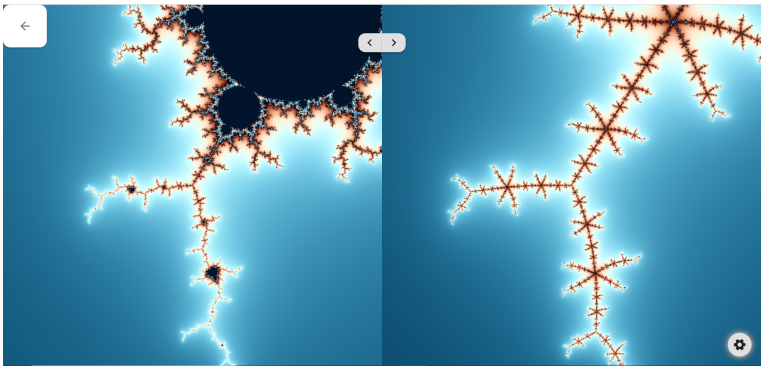


Extending Mandelbrot Maps to Visualise Tan's Theorem

Fraser Scott

MInf Project (Part 1) Report

Master of Informatics
School of Informatics
University of Edinburgh

April 12, 2021

Abstract

Mandelbrot Maps is a long-running project in the School of Informatics at the University of Edinburgh. It was originally a Java applet for visualisation and manipulation of fractals, specifically the Mandelbrot and Julia sets. In 2019, it was built from the ground up as a modern web app, providing an solid foundation for future projects.

This report showcases a new feature of Mandelbrot Maps which allows users to explore a known property of the Mandelbrot and Julia sets. To the naked eye, the sets might appear similar. In fact, as proven by Lei Tan, they are virtually indistinguishable around particular points. This extension takes users through the process of selecting a point on each set, then magnifying and rotating each set to demonstrate the similarity.

Mandelbrot Maps is available at the following URL:

`jmaio.github.io/mandelbrot-maps`

Source code is available on GitHub:

`github.com/JMaio/mandelbrot-maps`

Acknowledgements

I would like to thank

Philip Wadler, my project supervisor, for his guidance and infectious enthusiasm.

Joao Miao for his boundless knowledge and willingness to review my PR's.

Lei Tan for proving the wonderful theorem upon which this work relies.

Table of Contents

Chapter 1

Introduction

1.1 The Mandelbrot and Julia sets

"A fractal is a shape made of parts similar to the whole in some way."

- Benoit Mandelbrot (Feder 1998)

Much like fractals themselves, an exact definition of a fractal is difficult to pin down. Nonetheless, since the advent of the personal computer, these intricate objects have been renowned for their aesthetic appeal by both mathematicians and non-mathematicians. Among all fractals, the Mandelbrot set is often considered to be the most popular (Mandelbaum 2018).

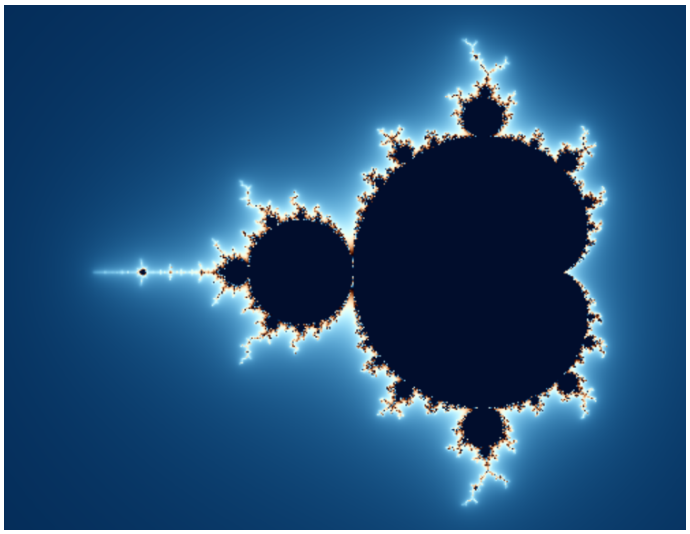


Figure 1.1: The Mandelbrot set

The Mandelbrot set is the set of complex numbers c for which the function $f_c(z) = z^2 + c$ does not diverge when iterated from $z = 0$. In other words, the sequence $f_c(0)$, $f_c(f_c(0))$, etc., remains bounded in absolute value. The behaviour of points under iteration is highly unpredictable, and when plotted in the plane, the Mandelbrot set exhibits an elaborate and infinitely complicated boundary.

Each point in the Mandelbrot set defines another fractal known as a Julia set. Here, we fix the complex number c. Then, the Julia set for c is the set of complex numbers z for which the function $f_c(z) = z^2 + c$ does not diverge when iterated from z. In other words, the sequence $f_c(z)$, $f_c(f_c(z))$, etc., remains bounded in absolute value.

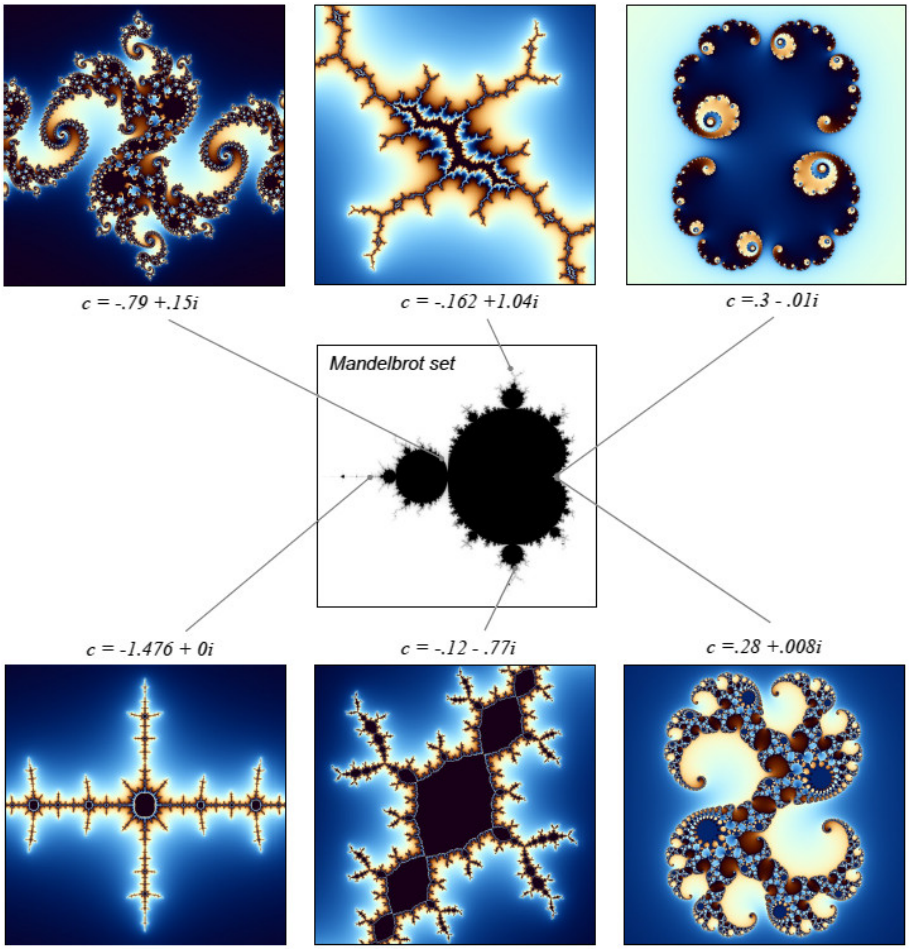


Figure 1.2: The corresponding Julia sets for various points in the Mandelbrot set (https://www.karlsims.com/julia.html)

1.2 The Previous Application

Mandelbrot Maps began life as a Java applet in 2008, with the aim of allowing "visualisation of the Mandelbrot set, and associated Julia sets" (Parris 2008). The app also offered a novel user interface for its time, which was designed to be appealing and intuitive. The project has gone through many iterations since, with the release of a successful Android app which targets touchscreen phones and tablets (Corbett 2014).

The most recent incarnation was developed in 2019 by Joao Miao (Figure 1.3). This web application aims to leverage modern technologies to replicate and surpass the functionality of the original app. It was well received, with an average rating of 4.1 out of 5 based on 31 reviews (Miao 2020).

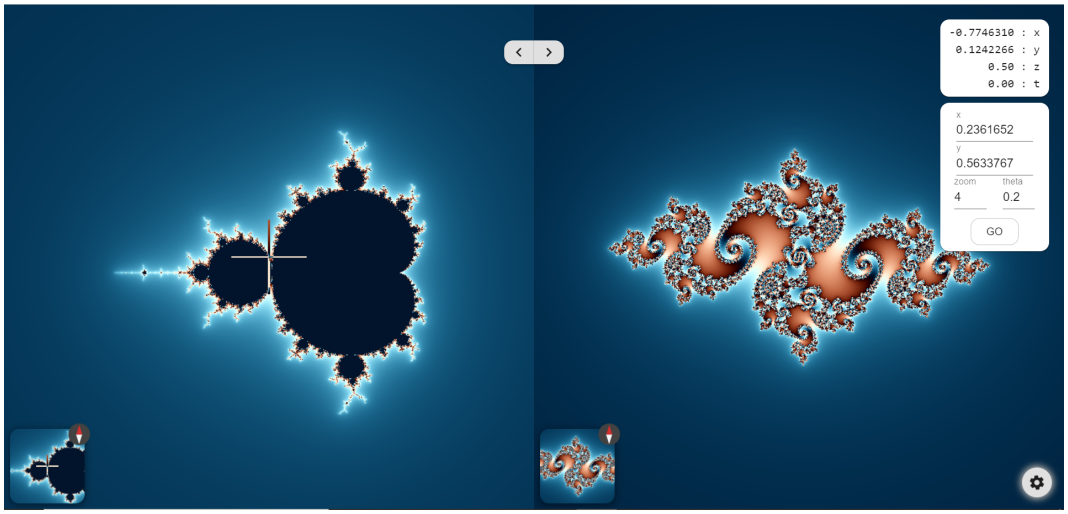


Figure 1.3: Mandelbrot Maps

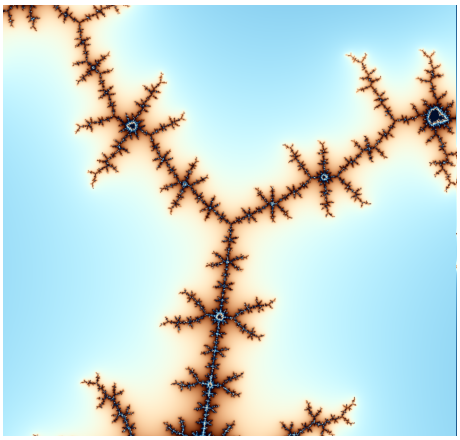
Since the official deprecation of Java applets in 2016, the newest version of *Mandelbrot Maps* is built with HTML5, JavaScript and WebGL. As a web app, it can provide a consistent experience across a large portion of devices regardless of platform, physical size, screen resolution, input devices, or other variables.

The user experience has also received an update. The app displays both the Mandelbrot and Julia set side by side, with them both updating in real time. Gone are the days of manually selecting a magnification - users can pinch to zoom on touchscreen, or scroll the mouse wheel on desktop. Dragging to pan feels fluid, and movements have a natural 'momentum' that gradually decays once a user stops interacting.

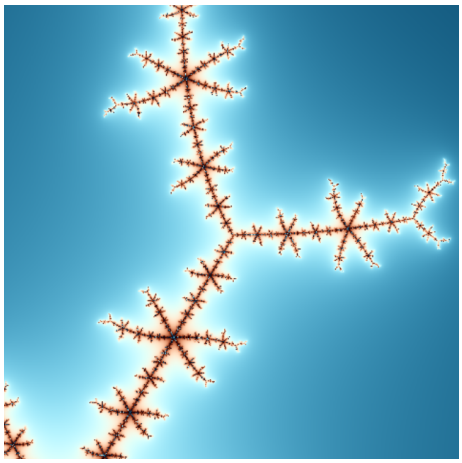
1.3 Tan's Theorem

Upon close inspection, one might notice that regions in the Mandelbrot set often bear a striking resemblance to the same region in the corresponding Julia set. This intuition is correct. In a seminal paper for the field (Tan 1990), Lei Tan proved that, at particular points, one can magnify and rotate the two sets such that they are almost indistinguishable.

This similarity occurs at points where ever Mandelbrot set splits into multiple branches, like the point $c = -0.101 + 0.956i$. Figure 1.4 shows the region around c in both the Mandelbrot set and the corresponding Julia set. Note the three converging branches in both images; if we were to magnify and rotate the sets, they would begin to look very much the same.



(a) The Mandelbrot set, centered at c



(b) The Julia set for c, centered at c

Figure 1.4: The Mandelbrot set and Julia set for $c = -0.101 + 0.956i$.

It is simple enough to animate this phenomenon with a computer. One could take a program like *Mandelbrot Maps*, find a branch point by eye, then manually magnify and rotate the views until the similarity can be seen. To achieve even better results, Tan's paper provides explicit magnification and rotation factors for each point.

Despite this, a thorough web search did not find a program that allows users to select an arbitrary branch point and visualise the similarity in real time. Along with the aesthetic appeal, this could be used as an intuitive 'visual proof' that would benefit mathematicians wishing to examine and understand the Mandelbrot set.

This project details an extension to *Mandelbrot Maps* that allows users to select a point on each set, then magnifies and rotates each set to best show the similarity. The following images demonstrate the results of the project, by applying to the feature to the aforementioned point $c = -0.101 + 0.956i$.

Unless stated otherwise, all the images in this paper were generated using Mandelbrot Maps and this new feature.

