which involves an initial time investment that can save time later in development.
Left: JavaScript; Right: TypeScript

Other projects such as Babylon.js (a web rendering engine) have also undertaken the migration to TypeScript to improve their tooling, both for existing developers to streamline their development process, but also for new developers to get started with the project more quickly because of the additional context that is provided by TypeScript, while simultaneously preparing their projects for the future of JavaScript [35].

3.3 Testing

As happened last year, most feature testing has been done manually. Automated testing of a WebGL-based application is not straightforward due to the nature of the drawing process performed by the GPU happening directly to an HTML `canvas` element, as opposed to creating DOM elements as is most common with user interfaces. This limitation is explained in detail by Rockwood in *Automated Cross-Browser Testing for WebGL – It's Not Going to Happen* [36].

The existing unit testing only gives some basic assurance that the application runs without crashing, but cannot test certain features affecting the display of the fractal viewers. This testing has been automated locally using `husky` (see section 3.4.5), which will require that all test suites pass on `pre-push`. Currently, `husky` will perform the following actions for testing:

- `pre-push` – After committing, but before pushing code to the repository, the project will be rebuilt to ensure that it compiles without errors, followed by running unit tests with the `CI` environment variable, which halts on error. If any errors are raised as part of the `pre-push` checks, `husky` will cancel the push.

Figure 3.6b shows the error produced by a failing test suite. After the code has been successfully tested locally, the push is approved. Once newly pushed code reaches the remote GitHub repository, it is once again put through a similar testing pipeline using GitHub Actions, which will independently validate that the project is able to successfully build and pass all tests.

3.4 Continuous Integration / Continuous Deployment

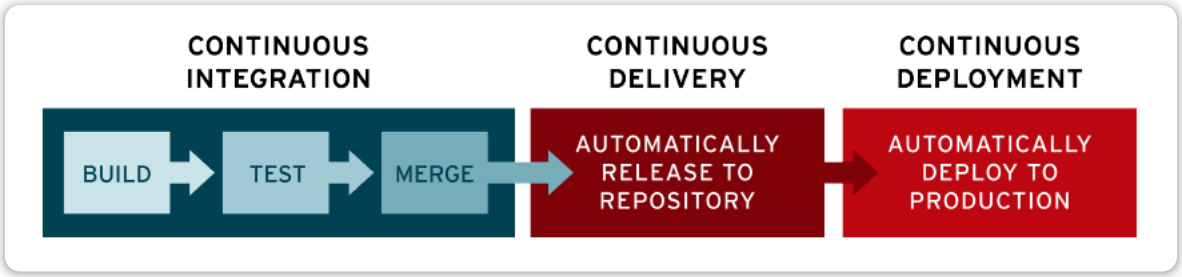


Figure 3.4: An example CI/CD pipeline

CI/CD, as it is often abbreviated, "*introduces automation into the stages of app development,*" helping with the integration of new code into existing projects. The literature from Red Hat explains:

> *The "CI" in CI/CD always refers to continuous integration, which is an automation process for developers.*
>
> *The "CD" in CI/CD refers to continuous delivery and/or continuous deployment, which are related concepts that sometimes get used interchangeably.*
>
> *Continuous delivery means changes to an application are automatically bug tested and uploaded to a repository (like GitHub).*
>
> *Continuous deployment (the other possible "CD") can refer to automatically releasing a developer's changes from the repository to production.*
>
> — What is CI/CD? – Red Hat [37]

Red Hat considers that "successful" CI means that code is "*regularly built, tested, and merged to a shared repository*". Regular testing is crucial as it ensures that new changes do not break the existing application. Regular merging also encourages a reduction in the number of branches in development at any one time.

The "CD" steps of the pipeline also contribute to seamless deployment of new code with minimal effort, largely because they "*address [the issues] that slow down app delivery*" by automating parts of the workflow between code being merged into a repository and being deployed to a production environment. The following sections detail how this was put into practice by automating the checking, testing, and subsequent deployment of new code.

3.4.1 Linting and Style Enforcement

> *Every major open-source project has its own style guide: a set of conventions (sometimes arbitrary) about how to write code for that project. It is much easier to understand a large codebase when all the code in it is in a consistent style.*
>
> *"Style" covers a lot of ground, from "use camelCase for variable names" to "never use global variables" to "never use exceptions".*
>
> — Google Style Guides [38]

Figure 3.5: Code editor on-save example. A supported code editor will call ESLint and Prettier for formatting as the file is being saved, removing the need to manually format code. Source file: `src/components/render/WebGLCanvas.tsx`

As evidenced by Google, creating and following a set of conventions for how to write code is imperative. The code in Mandelbrot Maps had previously consisted of JavaScript written mostly following a "best-guess", personal style, which itself inherits from existing styles like that of the React documentation, StackOverflow answers, or other projects, but without explicitly following a style convention.

Complementing the shift to TypeScript, this version of the project has adopted powerful tooling to enforce style while also finding and fixing code issues using ESLint [8] and Prettier [9]. These steps have been automated for local development using `husky` [10]. These and other tools listed below are distributed and executed in the same Node.js environment as the existing packages used for development (like React), meaning that no additional software should need to be installed for the workflow to function correctly, making it highly portable and easy to use.

3.4.2 ESLint – Code quality

ESLint is a tool which can "*find and fix problems in your JavaScript code*" – *linting* – by performing syntax-aware static analysis [8]. Through a system of rules, ESLint can be configured to specify the desired code style, and this can be extended with

plug-ins. These plug-ins allow for ESLint to find common React issues, such as `eslint-plugin-react-hooks` which helps in correctly applying the rules of React hooks used in Mandelbrot Maps.

Our configuration includes the standard "recommended" rules, as well as rules for React apps and TypeScript apps. Prettier is also included in the configuration to allow it to be executed together with ESLint, giving it rich code editor integration for making suggestions.

3.4.3 Prettier – Code styling

Prettier describes itself as "*an opinionated code formatter*" [9]. It uses predefined rules to automatically apply consistent code styling, and has been configured to run across the entire Mandelbrot Maps project, mostly for TypeScript and JavaScript files, but also on other supported file types like JSON, Markdown, or HTML. Its tight integration with editors like VS Code enables it to apply these rules on-save, making it effortless to use. Since VS Code can be configured by using a file named `settings.json`, this project stores its project-level preferences on version control to distribute them to all contributors, automatically enabling integrations like on-save linting. An example is shown in figure 3.5.

3.4.4 `commitlint` – Better commits

To aid versioning and encourage informative commit messages, we have adopted `commitlint` [14] to lint commit messages according to the Conventional Commits specification [39], which defines a set of guidelines for assigning a type, scope, and summary to all commit messages. Simple commit messages should be defined according to the format:

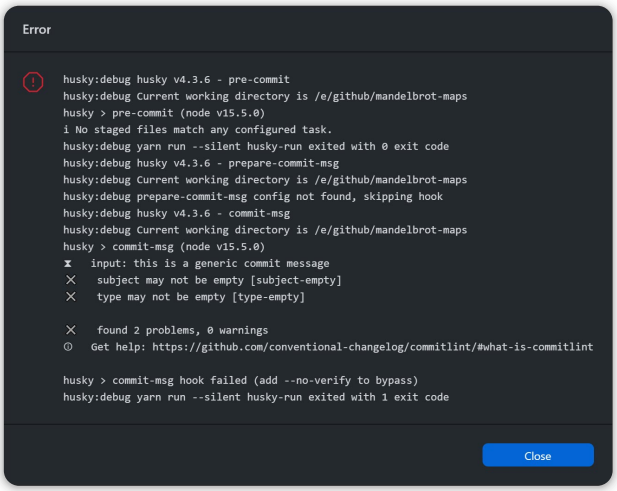
```
<type>[optional scope]:  <description>
```

We follow the Angular commit convention [40], where the `<type>` should be a keyword from the set `build|ci|docs|feat|fix|perf|refactor|test`, which indicates the context of the commit. The `[optional scope]` consists of a noun describing a section of the codebase, which in our case has included `controls`, `settings`, `perturbation-theory`, and others as required. The `<description>` should be a short summary of the code changes, such as "*add rotation indicator and reset to minimaps*". The commit message below (SHA `098d24f`) indicates that a new feature has been added within the scope of the application's controls:

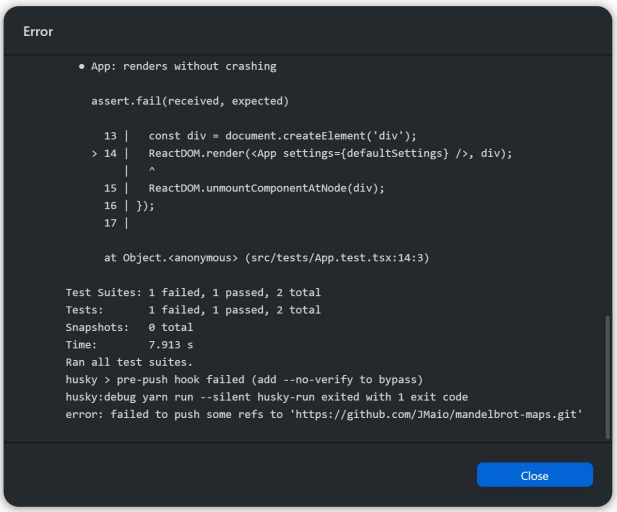
```
feat(controls):  add rotation indicator and reset to minimaps
```

3.4.5 `husky` – git hooks

Linting and style enforcement have been automated using `husky` [10], which uses git hooks [42, 8.3 Git Hooks] to perform predefined commands when certain events are triggered in a local git repository. If a new action such as a commit or push is detected, `husky` will intercept that action and execute commands which have been defined in



(a) `husky` throws an error after attempting to lint a commit with a nonconformant commit message ("*this is a generic commit message*"). `commitlint` requires the commit to contain at least a *subject* and *type*.



