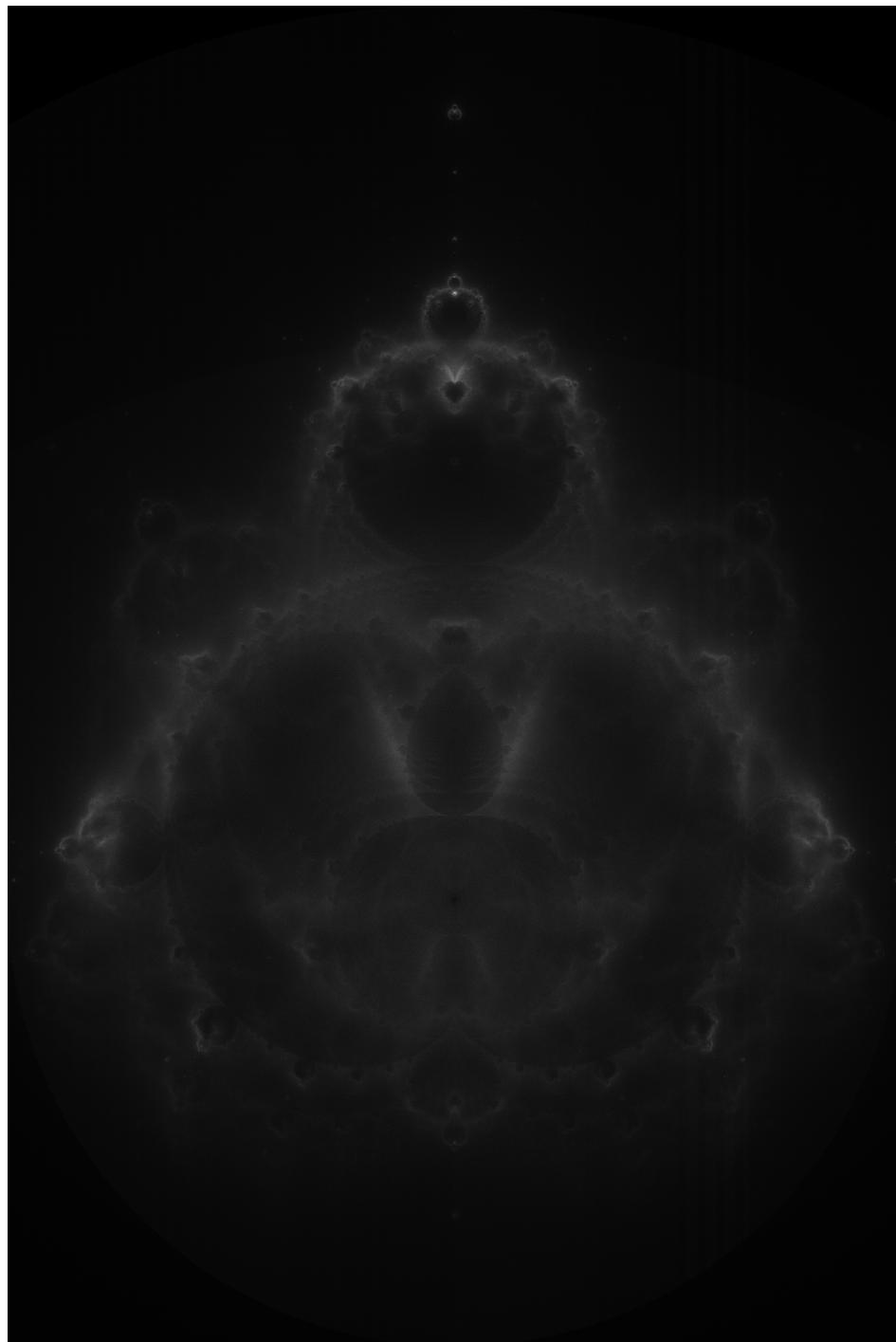


# Zoom in das Buddhabrot

Ambros D. Anrig  
6. Dezember 2021



Maturaarbeit  
Kantonsschule Glarus  
Betreuer: Fabio Thöny  
Referent: Beat Temperli

# Vorwort

Das Unendliche zieht die Menschen schon seit eh und je an, so auch mich. Die Unendlichkeit ist unbeschreiblich, doch spielen und rechnen wir gerne mit ihr. Da die Technologie immer besser und leistungsfähiger wird, ist es möglich, sich der Unendlichkeit immer schneller und näher zu approximieren. Schon seit meiner Jugend beeindrucken mich komplexere mathematische Vorgänge, weshalb ich mir in meiner Freizeit gerne YouTube-Videos über solche ansehe. Eines Tages entdeckte ich das Buddhabrot auf dem Titelbild eines Videos. Als ich mir dieses dann anschaut, verstand ich aufgrund der englischen Sprache nichts davon. So beschäftigte ich mich vorerst nur mit der Mandelbrot-Menge. Erst als ich in der Schule die Fraktale kennlernte, verstand ich die Thematik etwas besser. So entschloss ich mich, das Verstehen des Buddhabrotes nochmals zu versuchen und mich mit dem diesem auseinanderzusetzen. Aus diesem Grund befasse ich mich in dieser Arbeit mit dieser Thematik.

Vorab möchte ich einigen Personen meinen Dank aussprechen. In erster Linie möchte ich mich bei meiner Betreuungsperson Fabio Thöny, der mir bei jeglichen Fragen zur Verfügung stand und mich stets unterstützte, bedanken. Ebenfalls ist meiner Mutter, meiner Schwester und dem Freund meiner Schwester zu danken, die meine Arbeit gegenlasen und korrigierten. Auch Julian Steiner danke ich, der mir beim Programmieren mental zur Seite stand und mir bei allfälligen Problemen wie auch bei den Formulierungen in der Arbeit geholfen hat. Ebenfalls geht mein Dank an Linus Romer, der mir bei der CUDA-Implementierung half. Auch danke ich David Forchhammer, der mir bei den Formulierungen half.