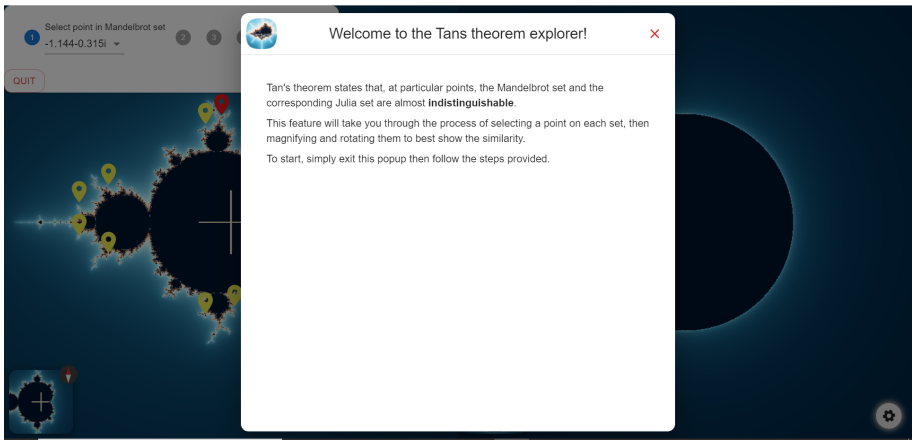


Figure 1.5: Upon opening the Tan's Theorem Explorer

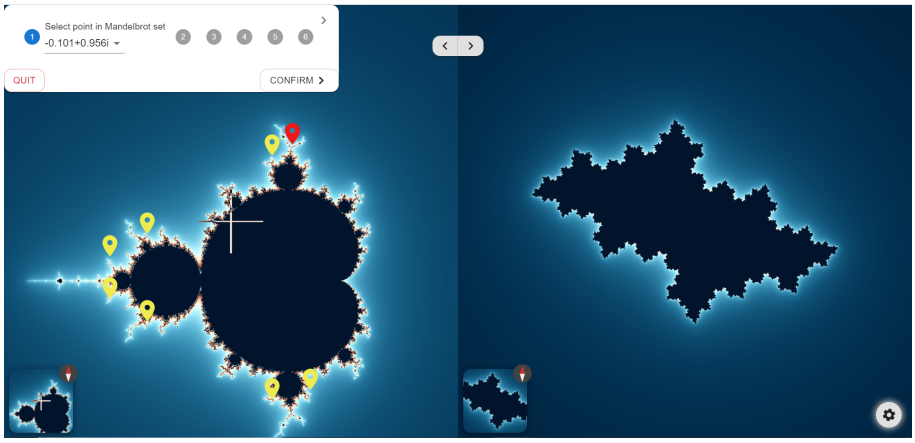


Figure 1.6: Selecting the point c on the Mandelbrot set

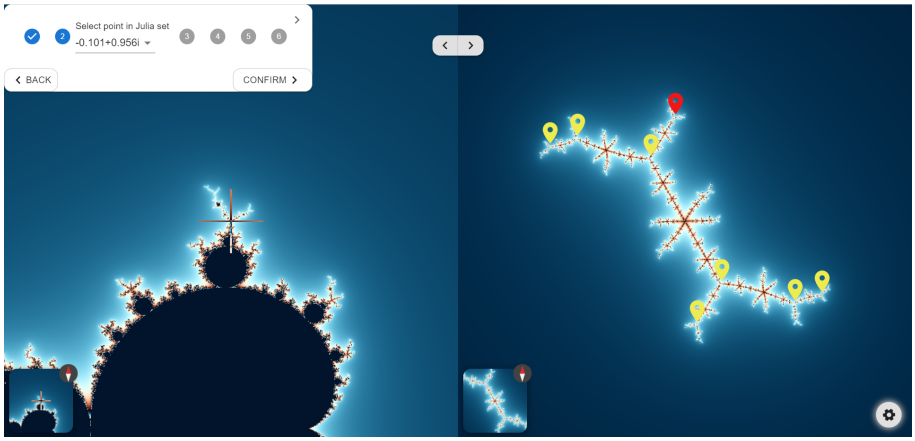


Figure 1.7: Selecting the point c on the Julia set for c

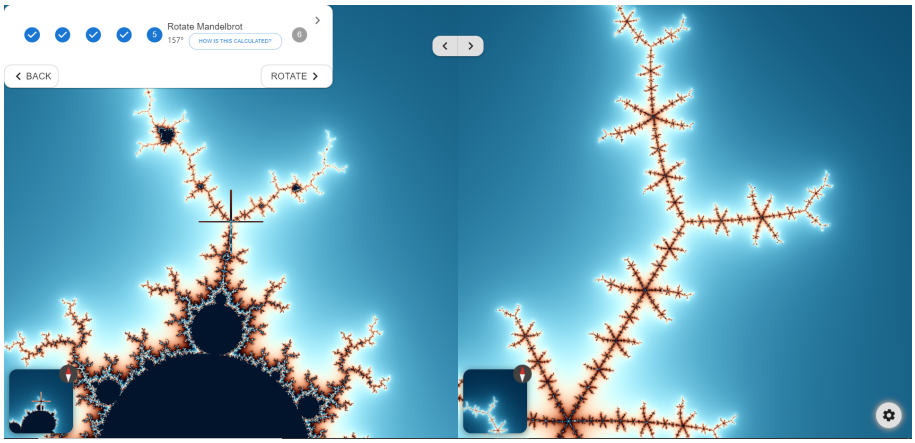


Figure 1.8: After magnifying both sets

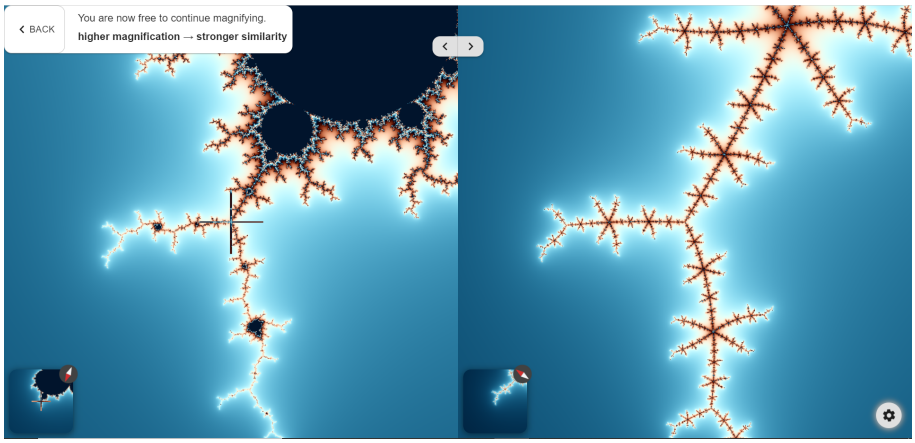


Figure 1.9: After magnifying and rotating both sets

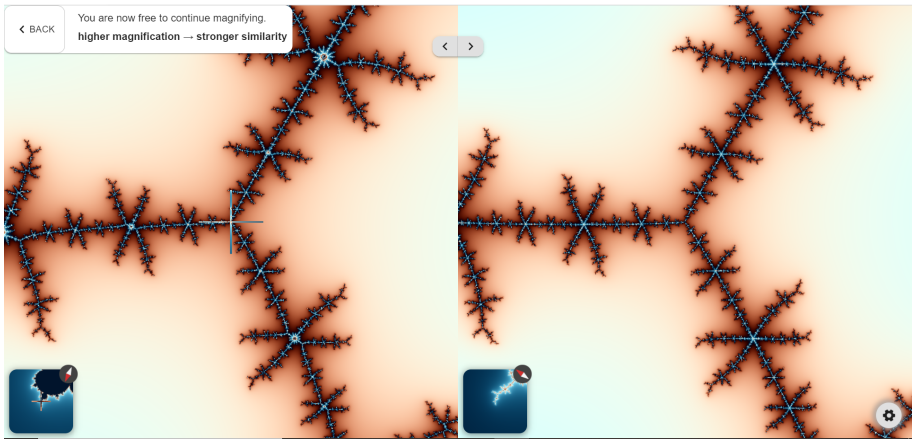


Figure 1.10: After magnifying and rotating both sets + additional manual magnification

1.4 Report Overview

This report is meant as a self-contained introduction and description of the project, outlining the design decisions and rationale behind them. The following chapters are:

Chapter 2 provides an in-depth introduction to the mathematical concepts and definitions required for the application.

Chapter 3 explains the implementation decisions and justifications of the application, with both a high-level overview and in-depth detail.

Chapter 4 evaluates the application by comparing the resulting images and analysing the findings from the user survey.

Chapter 4 concludes with an overview, and outlines the work to be done in future incarnations of Mandelbrot Maps.

1.5 Project Coordination

This year, myself and two other students, Georgina Medd and Joao Miao, were assigned to this project. Joao was responsible for building Mandelbrot Maps the previous year, so it was decided that he would work on porting the application to Typescript and adding new quality-of-life features. Meanwhile, Georgina and I would research different methods of visualising Tan's theorem, with the hope of comparing our work at the end.

Ultimately, we realised that there was a single, efficient solution to the problem that negated the need for more complex ideas. Therefore, I continued to focus on Tan's theorem, while Georgina moved to building an educational component to the app. Both my discussions with Georgina and the new functionality added by Joao throughout the year were fruitful and contributed to the success of the project.

Chapter 2

Mathematical Background

2.1 The complex plane

In school, I was often told something along the lines of:

> *"no number, when multiplied against itself, will result in a negative number"*

Thankfully, using the theory of complex numbers, it is in fact possible to study something like $\sqrt{-1}$. The key insight is to split complex numbers into a real component and an imaginary component:

$$z = a + bi \text{ where } a, b \in R \text{ (real numbers) and } i = \sqrt{-1} \tag{2.1}$$

With this, we can visualize a complex number as a point in 2D space, where the real part is mapped to the horizontal axis, and the imaginary part is mapped to the vertical axis. This extends the traditional number line into the complex plane.

This allows us to visualise two quantities associated with a complex number. For a point z, we have the **magnitude** $|z|$, which is the length of the vector that runs from the origin to z, and the **argument** $arg(z)$, which is the angle that same vector makes with the x-axis.

To add two complex numbers, simply add the corresponding real and imaginary parts. Similarly, to multiply two complex numbers, multiply their magnitudes and add their arguments.

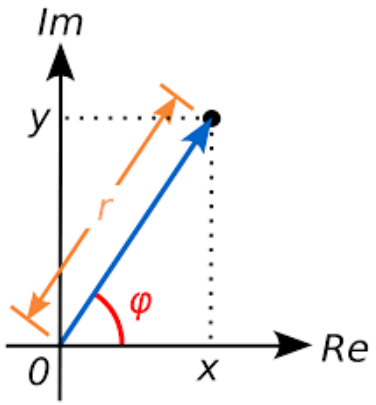


Figure 2.1: A plot of a complex number (© Wolfkeeper at English Wikipedia, CC BY-SA 3.0)

2.2 Forward iteration

A branch of mathematics, called complex dynamics, looks at the behavior of complex numbers under iteration with some function. So, not just applying a function once, but applying it repeatedly, using the output from one iteration as the input to the next. This study has led to two of the most famous fractals - the Mandelbrot set and Julia sets.

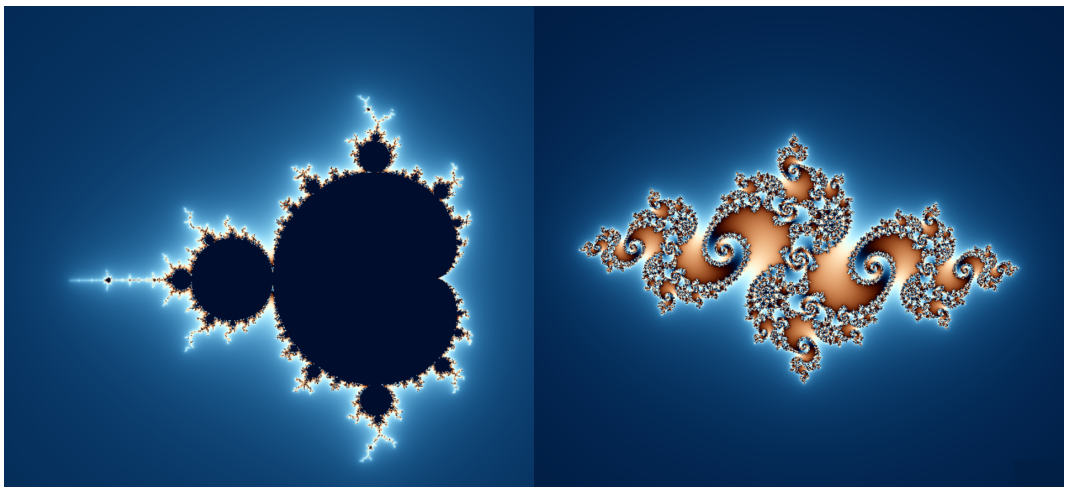


Figure 2.2: The Mandelbrot set and a Julia set.

In the study of the Mandelbrot and Julia sets, we are concerned with the following function f_c, where:

$$f_c(z) = z^2 + c \tag{2.2}$$

Here, z and c are complex numbers. To **iterate**, we start with some initial value of z, then repeatedly apply f_c to the output. For ease of notation, we use f_c^n to refer to n repeated applications of f_c. For example:

$$\begin{aligned} f_c^1(z) &= z^2 + c \\ f_c^2(z) &= (z^2 + c)^2 + c \\ f_c^3(z) &= ((z^2 + c)^2 + c)^2 + c \end{aligned}$$

The set of points visited in this process is then the **forward orbit** of z:

$$\bigcup_{n=1}^{\infty} f_c^n(z) \tag{2.3}$$

The members of the forward orbit of z are referred to as the **descendants** of z.

Geometrically, a single application of f_c can be visualised as doubling the argument of z, squaring its distance from the origin, then translating the result by c.

2.3 Backward iteration

Instead of iterating forward with f_c, we might wish to iterate backwards to find out where our current point 'came from'. We can calculate these points, known as **preimages**, by taking f_c and solving for z:

$$f_c(z) = z^2 + c$$
$$f_c(z) - c = z^2$$
$$z = \pm\sqrt{f_c(z) - c}$$

This gives us the **inverse** f_c^{-1} of f_c:

$$f_c^{-1}(z) = \pm\sqrt{z - c} \tag{2.4}$$

The plus-minus sign tells us that most points have two distinct preimages, with the only exception being $f_c^{-1}(c) = \sqrt{c-c} = \pm 0$.

Similarly to forward iteration, we use f_c^{-n} to refer to n repeated applications of f_c^{-1}. With this, we define the **backwards orbit** of z to be the set of points obtained by recursively taking the preimages of z:

$$\bigcup_{n=1}^{\infty} f_c^{-n}(z) \tag{2.5}$$

This process will keep returning new points, so backwards orbit are infinite in size (Milnor 1990). The members of the backwards orbit of z are referred to as the **ancestors** of z. As most points have two preimages, applying the inverse n times generates up to 2^n distinct ancestors. This can be visualised as a branching tree of points, as shown in Figure 2.3, where the tips of the arrows represent forward iteration.

