Bertolt-Brecht-Gymnasium
Schwarzenberg

Komplexe Leistung

# Fraktale

Kevin Hoang

Betreuer
Herr Heller

21. Dezember 2023

# Inhaltsverzeichnis

# Einleitung

Fraktale sind faszinierende Strukturen, die uns allgegenwärtig umgeben. Von der verzweigten Struktur eines Baumes bis hin zu den Wolken am Himmel, die sich in immer kleiner werdende Strukturen aufteilen, sind Fraktale ein fester Bestandteil unserer natürlichen Umgebung. Aber auch in der Mathematik sind Fraktale zu finden.
Der Begriff *Fraktal* wurde um 1975 vom Mathematiker Benoît Mandelbrot geprägt [5] und beschreibt natürliche und künstliche Formen, die bestimmte geometrische Eigenschaften aufweisen.
Fraktale zeichnen sich durch ihre *Selbstähnlichkeit* auf unterschiedlichen Größenskalen aus. Das bedeutet, dass Teile der Struktur in verschiedenen Vergrößerungen ähnliche Formen aufweisen.

Im theoretischen Teil dieser komplexen Leistung werde ich mich mit der mathematischen Theorie hinter Fraktalen beschäftigen, insbesondere mit der *Mandelbrot-Menge* und den *Julia-Mengen*.
Der praktische Teil beinhaltet die Algorithmen sowie deren Implementierung zur Darstellung von Fraktalen mithilfe von Computergrafik und der Programmiersprache *C++*, wobei insbesondere die *OpenGL*-Bibliothek genutzt wird.

Ziel dieser komplexen Leistung ist es, nicht nur die mathematischen Grundlagen von Fraktalen zu verstehen, sondern auch die Ästhetik von Fraktalen zu verdeutlichen, sowie auf die Anwendung von Fraktalen in der realen Welt einzugehen.

# 1 | Theoretischer Teil

In diesem Abschnitt werden die theoretischen Grundlagen erschlossen, um im praktischen Teil ein Programm zur Darstellung von Fraktalen erstellen zu können.

## 1.1 Was sind Fraktale?

Fraktale sind Strukturen, die durch die wiederholte Anwendung einfacher Schritte aus sich selbst entstehen. Dabei weisen Fraktale auf unterschiedlichen Größenskalen betrachtet ähnliche oder identische Formen auf, was als Selbstähnlichkeit bezeichnet wird. Durch *Iteration* werden bestimmte Anweisungen auf immer kleiner werdenden Maßstab wiederholt, um das gewünschte Fraktal darzustellen.

Ein anschauliches Beispiel für diese Konzepte ist die *Koch-Kurve*. Diese wird durch die iterative Umwandlung einer geraden Linie gebildet, wobei der Mittelpunkt zu einer Spitze umgeformt wird. Dieser Prozess wird auf den durch die Spitze neu entstandenen Linien wiederholt, wodurch mit jeder Iteration eine immer komplexer werdende, selbstähnliche Struktur entsteht.

Abbildung 1.1:
1. Iteration der Koch-Kurve

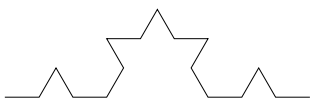

Abbildung 1.2:
2. Iteration der Koch-Kurve

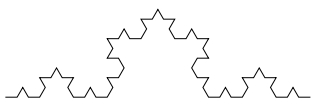

Abbildung 1.3:
3. Iteration der Koch-Kurve

## 1.2 Komplexe Zahlen

Um die Berechnung der Mandelbrot-Menge und der Julia-Mengen im nächsten Abschnitt zu verstehen, werden zunächst die grundlegenden Konzepte zu *komplexen Zahlen* erläutert.
Komplexe Zahlen sind eine Erweiterung der reellen Zahlen $\mathbb{R}$, mit dem Ziel, Gleichungen wie $x^2 = -1$ lösbar zu machen.
Eine komplexe Zahl $\mathbb{C}$ besteht aus der Summe eines *Realteils* $a$ und eines *Imaginärteils* $b \cdot i$, wobei $b$ mit einer *imaginären Einheit* $i = \sqrt{-1}$ erweitert wird und zu folgender Schreibweise führt.

$$z = a + b \cdot i$$

Anders als reelle Zahlen, welche auf einem 1-dimensionalen Zahlenstrahl dargestellt werden, verwendet man für die komplexen Zahlen die *Gauß'sche Zahlenebene*, welche einem 2-dimensionalen Koordinatensystem gleicht. Dabei wird die $x$-Achse in den Realteil und die $y$-Achse in den Imaginärteil eingeteilt.