(b) `husky` runs all tests before pushing code to a remote repository. Attempting to push with failing tests will fail.

Figure 3.6: `husky` automates testing and prevents errors from being committed. GitHub Desktop [41] displays errors as popup alerts.

its configuration file. Currently, `husky` has been configured to perform the following actions for linting and style enforcement on the following git hooks:

- `pre-commit` – Before a new commit is created, the `lint-staged` [12] utility is executed, which then runs the `pretty-quick` [11] utility to automatically re-apply the desired code style to selected files using the existing Prettier configuration. This step guarantees that even if Prettier had not been able to format the code in the editor on-save, which could happen if the editor used does not support on-save actions, then that code will now be correctly formatted outside of the editor by `pretty-quick`.
- `commit-msg` – When a new commit is created, its commit message is checked by `commitlint` [14], which requires the message to conform to the Angular commit convention [40].

Once all these checks have completed successfully, the commit is considered valid. Figure 3.6a shows the error produced by attempting to create a commit without the required structure.

3.4.6 Yarn – Dependency management

Dependencies have been managed using Yarn, a popular Node.js package manager originally developed by Facebook which focuses on speed and security [43]. Yarn provides improvements over the default `npm` package manager, and the most recent version Yarn 2 is even more stable and efficient [44].

Ensuring that dependencies stay up to date is a continuous process as dependencies are also updated periodically, which can happen because of feature updates, bugfixes, critical security updates, or other reasons. Part of this process can be automated using Dependabot [45], which scans the package files created by Yarn (it also supports `npm`) to determine the versions of dependencies used by the application, and automatically creates pull requests to update any dependencies where security vulnerabilities are identified. As part of its acquisition of Dependabot, GitHub now provides this feature free of charge for public repositories.

Because Mandelbrot Maps is a stateless application without connection to a central server, dependency security vulnerabilities are unlikely to cause any harm. All the code in Mandelbrot Maps is only ever executed locally in a web browser, and not having a central server means that there is no attack vector to be exploited by a potential attacker. Regardless, every project should maintain its dependencies up to date, especially those which are flagged for potential security vulnerabilities, and Mandelbrot Maps has been configured to encourage this.

3.4.7 `standard-version` – Versioning and changelog

Versioning has become more important in Mandelbrot Maps as more functionality was added. As a safeguard, versioning acts as a checkpoint for when the application is known to work as expected. When prototyping the application last year, proper versioning was not as relevant because the project was still subject to major changes without notice. However, after having completed that first implementation, that was taken as the de facto "`v1.0`", with some changes then justifying version `v1.0.1` towards the end of that project.

Already when experimenting with releases in the past, manual creation of versions was considered less than ideal, so we have automated this in Mandelbrot Maps with `standard-version` [13], which updates version numbers in files, creates new git tags, and generates changelogs when called to create a release. New versions follow Semantic Versioning 2.0 [46] as is common among software projects, and automatic changelog generation is done by retrieving information from commit messages, which as above should now follow the Conventional Commits specification.

3.4.8 GitHub Actions – Cloud pipelines

GitHub Actions provides a Continuous Integration pipeline which acts as a secondary testing mechanism that is configured to run whenever a new commit or pull request is created. The remote repository uses fresh environments when testing, which ensures that new code is built and tested in isolation. Specifically, a GitHub Actions configuration named "Yarn CI" has been created for the Mandelbrot Maps GitHub repository to build and test the application by performing the following:

1. Set up job environment
2. Checkout new code
3. Set up Node.js

4. Install all required packages using Yarn
5. Build the project using React
6. Test: run the project's unit tests
7. Clean up job
8. Complete job

After this job has completed successfully, the status of the commit or pull request is updated. If the job fails, the commit is labelled as "failing"; if this commit is the latest in a pull request, the pull request will also be labelled as "failing", and merging will be discouraged.

Another GitHub Actions job named "Auto-deploy new tag" has been configured to automatically deploy the application when a new version is pushed to the `master` branch. This job installs and builds the project like the "Yarn CI" job, but proceeds to deploy that new build to the `gh-pages` branch of the repository, which is then served by GitHub Pages to the application's URL at `jmaio.github.io/mandelbrot-maps`.

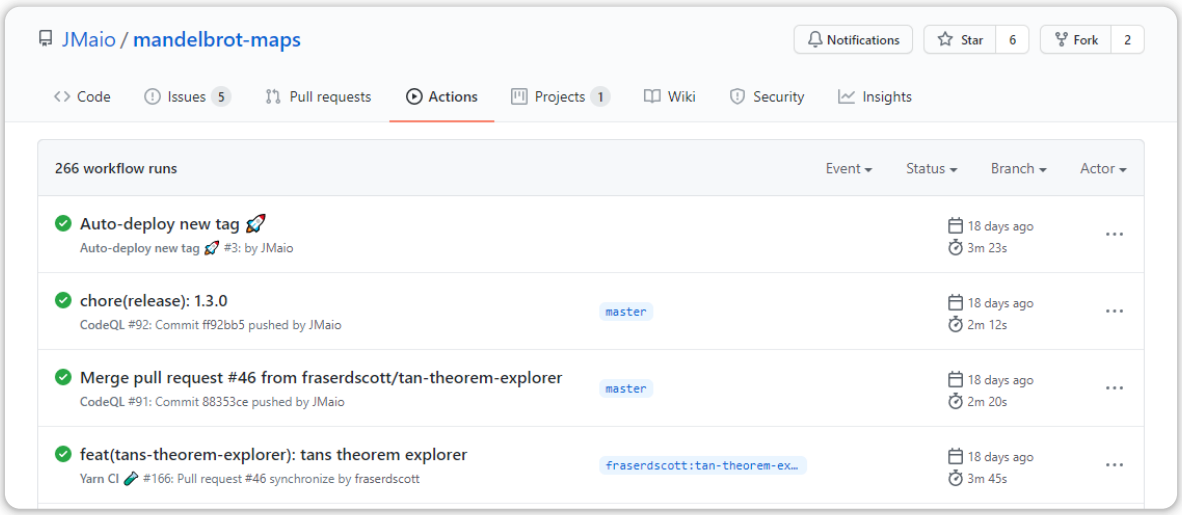


Figure 3.7: GitHub Actions acts as the remote CI/CD pipeline.

3.5 Documentation

Additional documentation has been written to include an outline of the steps required to deploy a public fork of the main Mandelbrot Maps repository to GitHub pages, so that those forks can be tested outwith the main application repository.

A Wiki has also been added to the GitHub repository, which currently contains a section titled "Committing guide" whose purpose is to act as a shortcut to the `commitlint` conventional configuration, so that it can quickly be referenced for available commit message *types* as set out by the Conventional Commits spec.

Chapter 4

The Application

4.1 Visual identity

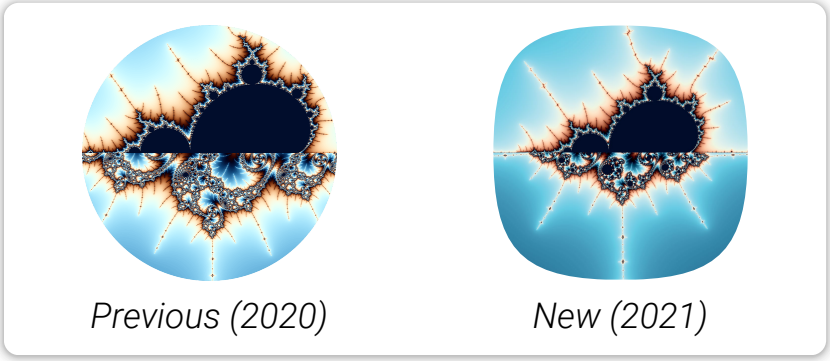


Figure 4.1: Last year's logo has been refreshed with a more modern look. The new icon is in the shape of a *squircle* [47] – a portmanteau of *square* and *circle*.

As part of improving the branding of the application, its logo has been updated to look more modern and professional, while also abiding by the standard for *maskable icons* defined in the Web Application Manifest [48, 2.3 - Icon masks and safe zone] set by the World Wide Web Consortium. Maskable icons can adapt to different icon styles set by a target device, be it a square, rounded square, teardrop, "squircle" (figure 4.1), circle, or other mask shape.