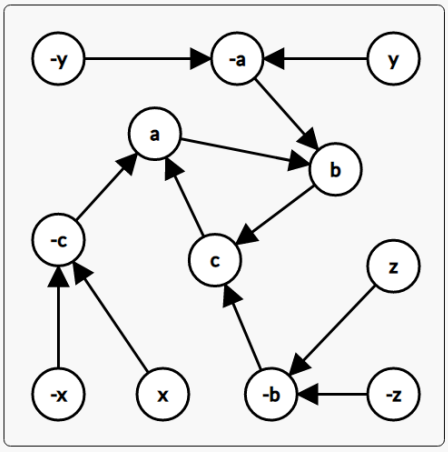


Figure 2.3: A cycle with members a, b, c and their ancestors (Graph drawn with `csacademy.com/app/graph_editor/`)

2.4 Julia sets

This paper is concerned with a visualising a particular connection between the Mandelbrot set and Julia sets. Although the Mandelbrot set is more well known, we hope to build to the connection by first studying Julia sets. Besides, it's only tradition - Benoit Mandelbrot himself started out using a computer to draw Julia sets (Feder 1998).

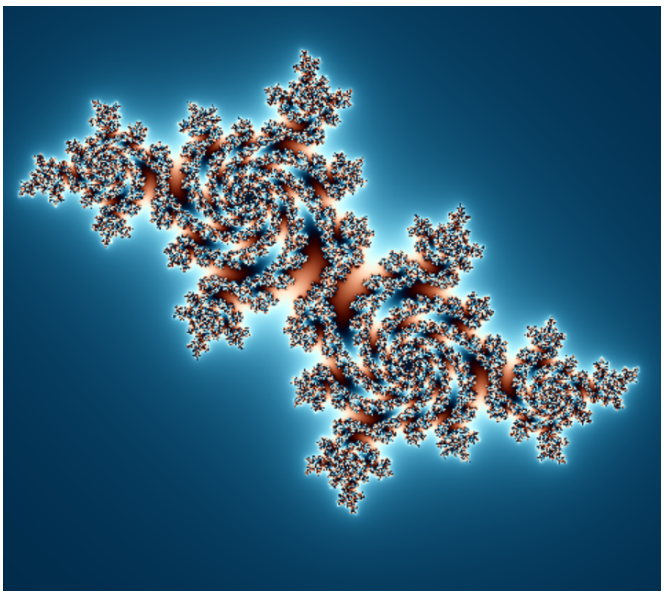


Figure 2.4: The Julia set for $c = -0.34 + 0.62i$

Julia sets are generated using the function from the previous chapters, $f_c(z) = z^2 + c$. Each value of c corresponds to a different Julia set, referred to as J_c.

To decide whether a number z is in a given Julia set, we fix the value of c, then iterate under f_c, starting with z as the initial value. If the resulting sequence remains bounded in magnitude, then $z \in J_c$. More formally, a **Julia set** is the following:

$$J_c = \{z \in C \mid f_c^n(z)_{n \in N} \text{ is bounded}\} \tag{2.6}$$

Here, C is the set of complex numbers. At first, it would seem that we need to compute the entire orbit to know if the magnitude remains bounded. However, there is a simple test: if the magnitude of a descendant ever exceeds 2, then the orbit diverges and will escape to infinity (Wikipedia 2021*a*). So, equivalently:

$$J_c = \{z \in C \mid \ |f_c^n(z)_{n \in N}| \leq 2\} \tag{2.7}$$

Images of Julia sets, like Figure 2.4, are often drawn using **escape-time algorithms**. Here, black points converge or cycle, while points outside the set are colored by counting iterations to divergence.

2.5 Periodic and preperiodic points

It is entirely possible that, while iterating under f_c, we might revisit a point and form a **cycle**. For instance, we could iterate 2 times, then enter a cycle of length 3, after which the orbit repeatedly visits the same 3 points.

More formally, a point z is strictly **periodic** if there exists a $p \geq 1$ such that:

$$z = f_c^p(z) \tag{2.8}$$

Similarly, a point z is strictly **preperiodic** if there exists some $q \geq 1, p \geq 1$, such that:

$$f_c^q(z) = f_c^{q+p}(z) \tag{2.9}$$

2.6 Differentiation

It turns out that we can learn a lot about orbits by differentiating the function f_c. Before this however, let us first provide a general definition of a derivative.

For a function f, we define its **derivative** f' at the point a as the following:

$$f'(x) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a} \tag{2.10}$$

Practically, we often cannot take this limit, so we can approximate the derivative by using a value of x that is sufficiently close enough to a.

We may also wish to differentiate the function f_c^n. The chain rule allows us to take the derivative of composite functions. For two functions, f and g, their composition F is the following:

$$F(x) = f(g(x)) \tag{2.11}$$

If both f and g are differentiable, then the derivative of their composition is given by the chain rule:

$$F'(x) = f'(g(x))g'(x) \tag{2.12}$$

In the case of repeated application of a function f, where $F(x) = f^n(x)$, we take the derivative recursively:

$$F'(x) = f'(f^{n-1}(x))(f^{n-1})'(x) \tag{2.13}$$

applying the chain rule until n=1.

2.7 Cycle eigenvalues

By differentiating the function f_c with respect to z, we can learn a lot about the points around z. For cyclic orbits, this is used to define a property known as the eigenvalue of the cycle.

To motivate this, let us look at a cycle of length 1; a **fixed point** α such that $\alpha = f_c(\alpha)$. Then, let the eigenvalue λ be the derivative of f_c at α:

$$\lambda = (f_c)'(\alpha) \tag{2.14}$$

Following from the definition of a derivative, we have:

$$\lambda = \lim_{z \to \alpha} \frac{f_c(z) - f_c(\alpha)}{z - \alpha} = \lim_{z \to \alpha} \frac{f_c(z) - \alpha}{z - \alpha} \tag{2.15}$$

As α is a fixed point, the numerator is simply the difference between $f_c(z)$ and α, and the denominator is the difference between z and α. In turn, if the magnitude of this ratio is less than 1, then $f_c(z)$ is closer to α than the initial z. We might say points near α are "attracted" towards α under iteration.

More generally, for a cycle of length p, the **eigenvalue** λ is defined as the following:

$$\lambda = (f_c^p)'(z) \tag{2.16}$$

Here, z is any point in the cycle. The eigenvalue is the same regardless of our choice of z, because we iterate p times and visit each point in the cycle.

Furthermore, we say that cycles are:

- attracting if $|\lambda| < 1$
- neutral if $|\lambda| = 1$
- repelling if $|\lambda| > 1$

As the eigenvalue is independent of the chosen point, we say that an individual periodic point is attracting (repelling etc.). Similarly, because its orbit contains a cycle, a preperiodic point is eventually attracting (repelling etc.).

2.8 Asymptotic self-similarity

The goal of this paper is to visualise a particular similarity between the Mandelbrot and Julia sets. In the subsequent chapters, we will build to this by first defining self-similarity, then similarity. These definitions are taken from Lei Tan's paper on similarity between the Mandelbrot and Julia set (Tan 1990).

To analyse the behaviour of a set around a particular point a, we use the translation function $\tau_{-a} : z \to z - a$. This function takes a set of points and translates them such that a is at the origin.

To measure the difference between two sets A and B, we use the Hausdorff distance. Informally, this is the maximum distance from a point in set A to the nearest point in set B. For a formal definition and motivation behind its use, I highly recommend the article *"Hausdorff distance between convex polygons"* (Grégoire & Bouillot 1998).

A set of complex numbers A is **asymptotically self-similar** with scale Q about a point $x \in A$ if there exists a set B such that:

$$Q^n(\tau_{-x}A) \to B \text{ while } n \to \infty \tag{2.17}$$

for the Hausdorff distance. The set B is known as the **limit model** of A at x.

2.9 Asymptotic self-similarity in Julia sets

Amazingly, Julia sets are self-similar. Take any repulsive periodic or preperiodic point z for f_c. J_c is asymptotically self-similar about z, with a scale equal to the eigenvalue λ of z (Tan 1990).

This has wonderful geometric consequences. Multiplying J_c by λ^n is equivalent to magnifying J_c by $|\lambda|$ and rotating J_c by $arg(\lambda)$ n times. If we increase n in integer increments, the same structure (the limit model) appears over and over! Figure 2.5 shows successive transformations on J_i, along with the magnification and rotation factors.

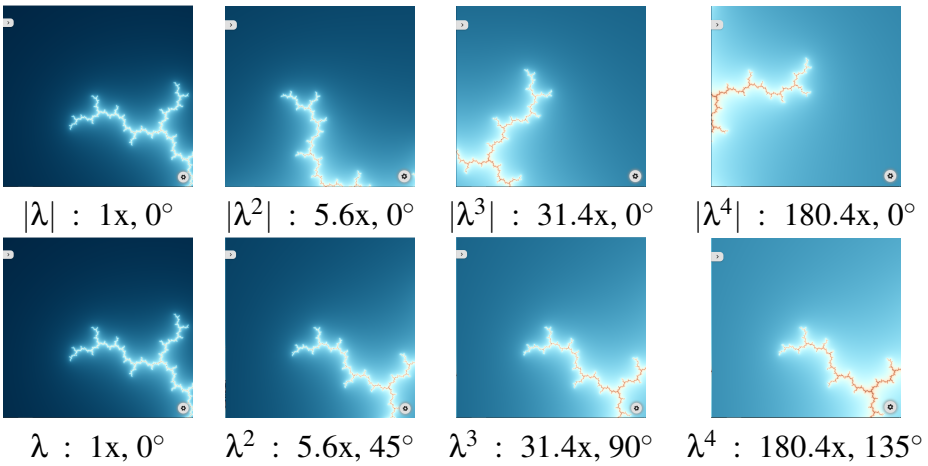


Figure 2.5: Successive transformations on J_c for $c = i$. Each image is centered at $i - 1$.

2.10 Asymptotic similarity in Julia sets

Julia sets also exhibit translational similarity. Under some special conditions, if we start with a point z and iterate under f_c, the resulting point is guaranteed to exhibit self-similarity in exactly the same way as z.

More formally, take any repulsive periodic or preperiodic point z. For any descendent $f^l(z)$ of z, provided that $(f^l)'(z) \neq 0$, then the limit models of J_c at z and at $f^l(z)$ are the same, up to a multiplication by $(f^l)'(z)$ (Tan 1990).

Points are **asymptotically similar** if this property holds. Upon inspection, this relates many points. Take two points $w, z \in J_c$ that share a common descendent $\alpha \in J_c$ such that $f_c^m(w) = \alpha$ and $f_c^n(z) = \alpha$. It is entirely possible that w and z are not descendants of *each other*; their orbits might take different routes to α. However, we can multiply J_c by either $(f^m)'(w)$ or $(f^n)'(z)$ and the resulting limit model will be the same as α. Therefore, this similarity is transitive - w and z are both similar to α, so are similar to each other. With some exceptions, points that enter the same cycle are similar.

To visualise this, let us examine the Julia set for $c = -0.10 + 0.96i$ (Figure 2.6). Note that it is composed of two distinct features; three-pronged branch points and six-pronged branch points.

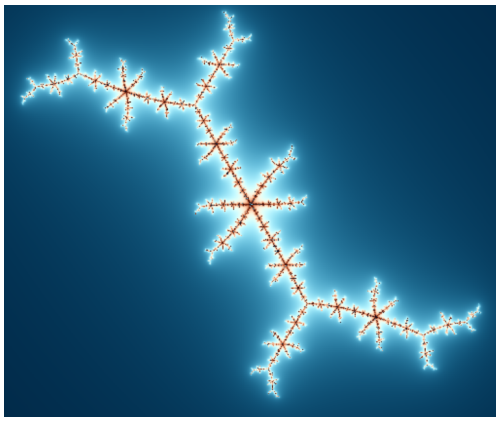


Figure 2.6: J_c

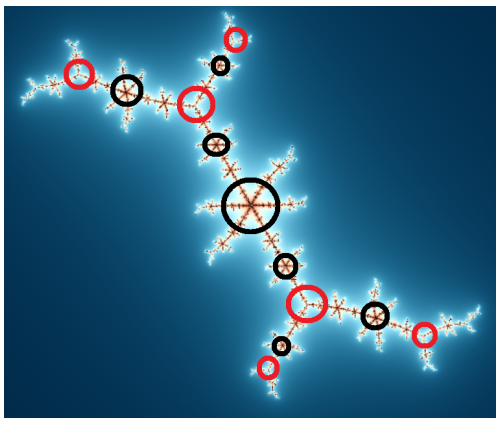
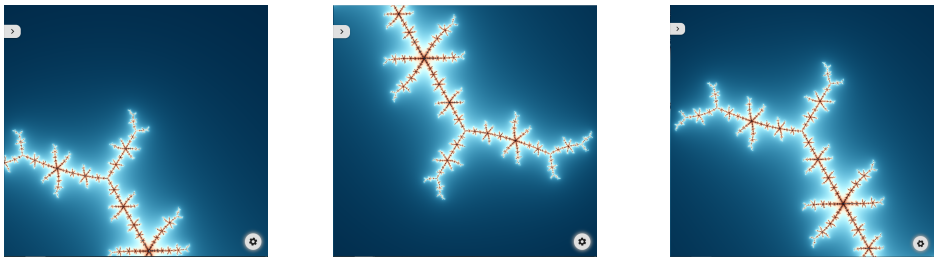


Figure 2.7: Three-pronged points are circled in red, six-pronged in black.

We will show the similarity by comparing three points: a periodic point, and two of its preperiodic ancestors. It turns out that this Julia set has only one cycle, a fixed point α with $f_c(\alpha) = \alpha$. Now, we can take two ancestors of α, namely c itself, and d such that $f_c^3(c) = f_c^1(d) = \alpha$. Therefore $m = 3$ and $n = 1$.

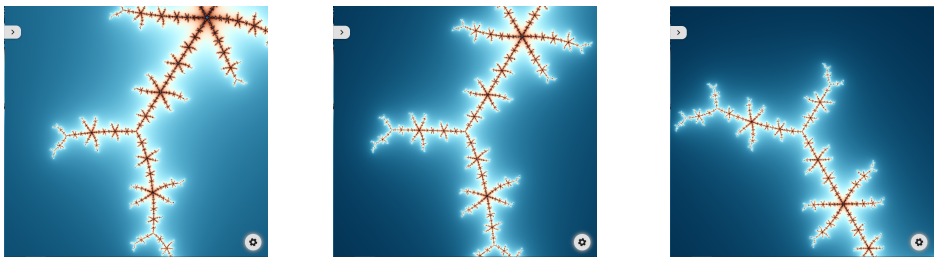
Firstly, as shown in Figure 2.8, we translate c, d and α to the origin.