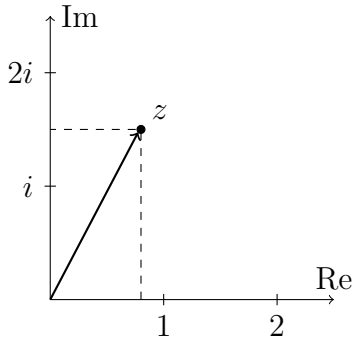


Abbildung 1.4: $z = 0,8 + 1,5i$ in der Gaußschen Zahlenebene

Um mit zwei komplexen Zahlen $z = a + bi$ und $w = c + di$ rechnen zu können, gelten die folgenden Rechenregeln.

- **Addition und Subtraktion**: $z \pm w = (a \pm c) + i \cdot (b \pm d)$ [4]
- **Multiplikation**: $z \cdot w = (ac - bd) + i \cdot (ad + bc)$ [4]

Der Betrag $|z| = \sqrt{a^2 + b^2}$ gibt die quantitative Menge der komplexen Zahl $z$ an.[1]

## 1.3 Mandelbrot-Menge

Die Mandelbrot-Menge gilt als das wohl bekannteste Fraktal und wurde im Jahr 1980 vom Mathematiker Benoît Mandelbrot entdeckt. Sie beschreibt die Menge aller komplexen Zahlen, deren Betrag bei Anwendung folgender iterativen Vorschrift gegen einen festgelegten Grenzwert konvergiert.

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

[2]
Man beginnt mit $z_0 = 0$ als Startwert und einer komplexen Zahl $c$ als Konstante. Der Grenzwert, gegen den der Betrag der komplexen Zahl $z$ konvergieren soll, wird auf 2 gesetzt. Zahlen, welche diesen Grenzwert nicht überschreiten, gehören zur Mandelbrot-Menge.
Zahlen, die diesen Grenzwert überschreiten, divergieren höchstwahrscheinlich ins Unendliche und gehören somit nicht zur Mandelbrot-Menge.

Wenn eine komplexe Zahl $c$ zur Mandelbrot-Menge gehört, zeichnet man diese farbig in die Gauß'sche Zahlenebene ein. Diese Farbe wählt man anhand der Anzahl der Iterationen, die benötigt werden, um den Grenzwert zu erreichen. Auf die Farbgebung wird im praktischen Teil nochmal genauer eingegangen.

Die grafische Darstellung der Mandelbrot-Menge zeigt vielfältige Strukturen, wie das charakteristische *Apfelmännchen* in der Grundform mit zahlreichen Verzweigungen am Rand. Diese Strukturen setzen sich in selbstähnlicher Weise auf immer kleiner werdenden Maßstab fort.