4.2 Functionality: New and Improved

4.2.1 Rotation

Both viewers have been updated with the ability to rotate around the complex plane. Rotation is obtained from the existing `react-use-gesture` controls, with rotation of the Mandelbrot set shader being one of my contributions, and rotation for the Julia set shader having been contributed by Fraser. Rotating a viewer can be achieved by rotating

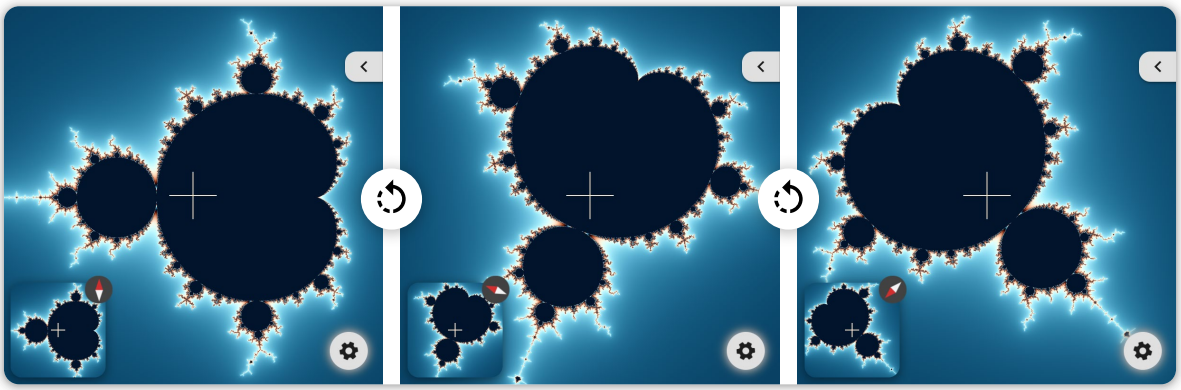


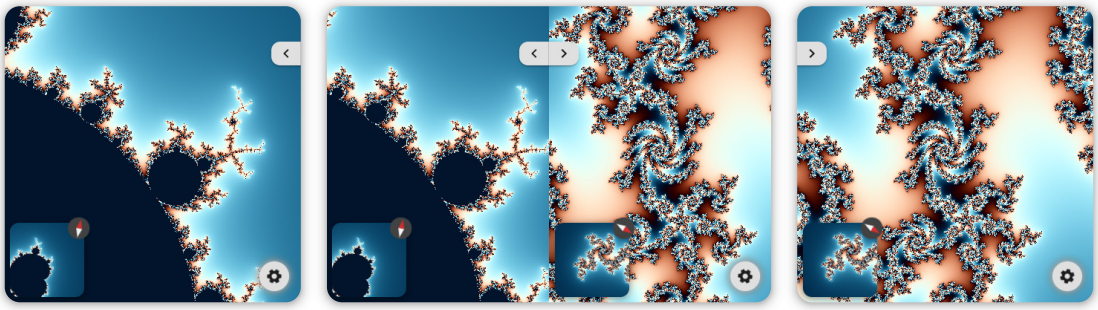
Figure 4.2: Viewers now support rotation. Minimaps update accordingly.

while performing a two-finger pinch gesture on a touch screen device, or by holding the Shift key while scrolling for larger devices.

Both fractal viewers have also been changed to allow for programmatically controlling their positions from within the code, which acts like an API for warping to specific points since it can be called from other components.

4.2.2 View modes

A new viewing mode switcher has been added, which allows for two layouts: the default split view, or a full-screen view which maximizes only one of the viewers. The switcher becomes hidden when a viewer is maximized, providing immediate user feedback by not allowing the user to perform the same action again as that would not be possible.



(a) Mandelbrot only (b) Split view (c) Julia only

Figure 4.3: New available view modes in Mandelbrot Maps.

4.2.3 URL parameters

Taking inspiration from Google Maps' URL parameter scheme, Mandelbrot Maps can now encode viewer locations within a URL. Google Maps encodes its parameters after the last forward slash, with an '@' symbol followed by the `<latitude>`, `<longitude>`, and `<zoom>` level (note the `z` suffix) delimited by commas:

`google.com/maps/@55.944781,-3.187282,17z` (`/@<lat>,<lng>,<zoom>z`)

This format benefits from being remarkably easy to read when compared to other encoding schemes – the example below encodes data using a JSON format which is highly verbose:

`mandelbrot.ophir.dev/#{"pos":{"x":-0.745,"y":0.125},"zoom":128000}`

Another consideration is that a URL encoding scheme for Mandelbrot Maps would need to support encoding two viewer locations: one for the location of the Mandelbrot set viewer and one for the location of the Julia set viewer. To accommodate this requirement, the Google Maps encoding scheme was adapted to explicitly denote whether an encoded location belonged to the Mandelbrot viewer (`m`) or the Julia viewer (`j`). With this, a URL for Mandelbrot Maps will encode each viewer's coordinates, zoom, and rotation: `<x>`, `<y>`, `<z>`, and `<t>`, respectively, by prefacing the `@` symbol with the viewer letter `<v>` (which is replaced with the letter `m` or `j`) in the following format:

`/<v>@<x>,<y>,<z>,<t>`

Encoding both viewers's positions with this scheme requires only concatenating the two resulting strings, producing a URL which can encode partial information either for a single viewer, or both viewers. In our case, these must be encoded as a "hash" route after the hash character `#`, to tell the browser that we are navigating to a location within the page and prevent it from navigating away. All the following URLs are valid:

- `jmaio.github.io/mandelbrot-maps/#/m@-0.5,0.6,8,0.2`
- `jmaio.github.io/mandelbrot-maps/#/j@0.4,-0.2,4.5,1.25`
- `jmaio.github.io/mandelbrot-maps/#/m@-0.6,-0.5,5,-0.1/j@0.4,0.3,5.1,0.85`

Figure 4.4: Google Chrome can generate QR codes for quickly sharing webpages. The code points to `jmaio.github.io/mandelbrot-maps/#/m@-0.6,-0.5,5,-0.1/j@0.4,0.3,5.1,0.85`

Thanks to the information embedded in these URLs, they can easily be shared, for example using Google Chrome's *share via QR code* feature (available since Chrome 85 [49]) as shown in figure 4.4. When navigating to a Mandelbrot Maps URL with encoded coordinates, the application smoothly animates the viewers to that location.

When either viewer changes position, the new coordinates are encoded again and the URL is updated, constantly changing to display the current location. Initially, because the URL is also used to drive the controls to the correct location, this resulted in an infinite loop where the URL would be updated, causing the application to warp to the location specified, which would trigger a URL update, again causing the application to

warp, and so on. This issue was fixed by changing the behaviour so that the URL is only updated when the view stops animating. By hooking into `react-spring`, it is possible to run a callback function when a spring stops animating as required. This callback now fires only once after the view has stopped animating, which prevents infinite update loops and is more efficient by not constantly refreshing the browser URL.

4.2.4 Deep zoom

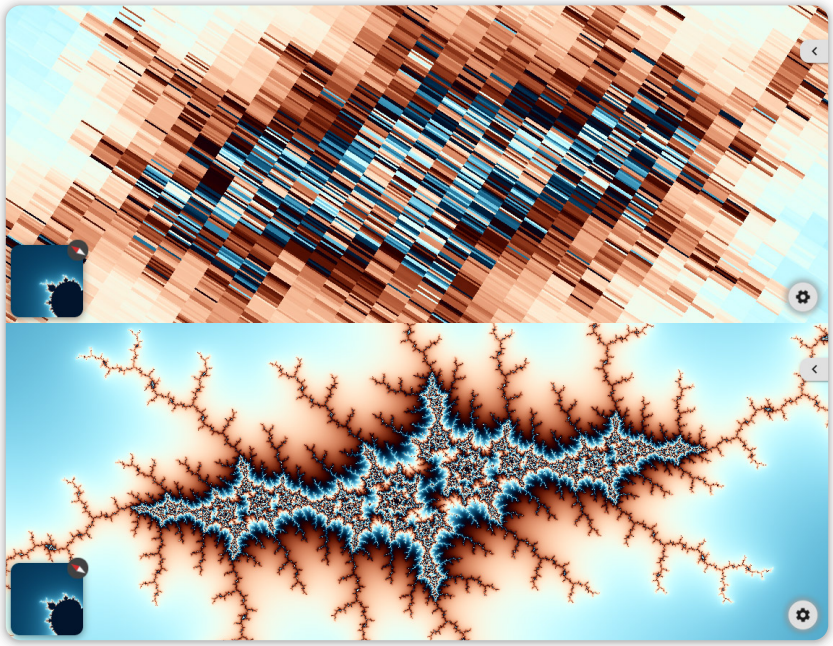


Figure 4.5: (Top) Deep zoom *disabled*. (Bottom) Deep zoom *enabled*. Deep zoom brings better magnification performance to most regions of the Mandelbrot set while maintaining the speed of WebGL.
`jmaio.github.io/mandelbrot-maps/#/m@-0.0399693,0.9861198,2005827.84,-1`

One of the standout features of a previously reviewed fractal explorer was the deep zoom in DeepFractal (now "DeepMandelbrot": `deep-mandelbrot.js.org`). Adding deep zoom posed several challenges, since the existing architecture needed to be tweaked before it could carry out the steps required for deep zoom to function correctly.

Integrating deep zoom required creating a new Mandelbrot renderer component with a different WebGL shader and toggling between the two when the feature is enabled. The deep zoom component then uses the CPU to perform a search for a reference point within the bounds of the Mandelbrot viewer. Finding a reference point involves iterating over multiple candidate points to obtain one which does not quickly diverge, such that it can be used in iterating other nearby points with the approximation in equation 2.9.

The results using deep zoom are extraordinarily detailed as shown in figure 4.5, but it can unfortunately suffer from random visual glitches if no good reference point is found as described above [50]. Since deep zoom can maintain detail up to immensely large factors of magnification, the precision of the `react-spring` controls needs to be able to change dynamically between a default level of precision (`1e-7`) and and an

enhanced level of precision (`1e-15`). Additionally, in situations where it does function correctly, deep zoom can reach such a high level of magnification that the precision of the controls themselves causes the view to feel like it snaps to certain points on a grid; it may be possible to mitigate this by specifying a higher precision for the controls, but testing this did not produce better results past this scale.