$\tau_{-c}J_c$ $\tau_{-d}J_c$ $\tau_{-\alpha}J_c$

Figure 2.8: J_c centered on c, d, α after magnification and rotation

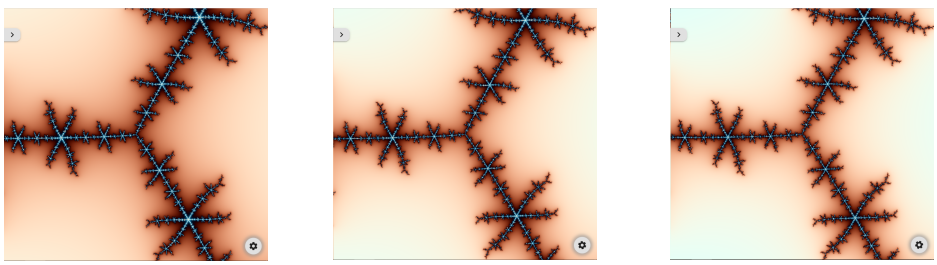
Secondly, we multiply the set translated to c by $(f_c^m)'(c)$ and the set translated to d by $(f_c^n)'(d)$. This ensures all of the sets have the same limit model, and has a fantastic geometric interpretation. Multiplying J_c by a complex number z is equivalent to magnifying J_c by $|z|$ and rotating J_c by $arg(z)$. Therefore, as seen in Figure 2.9, we simply have to magnify and rotate, and the regions around the three points look similar!



$(f_c^m)'(c)(\tau_{-c}J_c)$ $(f_c^n)'(d)(\tau_{-d}J_c)$ $\tau_{-\alpha}J_c$

Figure 2.9: J_c centered on c, d, α

Finally, each point is asymptotically self-similar in the same way, so the images become almost indistinguishable as we magnify.



$100(f_c^m)'(c)(\tau_{-c}J_c)$ $100(f_c^n)'(d)(\tau_{-d}J_c)$ $100(\tau_{-\alpha}J_c)$

Figure 2.10: J_c centered on c, d, α with additional magnification

The points we covered in the example were all three-pronged branch points, this raises a question. If there is only one cycle, how can there be both three-pronged and six-pronged branch points?

This is explained by the condition that $(f^l)'(z) \neq 0$. To understand when this happens, let us think about when f_c' is zero - the **critical points** of f_c. $(f^l)'(z)$ is a recursive derivative, so if it visits a critical point of f_c, then it will also be zero.

An interesting consequence of this is that points are only similar if they both visit the same critical point, *even* if they share some other common descendent. In some sense, critical points represent a 'crossover', because the critical point and its ancestors have one set of limit models, while its descendants have a different set of limit models. We can find the critical points of f_c by taking its derivative:

$$f_c(z) = z^2 + c \tag{2.18}$$

$$(f_c)'(z) = 2z \tag{2.19}$$

Solving for when the derivative is zero:

$$0 = 2z \tag{2.20}$$

$$z = 0 \tag{2.21}$$

We find that $z = 0$ is the only critical point of f_c. In answer to the question posed earlier, the six-pronged points are precisely zero and its ancestors! Meanwhile, because zero acts a crossover and $f_c(0) = c$, we know c and all of its descendants are three-pronged.

With this in mind, we define the set of points similar to c to be $S_c \subseteq J_c$. These points share a common descendent with c, but are not ancestors of c.

2.11 Computing similar points

In the previous chapter, we defined the set S_c to be the points in J_c that are similar to c. In this section, we discuss a algorithm to generate arbitrarily many members of S_c. This is of immense practical use - we can't visualise the similarity without examples of it! Indeed, this method was used to generate the points in the previous chapter.

An initial idea might be to simply take the forward orbit of c. These points clearly enter the same cycle as c, so comprise a subset of S_c. However, precisely because the orbit of c runs into a cycle, it has a finite number of descendants.

To generate more points, we can use the following property of Julia sets: for any point $z \in J_c$, the ancestors of z are guaranteed to also be in J_c (Milnor 1990). Therefore, we can iterate backwards from any point in S_c and generate an arbitrary number of points! Unless we visit a critical point, they will all be in S_c.

Upon closer inspection, this is a form of tree search. To illustrate this, let us examine the branching tree of points in Figure 2.11. Starting at point a, if we iterate forward, we will discover b, then c, then a again. If start at a and iterate backwards, we will discover $(c, -c)$, then $((x, -x),\ (b, -b))$, and so on.

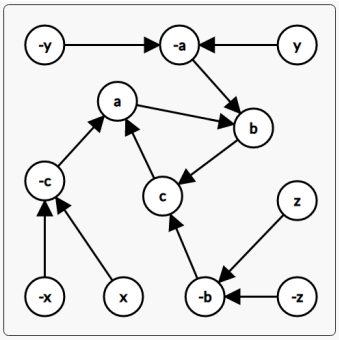


Figure 2.11: A cycle with members a, b, c and their ancestors (Graph drawn with `csacademy.com/app/graph_editor/`)

This algorithm generates at most 2^n points, where n is the depth of search. However, if we want to procedurally generate *all* of S_c, then there are two caveats to consider:

1. If we iterate backwards from c, then our list will not include the descendants of c, nor the ancestors of the descendants of c! To be thorough, we must first iterate forward as far as possible, until we reach the cycle, then begin our search.
2. We cannot include the ancestors of c, as the preimage of c is a critical point. We can simply terminate the branch of search that visits c.

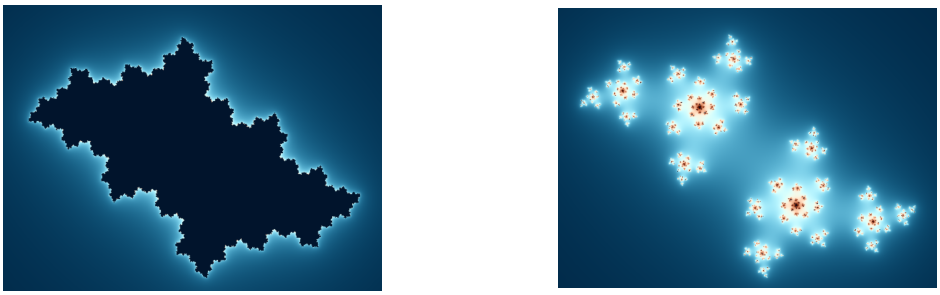
Given an infinite amount of time, this algorithm will generate all of S_c. However, there is a simple optimization we can make. If we start with a point in the cycle, then we will discover many duplicates as we loop back through the cycle. This can be avoided by undertaking p separate tree searches - one for each of the preperiodic preimages of the points in the cycle (*In Figure 2.11, this is* $-a, -b, -c$). The cycle itself can be appended to the list afterwards. This new algorithm returns at most $2^n p + p$ points.

2.12 The Mandelbrot set

"The real power in the Mandelbrot set is that this one object encompasses, in some sense, almost all of the dynamically interesting stuff - the stuff that happens under iteration - for all functions $z^2 + c$."

- Holly Kreiger (Numberphile 2014)

There are properties of Julia sets that seemingly require an infinite amount of information to calculate. For instance, which Julia sets are *connected* - densely packed, instead of forming a 'dust'?



(a) J_c for $c = -0.4 + 0.5i$ (b) J_c for $c = -0.4 + 0.7i$

Figure 2.12: Connected and disconnected Julia sets

It turns out that Julia sets J_c are connected exactly when c is in the Mandelbrot set (Peherstorfer & Stroh 2001).

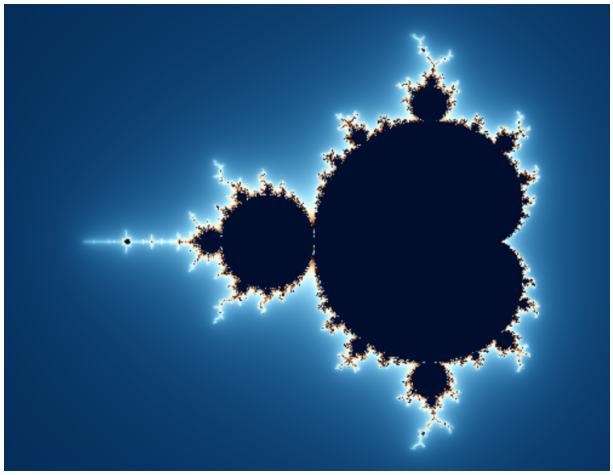


Figure 2.13: The Mandelbrot set

The Mandelbrot set is generated using our familiar function $f_c(z) = z^2 + c$. This time, we let c vary, and check which values cause the orbit of the critical point, zero, of f_c to remain bounded. By bounded, we mean that the magnitude never grows greater than 2 (Boston University 2021). Formally, the **Mandelbrot set** is the following:

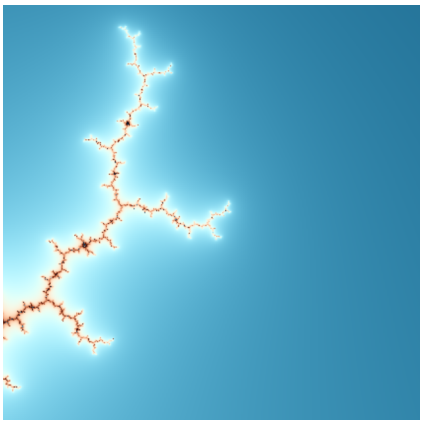
$$M = \{c \in C \mid |f_c^n(0)_{n \in N}| \leq 2\} \tag{2.22}$$

2.13 Misiurewicz points

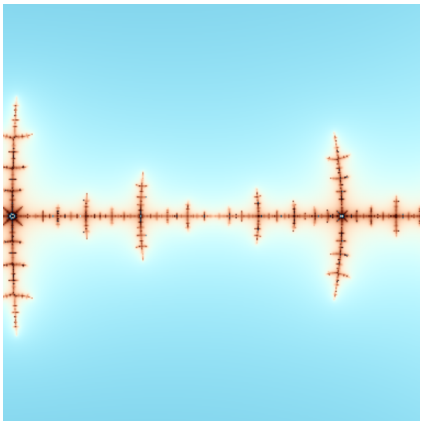
In a previous chapter, we defined a point as being preperiodic if its forward orbit contained a cycle. In the context of the Mandelbrot set, we can instead study when the forward orbit of zero contains a cycle. We say c is a **Misiurewicz point** if there exists some $q \geq 1, p \geq 0$ such that:

$$f_c^q(0) = f_c^{q+p}(0) \tag{2.23}$$

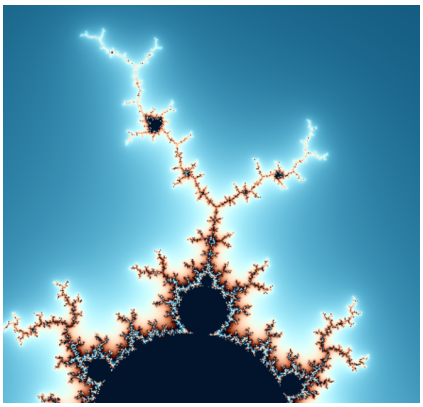
Misiurewicz points are only found in the boundary of the Mandelbrot set and are in fact, dense in the boundary of the Mandelbrot set. Geometrically, they are the centres of 'branch points'.



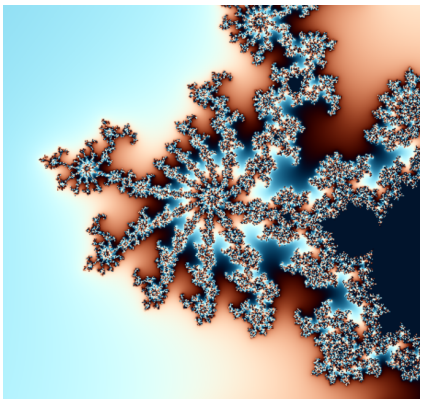
(a) a branch tip



(b) a 2-pronged branch point



(c) a 3-pronged branch point



(d) an 11-pronged branch point

2.14 Computing Misiurewicz points

The goal of this paper is to visualise a particular similarity between the Mandelbrot and Julia sets. It turns out that the similarity is exhibited at Misiurewicz points, so we need some way of computing them. In this section, we discuss two different methods, each with a specific use case. The first allows us to compute the entire set in a procedural manner, while the second allows us to take a region of the Mandelbrot set and find a nearby Misiurewicz point.

2.14.1 Solving polynomials

Recall that c is a Misiurewicz point of period p and preperiod q if it satisfies the following equality:

$$f_c^q(0) = f_c^{q+p}(0) \tag{2.24}$$

If we expand out the applications of f_c, we have:

$$\underbrace{((0^2+c)^2+c\ldots)^2+c}_{q \text{ applications}} = \underbrace{((0^2+c)^2+c\ldots)^2+c}_{q+p \text{ applications}} \tag{2.25}$$

This leaves us with a polynomial in c:

$$a_1c + a_2c^2 + \ldots + a_{q+p}c^{2^{q+p}} = 0 \tag{2.26}$$

Given some period and preperiod, we can solve for c by finding the roots of this equation. This allows us to enumerate the set of Misiurewicz points, handily indexing each point by its period and preperiod.

However, this method does have a practical limit - the polynomals are of order 2^{q+p}. Therefore, they become computationally infeasible to solve as the period and preperiod increase.

2.14.2 Newton's method with Misiurewicz domains

First documented in a blogpost (Heiland-Allen 2015), Misiurewicz domains shade the complex plane in a way that hints at the location of Misiurewicz points. Essentially, we group points based when their orbits are 'closest' to entering a cycle, using the function w:

$$w(z) = f_z^{q+p}(0) - f_z^q(0) \tag{2.27}$$

To generate the Misiurewicz domains of period p, we label each point z with the value q for which $|w(z)|$ is minimized. Misiurewicz points are surrounded by points with a value of q equal to the preperiod of the Misiurewicz point.