Falls Sie diese Arbeit und die aus ihr resultierenden Bilder samt Codes interessieren, besuchen Sie den unten angefügten Link:

[https://github.com/SchwammPizza/Maturaarbeit\\_Ambros](https://github.com/SchwammPizza/Maturaarbeit_Ambros)

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Präliminarien</b>	<b>2</b>
2.1 Komplexe Zahlen . . . . .	2
2.2 Fraktale Geometrie . . . . .	2
2.2.1 Allgemein . . . . .	2
2.2.2 Mandelbrotmenge . . . . .	3
2.2.3 Buddhabrot . . . . .	4
<b>3 Methode</b>	<b>5</b>
3.1 Allgemeines Rechnen . . . . .	5
3.2 Erstellen eines ersten Zooms . . . . .	5
3.3 CUDA-Optimierung . . . . .	5
3.4 Analyse . . . . .	5
3.5 Implementierung der Ergebnisse . . . . .	6
<b>4 Ergebnisse</b>	<b>7</b>
4.1 Zahlen ohne Optimierung . . . . .	7
4.2 Analyse der Ergebnisse . . . . .	7
4.3 Implementierungszahlen . . . . .	8
<b>5 Diskussion</b>	<b>9</b>
<b>6 Fazit</b>	<b>10</b>
<b>7 Selbständigkeitserklärung</b>	<b>11</b>
<b>8 Quellenverzeichnis</b>	<b>12</b>
<b>9 Anhang</b>	<b>13</b>

# 1 Einleitung

An der Kantonsschule Glarus werden im Schwerpunkt fach Anwendung der Mathematik und Physik unter anderem die Julia-Mengen und die Mandelbrotmenge thematisiert. Auch das Buddhabrot wird in diesem Zusammenhang erwähnt, wobei auch ein Zoom, welcher in die Mandelbrotmenge hineinfokussiert, angeschaut wird. Es sind jedoch wenige Zooms ins Buddhabrot zu finden, geschweige denn solche, welche gleich tief in das Buddhabrot gehen, wie die beim Mandelbrot.

Ziel dieser Arbeit ist es, einen Zoom in das Buddhabrot zu berechnen. Es wird nach einer möglichst effizienten Methode für die Berechnung gesucht. Dies soll mit rein mathematischen Algorithmen bewerkstelligt werden, jedoch werden schon zum Anfang leichte Hilfen von Programmiertricks benutzt. Diese Arbeit wird mit der Programmiersprache Julia erstellt, einer schnellen und verständlichen Sprache. Es wird jedoch kein Video erstellt, sondern ein Bild, da ein Video eine rasche Abfolge von Bildern ist. Findet man also eine Methode für das Bild, ist der Schritt zum Video bereits erleichtert. Jedoch würde dieser zusätzliche Schritt den Rahmen dieser Arbeit strapazieren.

## 2 Präliminarien

### 2.1 Komplexe Zahlen

Wird mit reellen Zahlen gerechnet, bekommt man Probleme, wenn die Wurzel aus einer negativen Zahl gezogen wird. Jedoch fanden Mathematiker bereits im 17. Jahrhundert eine Lösung für dieses Problem, indem dieses Zahlensystem mit den imaginären Zahlen, die die imaginäre Einheit  $i$  besitzen, erweitert wurde (Helmuth Gericke 1970, S.66). Addieren und Subtrahieren zweier imaginären Zahlen funktioniert genau gleich, wie wenn mit einer reellen Zahl gerechnet wird. Dies heisst, dass das  $i$  wie die 'herkömmlichen' Variablen behandelt werden kann. Jedoch muss man beim Multiplizieren, Dividieren und beim somit entstehenden Rechnen mit Potenzen aufpassen, denn es gilt für  $n \in \mathbb{Z}$ :

$$\begin{aligned} i^{4n} &= 1 \\ i^{4n+1} &= i \\ i^{4n+2} &= -1 \\ i^{4n+3} &= -i \end{aligned}$$

Potenzen mit der Basis  $i$  sollten nach den oben genannten Regeln vereinfacht werden. Hierbei ist gut ersichtlich, dass eine Verknüpfung zwischen den imaginären und reellen Zahlen in die andere Richtung ebenfalls existiert. Wenn man nun eine imaginäre Zahl  $ib$  mit einer reellen Zahl  $a$  addiert, bekommt man eine komplexe Zahl  $c = a + ib$  mit dem Realteil  $a$  und dem Imaginärteil  $b$ .  $a$  und  $b$  sind hier reelle Zahlen. Beim Addieren von komplexen Zahlen, wie  $z = a + ib$  und  $w = e + if$ , folgt man diesem Beispiel:

$$\begin{aligned} z + w \\ a + ib + e + if \\ a + e + (b + f)i \end{aligned}$$

Nun kann man feststellen, dass eine komplexe Zahl mit einem Vektor vergleichbar ist. Um die Zahl darstellen zu können, benutzt man nämlich die 2-dimensionale komplexe Ebene (Bertram Maurer 2015, S. 144). Daraus schliesst man, dass komplexe Zahlen 2-dimensional sind. Wird eine komplexe Zahl multipliziert und dies auf der komplexen Ebene beobachtet, fängt der Punkt an scheinbar unkontrolliert herumzuspringen. Der Punkt folgt jedoch weiterhin logischen Regeln. Beim Quadrieren verschiebt sich der Punkt in die positive Drehrichtung (Gegenuhrzeigersinn).

Ebenfalls kann, da die Zahl vergleichbar mit einem Vektor ist, der absolute Betrag der komplexen Zahl  $c$  bestimmt werden, welcher mit dem Pythagoras berechnet wird (Reinhart Behr 1989, S. 22):

$$|c| = |a + ib| = \sqrt[2]{a^2 + b^2}$$

### 2.2 Fraktale Geometrie

#### 2.2.1 Allgemein

Um zum Buddhabrot zu kommen, müssen wir noch einen weiteren Begriff klären: das Fraktal. Fehlt bei einem  $3n$  grossen Strich das mittlere Drittel und stehen an dieser Stelle die anderen zwei Seiten eines gleichseitigen Dreiecks mit der Seitenlänge  $n$ , so hat man die erste Iteration einer Kochkurve. Fügt man nun in die einzelnen  $n$  grossen Striche die vorige Iteration der Kochkurve ein, entsteht die zweite Iteration. Dies kann nun immer wiederholt werden, sodass ein immer detaillierteres und komplizierteres Bild entsteht.

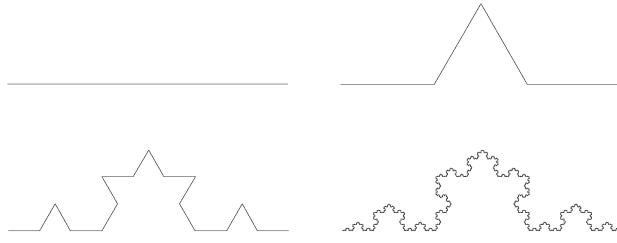


Abbildung 1: Die Entstehung der Kochkurve

Wenn man in die Kochkurve hineinzoomt, findet man die Kochkurve immer wieder: ein rekursives Bild oder eben ein Fraktal (Prof. Dr. Guido Walz 2001, S. 128).