4.2.5 Controls

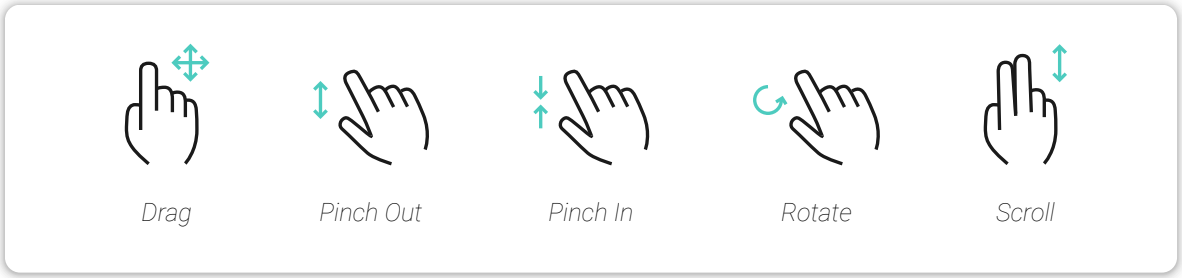


Figure 4.6: Multi-touch gestures illustrated.

The ability to rotate a viewer had previously not been implemented due to issues with rotation when using the two-finger touch gesture. The first attempt at adding rotation last year was not reliable because the method for calculating the desired rotation angle did not correctly consider full rotations, causing it to wrap around between 0 and 360 degrees with a sudden jump between the two. That issue has since been addressed by using the correct two-finger rotation calculation provided by `react-use-gesture`.

Another issue raised during the evaluation was the instability of the panning gesture after releasing from a mouse click, where the view would suddenly move even though there was no residual velocity in the gesture. This has been fixed by changing the precision of the spring as described in the documentation for `react-spring`.

When zooming, some users have also reported differences when using a touchpad to zoom. Some touchpads support two methods for zooming: scrolling with two fingers, which is equivalent to using a scroll wheel on a mouse; or using a two-finger pinch, which is equivalent to a pinch gesture on a phone. Unfortunately, there does not appear to be a consistent way to identify the difference in gesture, which can result in very slow zooming when using the two-finger gesture on a touchpad where it may be perfectly fine on a mobile device.

4.2.6 Colour picker

A simple colour picker widget (from `react-colorful`) has been added to the settings menu, allowing users to select the primary colour for displaying the fractals. The colours are applied to both the regular and deep zoom modes, and the primary colour is complemented by its inverse: blue with orange, green with purple, black with white. The colour picker can be seen in figure 4.8 below the iterations slider.

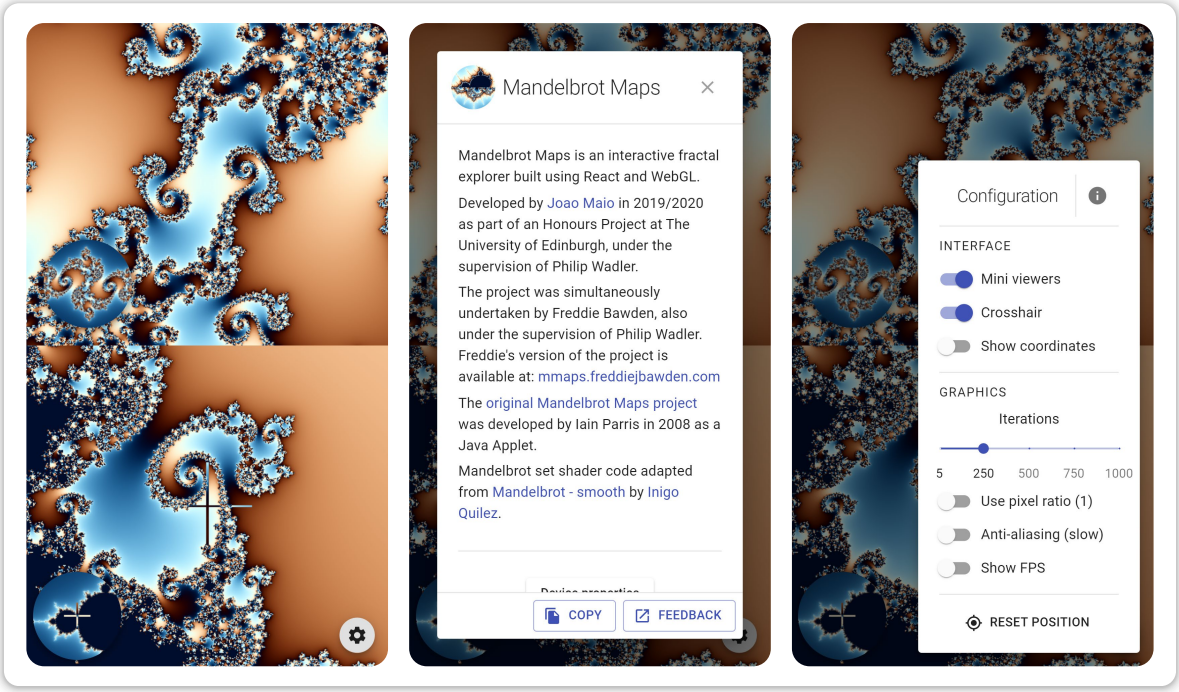


Figure 4.7: The User Interface of the final version of last year's application. (v1.0.1)

4.3 User Interface

4.3.1 Cosmetic changes

The UI features rounder corners for a more modern look. For clarity, "mini-viewer" has been renamed to "minimap", and minimaps have changed from a circular shape to a rounded rectangular shape, mimicking Google Maps' *Satellite/Map* view switcher. More colour has been added to provide better context, such as the information buttons and the reset button. This includes the settings menu, which has been renamed from *Configuration* to *Settings*, and its layout and spacing have been tweaked.

The settings icon on the bottom right of the main screen now has an animation which makes it slowly rotate around its centre. This slow rotation provides a visual cue that draws attention towards the settings button without being too distracting. Animating this icon has an additional benefit that the rotation will continue as long as the application is responding, signalling to a user that it is not frozen if the icon is still rotating, or conversely that it is not responding (and should be refreshed) if the icon stops rotating. The rotation may also become less smooth if the current rendering settings are too demanding for the device, which is an unintended side-effect, but one which provides good insight into how much computation the device is expending for rendering. A smoother rotation indicates a faster, more capable device, whereas a choppy rotation indicates a device which may be struggling to keep up with the application.

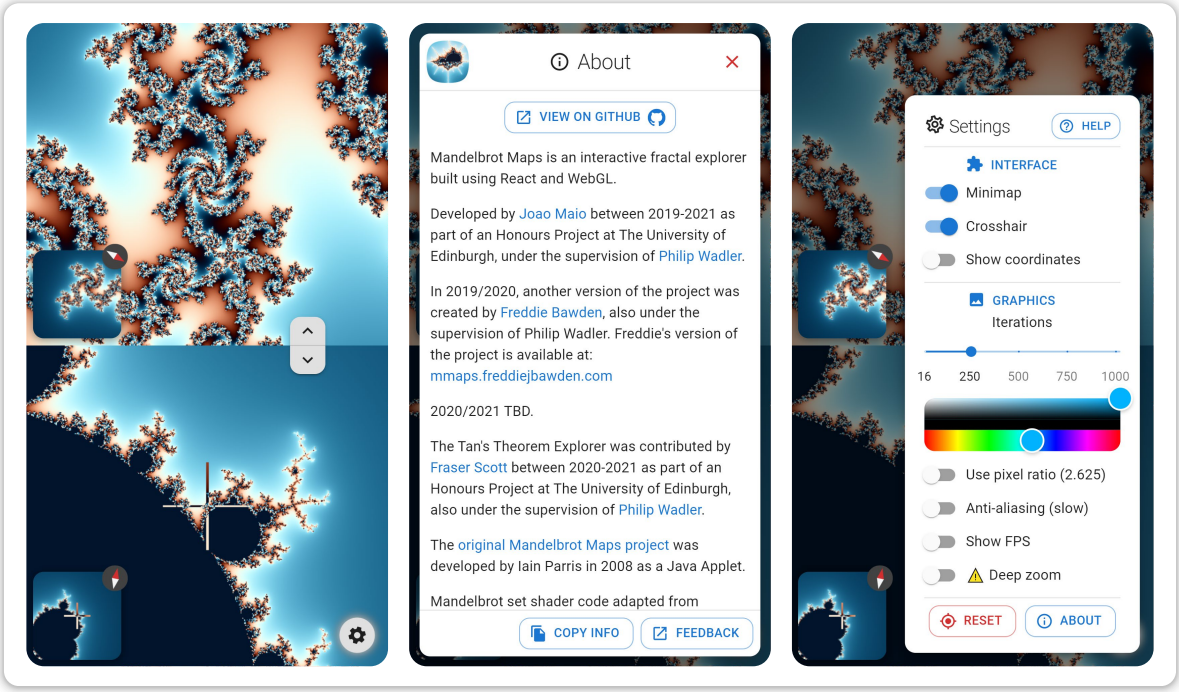


Figure 4.8: The application's User Interface has had multiple additions and visual changes to make it more appealing. (v1.2.0)

4.3.2 Usability

Minimaps

The minimap was intended to show a zoomed-out view of the region where a user was zoomed into on the fractal, similarly to Google Maps' *Satellite/Map* view switcher, or even a video game minimap [51]. Minimaps would previously become gradually more transparent or "fade out" as the viewer approached a zoom level of 1 or lower to remove clutter from the view, since the minimap would look almost the same as the regular view. This fading behaviour was not obvious or documented and caused confusion to some users, so it has been removed, giving only the option of always showing or hiding the minimaps.

Rotation indicator

Minimaps have also been updated with a small visual element on the top-right corner which displays the current rotation level of a viewer. This visual element is shaped like a compass: a large red and white needle inside a disc of contrasting colour, similarly to how popular maps applications convey the orientation of the map to the user. The compass element is clickable – doing so resets the rotation of the viewer and sets the red half of the compass needle pointing North, equivalent to a rotation of zero radians.