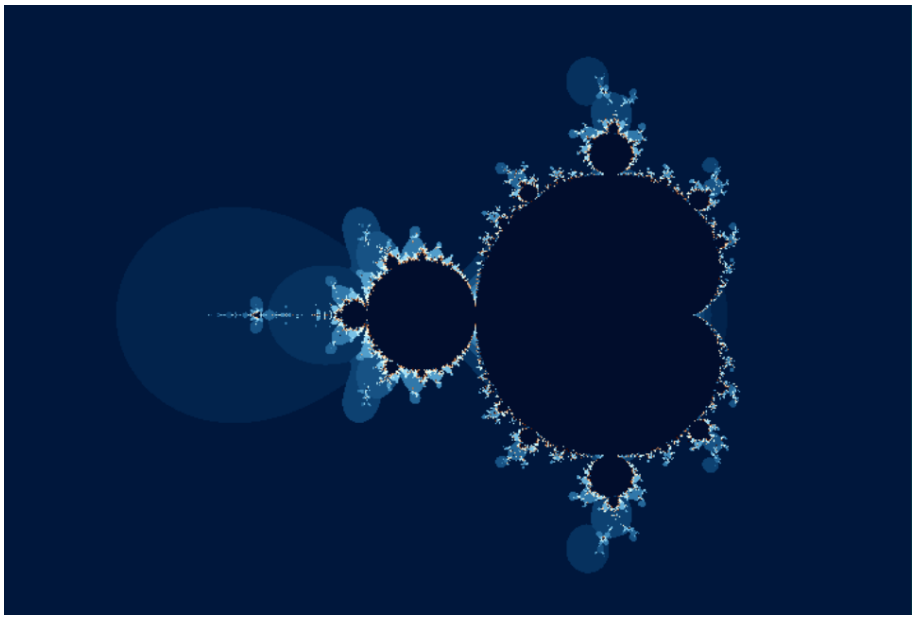


Figure 2.15: Misiurewicz domains of period 1

Note that if z is a Misiurewicz point and p is equal to its period, then $w(z) = 0$. Therefore, we aim to find the zeros of w. We can do this be applying Newton's method (Wikipedia 2021*b*), which tells us that the zeros of a function F are equal to the limit of the following sequence:

$$y_{i+1} = y_i - \frac{F(y_i)}{F'(y_i)} \tag{2.28}$$

With some initial guess y_0, each value y_i is a closer approximation of the root. In the context of Misiurewicz domains, we let $F = w$, while F' is:

$$F'(y) = (f_y^{q+p})'(0) - (f_y^q)'(0) \tag{2.29}$$

We can simply choose some point in a Misiurewicz domain to be the initial guess, then generate the sequence y_i to approximate the corresponding Misiurewicz point.

2.15 Asymptotic similarity between the Mandelbrot and Julia sets

It turns out that Misiurewicz points exhibit self-similarity in much the same way as their preperiodic counterparts in Julia sets. Before we state this formally however, we must define some functions.

First, recall the function w from subsection 2.14.2 :

$$w(c) = f_c^{q+p}(0) - f_c^q(0) \tag{2.30}$$

Then, let u' be the ratio of the derivative of w to the eigenvalue λ for c:

$$u'(c) = \frac{w'(c)}{\lambda - 1} \tag{2.31}$$

The similarity between the Mandelbrot and Julia sets is as follows. Take any Misiurewicz point c. Let α be the *first* periodic point visited in the orbit of c. For any point $z \in S_c$ with $f^l(z) = \alpha$, then the limit models of M at c and J_c at z are the same, up to a multiplication by $u'(c)$ and $(f^l)'(z)$ respectively (Tan 1990).

To visualise this, let us examine the Mandelbrot set at $c = 0.37 + 0.59i$. Note that the region around this Misiurewicz point bares a striking resemblance to the four-pronged branch points in the corresponding Julia set. Further examples of this similarity are given in the Implementation chapter.

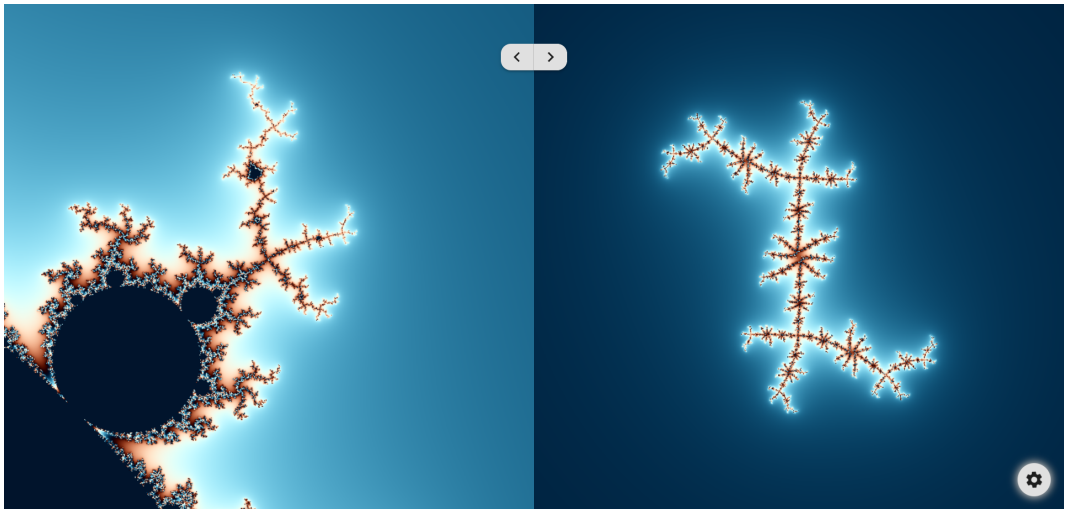


Figure 2.16: The Mandelbrot set at $c = 0.37 + 0.59i$, with the corresponding Julia set

Chapter 3

Implementation

The process of showing the similarity between the Mandelbrot and Julia sets has many stages, so this application uses a wizard-style workflow that leads the user through each step. Users can access this feature, termed the *"Tan's Theorem Explorer"* via the Settings menu in Mandelbrot Maps.

First, users are presented with an introduction popup which explains the process. Then, they select a point on each set and click to magnify and rotate the viewers. At the final stage, users are free to magnify further or select a new pair of points. Throughout, their current stage is indicated with a stepper - a UI element which displays progress through a sequence of numbered steps.

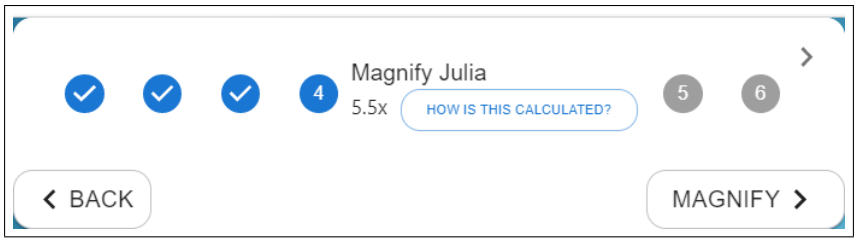


Figure 3.1: The stepper, after completing stages 1, 2 and 3

The stepper complements the wizard-style workflow by giving an overview of the process (Material UI 2021). However, to draw the users focus and maximise screen real estate, only the current step is fully labeled. Users interested in seeing all of the stages in full can expand the stepper via a small arrow in the top corner.

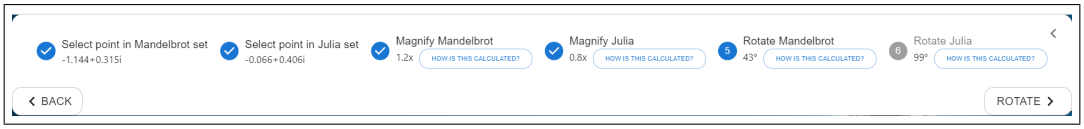


Figure 3.2: The expanded stepper

3.1 Step 0: Accessing the feature

The explorer is activated via a button in the Settings menu. This follows the pattern set by other features in Mandelbrot Maps, like the "About" button, and frees up screen real estate in the main application.

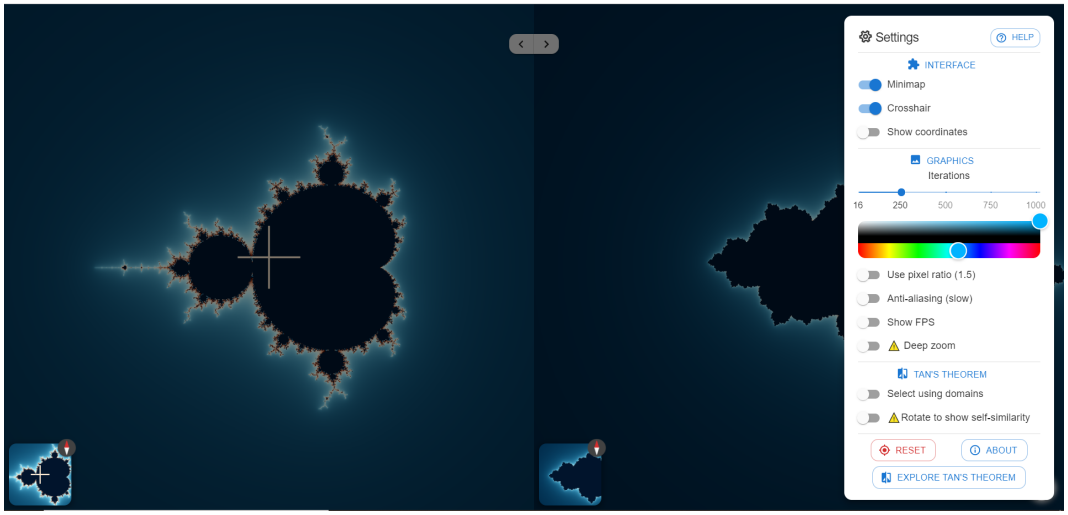


Figure 3.3: The Settings menu, with "Explore Tan's theorem" in the bottom right

After pressing the "Explore Tan's Theorem" button, users are presented with an introduction popup, which provides an informal statement of Tan's theorem and summarises the process. Future work could change this to only display the first time that a user accesses the explorer, and also make it accessible via some "help" button.

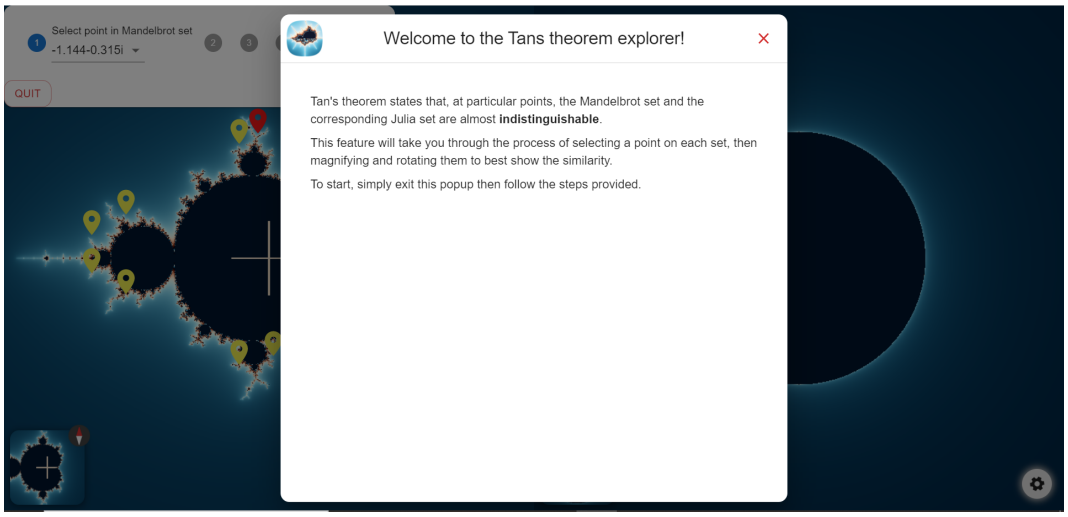


Figure 3.4: The introduction dialog

3.2 Step 1: Selecting a point in the Mandelbrot set

In the first step of the wizard, users are prompted with the message:

"Select point in Mandelbrot set"

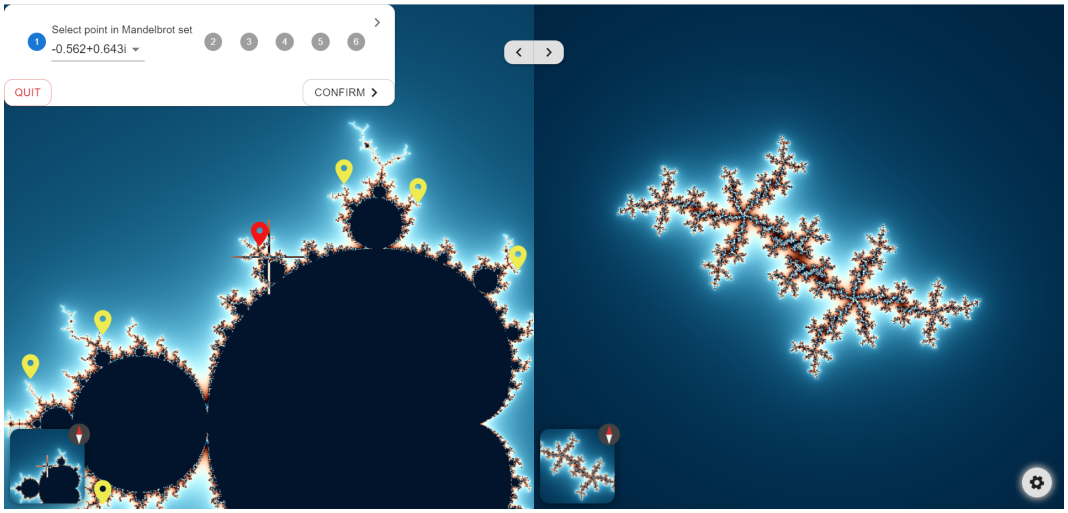


Figure 3.5: Step 1

Users select from a set of 200 Misiurewicz points. These points have been precomputed using the "solving polynomials" method detailed in subsection 2.14.1. As detailed in the following subsections, nearby points are highlighted with map markers, while a dropdown lists the entire set.

Once they have selected a point, users press "confirm" to progress to the next stage. After this, the translation, magnification and rotation controls for the Mandelbrot set are made inactive, so the user can only interact with the Julia set. Behind the scenes, the magnification and rotation factors to show the similarity for the Mandelbrot set are calculated. In addition, the set of points for the Julia set are generated.

3.2.1 Map markers

The Misiurewicz points are individually labelled with "Google Maps"-style map markers, a concept likely familiar to most users. Markers are coloured yellow to differentiate them from the standard blue shader used for the Mandelbrot set. The currently selected marker is coloured red to stand out further.

To aid performance and prevent screen clutter, up to eight points are marked at any given moment. The points that are displayed is based on the users position, magnification and rotation; the portion of the Mandelbrot curve in view. This is analogous to the way Google Maps highlights geographical features. At a low magnification, larger objects like towns and cities are marked. At high magnifications, individual streets and building are marked.

It turns out that we have a way of measuring the 'size' of Misiurewicz points. As discussed in section 2.15, the quantity $u'(c)$ is used to calculate the magnification and

rotation factors to show the similarity with the corresponding Julia set. If $|u'(c)|$ is small, then the magnification factor is small, so the similarity is visible from a low magnification. The branch point is 'large' in the sense that its structure is easily visible at a low magnification!

The points that are marked at any given time is determined in the following way: upon loading the application, the list of points is sorted by $|u'(c)|$ in ascending order. Then, at regular intervals, the list is searched in order for points that are within the bounds of the viewer and not currently selected. We can stop searching once we have the desired amount; the subsequent points will only have a higher value of $|u'(c)|$.