Man definiert nun das Fraktal als eine Figur, bei der sehr oft Selbstähnlichkeit auffindbar ist, das heisst, dass das gesamte Fraktal oder Teile davon mehrfach im Fraktal vorkommen und das Fraktal selbst eine gebrochene und somit keine ganzzahlige Dimension besitzt (Bertram Maurer 2015, S. 258).

## 2.2.2 Mandelbrotmenge

Die nach dem Mathematiker Benoît B. Mandelbrot (\*20.11.1924; †14.10.2010) benannte Menge ( $\mathbb{M}$ ) beinhaltet jede Zahl  $c$ , die nicht gegen  $\infty$  für die rekursive Folge divergiert (Reinhart Behr 1989, S. 54):

$$\begin{aligned} z_0 &= 0 \\ z_{n+1} &= z_n^2 + c \end{aligned}$$

Man fand heraus, dass wenn  $|z_n| > 2$  gilt, wird die Folge gegen  $\infty$  divergieren (Reinhart Behr 1989, S. 74).

Die erstellte Abbildung ergibt ein sehr schönes Gebilde, dem auch Farbe dazugegeben werden kann. Personen des deutschen Sprachraums sahen aufgrund seiner Form ein 'Apfelmännchen' und benannten diese Abbildung danach (Reinhart Behr 1989, S. 54).

Die  $\mathbb{M}$  ist ein Fraktal (Bertram Maurer 2015, S. 258). Man findet das erst gesehene Bild der Menge beim Hineinzoomen immer wieder. Somit ist es ebenfalls selbstähnlich. Immer wieder findet man im Mandelbrot verschiedene Julia-Mengen mit dem zugehörigen  $c$  (Reinhart Behr 1989, S. 54). Diese sind ebenfalls Fraktale und selbstähnlich.  $\mathbb{M}$  besitzt durch die vorgegebene Formel ein chaotisches System (Reinhart Behr 1989, S. 32).

All diese Faktoren führen sicherlich dazu, dass es einige YouTube-Videos gibt, die einen Zoom in die Menge zeigen, welche zur Herstellung auch viel Rechenleistung benötigen.

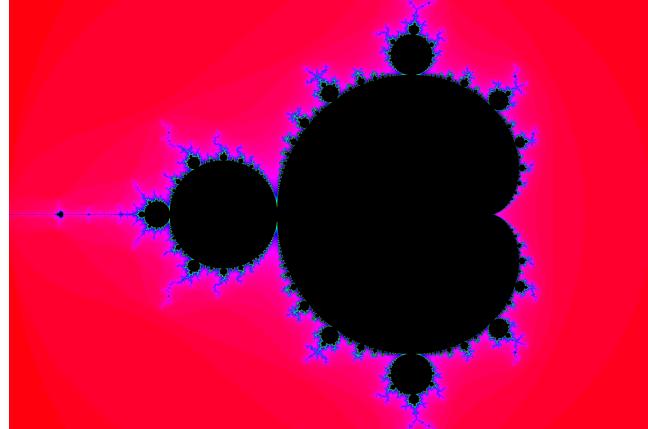


Abbildung 2: Das Mandelbrot, die  $\mathbb{M}$  wird hier schwarz dargestellt

### 2.2.3 Buddhabrot

Wenn man sich diese Abbildung als Erstes anschaut, ist der meditierende Buddha darin ersichtlich. Daher auch der Name 'Buddhabrot'. Das 'Brot' ist eine Andeutung, dass diese Abbildung etwas mit dem Mandelbrot zu tun hat, denn es stellt eine andere Variante dar, die  $\mathbb{M}$  abzubilden.

Das Bild zum Buddhabrot entsteht, indem das Mandelbrot nochmals berechnet wird, allerdings nur die Punkte, die bei der Mandelbrotberechnung gegen das  $\infty$  divergieren. Nun wird auch nicht mehr geschaut, nach wie vielen Schritten der Punkt  $c$  ins  $\infty$  abdriftet, sondern bei welchen Punkten  $c$  nach jeder Iteration landet.

Zudem wird ein Zoom in das Buddhabrot durch das chaotische System von  $\mathbb{M}$  und durch die fraktalen und selbstähnlichen Eigenschaften von  $\mathbb{M}$  interessant (Melinda Green 2017).

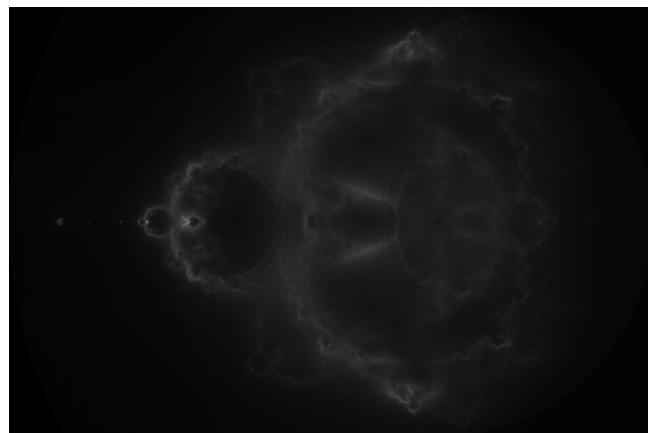


Abbildung 3: Das Buddhabrot

## 3 Methode

### 3.1 Allgemeines Rechnen

Um Bilder vom Buddhabrot zu generieren, wurde mit der Programmiersprache Julia eine zweidimensionale Matrix erstellt, bei der jeder Wert der Matrix zu einem Pixel zugeordnet wurde. Zuerst mussten die Punkte, welche man iterieren sollte, jedoch bekannt sein. So wurde zuerst das Mandelbrot ausgerechnet. Um Speicherplatz zu sparen, ordnete man ihnen die Werte 0 und 1 zu: 1 divergiert gegen  $\infty$ , 0 gehört der Mandelbrotmenge an. Alle Punkte, die gegen  $\infty$  divergieren, wurden nochmals iteriert. Anschliessend wurde geschaut, wo die Punkte durchgingen. Am Ende wurde ermittelt, welcher Punkt die meisten Treffer bekam, denn dieser Wert war nötig, um die Graustufung zu machen. Daher musste man dreimal die riesige Matrix durchgehen und auch zweimal iterieren. Dies war somit im Vergleich zur Mandelbrotmenge sehr rechenaufwändig, bei der nur ein Durchlauf nötig gewesen wäre. Als Definitionsbereich wurde  $\{z \in \mathbb{C} \mid -2 < \Re(z) < 1 \& -1 < \Im(z) < 1\}$  gewählt.