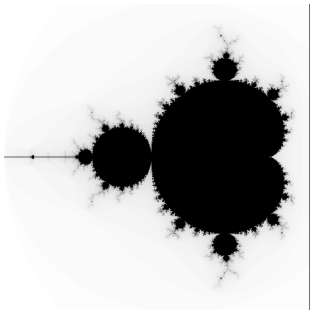

Abbildung 1.5: Apfelmännchen

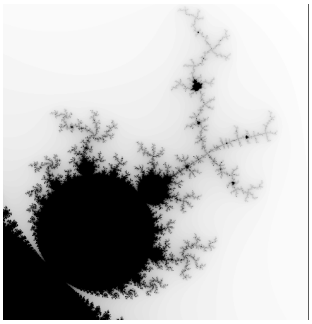

Abbildung 1.6: Vergrößerung einer Abzweigung

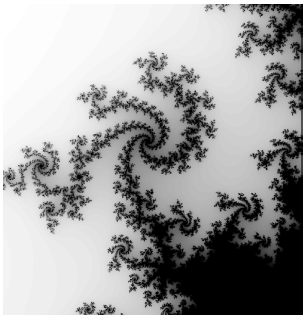

Abbildung 1.7: Spiralförmige Verzweigungen

## 1.4 Julia-Mengen

Ein weiteres Fraktal, welches eng mit der Mandelbrot-Menge zusammenhängt, sind die Julia-Mengen. Jeder komplexen Zahl $c$ ist eine Julia-Menge zugeordnet, wobei die komplexen Zahlen, die auch zur Mandelbrot-Menge gehören, Julia-Mengen erzeugen, welche ähnliche Strukturen der Mandelbrot-Menge aufweisen.
Sie wurden erstmals 1919 von dem französischen Mathematiker Gaston Julia und dem französischen Physiker Pierre Fatou beschrieben. [6][7]

Julia-Mengen werden ebenfalls durch die iterative Zahlenfolge $z_{n+1} = z_n^2 + c$ gebildet. Anders als bei der Mandelbrot-Menge wird jedoch der Startwert $z_0$ nicht auf 0 gesetzt, sondern auf die jeweilige komplexe Zahl $z$, für die berechnet wird, ob sie zur jeweiligen Julia-Menge der komplexen Zahl $c$ gehört.

In der grafischen Darstellung weisen Julia-Mengen vergleichbare Eigenschaften auf wie die Mandelbrot-Menge. Ihre Strukturen sind jedoch weniger komplex, gleichen aber den jeweiligen Strukturen der Mandelbrot-Menge.

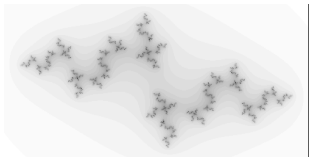

Abbildung 1.8:
Julia-Menge für
$c = -1,0 + 4,0i$

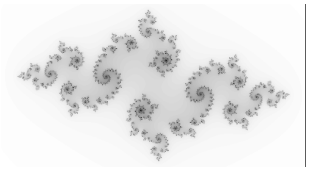

Abbildung 1.9:
Julia-Menge für
$c = -0,8 + 0,2i$

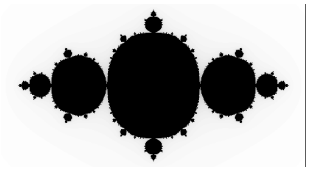

Abbildung 1.10:
Julia-Menge für
$c = -0,8 + 0,0i$

# 2 Praktischer Teil

Im praktischen Teil dieser komplexen Leistung werden die in der Theorie erläuterten Konzepte angewendet und in ein Programm umgesetzt.
Da die Erklärung des gesamten Programms den Rahmen dieser Arbeit sprengen würde, wird hier nur auf die wichtigsten Aspekte eingegangen. Der Quellcode des Programms ist im Anhang zu finden. Der Quellcode ist an den wichtigen Stellen mit Kommentaren versehen.

## 2.1 Benutzeroberfläche

Das Programm *Fraktale* ist ein interaktives Programm, um Fraktale zu visualisieren und zu erkunden.
Es besteht aus zwei Teilen: Einer Zeichenfläche, auf die Fraktale gerendert werden, und einem Einstellungsfenster, in dem man das aktuelle Fraktal und dessen Parameter, wie die Anzahl der Iterationen und den Farbgebungsalgorithmus, einstellen kann. Das Fraktal kann man ganz einfach mit der Maus verschieben und mit dem Mausrad zoomen.