To ensure users do not perceive the points 'lagging' behind as they move the viewer, the list is searched every 10 milliseconds. This is intentionally less than 13ms, the shortest interval of time that a human can process an image in (Potter et al. 2014). The relatively small size of the list ensures the search terminates in time - in the worst case, there are no points within the bounds of the viewer, so we have to search the whole list.

3.2.2 Dropdown

Users can also select a point via a dropdown under their current stage. This lists all 200 of the precomputed points (not just the visible map markers) by their coordinates, so users can more quickly find a specific point. As a dropdown, it is compact and unobtrusive when not in use.

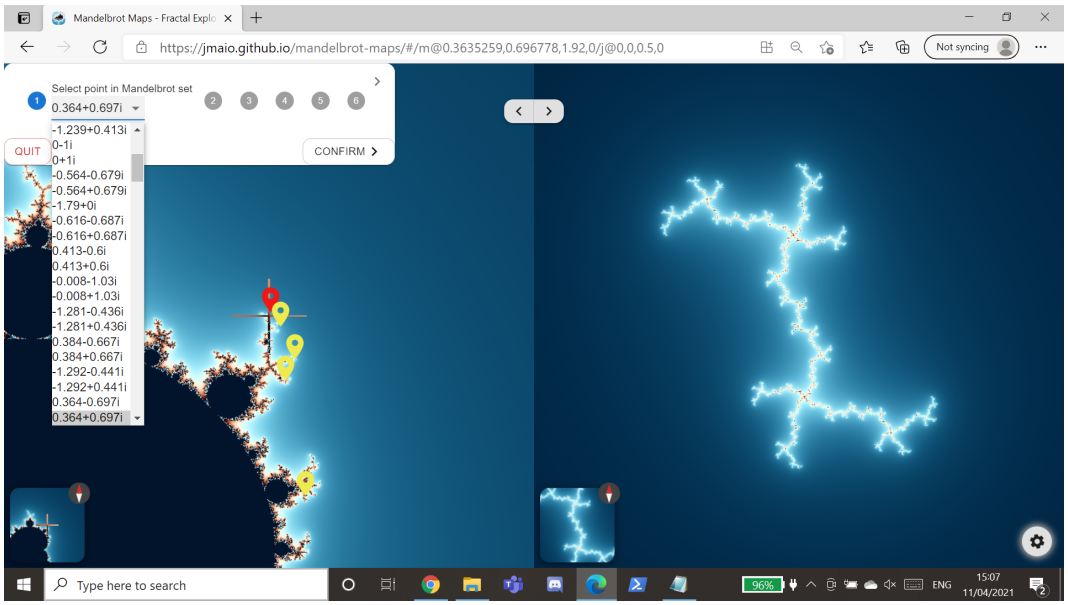


Figure 3.6: Step 1, with the dropdown selected

The currently selected point is shared between the markers and dropdown, so the dropdown serves a double purpose. Clicking on a marker will display its coordinates in the dropdown. Similarly, selecting a point from the dropdown will highlight the corresponding map marker and translate the viewer to the appropriate coordinates.

3.2.3 Misiurewicz domains

Alternatively, users can select a point using Misiurewicz domains (subsection 2.14.2). This toggle for this option is located in the Settings menu, under the "Tan's Theorem" options. While active, and the user is in step 1, the standard Mandelbrot shader is replaced with a Misiurewicz domains shader. Users simply center the viewer over a domain, and press a button to compute the nearest Misiurewicz point.

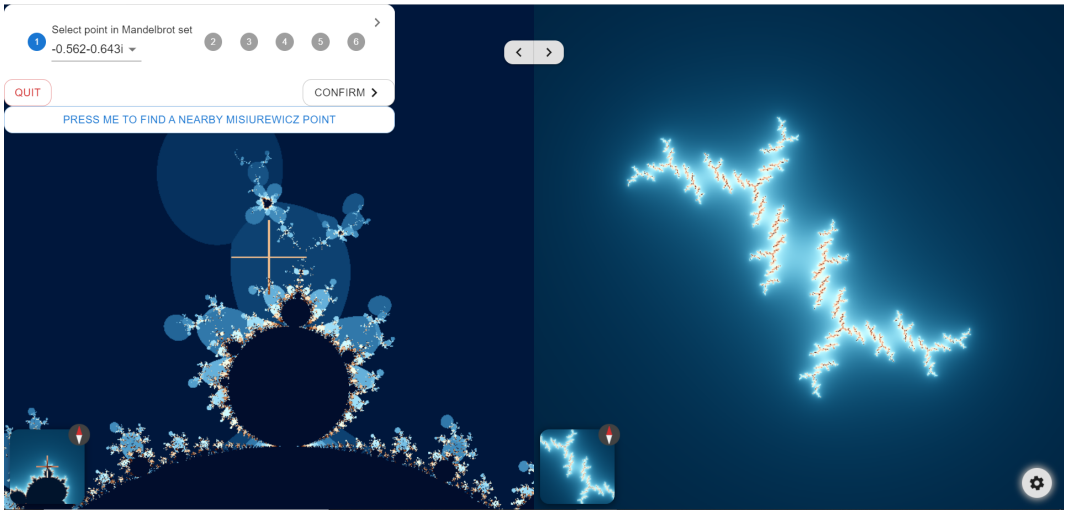


Figure 3.7: Before pressing the button to find a Misiurewicz point

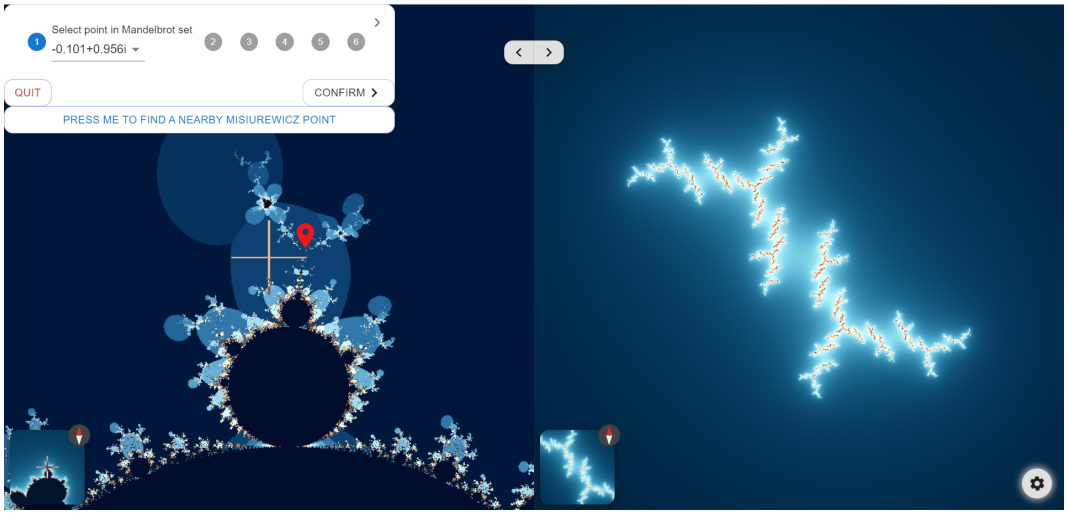


Figure 3.8: After the Misiurewicz point is found

This setting is aimed somewhat towards advanced users, as it requires more input and an understanding of what Misiurewicz domains are. However, it greatly increases the range of available points. The list of precomputed points is fixed, but here, we can iterate as much as necessary to approximate the nearest point.

To simplify the computations, the current shader searches for domains of period 1. Future work could add an option to find Misiurewicz points with different periods.

3.3 Step 2: Selecting a point in the Julia set

In the second step of the wizard, users are prompted with the message:

"Select point in Julia set"

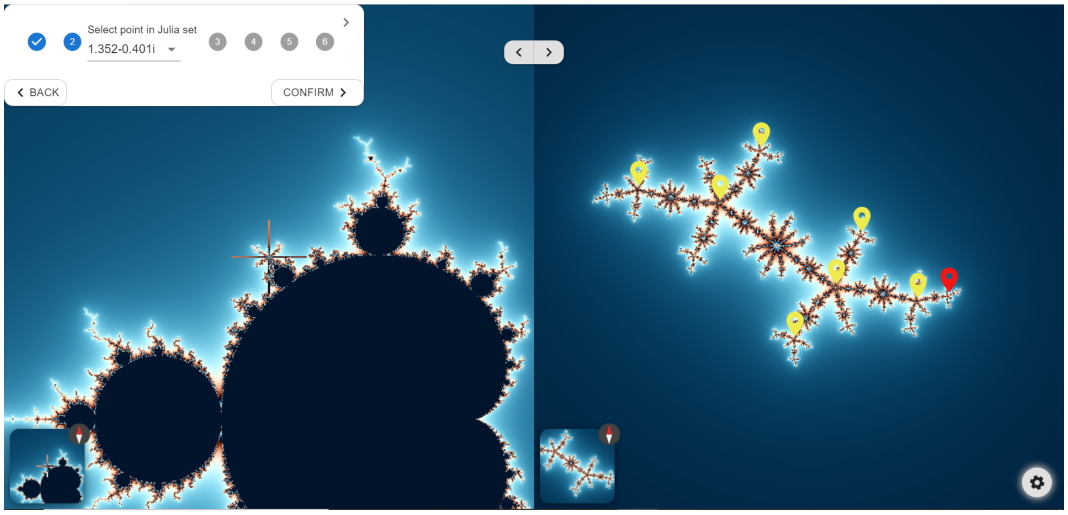


Figure 3.9: Step 2

This stage is much the same as the previous, with three differences:

- The map markers are positioned on the Julia set.
- There is no selection method equivalent to Misiurewicz domains. Users can only select from a precomputed list of points.
- The precomputed points are not Misiurewicz points. Instead, given that the user chose some point c in step 1, they are now presented with the set S_c. This set is usually infinite, so only a fixed number of points is computed, using the depth first search method described in section 2.11 to a fixed depth.

Once they have selected a point, users press "confirm" to progress to the next stage. After this, the translation, magnification and rotation controls for the Mandelbrot and Julia set are made inactive, so the user can only interact by clicking buttons when prompted. Behind the scenes, the magnification and rotation factors to show the similarity for the Julia set are calculated.

3.4 Step 3 - 6: Magnifying and rotating each set

In step three, the user clicks to magnify the Mandelbrot set, then in step four, they click to magnify the Julia set. In steps five and six, the sets are rotated in the same order. These steps are exhibited in Figure 3.10, Figure 3.12, Figure 3.13 and Figure 3.14.

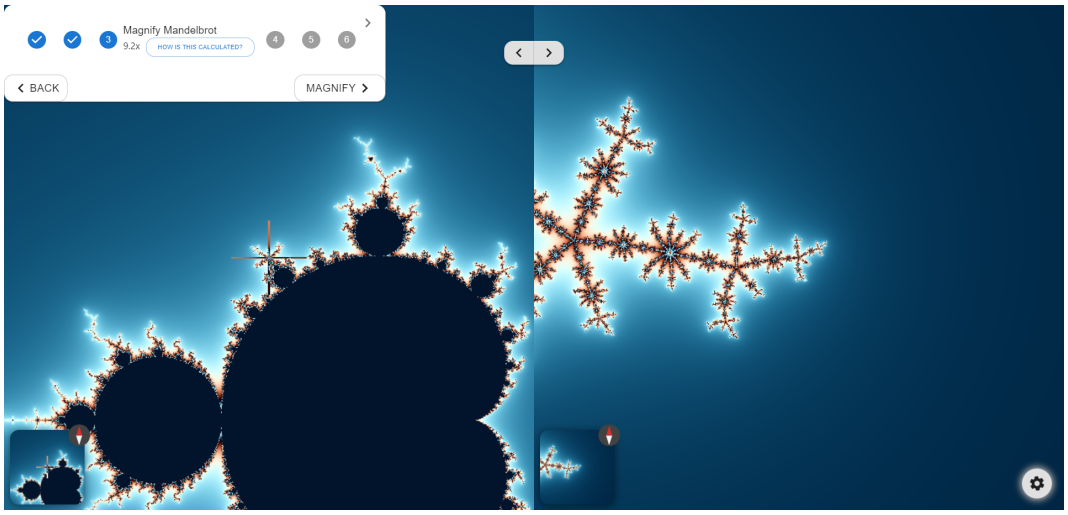


Figure 3.10: Step 3

These steps could have been combined into a single, automated animation which would require less input from the user. However, separating them out allows the user to click through the stages at their own speed, making it is easier to follow. A single step animation would require careful tuning of the rotation and magnification speed to ensure it was not jarring. In addition, the similarity manifests more gradually - after step four, both sets have been magnified and are similar up to a rotation.

We can also display more information about the process. Namely, a *"how is this calculated?"* button has been added which explains how the particular magnification or rotation factor was calculated in a popup.