Mouse cursor context

Both viewers have been updated to display a more contextual mouse cursor. Previously, if a user hovered over or interacted with a viewer, their mouse cursor would not change,

remaining the default system mouse cursor. This has been improved so that the mouse cursor is now changed using the `cursor` CSS property [52], setting it to `grab` when hovering over a viewer or `grabbing` when clicking and dragging to pan around the view. When using a mouse, this small change provides a much clearer visual metaphor of what action is currently being performed.

Figure 4.9: The mouse cursor changes contextually based on what action a user is performing on a viewer. *Grab* hints that the viewer's content can be moved around, and *Grabbing* shows that the content is actively being moved by the user.

Explicit updates

An issue found towards the end of last year's project was that updating the application was difficult. The application relies on its Progressive Web App service worker [53] to store the application offline, after which it periodically checks if a new update is available online. If an update is available, the worker downloads and applies it over the existing version.

This update process would complete in the background, but the existing version would still be displayed until the page was reloaded to display the new version, making it appear to the user as though nothing had happened.

To make updates more explicit, the application now hooks into the service worker's lifecycle to determine when an update is available and has been applied. The service worker emits an event if it has found and applied a new update, which the application then actions by displaying a persistent pop-up snackbar to the user, informing them that an update is available and prompting them with a button that can be clicked to reload the page and apply the update.

4.4 Help Menu

A new pop-up dialog has been added containing helpful information about the layout of the Mandelbrot and Julia sets, the available ways of controlling viewers, and information about each of the settings available in the settings menu. Ideally, first-time users would be prompted to view this menu on their first visit, such as with a snackbar, but this feature is not currently available. To reach this menu, a user must first enter the *Settings* menu, then click the *Help* button on the top right corner of the popover to reveal the pop-up dialog consisting of three pages (figure 4.10).

The first page of the help menu contains basic information about the application, such as how to re-enter the help menu, what the application does, and a short line about fractals.

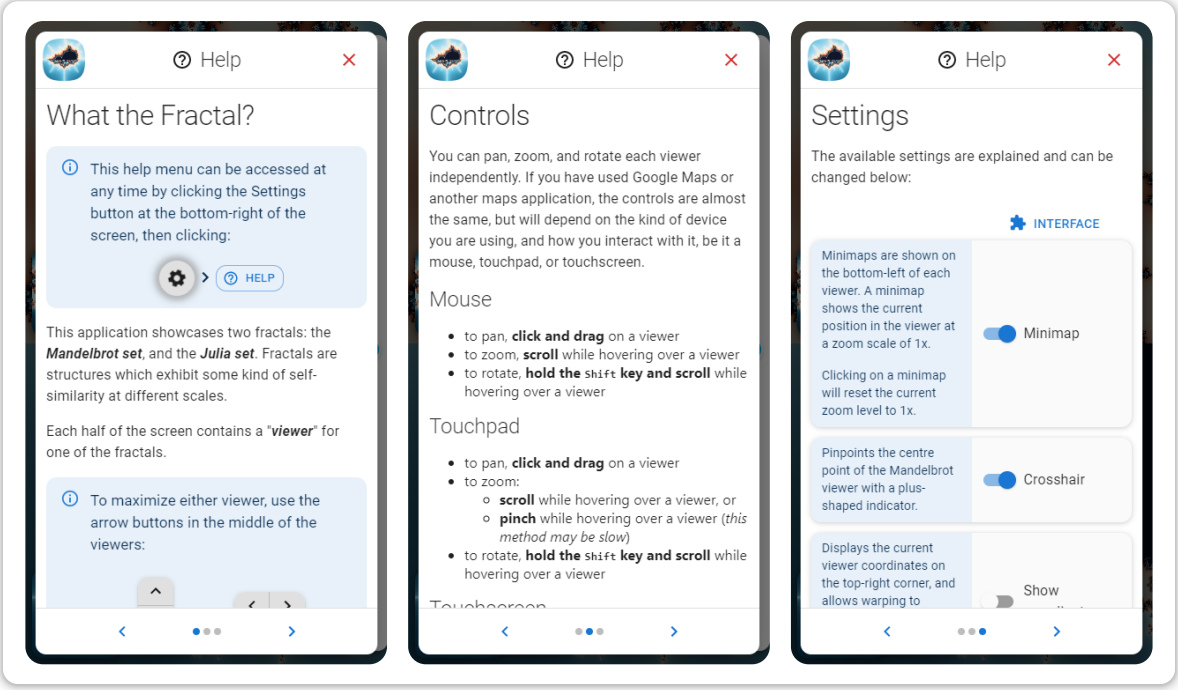


Figure 4.10: The top of the three help menu pages.

Further down, the viewer layouts are explained using figure 4.11. On the second page, the control schemes for different devices are explained.

The third page contains the same settings widgets as the *Settings* menu. Settings can be toggled from here instead of having to alternate back and forth. Reusing the actual control component was considered a clearer, simpler design decision, as opposed to displaying the control using, say, a static image. This also means that new controls are automatically added to the help menu without any additional input required.

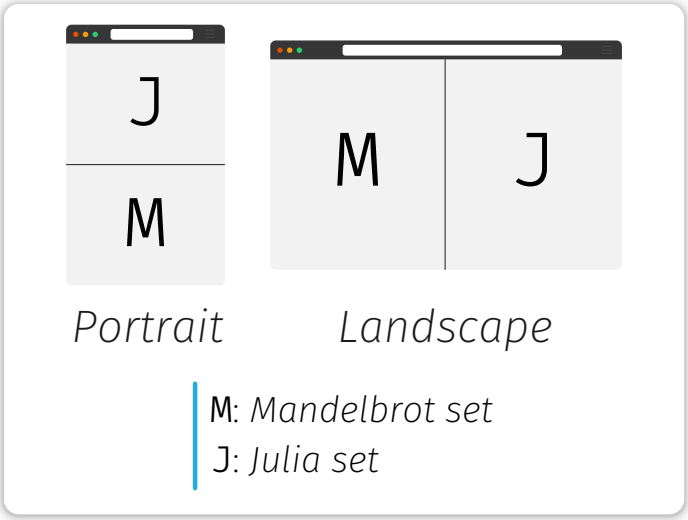


Figure 4.11: The diagram explaining the layout of the viewers in the first help page.

Chapter 5

Evaluation

5.1 Search Engine Optimization

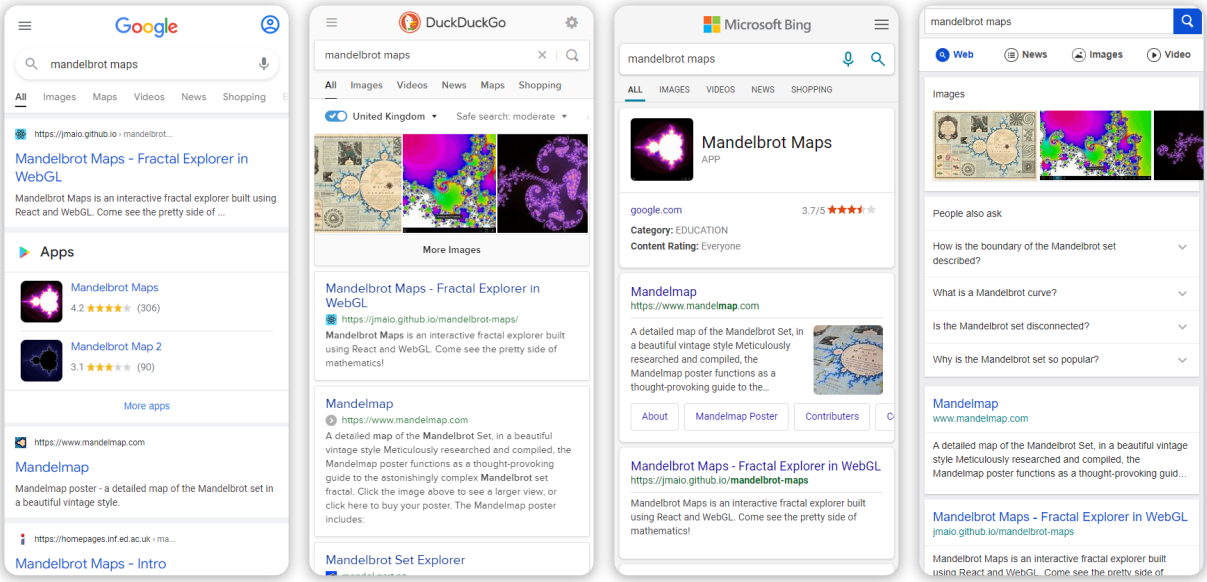


Figure 5.1: Search results for popular search engines with the query "*mandelbrot maps*" on a simulated mobile device (Pixel 2 XL). (Left to right: Google, DuckDuckGo, Bing, Yahoo)

Depending on certain factors, search results can appear in different positions on search engines. Geographical location, for example can affect how a search engine ranks results based on local trends [16]. Searches were therefore carried out using the School of Informatics' VPN service, and location was set to *United Kingdom* where possible, to simulate a query performed in Edinburgh. The search query "Mandelbrot Maps" ranks our application at number 1 on both Google and DuckDuckGo (discounting image search suggestions). Bing and Yahoo rank it in second place in terms of webpages, with both suggesting "Mandelmap" as the most relevant result. Bing also suggests the existing Android application "Mandelbrot Maps" developed by Alasdair Corbett (see

appendix A). According to Google search console, the most popular search term leading to the application being seen by users is “mandelbrot map”, which resulted in 225 views out of 719 total views ($\sim$ 30%) over the past 16 months on Google search.