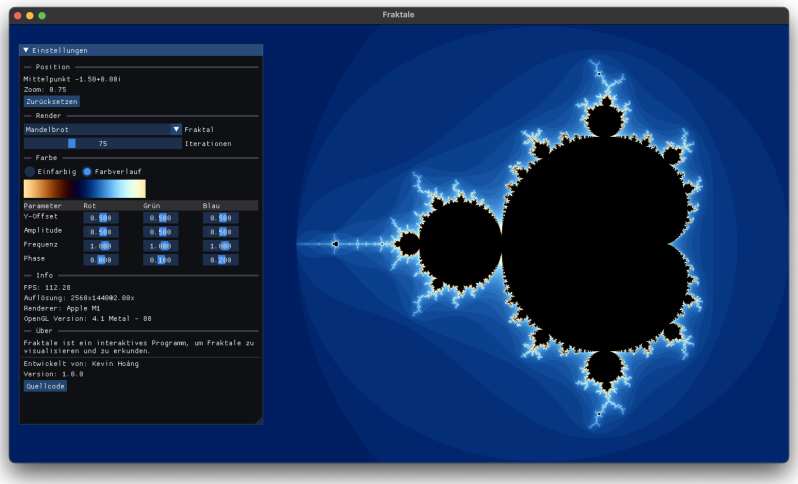


Abbildung 2.1: Das Programm *Fraktale*

## 2.2 Programmaufbau

Das Programm wurde in der Programmiersprache *C++* geschrieben und nutzt die Programmierschnittstelle *OpenGL* um eine effiziente und leistungsstarke Umsetzung der Algorithmen zu ermöglichen. Die Algorithmen wurden in Form von *Shadern* implementiert, die auf der Grafikkarte ausgeführt werden. Diese werden in der Programmiersprache *GLSL* geschrieben.

Zusätzlich wurden Bibliotheken verwendet, deren genauere Beschreibung und Funktion sich im Anhang befindet.

Das Programm besteht aus folgenden Dateien:

- `main.cpp` - Die Hauptdatei, in der das Programm initialisiert wird.
- `window.h` - In dieser Datei befinden sich Funktionen, die das Fenster erstellen und verwalten.
- `dearimgui.h` - In dieser Datei befinden sich Funktionen, um die Benutzeroberfläche zu erstellen und zu verwalten.
- `callbacks.h` - In dieser Datei befinden sich Funktionen, um die Fraktale mit der Maus zu verschieben und zu zoomen.
- `shader.h` / `shader.cpp` - Eine Hilfsdatei, die das Laden und Kompilieren von Shadern ermöglicht.
- `misc.h` - Hilfsdatei, die Funktionen enthält, die in anderen Dateien verwendet werden.
- `VertexShader.vert` - Der Vertex Shader, welcher für die Positionierung der *Vertices* zuständig ist.
- `FragmentShader.frag` - Der Fragment Shader, wo die eigentliche Berechnung des Fraktals stattfindet.

In der Datei `main.cpp` werden in der `main`-Funktion die Funktionen zum Erstellen des Fensters und der grafischen Benutzeroberfläche sowie OpenGL Funktionen aufgerufen. Die weitere Ausführung des Programms findet in der *Hauptschleife* statt, welche durch die Bedingung `window_should_close()` gesteuert wird. In dieser Schleife werden die C++ Variablen an den Shader übergeben, der *Framebuffer* gerendert und die grafische Benutzeroberfläche mittels *Dear ImGui* gezeichnet.

## 2.3 Implementierung der Algorithmen

Die Algorithmen wurden in der Programmiersprache GLSL implementiert. In dieser Programmiersprache gibt es die Datentypen `int` und `float` für ganze und gebrochene Zahlen. Außerdem gibt es die Typen `vec2`, `vec3` und `vec4`, welche einen jeweils n-dimensionalen Vektor repräsentieren.
Der GLSL Code des Shaders wird für jedes Pixel des Framebuffers ausgeführt. Die Position des aktuellen Pixels ist durch die Variable `gl_FragCoord` gegeben. Dieser Pixel wird zunächst in eine komplexe Zahl umgewandelt, welches in der `map`-Funktion geschieht. Mit den *Uniform*-Variablen `u_resolution`, `u_offset` und `u_zoom` werden die Verschiebung und der Zoom unter Berücksichtigung der Auflösung des Fensters berechnet. Dies ermöglicht es dem Benutzer, das Fraktal mit der Maus zu verschieben und zu zoomen.