### 3.2 Erstellen eines ersten Zooms

Beim ersten Ansatz wurde für den Zoom eine riesige Matrix erstellt, in die anschliessend mit all den darin enthaltenen nötigen Werten gezoomt wurde. Die Grösse der Matrix war die Zoomtiefe im Quadrat, wie das vom erwarteten Bild. Dann wurde ein Ausschnitt gewählt und liess diesen zeichnen. Dies mithilfe der Berechnung vom Offset im Array, welcher durch die Angabe, zu welchem Punkt man zoomen möchte, berechnet wurde.

### 3.3 CUDA-Optimierung

Um lange Wartezeiten zu vermeiden, wurde der Code mit CUDA formuliert. Es wurde CUDA.jl hinzugefügt, um so mehrere Punkte gleichzeitig rechnen zu lassen. Dadurch, dass nun der grösste Teil des Codes auf der Grafikkarte (GPU) gerechnet wurde, war weniger Speicherplatz für die Berechnung verfügbar. Drei Funktionen wurden hierbei auf der GPU gerechnet: die Berechnung der Punkte die nach  $\infty$  divergierten, die Berechnung des Buddhabrotes und das Zeichnen des Ausschnitts. Für dies mussten die Funktionen umgeschrieben werden, damit CUDA diese nutzen konnte. Zu erwähnen ist, dass CUDA eine Plattform ist um auf der Grafikkarte zu rechnen. CUDA wird in Julia durch CUDA.jl genutzt. CUDA ist von Nvidia, was dazu führt, dass das Programm nur noch auf Computern laufen kann, die eine Grafikkarte von Nvidia haben.

### 3.4 Analyse

Da das Buddhabrot als chaotisches System gilt, wurde versucht, im Chaos ein Muster zu finden. Eingangs wurde geschaut, ob ein gegebener Bereich gegenüber anderen Bereichen einen überwiegenden Einfluss ausübt oder sehr geringen Einfluss hat. So könnte man in einem Zoom nur noch diesen Bereich anschauen oder vernachlässigen. Dies wurde einfach bewerkstelligt, indem man Bilder erstellte, die zeigten, wie sich Punkte aus diesen Bereichen iterierten. Zuerst wurden die 4 Quadranten als Startbereiche gewählt.

Anschliessend wurde noch zusätzlich die Überlegung gemacht, dass der absolute Wert von  $c$  ebenfalls einen Einfluss haben könnte, wie wenn er kleiner als 1 wäre. Dies wurde getestet, indem man eine Variable dem vorigen Aufbau mitgab, welcher mit einem XOR dafür sorgte, dass entweder der Bereich, bei dem  $|c| \geq 1$  gilt, oder der andere ausgewertet wurde.

### 3.5 Implementierung der Ergebnisse

Um Zeit zu sparen, wurde der zu verwerfende Bereich schon in der Mandelbrotberechnung verworfen. Bei den anderen nützlichen Ergebnissen (vgl. Kapitel Ergebnisse) wurde geschaut, in welchen Bereichen es auf den Analysebildern schwarz war oder nur ein Treffer gezeigt wurde. Danach wurde diese Fläche zu einer Funktion umgewandelt. Mit dieser wurde geschaut, ob der massgebende Eckpunkt des Zoombereichs in der Fläche war. Wenn es so war, wurde deren Quadrant nicht miteinbezogen. So musste man je nachdem nur noch 2 Quadranten für ein Bild berechnen.

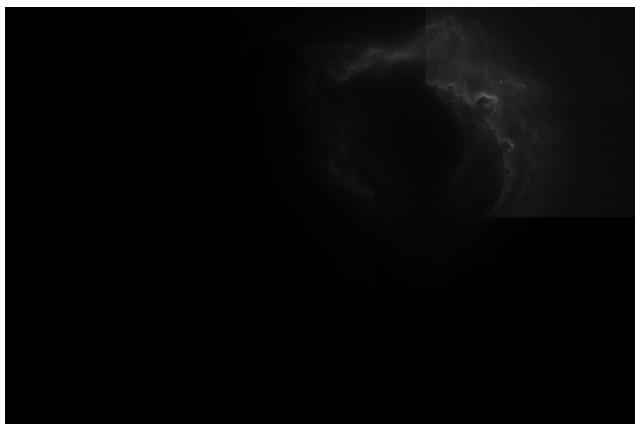
## 4 Ergebnisse

### 4.1 Zahlen ohne Optimierung

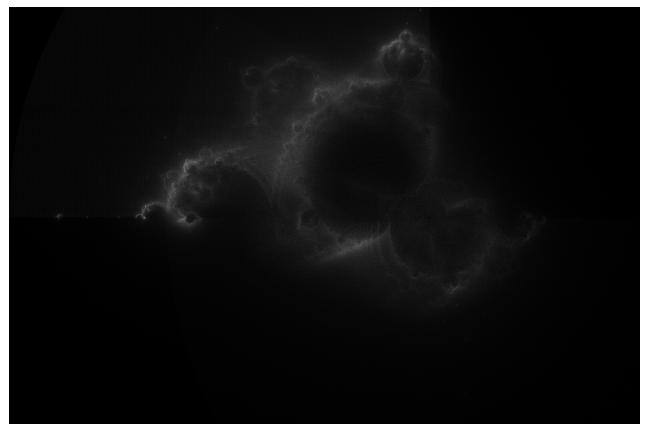
Beim ersten Zoom ist ein 16-facher Zoom nicht mehr möglich, da die Matrix so gross wird, dass der Computer nicht genügend RAM freiräumen kann. Ausserdem dauert es dann schnell mal eine lange Zeit. Einen 12-fachen Zoom mit der Auflösung 4'001 auf 2'667 Pixel zum Punkt -1.25 und mit 150 Iterationen zu berechnen, dauert etwa 45.6 Stunden.

Durch die CUDA-Optimierung kann man nur noch einen 6.25-fachen Zoom machen, dies liegt daran, dass die GPU nur noch 8GB VRAM und der PC 16GB RAM haben. Bei gleicher Einstellung, bis auf die Zoomtiefe von 6.25, konnte das Programm innerhalb von 2 Minuten und 34 Sekunden fertig rechnen. Dass ein Zoom nicht gleich tief möglich ist, ist eigentlich irrelevant, denn es geht ja darum, die Rechenleistung zu verringern und einen allfälligen Algorithmus zu finden. Mit demselben Programm sowie einem 1-fachen Zoom dauert die Berechnung nun nur noch 49 Sekunden statt drei Stunden.