5.2 Performance

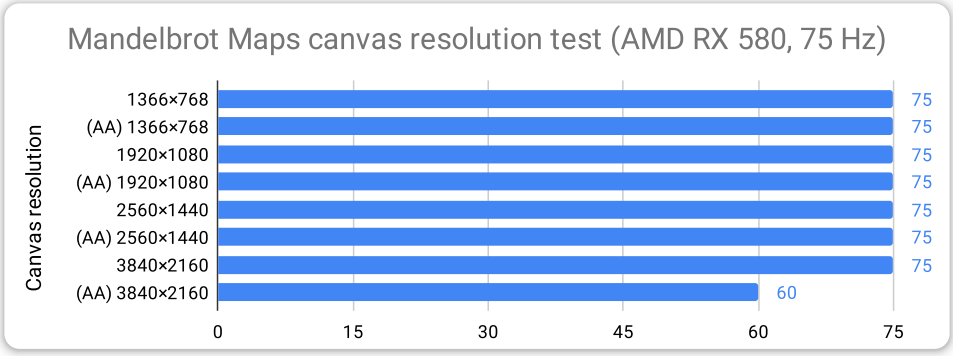


Figure 5.2: Desktop benchmarking results for Mandelbrot Maps using popular screen resolutions from StatCounter (`gs.statcounter.com`).

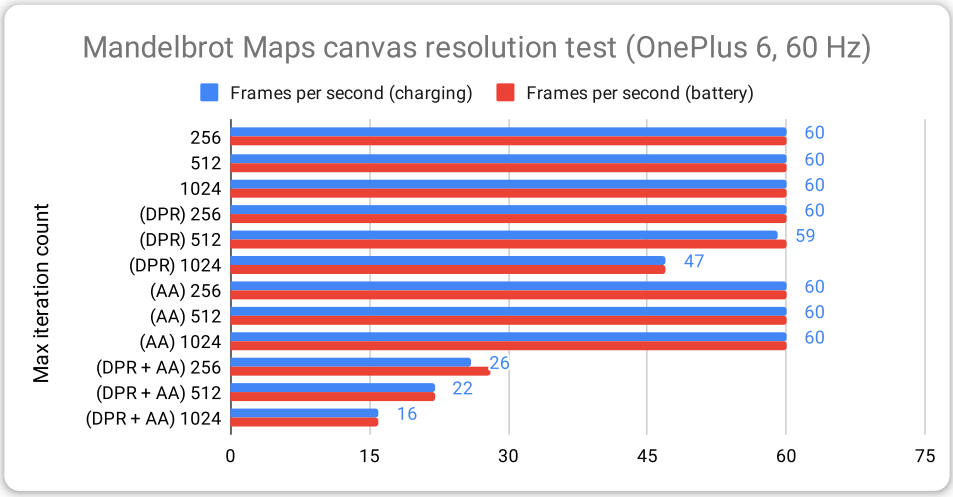


Figure 5.3: Mobile phone benchmarking results for Mandelbrot Maps using a OnePlus 6 mobile device. Tests carried out both while charging and on battery for completeness. (DPR = Device Pixel Ratio [enabled], AA = Anti-Aliasing [enabled])

Raw graphics performance of the application was tested to verify that its performance has remained fast despite the addition of new functionality. Statistics obtained with `stats.js` (`github.com/mrdoob/stats.js`).

Benchmarking on a mobile phone was done both while charging and on battery to ensure that this variable was removed, since overall performance could be lowered by power-saving features or by heat generated by the charging process throttling the device. As tested, the device performed almost identically in both scenarios (within a small margin of error), indicating that performance was unaffected by the charging status.

The application once again performs smoothly in benchmarks, capping out at the screen's native refresh rate in most tests, which indicates that there is still a lot of performance available beyond that, and that this year's additions have not noticeably lowered its performance. Deep zoom has not been tested extensively as it provides unpredictable performance, tends to be subject to glitches (figure 5.4).

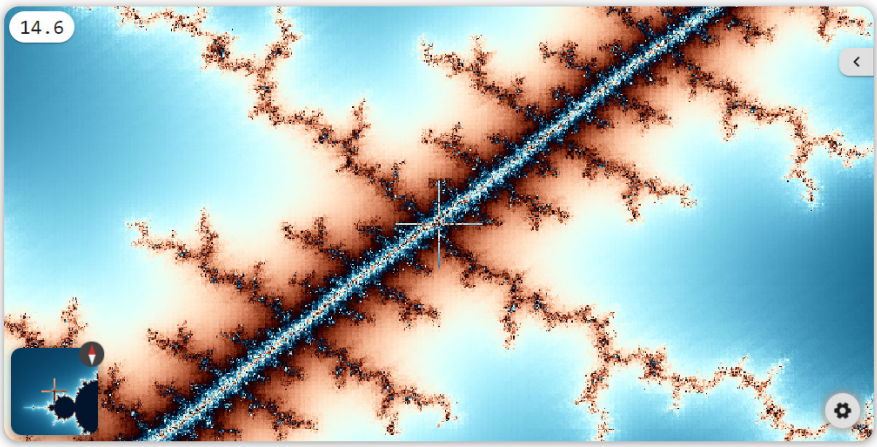


Figure 5.4: Deep zoom can suffer from glitches which lead to reduced performance: jmaio.github.io/mandelbrot-maps/#/m@-1.2532036,0.3850113,25253719.3,0/j@0.4364131,-0.6468786,4,2.12

5.3 Participant Survey

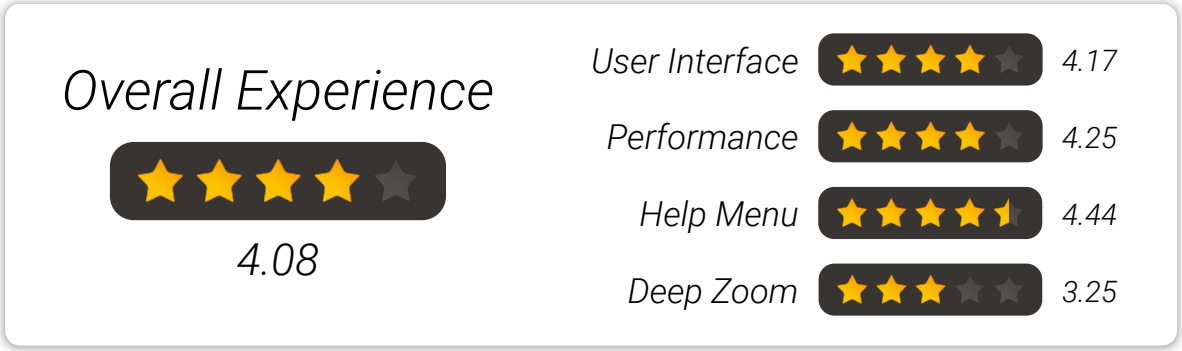


Figure 5.5: Survey statistics collected using a star scale ranging from one to five.

A participant survey was circulated where anyone could anonymously submit feedback, and 12 responses were submitted. The key statistics are highlighted in figure 5.5.

Overall, the results from only twelve participants carry a larger margin of error compared to last year's 31 participants, but remain comparable, if marginally lower (table 5.1). Reception was once again positive, with diverse feedback from multiple areas of the application. Responses indicated that 5 respondents encountered issues using the application, mostly with the controls when zooming on the viewers or due to overlapping UI elements. Despite this, 8 respondents indicated that they would be interested in using the application again, 1 would not be interested, and 3 were not sure.

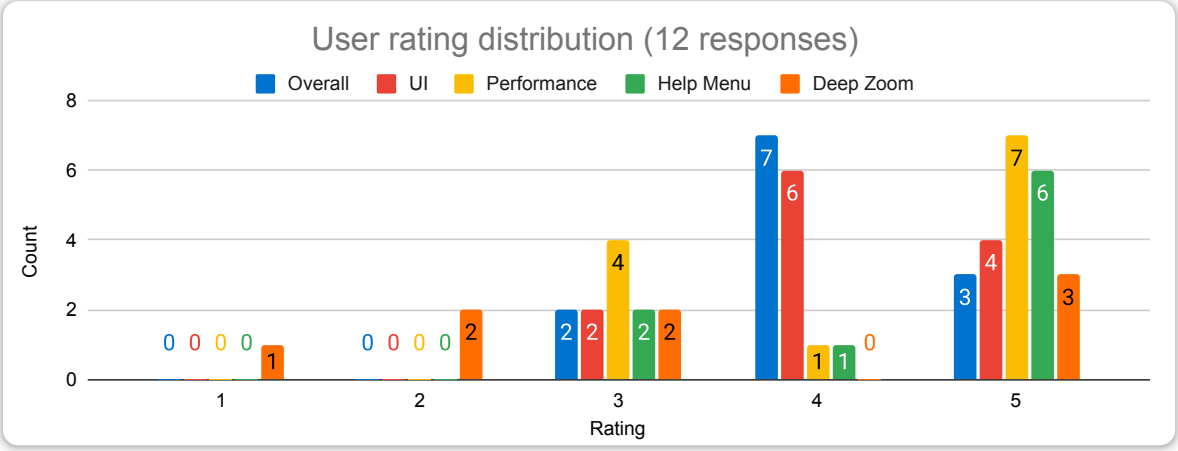


Figure 5.6: Distribution of user ratings per category. The first three questions were required; the last two were not and therefore do not add up to 12.

Measure	Last year	σ	This year	σ
Overall	4.13	0.72	4.08	0.67
User Interface	4.23	0.84	4.17	0.72
Performance	4.45	0.72	4.25	0.97
Help Menu	–	–	4.44	0.88
Deep Zoom	–	–	3.25	1.58

Table 5.1: Comparison of last year's user survey results with this year's survey.