Da es in GLSL keine komplexen Zahlen gibt, wird in der Implementierung der Typ `vec2` verwendet, um eine komplexe Zahl $z = a + bi$ darzustellen, was zur folgenden Variablendeklaration führt:

```
vec2 z = vec2(a, b);
```

### 2.3.1 Mandelbrot-Menge

Die Mandelbrot-Menge wird in der Funktion `mandelbrot` berechnet. Diese hat als Parameter eine komplexe Zahl `c` und gibt die Anzahl der Iterationen `n` zurück, die benötigt werden, um festzustellen, ob die komplexe Zahl `c` in der Mandelbrot-Menge liegt.

```
float mandelbrot(vec2 c) {
    vec2 z = vec2(0.0, 0.0);
    float n = 0.0;
    for(int i = 0; i < int(u_iterations); i++) {
        z = vec2(z.x * z.x - z.y * z.y, 2.0 * z.x * z.y);
        z += c;
        n += 1.0;
        if(length(z) > 2.0) {
            break;
        }
    }
    return n;
}
```

Um die iterative Vorschrift $z_{n+1} = z_n^2 + c$ zu implementieren, wird die Variable `z` mit einem Startwert von 0 festgelegt. In der Schleife wird die `z` zuerst quadriert und dann die komplexe Zahl `c` addiert.
Die Bildung des Quadrates von `z` erfolgt durch Anwendung der 2. binomischen Formel:

$$\begin{aligned} z^2 &= (a + bi)^2 \\ z^2 &= a^2 + 2abi - b^2 \\ z^2 &= (a^2 - b^2) + (2ab)i \end{aligned}$$

Folglich in GLSL: `z = vec2(z.x * z.x - z.y * z.y, 2.0 * z.x * z.y);`.

Die Addition komplexer Zahlen ist analog zur Addition von Vektoren: `z += c`.

Die Funktion `length` berechnet die Länge eines Vektors, welche dem Betrag einer komplexen Zahl entspricht. Falls dieser größer als 2 ist, wird die Schleife abgebrochen und die Anzahl der Iterationen zurückgegeben.
Falls die Schleife nicht abgebrochen wird, wird die Anzahl der maximal eingestellten Iterationen `u_iterations` zurückgegeben.

### 2.3.2 Julia-Mengen

Die Julia-Menge wird in der Funktion `julia` berechnet. Diese ist analog zur Funktion `mandelbrot` aufgebaut, die komplexe Zahl `c` wird hier jedoch als Startwert verwendet. Nach dem Quadrieren wird die komplexe Zahl `u_julia_c` addiert, welche vom Benutzer eingestellt werden kann.

```
float julia(vec2 c) {
    vec2 z = c;
    float n = 0.0;
    for(int i = 0; i < int(u_iterations); i++) {
        z = vec2(z.x * z.x - z.y * z.y, 2.0 * z.x * z.y);
        z += u_julia_c;
        n += 1.0;
        if(length(z) > 2.0) {
            break;
        }
    }
    return n;
}
```

### 2.3.3 Farbgebung

Nachdem man alle Punkte der Mandelbrot-Menge und der Julia-Menge berechnet hat, kann man die zugehörigen schwarzen Bereiche einzeichnen. Punkte, die nicht dazugehören, werden eingefärbt. Dies kann entweder einfarbig, mit verschiedenen Helligkeitsstufen einer Farbe erfolgen, die je nach Anzahl der benötigten Iterationen variiert, wenn der Betrag von $z$ gegen unendlich divergiert. Alternativ kann auch eine Farbpalette verwendet werden, was zu interessanten Abbildungen führt.

#### Einfarbig

Die einfache Farbgebung erfolgt durch die Funktion `colorize_single`.

```
vec3 colorize_single(float n) {
    if(n == u_iterations) {
        return vec3(0.0, 0.0, 0.0);
    }
    float hue = n / u_iterations;
    return u_color * vec3(hue, hue, hue);
}
```

Die Farbe wird durch einen *RGB-Vektor* repräsentiert. Die vom Benutzer eingestellte Farbe `u_color` wird mit dem `hue`-Wert multipliziert, der den Anteil der benötigten Iterationen zur Iterationsgrenze darstellt.