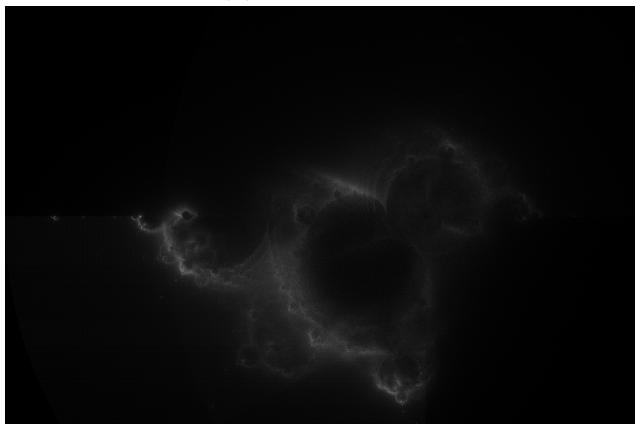
### 4.2 Analyse der Ergebnisse



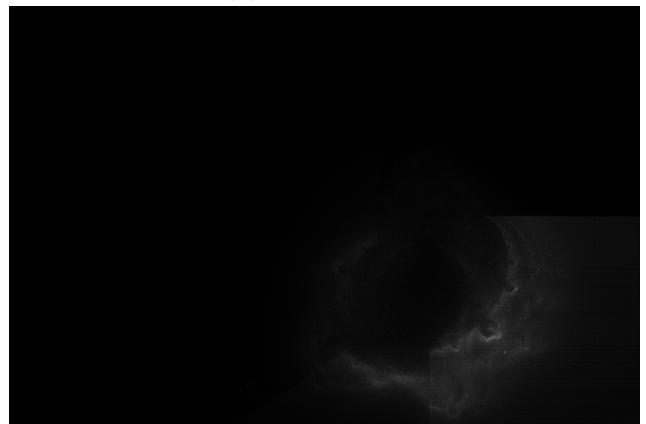
(a) 1. Quadrant



(b) 2. Quadrant



(c) 3. Quadrant



(d) 4. Quadrant

Abbildung 4: Das Buddhabrot der einzelnen Quadranten

Bei Abbildung 5 ist der auf einem Pixel maximal erreichte Wert 4.

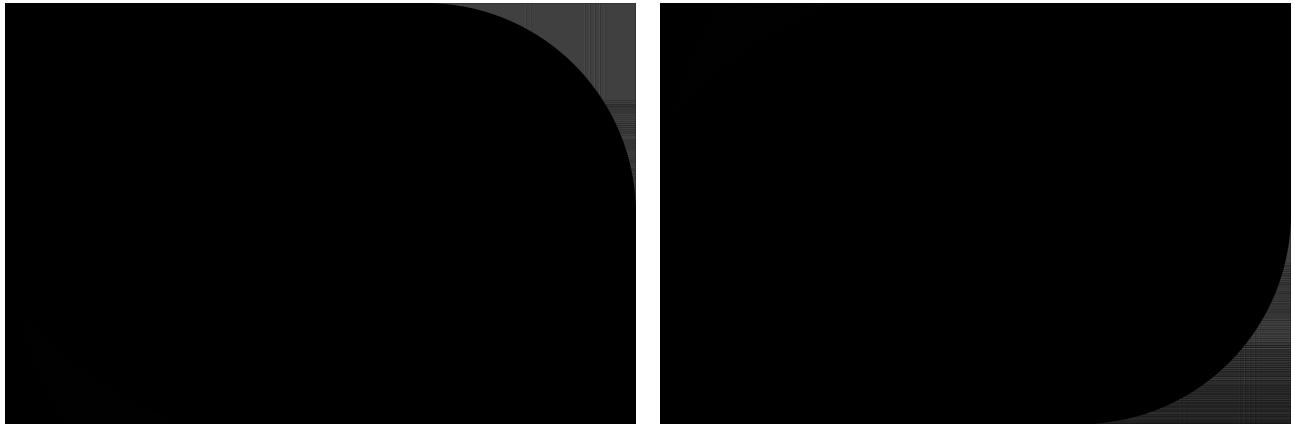


Abbildung 5: Das Buddhabrot für  $\{z \in \mathbb{C} \mid |z| > 1\}$  in den Quadranten 1 & 4

### 4.3 Implementierungszahlen

Durch die gefundene Lösung ist ein 6.48-facher Zoom möglich. Bei einem wie zuvor erreichten Zoom von 6.25 braucht es nun mehr Zeit. Dies liegt daran, dass nun grosse Matrizen in einer Liste zu finden sind und so der Aufruf durch mehrere if-Konditionen länger dauert. Dies stellt jedoch einen lediglich geringen Verlust an Effizienz dar und ist somit irrelevant. Ein 6.48-facher Zoom zum Punkt -1.25 mit der Auflösung 4'002 auf 2'668 Pixel und mit 150 Iterationen dauert 3 Minuten und 53 Sekunden. Bei einem 6.25-fachem Zoom benötigt man bei dieser Lösung 3 Minuten und 41 Sekunden.

## 5 Diskussion

Bei den Analysen der Quadranten hat sich gezeigt, dass die Quadranten auf bestimmte Bereiche keinen, hingegen auf gewisse Bereiche einen starken Einfluss ausüben. Die Quadranten hatten auf folgende Bereiche im Definitionsbereich keinen Einfluss:

### 1. Quadrant

$$\{z \in \mathbb{C} \mid \Im(z) > 57.4 \frac{\Re(z)+2}{11.4} - \frac{29.8}{3.8}\}$$

### 2. Quadrant

$$\{z \in \mathbb{C} \mid ((\Re(z) - \frac{3'504}{667})^2 + \Im(z)^2 > 6.25 \text{ \& } \Im(z) < -\frac{25}{667}) \vee (\Im(z) > (\Re(z) - 1)^2)\}$$

### 3. Quadrant

$$\{z \in \mathbb{C} \mid ((\Re(z) - \frac{3'504}{667})^2 + \Im(z)^2 > 6.25 \text{ \& } \Im(z) > \frac{25}{667}) \vee (\Im(z) < -(\Re(z) - 1)^2)\}$$

### 4. Quadrant

$$\{z \in \mathbb{C} \mid \Im(z) < -57.4 \frac{\Re(z)+2}{12} + \frac{24.5}{4}\}$$

Bei den durchgeführten Analysen mit den Radien hat sich gezeigt, dass vom Bereich  $\{z \in \mathbb{C} \mid |z| > 1 \text{ \& } 1 \geq \Re(z) \geq -1 \leq \Im(z) \leq 1\}$  nur ein maximaler Treffer von 4 erreicht wird. Somit kann dieser Bereich in der Berechnung verworfen werden, da 4 in Relation zu den maximal erreichten Treffern von 69 vernachlässigbar ist und es unter anderem beim Anblick des Buddhabrotes nicht gut erkenntlich ist.