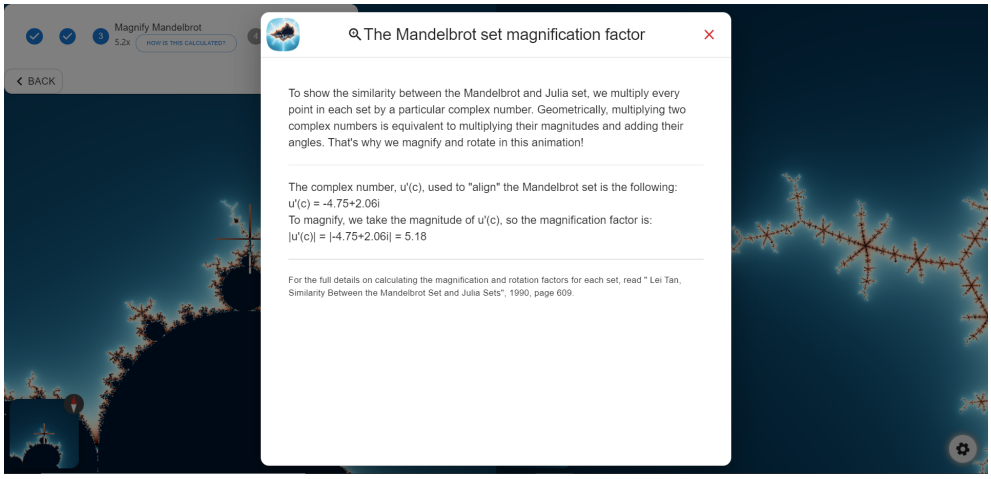


Figure 3.11: The *"how is this calculated?"* popup in stage 3

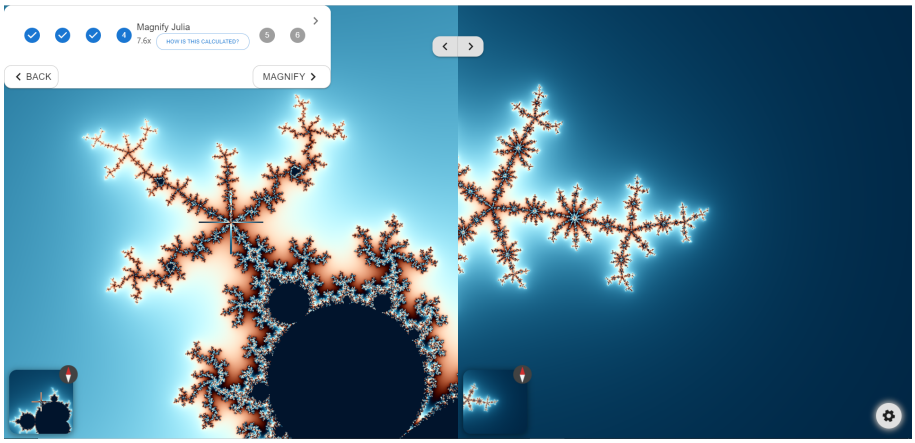


Figure 3.12: Step 4

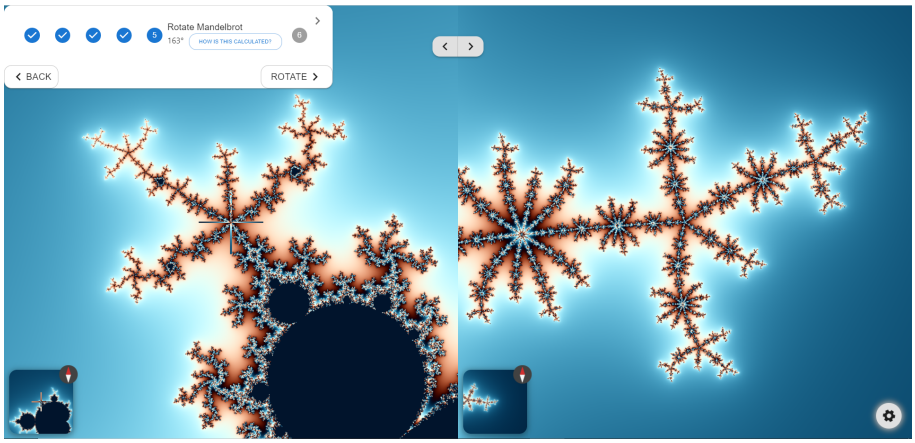


Figure 3.13: Step 5

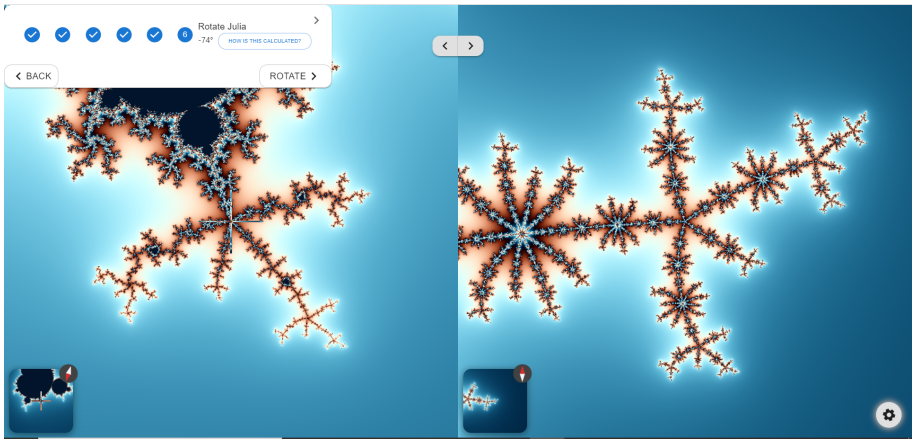


Figure 3.14: Step 6

3.5 Step 7: Magnifying each set further

In the seventh and final step of the animation, users are prompted with the message:

"You are now free to continue magnifying"

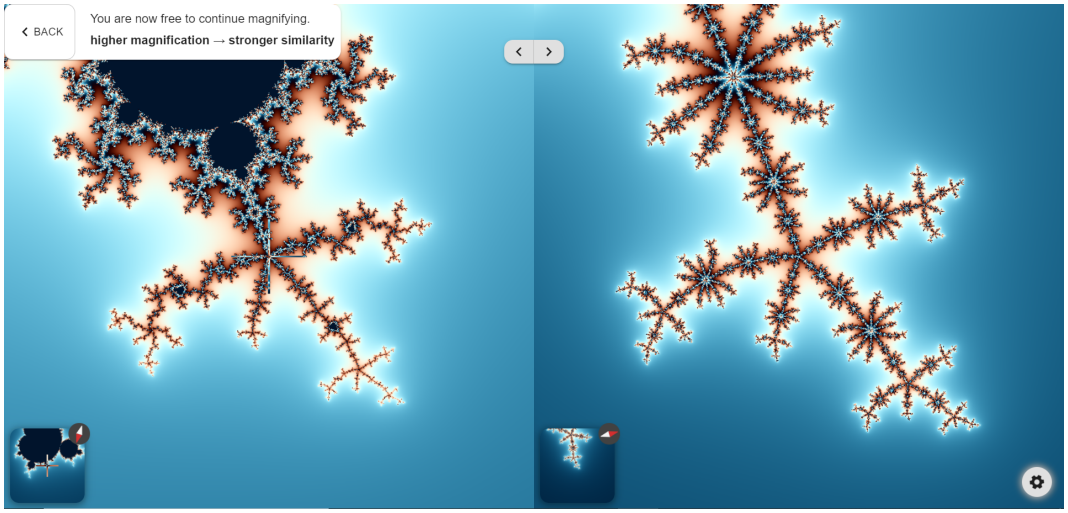


Figure 3.15: Step 7

The similarity between the Mandelbrot and Julia set is asymptotic, so here, users can magnify further to see the similarity more clearly. To improve visibility, the progress bar is hidden.

The translation and rotation controls for each set are still inactive, but the magnification controls become **synchronised** in the following sense: magnifying one set magnifies *both* sets by the same amount (multiplied by the magnification factors from the previous steps to maintain the similarity).

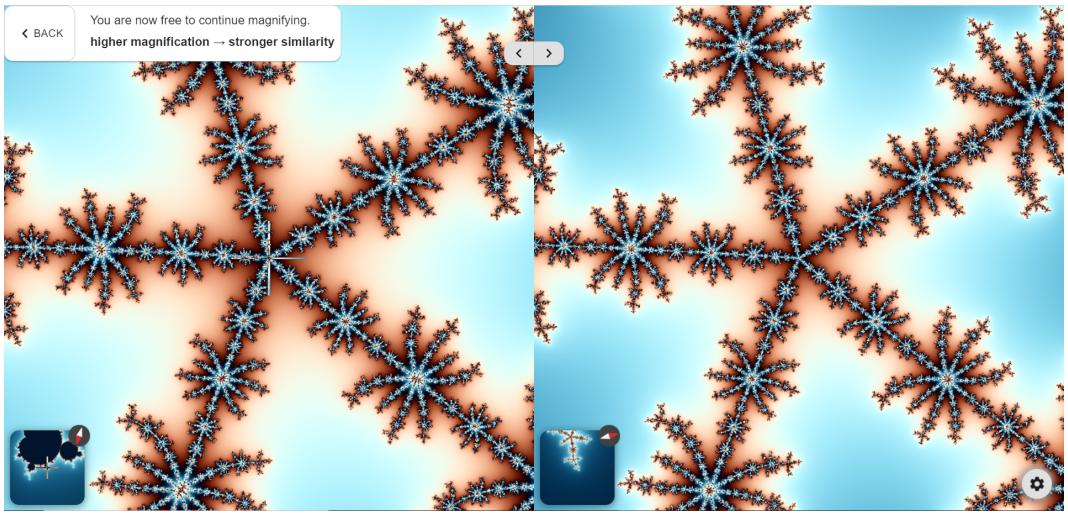


Figure 3.16: Step 7, after some manual magnification

3.5.1 Rotating to show self-similarity

The Tan's Theorem Explorer is intended to show the similarity between the Mandelbrot and Julia set. However, as described in section 2.9, the Mandelbrot set and Julia set are *self-similar* about Misiurewicz points c and points $z \in S_c$, respectively.

Namely, if the point has the eigenvalue λ, the same structure appears repeatedly as the set is multiplied by λ^n, providing that n is increased in integer increments. It would seem λ is not present in the Tan's Theorem Explorer, but there is a subtlety here. Provided that $|\lambda| > 1$, we can interpret the users manual magnification (a positive real number) as multiplying the sets by $|\lambda^n|$ for some n! To show the self-similarity, we can use λ and the current magnification to solve for n, then apply $arg(\lambda^n)$ as an additional rotation factor. This is equivalent to multiplying the sets by λ^n.

This toggle for this setting is marked with an exclamation mark in the Settings menu, as large eigenvalues can cause excessive rotation. A slider in the corner displays the current value of n (modulo 1), indicating which structure the user should see. For example, the following four figures show the Julia set for $c = -1.14 + 0.32$ at two different magnifications. The latter two figures have the added rotation; the slider is in the same place (around one fifth of the way), and the images are almost indistinguishable.

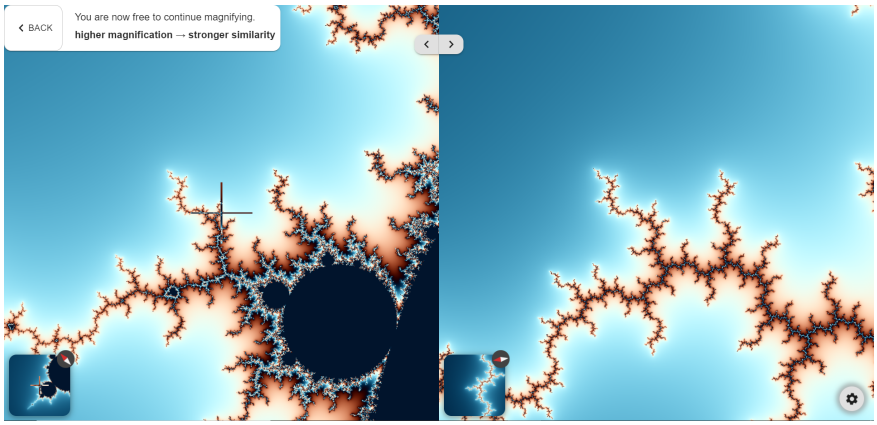


Figure 3.17: Without self-similarity rotation at $c = -1.14 + 0.32$ with $n = 2.2$

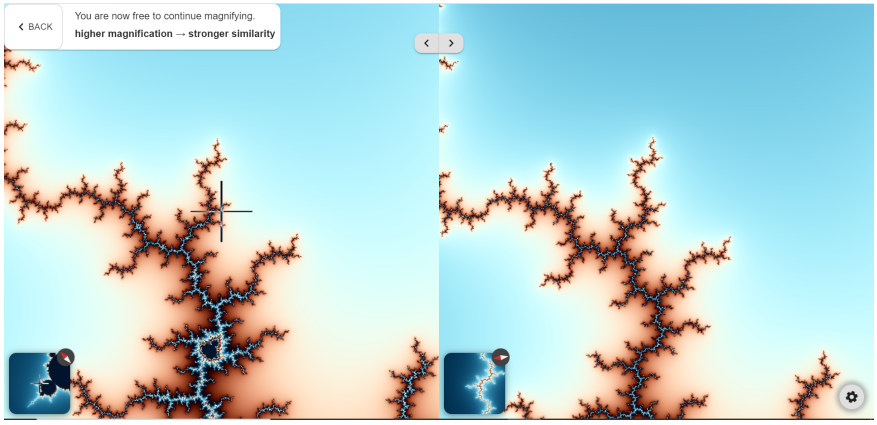


Figure 3.18: Without self-similarity rotation at $c = -1.14 + 0.32$ with $n = 3.2$

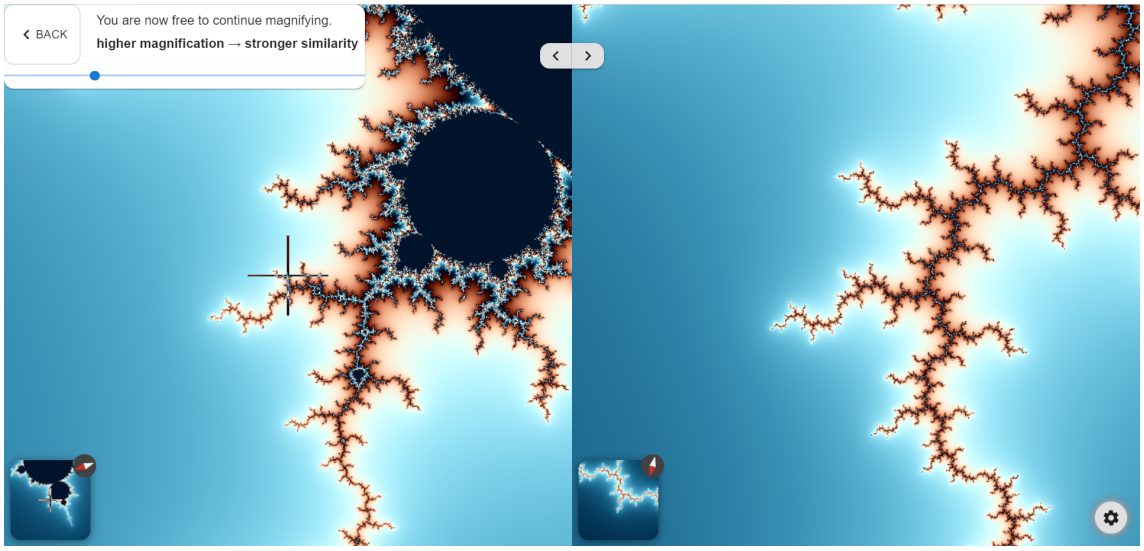


Figure 3.19: With self-similarity rotation at $c = -1.14 + 0.32$ with $n = 2.2$

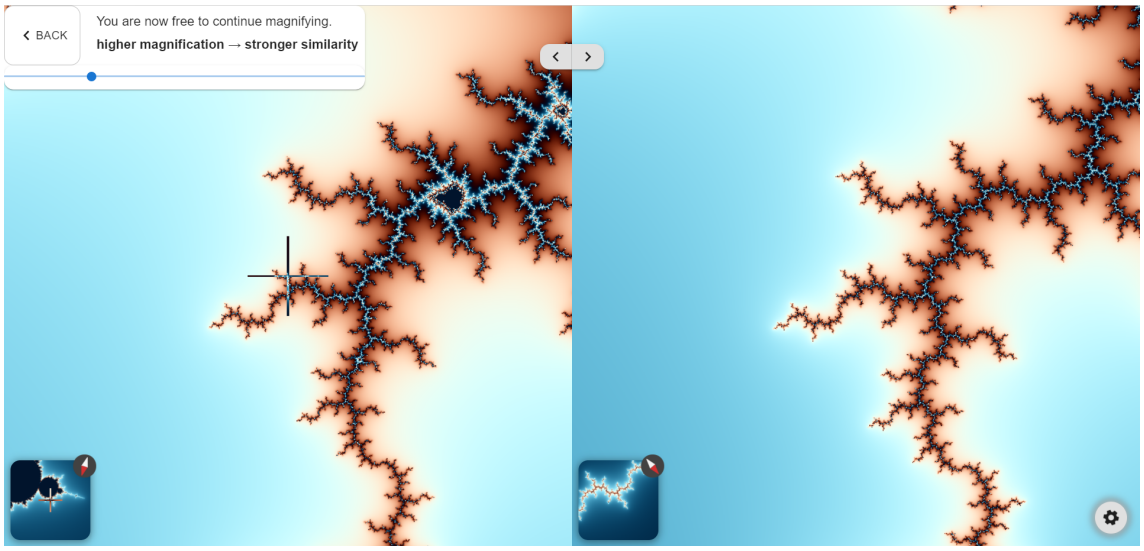


Figure 3.20: With self-similarity rotation at $c = -1.14 + 0.32$ with $n = 3.2$

Chapter 4

Evaluation

4.1 User Survey

With three students contributing to Mandelbrot Maps this year, the decision was made to merge all of the changes and publicise a single version of Mandelbrot Maps, along with single survey. This was done to save respondents time and maximise responses - they did not need to try a different website and fill in a different survey for each students contributions. I would like to thank everyone who took part.