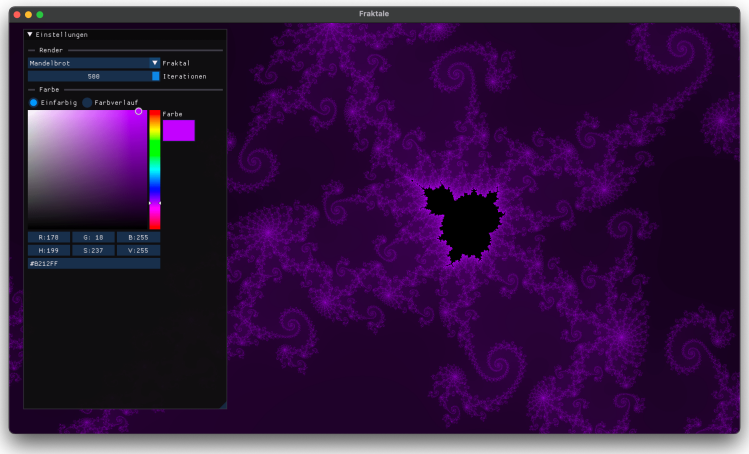


Abbildung 2.2: Mandelbrot-Menge mit verschiedenen Lila-Tönen

### Farbverlauf

Die Farbgebung mit Farbverlauf erfolgt durch die Funktion `colorize_gradient`.

```
vec3 colorize_gradient(float n) {
    if(n == u_iterations) {
        return vec3(0.0, 0.0, 0.0);
    }
    float t = n / u_iterations + 0.5;
    return u_a.xyz + u_b.xyz * cos(6.28318 * (u_c.xyz * t + u_d.xyz));
}
```

Die Funktion `colorize_gradient` wird verwendet, um einen Farbverlauf basierend auf der Gleichung $f(t) = a + b \cdot \cos(2\pi(c \cdot t + d))$ [3] zu generieren. Hierbei handelt es sich um 3 Funktionen für die Farbwerte Rot, Grün und Blau mit 4 unterschiedlichen Parametern, wobei $a$, $b$, $c$ und $d$ RGB-Vektoren sind, die vom Benutzer festgelegt werden können.

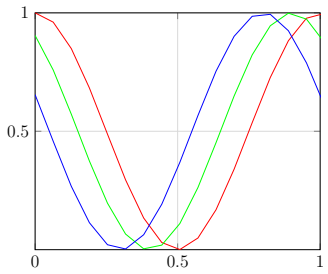


Abbildung 2.3: Die entgültige Farbe wird durch die Addition der einzelnen Farbwerte für den Wert $t$ berechnet.

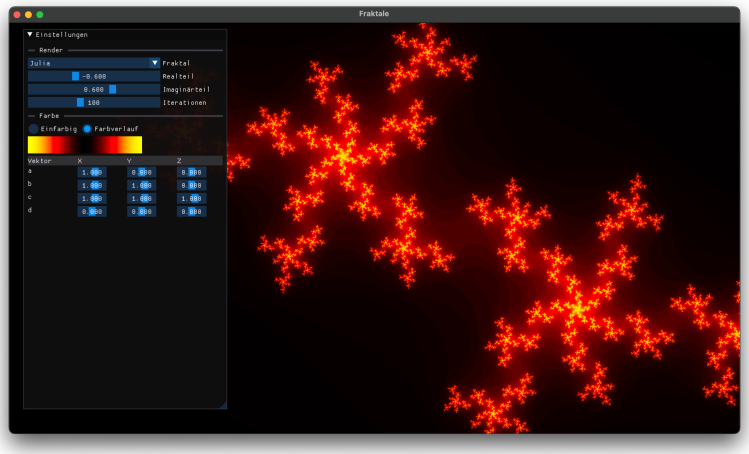


Abbildung 2.4: Julia-Menge für $c = -0.6 + 0.6i$ mit Gelb-Rot-Farbverlauf

# 3 | Fazit

Die Auseinandersetzung mit der faszinierenden Welt der Mathematik durch die Visualisierung von Fraktalen hat nicht nur Einblicke in mathematische Konzepte, sondern auch in die Computergrafik gegeben aber auch die ästhetische Schönheit dieser abstrakten Strukturen hervorgehoben.