Feedback for individual components was positive overall. Choosing between the ratings of "Don't know / Did not use", "Not good", "Okay", and "Great!", 75% of respondents rated the Mandelbrot set, Julia set, and Minimaps as "Great!". Most feedback fell between "Okay" and "Great!", although some respondents noted that they did not know or did not use certain functionality, most notably the *Coordinates card*, *Iteration slider*, *Colour changer*, and *Frames per second (FPS) meter*.

Regarding controls, feedback was overwhelmingly positive. All ratings for Touchscreen input were "Great!" Nearly all other ratings given were "Great!", with only one user rating Mouse input as "Not good", and one other rating Touchpad as "Okay". As outlined in section 4.2.5, the improvements made appear to have fixed most remaining issues, greatly improving the user experience as a result.

Survey responses also showed that users were happy with the level of performance, even if at times they may have encountered some issues with zooming around for a while, especially if Deep Zoom was enabled. Unfortunately, it is not always possible to test on extensive numbers of devices ahead of release, and as discussed previously, unit testing WebGL applications cannot give accurate insight into what potential issues different users will experience on their own devices, which makes testing the application a highly manual, time-consuming process that will inevitably not find all existing bugs.

Insightful feedback snippets from the survey respondents are included in the subsections below.

5.3.1 Positive feedback

- information in the “Help” menu was super helpful, although I don’t understand what half the settings do as they were technical terms
- things are explained well and I like the interface and look of the help menu - suits the app well!

5.3.2 Suggestions for improvement

- perhaps hovering over the different features could display a short help card (for a quick reminder/explanation) - and then a pointer to the help section for more help?
- didn’t really use the help menu, the descriptions of each setting would be more useful if accessible directly next to that setting
- the features I used so far are great, you should think about performance
- maybe change the colour of the crosshair, sometimes it blends in with the Mandelbrot set

5.3.3 Issues

- got stuck when zooming in, reloading worked
- I couldn’t zoom infinitely
- scroll wheel was not that sensitive: I had to resort to use pinch to zoom to zoom faster
- pinching to zoom often rotated everything almost 180°, it would be nice to have an option to disable rotation
- the map freezes after I zoom in and out it for a while (about 1 min) on iPad
- after exploring and zooming for a few minutes, the graphics freeze (but the settings menu still responds)
- when I started the “Tans Theorem Explorer” I couldn’t expand the Julia set because the arrow was overlapped by the explorer popup, then I didn’t find a way to close the explorer because the cross is at the very top left and was covered by the FPS display; maybe some rearrangement of the components would be useful
- it was pixelated, only soon after I found that I could change the DPI; you might want to configure the DPI behind the scenes to keep things sharp by default: if framerate [is low], fall back to a smaller resolution; this is what many games do behind the scenes when they render the 3D scene (and keep the UI high resolution to fool you)

5.3.4 Deep Zoom

- really loved deep zoom: information was super helpful, good to know performance can be affected as it definitely was but didn’t surprise me since I knew
- it was slow, but yes, that was mentioned in the help menu already
- I didn’t know there was this functionality; maybe when you reach a zoom limit, you could make a pop up that suggests to activate Deep Zoom or keep it on by default
- (on mobile) unusually glitchy with the Mandelbrot fractal shapes moving around as I zoom

5.4 Workflow and Collaboration

The workflow presented in chapter 3 has undoubtedly made the development process easier, but this measure can be subjective and difficult to quantify. A good way to demonstrate the efficiency of this workflow is to look at the new functionality added, especially from external contributions. Despite not having worked on the project before, Fraser contributed functionality which was merged into the main branch of the repository from 22 March 2021 (Tan's theorem explorer).

> *It was really easy! The code is laid out into understandable folders, and it's easy to find e.g. the settings menu, shaders or UI, plus there are setup instructions in the README.*
>
> *TypeScript was great, especially because there are custom types. For instance,* `XYType` *made it clear I should be passing in coordinates.*
>
> *I LOVE Prettier... the code looks consistent and you spend less time tabbing lines out.*
>
> `husky` *gave me reassurance because I was working on a public repo, although initially part of it couldn't handle my username having a space in it. :'(*
>
> — Fraser Scott on extending Mandelbrot Maps

5.5 The future of WebGL

Upgrading to WebGL 2 was considered, and has been implemented and tested to work, but this feature has not been integrated into the main branch because it is not supported by the mobile Safari browser, which represents 24.85% of the global mobile browser market share as of March 2021 [54]. Updating to alienate about 25% of the potential user base in our case is not a justifiable business decision. While WebGL 2 brings significant improvements for 3D content on the web, in the context of Mandelbrot Maps, it does not contribute any appreciable benefits since we only make use of a basic subset of the existing WebGL features.

Although this Mandelbrot Maps project makes direct use of WebGL, there are other projects such as Babylon.js [55] and Three.js [56] which provide wrappers around the base WebGL functionality to make it easier to create 2D and 3D content on the Internet. As these engines look to provide more functionality, they have inevitably run into the limitations of the underlying WebGL technology (which is seeing development winding down) [57]. This is also true of other projects looking to make use of advanced graphics on the web, which has led to the joint development of WebGPU: a new web standard for "*modern 3D graphics and computation capabilities*" [58]. WebGPU is still a work-in-progress, but a future version of Mandelbrot Maps could see more advanced functionality by transitioning to WebGPU if the specification becomes widely adopted.

Chapter 6

Conclusion

This report has showcased updates made to Mandelbrot Maps which have made it easier to develop for, maintain, and extend with new functionality. The conversion to TypeScript and the workflow proposed serve as a demonstration of how an existing JavaScript application could be adapted to follow a streamlined workflow and maximize developer productivity by leveraging process automation.

6.1 Workflow

As it stands, this project could serve as a "demo project" for new and existing JavaScript and TypeScript projects to reference how to:

- use React and TypeScript (together or separately)
- use Yarn 2 (with Plug'n'Play) for project and dependency management
- set up linting and style rules with ESLint and Prettier
- create git hooks using `husky`
- enforce commit message format with `commitlint`
- release new versions with `standard-version`
- create CI/CD pipelines for testing and deployment, using GitHub Actions
- detect dependency vulnerabilities using Dependabot

6.2 Application

The application has proved to be a solid base to build upon, as shown by the new functionality, including Fraser's contribution of the *Tan's Theorem Explorer*. After extensively searching for and evaluating existing fractal viewers on the Internet over the past two years, we believe that this new version of Mandelbrot Maps combines the functionality from some of the best, most unique fractal viewers into a single application, making it one of the most complete fractal viewers currently available online. Its key features are:

- Supports Mandelbrot set and Julia set

- High performance using GPU acceleration (WebGL)
- Native support desktop and mobile devices and gestures (React)
- Deep Zoom (Perturbation theory)
- Movement: Pan, Zoom, Rotate
- Modern, simple User Interface
- Fractal options: iteration control, colour selection
- Progressive Web App: works like a native app, including offline
- Embedded help menu

6.3 Future work

The user survey showed that most users feel positively towards the application, but that some control issues and rendering bugs are still lingering. As suggested by one of the respondents, having an educational component that is "99% visual representation and 1% text" would be a good addition. The application could be made more accessible to younger audiences by including a didactic component that assumes little to no prior knowledge, taking steps to teach the basics and aiming to provide a better understanding of the mathematics behind the fractal shapes. Adding support for more exploration possibilities, which could include other fractals such as the Burning Ship [59], or even allowing arbitrary functions may make the application more attractive for education purposes.

With WebGL potentially becoming less relevant as the WebGPU specification gets closer to release, it may be useful to migrate to a graphics engine such as Babylon.js or Three.js, as they would be be updated internally to bring support for newer features and specifications.

Although Deep Zoom is currently only available on the Mandelbrot set viewer, adding this to the Julia set will likely require just a few minor tweaks to the existing code. The application's Deep Zoom functionality may also benefit from experimenting with other techniques for fast arbitrary precision calculations. Fractalforums (previously `fractalforums.com`, now `fractalforums.org`) is a high-quality resource where some amazing fractal-specific improvements have been proposed.

Regarding automation, the current configuration works well as is. In future, new functionality may require manually triggering certain tasks, such as packaging the application before release. A tool that could help with this is gulp [60], which lets developers automate parts of their workflow by defining arbitrary tasks that can be declared as JavaScript code.

Bibliography

[1] João Filipe Maio. *Mandelbrot Maps: WebGL Application for Exploring Fractals*. 2020.

[2] Lei Tan. "Similarity between the Mandelbrot set and Julia sets". In: *Comm. Math. Phys.* 134.3 (1990), pp. 587–617. URL: https://projecteuclid.org:443/euclid.cmp/1104201823.

[3] *React - A JavaScript library for building user interfaces*. URL: http://reactjs.org/.

[4] *WebGL - OpenGL ES for the Web*. 2011. URL: http://khronos.org/webgl/.

[5] *Adobe Flash Player EOL General Information Page*. URL: https://www.adobe.com/products/flashplayer/end-of-life.html.

[6] Microsoft. *TypeScript: Typed JavaScript at Any Scale*. URL: https://www.typescriptlang.org/.

[7] Microsoft. *IntelliSense in Visual Studio Code*. URL: https://code.visualstudio.com/docs/editor/intellisense.

[8] OpenJS Foundation. *ESLint - Pluggable JavaScript linter*. URL: https://eslint.org/.

[9] *Prettier – Opinionated Code Formatter*. URL: https://prettier.io/.