The survey is unsupervised; it asks participants to explore the app on their own, then answer the questions through the Microsoft Forms platform. Within the survey, there was a section labelled “Tan’s Theorem Explorer”. The four questions, all optional, are detailed in Table 4.1. Users rate the features from 1 to 5 stars: one star being a negative experience and five stars being a positive experience.

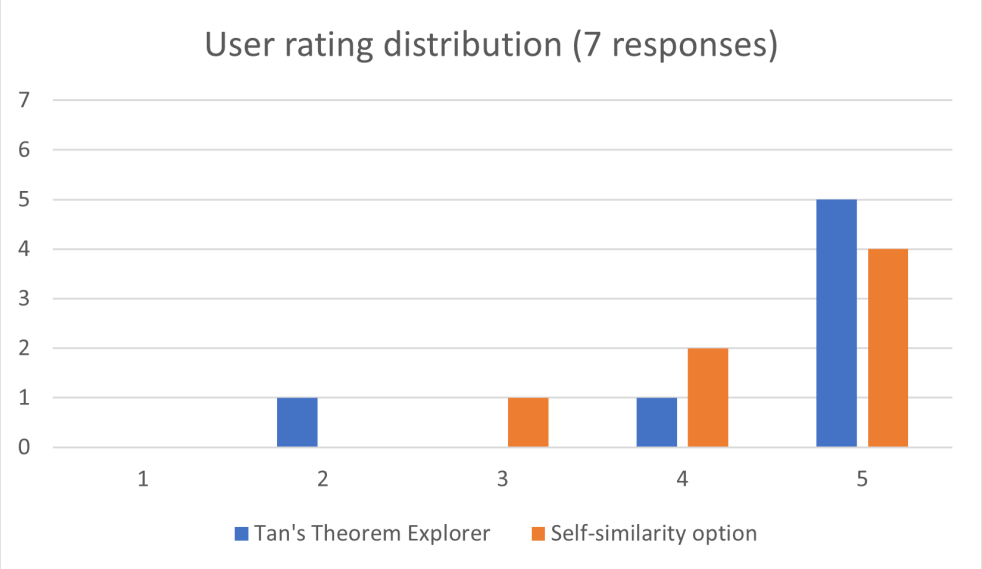


Figure 4.1: User rating statistics indicate positive reception of the application.

Question	**Answer**
If you tried the "Explore Tan's theorem" mode, how satisfied were you with the result?	1-5
After following the steps provided, did the Mandelbrot and Julia sets look similar?	Yes / No
If you tried the "Rotate to show self-similarity" option, how satisfied were you with the result? (more information on this option is available in the [Help] menu)	1-5
Do you have specific feedback for the Tan's theorem Explorer? Were the instructions helpful?	[Short Text]

Table 4.1: Questions asked in the user survey. A $1 - 5$ scale corresponds to [1 (Very Negative)] – [5 (Very Positive)].

The results from the survey were positive, with the "Explore Tan's theorem" mode receiving an average satisfaction rating of 4.43 out of 5 from 7 respondents.

Respondents were satisfied with the "Rotate to show self-similarity" option, with it also receiving an average rating of 4.43 out of 5 from 7 respondents.

When asked whether the Mandelbrot and Julia sets looked similar after following the steps provided, 7 respondents said Yes, while 1 said No. The one negative response requires further investigation - the similarity is a proven theorem, so there could be some issue with the code. It may also be that they did not manually magnify, or, as indicated in the feedback, they were on mobile and could not progress with the feature.

At the end of the section, users were free to leave short-text feedback. There were six comments; they are follows:

- The instructions were really clear and I liked seeing it step by step.
- good explanation
- It would be cool if it would zoom in a bit more in the end. Math font maybe? Otherwise it was so interesting to see how the two sets are similar and relate
- Good stuff!
- Instructions were helpful. As said before: Including some theoretical details would be interesting :)
- The UI was basically unusable on mobile (small iPhone SE 1 screen) with elements overlapping the parts I was supposed to interact

Overall, these responses were positive, complementing that the process is broken up into steps with helpful instructions. One respondent wished more theoretical details. An explanation of the mathematics with the appropriate typesetting is sorely missing from the app, in part because Tan's theorem is complex! Lastly, one user complained about the experience on mobile. The feature was not built with mobile in mind, and, for instance, the progress bar does not fit on phone screens. Future work could build a separate layout optimized for mobile.

Chapter 5

Conclusion

5.1 Closing remarks

The ultimate goal of this report, and the application presented in it, is to teach people interesting properties of the Mandelbrot set and Julia sets. If just one person has learnt something new, I would say it has been successful.

More practically however, this report showcases an entirely new feature of Mandelbrot Maps. Users can easily select two points and see the similarity between the Mandelbrot set and the corresponding Julia set. In addition, using novel features like Misiurewicz domains, they can find infinitely many examples of it!

This new functionality was added to Mandelbrot Maps with a view to support future work, be that with Tan's theorem, or visualising other properties of the Mandelbrot set. We have entered an era where real-time fractal viewers are the norm, and we can leverage this to build tools that communicate mathematics to the masses.

5.2 Further work

5.2.1 A tour of the application

As currently presented, this feature is not easily accessible to new users. Firstly, the button is not very noticeable, being located at the bottom of the Settings menu. Secondly, it is labelled "Explore Tan's theorem", which may be confusing to users unfamiliar of Tan's theorem. Equally, this name does not reference the similarity, which would communicate the purpose of the feature better. As the survey indicated, many respondents were unsure of the purpose of the application.

A future version of the application could include an introduction to the apps features, which would highlight each button and explain their purpose.

5.2.2 More in-depth explanations of the mathematics

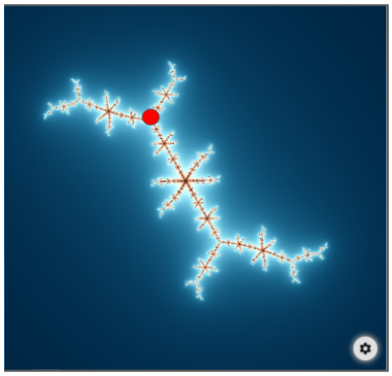
Although this application can display the similarity between the Mandelbrot and Julia set, it does little to prove to users *why* the similarity exists. The only attempt is with the *"how is this calculated?"* button, which essentially prompts users to read Tan's original paper.

Future work could include more definitions and plain language explanations in-app. For example, a *"what are Misiurewicz points?"* button. Ideally, these concepts would be explained visually - the next section discusses a specific example of this.

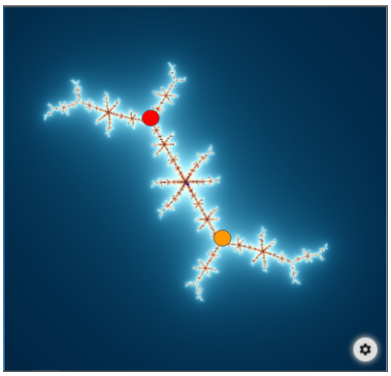
5.2.3 Visualising the relationship between Julia set points

Currently, users are prompted to select a point on the Julia set from a preset list, with no explanation of how the list was generated. Here, we propose a way of visualising this process.

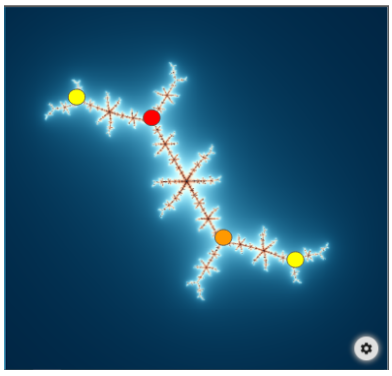
As discussed in section 2.11, we generate a list of points in the Julia set with a tree search, which discovers points at an exponential rate relative to the depth of the search. Future work could add an additional stage to the explorer that explains the search method, and highlights the points generated at each depth.



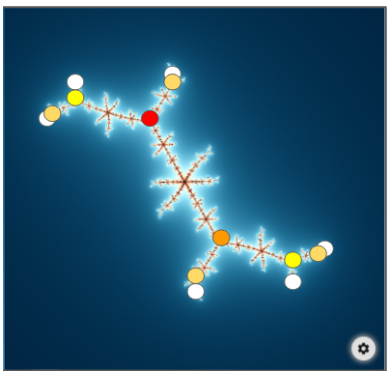
(a) Points with preperiod 0 (the cycle)



(b) Points with preperiod 1



(a) Points with preperiod 2



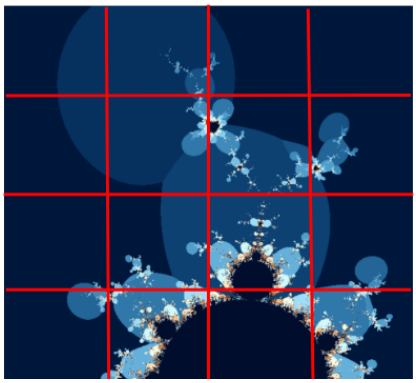
(b) Points with preperiod 3

5.2.4 Misiurewicz domain sampling

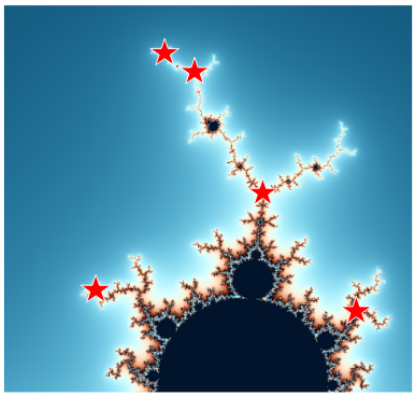
Currently, users can either select Misiurewicz points from a preset list, or use Misiurewicz domains to discover points themselves. A new approach, here referred to as *Misiurewicz domain sampling*, might unify these two approaches.

The insight behind Misiurewicz domain sampling is the following: in an ideal world, users shouldn't have to discover points themselves. Instead, they should be free to navigate the Mandelbrot set as normal, and any nearby Misiurewicz points should be instantly highlighted for them.

Domain sampling would work in the following way: at regular intervals, the computer takes an array of initial points and applies the gradient descent method from subsection 2.14.2 to each. If a given initial point was in a domain, this will find the corresponding Misiurewicz point for that domain. These points are then highlighted to the user as normal. In the example below, the initial points are placed in a 3x3 grid.



(a) What the computer sees



(b) What the user sees

This method combines the strengths of the current selection methods, as it requires no input from the user, but provides many points. It also doesn't need to be computationally expensive, because the coordinates of the Misiurewicz points do not have to be exact until the user selects them. For instance, if we updated the list every 100ms, had 10 initial points, and iterated 100 times for each, then this only equates to $\frac{10*100}{0.1} = 10000$ operations per second.

Bibliography

Boston University (2021), 'The mandelbrot set explorer mathematical glossary.'. http://math.bu.edu/DYSYS/explorer/def.html.

Corbett, A. (2014), 'Mandelbrot maps – google play store.'. https://play.google.com/store/apps/details?id=uk.ac.ed.inf.mandelbrotmaps.

Feder, J. (1998), *Fractals*, Plenum Press.

Grégoire, N. & Bouillot, M. (1998), 'Hausdorff distance between convex polygons'. http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html.

Heiland-Allen, C. (2015), 'Misiurewicz domain coordinates and size estimates'. https://mathr.co.uk/blog/2017-11-21_misiurewicz_domain_coordinates_and_size_estimates.html.

Mandelbaum, R. F. (2018), 'This trippy music video is made of 3d fractals'. https://gizmodo.com/this-trippy-music-video-is-made-of-3d-fractals-1822168809.

Material UI (2021), 'Stepper'. https://material-ui.com/components/steppers/.

Miao, J. (2020), 'Mandelbrot maps webgl application for exploring fractals'.

Milnor (1990), *Dynamics in One Complex Variable, 3rd ed.*, Southgate Publishers.

Numberphile (2014), 'Filled julia sets'. https://www.youtube.com/watch?v=oCkQ7WK7vuY&ab_channel=Numberphile2.

Parris, I. (2008), '"fractals don't have to be scary." mandelbrot maps: Creating a real-time mandelbrot/julia fractal explorer'.

Peherstorfer, F. & Stroh, C. (2001), 'Connectedness of julia sets of rational functions', *Comput. Methods Funct. Theory* **1**, 61–79.

Potter, M. C., Wyble, B., Hagmann, C. E. & McCourt, E. S. (2014), 'Detecting meaning in rsvp at 13 ms per picture', *Attention, Perception, Psychophysics* **76**, 270–279.

Tan, L. (1990), 'Similarity between the mandelbrot set and julia sets', *Communications in Mathematical Physics* **134**(3), 587–617.

Wikipedia (2021*a*), 'Mandelbrot set'. https://en.wikipedia.org/wiki/Mandelbrot_set.

Wikipedia (2021*b*), 'Newton's method'. https://en.wikipedia.org/wiki/Newton%27s_method.