Der technologische Fortschritt, insbesondere im Bereich der Computergrafik und Programmierung, hat es ermöglicht, Fraktale in einer Weise zu visualisieren, die vor einigen Jahrzehnten undenkbar war. Das Programm *Fraktale* bietet die Möglichkeit, interaktiv in die Welt der Fraktale einzutauchen, sie zu erforschen und ihre Schönheit zu erleben.

Die Anwendungsmöglichkeiten von Fraktalen erstrecken sich über viele verschiedene Bereiche, von reiner Kunst bis hin zu wissenschaftlichen Analysen. Die Selbstähnlichkeit und Komplexität von Fraktalen finden in Bildkompression, digitaler Kunst, Signalverarbeitung, Wettermodellierung und Finanzanalyse Anwendung.

Insgesamt zeigt diese komplexe Leistung, wie moderne Technologien und mathematische Konzepte zusammenarbeiten können, um eine Anwendung zu entwickeln, die sowohl informativ als auch ästhetisch ansprechend ist.

# 4 Anhang

## 4.1 Quellcode

Der Quellcode und das Programm sind über den beigelegten USB-Stick oder über folgenden GitHub Link verfügbar:
`https://www.github.com/kevinoverflow/Fraktale`

## 4.2 Bibliotheken

**CMake - `https://cmake.org/`**

*Ein plattformübergreifendes Open-Source-Build-System.*

**CMakeRC - `https://github.com/vector-of-bool/cmrc`**

*Ein Ressourcencompiler in einem einzigen CMake-Skript.*

**Dear ImGui - `https://github.com/ocornut/imgui`**

*Eine einfache Immediate-Mode-Grafikbenutzeroberfläche für C++.*

**GLFW - `https://www.glfw.org/`**

*Eine plattformübergreifende Bibliothek zur Erstellung von Fenstern mit OpenGL-Kontexten.*

**GLEW - `http://glew.sourceforge.net/`**

*Die OpenGL Extension Wrangler Library für eine einfachere Handhabung von OpenGL-Erweiterungen.*

**GLM - `https://glm.g-truc.net/0.9.9/index.html`**

*OpenGL Mathematics (GLM) ist eine rein Header basierte C++-Mathematikbibliothek für Grafiksoftware, basierend auf den Spezifikationen der OpenGL Shading Language (GLSL).*

## 4.3 Literaturverzeichnis

[1] David Plotzki. *Komplexe Zahlen*. https://www.informatik.uni-leipzig.de/~meiler/Schuelerseiten.dir/DPlotzki/html/complex.htm. (Online; Stand 21. Dezember 2023). 1999.

[2] David Plotzki. *Mandelbrotmenge*. https://www.informatik.uni-leipzig.de/~meiler/Schuelerseiten.dir/DPlotzki/html/mndlbrt.htm. (Online; Stand 21. Dezember 2023). 1999.

[3] Inigo Quilez. *computer graphics, mathematics, shaders, fractals, demoscene and more*. https://iquilezles.org/articles/palettes/. (Online; Stand 21. Dezember 2023).

[4] Andreas Schneider. *Komplexe Zahlen | Mathebibel*. https://www.mathebibel.de/komplexe-zahlen. (Online; Stand 21. Dezember 2023).

[5] Wikipedia. *Fraktal - Wikipedia*. https://de.wikipedia.org/w/index.php?title=Fraktal&oldid=238550386. (Online; Stand 21. Dezember 2023). 2023.

[6] Wikipedia. *Julia-Menge – Wikipedia*. https://de.wikipedia.org/wiki/Julia-Menge. (Online; Stand 21. Dezember 2023). 2023.

[7] www.mathematik.ch. *Juliamengen und Mandelbrotmenge*. https://www.mathematik.ch/anwendungenmath/fractal/julia/. (Online; Stand 21. Dezember 2023). 1997.

# Selbstständigkeitserklärung

Ich versichere, dass ich die Komplexe Leistung ohne fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe. Alle wörtlich oder sinngemäß übernommenen Textstellen habe ich als solche kenntlich gemacht.

Schwarzenberg, den 21. Dezember 2023

Kevin Hoang