## 6 Fazit

Eine Steigerung vom ersten Ansatz bis zur letzten optimierten Fassung ist klar ersichtlich, abgesehen davon, dass das Bild nun am hellsten ist. Zwar wurde ein Punkt gewählt, bei dem sich die letzte Variante gelohnt hat, wobei auch die Punkte in dieser Umgebung sehr spannend sind. Hätte man einen anderen Punkt ausgewählt, wäre keine klare Verbesserung ersichtlich, womöglich denn sogar nicht mal vorhanden gewesen. Ebenfalls ist das letzte Programm nicht gleich effizient wie das zweite, da es nun mehr if-Konditionen hat, welche so das Programm verlangsamen.

Es gibt einiges, was man hätte probieren können, um einen tieferen Zoom zu ermöglichen. Beispielsweise die Auflösung des Bildes auf 1080p zu verringern. Heutzutage ist ein 4K-Bild jedoch üblicher und die Bilder werden heller, da mehr Punkte iteriert werden. Ein weiteres Beispiel wäre eine Datenbank, welche während des Rechnens erstellt würde, so dass die Threads bei alten Resultaten weiter rechnen könnten. Das Programm wäre zwar wiederum langsamer, da nun mehr Aufrufe ausserhalb der CUDA geschehen. Eine weitere Methode wäre, den Metropolis-Hashings Algorithmus von Alexander Boswell zu nutzen. Dies wurde nicht gemacht, da die Wahrscheinlichkeit, nicht alle Punkte miteinzubeziehen, vorhanden ist. Hier wird nämlich die Wahrscheinlichkeit vom Divergieren eines Punktes berechnet. Dass alle Punkte miteinbezogen werden, ist durch den Verwerfungsbereich in der letzten Variante dieser Arbeit ebenfalls nicht gegeben, jedoch war der maximale Treffer 4 bei verschiedenen grossen Iterationsstufen vernachlässigbar. Ebenfalls sind hier die Punkte bestimmt nicht vorhanden und nicht von einer Wahrscheinlichkeit abhängig.

## 7 Selbständigkeitserklärung

Hiermit bestätige ich, Ambros Daniel Anrig, meine Maturaarbeit selbständig verfasst und alle Quellen angegeben zu haben.

Ich nehme zur Kenntnis, dass meine Arbeit zur Überprüfung der korrekten und vollständigen Angabe der Quellen mit Hilfe einer Software (Plagiaterkennungstool) geprüft wird. Zu meinem eigenen Schutz wird die Software auch dazu verwendet, später eingereichte Arbeiten mit meiner Arbeit elektronisch zu vergleichen und damit Abschriften und eine Verletzung meines Urheberrechts zu verhindern. Falls Verdacht besteht, dass mein Urheberrecht verletzt wurde, erkläre ich mich damit einverstanden, dass die Schulleitung meine Arbeit zu Prüfzwecken herausgibt.

Ort

Datum

Unterschrift

## 8 Quellenverzeichnis

### Literaturverzeichnis

- [1] Reinhart Behr, *Ein Weg zur fraktalen Geometrie*, Ernst Klett Schulbuchverlag, Stuttgart, 1989.
- [2] Helmuth Gericke, *Geschichte des Zahlenbegriffs*, Bibliographisches Institut, Mannheim, 1970.
- [3] Melinda Green, *The Buddhabrot Technique* (2017), <https://superliminal.com/fractals/bbrot/>. [Online; Stand 29.11.2021].
- [4] Bertram Maurer, *Mathematik - Die faszinierende Welt der Zahlen*, Fackelträger Verlag GmbH, Köln, Emil-Hoffmann-Strasse 1, D-50996 Köln, 2015.
- [5] Prof. Dr. Gudio Walz (ed.), *Lexikon der Mathematik*, Spektrum Akademischer Verlag GmbH Heidelberg, Berlin & Heidelberg, 2001.

### Abbildungsverzeichnis

1	Kochkurve (eigenes Bild)	3
2	Mandelbrot (eigenes Bild)	4
3	Das Buddhabrot (eigenes Bild)	4
4	Das Buddhabrot der Quadranten (eigenes Bild)	7
5	Der Verwerfungsbereich (eigenes Bild)	8
6	Erstes Bild im Anhang (eigenes Bild)	22
7	Zeites Bild im Anhang (eigenes Bild)	22
8	Drittes Bild im Anhang (eigenes Bild)	23
9	Viertes Bild im Anhang (eigenes Bild)	23