[10] @typicode. *typicode/husky: Git hooks made easy - woof!* URL: https://github.com/typicode/husky.

[11] @azz. *azz/pretty-quick: Get Pretty Quick*. URL: https://github.com/azz/pretty-quick.

[12] @okonet. *okonet/lint-staged: Run linters on git staged files*. URL: https://github.com/okonet/lint-staged.

[13] @conventional-changelog. *conventional-changelog/standard-version: Automate versioning and CHANGELOG generation, with semver.org and conventionalcommits.org*. URL: https://github.com/conventional-changelog/standard-version.

[14] @conventional-changelog. *conventional-changelog/commitlint: Lint commit messages*. URL: https://github.com/conventional-changelog/commitlint.

[15] John Gruber and Aaron Swartz. *Markdown – text-to-HTML conversion tool for web writers*. URL: https://daringfireball.net/projects/markdown/.

[16] Wikipedia contributors. *Google Personalized Search — Wikipedia, The Free Encyclopedia*. [Online; accessed 7-April-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Google_Personalized_Search&oldid=1003846204.

[17] Poorvu Center for Teaching and Learning. *Citing Internet Sources | Poorvu Center for Teaching and Learning*. URL: https://poorvucenter.yale.edu/writing/using-sources/citing-internet-sources.

[18] Samvid Mistry and Prakash Patel. "A Guide to Material Design, a Modern Software Design Language". In: *Open Source For You* (Apr. 2016), pp. 64–66.

[19] Gavin Bierman, Martín Abadi, and Mads Torgersen. "Understanding TypeScript". In: *ECOOP 2014 – Object-Oriented Programming*. Ed. by Richard Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 257–281. ISBN: 978-3-662-44202-9.

[20] Wikipedia contributors. *Patterns in nature > Trees, fractals — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Patterns_in_nature&oldid=1011041441. [Online; accessed 24-March-2021]. 2021.

[21] Eric W. Weisstein. "Mandelbrot Set". In: *From MathWorld–A Wolfram Web Resource.* (2020). [Online; accessed 11-April-2020]. URL: https://mathworld.wolfram.com/MandelbrotSet.html.

[22] Mike Hurley (mgh3@po.cwru.edu). *Proof that 2.0 is sufficient*. URL: http://mrob.com/pub/muency/escaperadius.html.

[23] Eric W. Weisstein. "Julia Set". In: *From MathWorld–A Wolfram Web Resource.* (2020). [Online; accessed 11-April-2020]. URL: https://mathworld.wolfram.com/JuliaSet.html.

[24] Kevin I. Martin. *SuperFractalThing – Arbitrary precision Mandelbrot set rendering in Java*. Paper: science.eclipse.co.uk/sft_maths.pdf. URL: http://www.science.eclipse.co.uk/SuperFractalThing.html.

[25] D.H. Bailey, R. Barrio, and J.M. Borwein. "High-precision computation: Mathematical physics and dynamics". In: *Applied Mathematics and Computation* 218.20 (2012), pp. 10106–10121. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2012.03.087. URL: https://www.sciencedirect.com/science/article/pii/S0096300312003505.

[26] Islam ElShaarawy and Walid Gomaa. "Ideal Quantification of Chaos at Finite Resolution". In: *Computational Science and Its Applications – ICCSA 2014*. Ed. by Beniamino Murgante et al. Cham: Springer International Publishing, July 2014, pp. 162–175. ISBN: 978-3-319-09144-0. DOI: 10.1007/978-3-319-09144-0_12.

[27] *Mathematics Stack Exchange: Perturbation of Mandelbrot set fractal (NightElfik)*. URL: https://math.stackexchange.com/a/1071945/757765.

[28] Atlassian. *Atlassian Agile Coach – What is Agile?* URL: https://www.atlassian.com/agile.

[29] Atlassian. *Kanban - A brief introduction*. URL: https://www.atlassian.com/agile/kanban.

[30] Jeff Sutherland and Ken Schwaber. *The 2020 Scrum Guide*. URL: https://scrumguides.org/.

[31] GitHub. *Features | GitHub – GitHub*. URL: https://github.com/features.

[32] Mozilla. *MDN Web Docs Glossary: Dynamic typing*. URL: https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing.

[33] @microsoft. *microsoft/TypeScript: TypeScript is a superset of JavaScript that compiles to clean JavaScript output.* URL: https://github.com/microsoft/TypeScript.

[34] Poimandres. *react-spring – bring your components to life with simple spring animation primitives*. URL: https://www.react-spring.io/.

[35] David Catuhe. *Why we decided to move from plain JavaScript to TypeScript for Babylon.js*. URL: https://www.eternalcoding.com/why-we-decided-to-move-from-plain-javascript-to-typescript-for-babylon-js/.

[36] Amber Rockwood. *Automated Cross-Browser Testing for WebGL – It's Not Going to Happen*. 2018. URL: https://www.eventbrite.com/engineering/automated-cross-browser-testing-webgl-not-going-happen/.

[37] Red Hat. *What is CI/CD?* URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd.

[38] @google. *google/styleguide: Style guides for Google-originated open-source projects*. URL: https://github.com/google/styleguide.

[39] OpenJS Foundation. *Conventional Commits – A specification for adding human and machine readable meaning to commit messages*. URL: https://www.conventionalcommits.org/.

[40] @angular. *Angular – Contributing to Angular > Commit Message Format*. URL: https://github.com/angular/angular/blob/master/CONTRIBUTING.md#-commit-message-format.

[41] GitHub. *GitHub Desktop | Simple collaboration from your desktop*. URL: https://desktop.github.com/.

[42] Scott Chacon and Ben Straub. *Pro Git*. Expert's voice in software development. Apress, 2009. ISBN: 9781430218340.

[43] @yarnpkg. *Yarn - Package Manager*. URL: https://yarnpkg.com/.

[44] Yarn. *Why should you upgrade to Yarn Modern?* URL: https://yarnpkg.com/getting-started/qa.

[45] Dependabot. *Automated dependency updates*. URL: https://dependabot.com/.

[46] Tom Preston-Werner. *Semantic Versioning 2.0.0*. URL: https://semver.org/.

[47] Eric W. Weisstein. "Squircle". In: *From MathWorld–A Wolfram Web Resource.* (2021). [Online; accessed 9-April-2021]. URL: https://mathworld.wolfram.com/Squircle.html.

[48] World Wide Web Consortium. *Web Application Manifest*. URL: https://w3c.github.io/manifest/.

[49] Abner Li. "Chrome 85 for Android adds share menu, desktop QR codes". In: *9to5Google* (Aug. 2020). URL: https://9to5google.com/2020/08/25/chrome-85-share/.

[50] Fractal Wiki Contributors. *Perturbation theory – Fractal Wiki*. URL: https://fractalwiki.org/index.php?title=Perturbation_theory&oldid=29.

[51] Wikipedia contributors. *Mini-map — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-April-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Mini-map&oldid=1002395177.

[52] *cursor - CSS: Cascading Style Sheets | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/cursor.

[53] *Service Worker API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.

[54] *Can I use WebGL 2.0?* URL: https://caniuse.com/webgl2.

[55] *Babylon.js: Powerful, Beautiful, Simple, Open - Web-Based 3D At Its Best*. URL: https://www.babylonjs.com/.

[56] *Three.js JavaScript 3D Library*. URL: https://threejs.org/.

[57] *WebGL Happenings – The Khronos Group Inc*. URL: https://www.khronos.org/blog/webgl-happenings.

[58] *W3C – GPU for the Web Community Group*. URL: https://www.w3.org/community/gpu/.

[59] Fractal Wiki Contributors. *Burning Ship – Fractal Wiki*. URL: https://fractalwiki.org/wiki/Burning_Ship.

[60] @gulpjs. *gulp – A toolkit to automate & enhance your workflow*. URL: https://github.com/gulpjs/gulp.

Appendix A

Project Lineage

This section is dedicated to those who have previously worked on versions Mandelbrot Maps. GitHub @usernames are included where available.

- 2008 – Iain Parris creates the first version of Mandelbrot Maps, a Java applet which can display the Mandelbrot set and the Julia set side-by-side.
- 2009 – Edward Mallia improves upon Parris' applet by adding multithreading, among others; notably, the introduction includes an email exchange between Edward and the late Professor Benoît Mandelbrot.
- 2010 – Taige Liu builds upon Mallia's version of the applet.
- 2012 – Alasdair Corbett (@withad) creates a version of Mandelbrot Maps for Android, available on the Google Play Store at the time of writing: `play.google.com/store/apps/details?id=uk.ac.ed.inf.mandelbrotmaps`
- 2015 – Skye Welch (@CarrotCodes) updates Corbett's version of Mandelbrot Maps for Android with faster rendering and other improvements
- 2019 – Joseph Hunter further updates the Android version to display Tan Lei's theorem (*Similarity between the Mandelbrot set and Julia sets*)
- 2020 – Freddie Bawden (@freddiejbawden) creates a web version of Mandelbrot Maps using React and WebAssembly + Rust (`http://mmaps.freddiejbawden.com/`); Joao Maio (@JMaio) creates a web version of Mandelbrot Maps using React and WebGL.
- 2021 – Joao Maio updates and ports the project to TypeScript, proposing a new development and CI/CD workflow, and adding perturbation theory deep zoom. Fraser Scott (@fraserdscott) builds on the existing WebGL version of Mandelbrot Maps to add a demonstration of Tan's theorem, displaying the similarity between the Mandelbrot set and Julia sets, which is integrated into the main application. Georgina Medd (@GeorginaMedd) researches and implements an educational component to Mandelbrot Maps.