9 Anhang

## **1. Fassung des Programmes**

```

1 # Dies ist der erst Versuch der Arbeit
2
3 using Images, Colors
4
5 # variabeln deffinieren
6 @time begin
7     #Varierende variabeln
8     const n = Int(2668)
9     const m = Int(floor(n/2*3))
10
11    const iteration = 1000
12
13    const zoom = 12 #zoom != 0
14    const zoomPoint = -1.25 + 0im
15
16    #Berechnete variabeln
17    const zoomPointAsMatrixPoint = ((-imag(zoomPoint) + 1)*n*zoom/2 + 1, (
18        real(zoomPoint) + 2)*m*zoom/3 + 1)
19    println(zoomPointAsMatrixPoint)
20
21    # verschiebung des Bildes im gesamt array
22    const horizontal = Int(floor(zoomPointAsMatrixPoint[1] - n/2))      #
23    zuerst die auf der Komplexenebene rechtere
24    const vertical = Int(floor(zoomPointAsMatrixPoint[2] - m/2))      #
25    zuerst die auf der Komplexenebene hoechere
26    println(string(horizontal) * " " * string(vertical))
27    maxLanding = 0
28
29    # erstellen der array
30    img = zeros(RGB{Float64}, n, m)
31    maxValues = zeros(Int128, n*zoom, m*zoom)
32
33    mandelbrot = zeros(Int8, n*zoom, m*zoom)
34
35    # erstellen der Funktionen
36
37    # bearbeitet das mandelbrot array so das nunroch 1 und 0 gibt, 1 fuer
38    # drausen und 0 fuer in der Menge
39    function mandelbrotmenge()
40        for i = 1:(n*zoom)
41            for j = 1:(m*zoom)
42                y = (2*i - n)/n * 1im # definitionsbereich = [-1im, 1im]
43                x = (3*j - 2*m)/m # definitionsbereich = [-2, 1]
44                z = x + y
45                c = x + y
46                f = []
47                for _ = 1:iteration
48                    if z in f
49                        break
50                    elseif abs(z) >= 2
51                        mandelbrot[i, j] = 1
52                        break
53                    end
54                    append!(f, z)
55                    z = z^2 + c
56                end
57            end
58        end
59    end
60
61    #mandelbrotmenge()
62
```

```

52         end
53     end
54   end
55 end

56 # schaut was die Maximale Iterationzahl war, und speichert in maxValues
57 wie oft dort ein Punkt ankam
58 function berechnungBuddhaBrot(i, j)
59   global maxLanding
60
61   y = (2*i - n) / n * 1im # definitionsbereich = [-1im, 1im]
62   x = (3*j - 2*m) / m # definitionsbereich = [-2, 1]
63   z = x + y
64   c = x + y
65   f = []
66   for _ = 1:iteration
67     y = PointImagToIndex(z)
68     x = PointRealToIndex(z)
69
70     if z in f
71       break
72     elseif abs(z) >= 2
73       break
74     elseif true in (x > (m*zoom) , x < 1 , y > (n*zoom) , y < 1)
75       break
76     end
77
78     maxValues[y, x] += 1
79     if maxValues[y, x] > maxLanding
80       maxLanding = maxValues[y, x]
81     end
82     append!(f, z)
83     z = z^2 + c
84   end
85 end

86
87 function zeichnen(y, x)
88   return RGB{Float64}(maxValues[y, x]/maxLanding, maxValues[y, x]/
89 maxLanding, maxValues[y, x]/maxLanding)
90 end

91 PointImagToIndex(c) = floor(Int64, ((imag(c) + 1) * (n*zoom) / 2))
92
93 PointRealToIndex(c) = floor(Int64, ((real(c) + 2) * (m*zoom) / 3))

94
95 # Hauptprogramm
96 if (real(zoomPoint) - 3/(zoom*2) < -2) || (real(zoomPoint) + 3/(zoom*2)
97 > 1) || (imag(zoomPoint) - 1/(zoom) < -1) || (imag(zoomPoint) + 1/(zoom)
98 > 1)
99   println("Zoom auserhalb der Vordefinierte Bildreichweite")
100  exit()
101 end

102 mandelbrotmenge()

103 for i = 1:(n*zoom)
104   for j = 1:(m*zoom)
105     if mandelbrot[i, j] != 0
106       berechnungBuddhaBrot(i, j)

```

```

107         end
108     end
109   end
110   for i = 1:n
111     for j = 1:m
112       img[i, j] = zeichnen(i + horizontal - 1, j + vertical - 1)
113     end
114   end
115
116   # Bildstellung
117   save(string(@__DIR__) * "/Pictures/BuddhabrotmengeWithZoom$(zoom)
118   ToPoint$(zoomPoint)WithIteration$(iteration)withResolution$(m)x$(n).png",
119   img)
120 end

```

## 2. Fassung des Programmes, mit CUDA optimierte

```

1 # Dies ist die mit CUDA optimierte Fassung, somit die 2.
2
3 using Images, Colors, CUDA
4
5 @time begin
6     #Varierende variabeln
7     const n = Int(2668)
8     const m = Int(floor(n/2*3))
9
10    const iteration = 150
11    const anzahlThreads = 256
12
13    const zoomPoint = -1.25 + 0im
14    const zoom = 6.25
15
16    #Berechnete variabeln
17    const zoomPointAsMatrixPoint = ((-imag(zoomPoint) + 1)*n*zoom/2 + 1, (
18        real(zoomPoint) + 2)*m*zoom/3 + 1)
19    println(zoomPointAsMatrixPoint)
20
21    # verschiebung des Bildes im gesamt array
22    const horizontal = Int(floor(zoomPointAsMatrixPoint[1] - n/2))           #
23    zuerst die auf der Komplexenebene rechtere
24    const vertical = Int(floor(zoomPointAsMatrixPoint[2] - m/2))           #
25    zuerst die auf der Komplexenebene hoechere
26    println(string(horizontal) * " " * string(vertical))
27
28    # erstellen der array
29    mandelbrot = CUDA.zeros(Int8, round(Int, n*zoom), round(Int, m*zoom))
30
31    # erstellen der Funktionen
32
33    # bearbeitet das mandelbrot array so das nunroch 1 und 0 gibt, 1 fuer
34    # drausen und 0 fuer in der Menge
35    function mandelbrotmenge!(nzoom::Int64, mzoom::Int64, n::Int64, m::Int64
36    , iteration::Int64, mandelbrot, f::CuDeviceVector{ComplexF64, 1})
37        indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
38        strideX = blockDim().x * gridDim().x
39        indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y
40        strideY = blockDim().y * gridDim().y
41
42        for i = indexX:strideX:nzoom
43            for j = indexY:strideY:mzoom
44                mandelbrot[i + j*gridDim().x] = 0
45            end
46        end
47    end
48
49    mandelbrotmenge!(nzoom, mzoom, n, m, iteration, mandelbrot, f)
50
51    # Speichern des Mandelbrots
52    save("mandelbrot.png", mandelbrot)
53
```

```

38     for j = indexY:strideY:mzoom
39         y = (2*i - n) / n * 1im # definitionbereich = [-1im, 1im]
40         x = (3*j - 2*m) / m # definitionbereich = [-2, 1]
41         z = x + y
42         c = x + y
43
44         for r = 1:iteration
45             if z in f
46                 break
47             elseif abs(z) >= 2
48                 mandelbrot[i, j] += 1
49                 break
50             end
51             f[r] = z
52             z = z^2 + c
53         end
54     end
55     return nothing
56 end
57
58
59 function bench_mandel!(nzoom::Int64, mzoom::Int64, n::Int64, m::Int64,
60 iteration::Int64, mandelbrot)
61     f = CUDA.zeros(ComplexF64, iteration)
62     f .= 3
63     numblocks = ceil(Int, length(mandelbrot)/anzahlThreads)
64     CUDA.@sync begin
65         @cuda threads=anzahlThreads blocks=numblocks mandelbrotmengen!(
66         nzoom, mzoom, n, m, iteration, mandelbrot, f)
67     end
68 end
69
70 # schaut was die Maximale Iterationzahl war, und speichert in maxValues
71 wie oft dort ein Punkt ankam
72 function berechnungBuddhaBrot!(nzoom::Int64, mzoom::Int64, n::Int64, m::Int64,
73 iteration::Int64, maxValues, mandelbrot, f::CuDeviceVector{ComplexF64, 1})
74     indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
75     strideX = blockDim().x * gridDim().x
76     indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y
77     strideY = blockDim().y * gridDim().y
78
79     for i = indexX:strideX:nzoom
80         for j = indexY:strideY:mzoom
81             if mandelbrot[i, j] != 0
82                 y = (2*i - n) / n * 1im # definitionbereich = [-1im, 1im]
83                 x = (3*j - 2*m) / m # definitionbereich = [-2, 1]
84                 z = x + y
85                 if !((abs(z) > 1) & (x >= 0))
86                     c = x + y
87                     for r = 1:iteration
88                         y = CUDA.floor(Int64, (imag(z)+1)*nzoom/2)
89                         x = CUDA.floor(Int64, (real(z)+2)*mzoom/3)
90
91                         if z in f
92                             break
93                         elseif abs(z) >= 2
94                             break
95                         end
96                     end
97                 end
98             end
99         end
100    end
101    return maxValues
102 end

```

```

92             if (1 < x < mzoom) & (1 < y < nzoom)
93                 maxValues[y, x] += 1
94             end
95
96             f[r] = z
97             z = z^2 + c
98         end
99     end
100    end
101    return nothing
102 end
103
104 function bench_buddhi!(nzoom::Int64, mzoom::Int64, n::Int64, m::Int64,
105 iteration::Int64, maxValues, mandelbrot)
106     f = CUDA.zeros(ComplexF64, iteration)
107     f .= 3
108     numblocks = ceil(Int, length(mandelbrot)/anzahlThreads)
109     CUDA.@sync begin
110         @cuda threads=anzahlThreads blocks=numblocks
111         berechnungBuddhaBrot!(nzoom, mzoom, n, m, iteration, maxValues,
112         mandelbrot, f)
113     end
114 end
115
116 function zeichnen(n::Int64, m::Int64, horizontal::Int64, vertical::Int64
117 , maxValues, img, maxLanding::Int128)
118     indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
119     strideX = blockDim().x * gridDim().x
120     indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y
121     strideY = blockDim().y * gridDim().y
122
123     for i = indexX:strideX:n
124         for j = indexY:strideY:m
125             img[i, j] = RGB{Float64}(maxValues[i+horizontal-1, j+
126 vertical-1]/maxLanding, maxValues[i+horizontal-1, j+vertical-1]/
127 maxLanding, maxValues[i+horizontal-1, j+vertical-1]/maxLanding)
128     end
129     end
130 end
131
132 function bench_zeich!(n::Int64, m::Int64, horizontal::Int64, vertical::
133 Int64, maxValues, img, maxLanding::Int128)
134     numblocks = ceil(Int, length(maxValues)/anzahlThreads)
135     CUDA.@sync begin
136         @cuda threads=anzahlThreads blocks=numblocks zeichnen(n, m,
137         horizontal, vertical, maxValues, img, maxLanding)
138     end
139 end
140
141 # Hauptprogramm
142 if (real(zoomPoint) - 3/(zoom*2) < -2) || (real(zoomPoint) + 3/(zoom*2)
143 > 1) || (imag(zoomPoint) - 1/(zoom) < -1) || (imag(zoomPoint) + 1/(zoom)
144 > 1)
145     println("Zoom auserhalb der Vordefinierte Bildreichweite")
146     exit()
147 end

```

```

141  bench_mandel!(round(Int64, n*zoom), round(Int64, m*zoom), n, m,
142    iteration, mandelbrot)
143    maxValues = CUDA.zeros(Int128, round(Int,n*zoom), round(Int,m*zoom))
144    bench_buddhi!(round(Int64, n*zoom), round(Int64, m*zoom), n, m,
145      iteration, maxValues, mandelbrot)
146      mandelbrot = nothing
147
148      img = CUDA.zeros(RGB{Float64}, n, m)
149      println(maximum(maxValues))
150      bench_zeich!(n, m, horizontal, vertical, maxValues, img, maximum(
151        maxValues))
152
153      # Bildstellung
154      img_cpu = zeros(RGB{Float64}, n, m)
155      img_cpu .= img
156      save(string(@__DIR__) * "/Pictures/gpu/BuddhabrotmengeWithZoomGPU$(zoom)
157      ToPoint$(zoomPoint)WithIteration$(iteration)withResolution$(m)x$(n)high.
158      png", img_cpu)
159 end

```

## Endfassung des Programmes, mit CUDA und eigen Optimierung

```

1 # Dies ist die letzte und somit Optimierte Fassung der Arbeit
2
3 using Images, Colors, CUDA
4
5 @time begin
6   #Varierende variablen
7   const n = Int(2668) # muss eine Gerade Zahl sein
8   const m = Int(floor(n/2*3))
9
10  const iteration = 150
11  const anzahlThreads = 256
12
13  const zoomPoint = -1.25 + 0im
14  const zoom = 6.25 #zoom != 0
15
16  #Berechnete variablen
17  zoomPointAsMatrixPoint = ((-imag(zoomPoint) + 1)*n*zoom/2 + 1, (real(
18    zoomPoint) + 2)*m*zoom/3 + 1)
19
20  # verschiebung des Bildes im gesamt array
21  const horizontal = floor(Int, zoomPointAsMatrixPoint[1] - n/2)          #
22  zuerst die auf der Komplexenebene rechtere
23  const horizontal2 = floor(Int, zoomPointAsMatrixPoint[1] + n/2)
24  const vertical = floor(Int, zoomPointAsMatrixPoint[2] - m/2)             #
25  zuerst die auf der Komplexenebene hoechere
26  const vertical2 = floor(Int, zoomPointAsMatrixPoint[2] + m/2)
27  zoomPointAsMatrixPoint = nothing
28
29  # erstellen der Funktionen
30
31  # bearbeitet das mandelbrot array so das nunroch 1 und 0 gibt, 1 fuer
32  drausen und 0 fuer in der Menge
33  function mandelbrotberechnung!(nn::Int64, mm::Int64, iteration::Int64,
34    mandelbrotPart, f::CuDeviceVector{ComplexF64}, 1}, r::Int8)
35    indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
36    strideX = blockDim().x * gridDim().x
37    indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y

```

```

33     strideY = blockDim().y * gridDim().y
34
35     for i = indexX:strideX:nn
36         for j = indexY:strideY:mm
37             y = floor((r-1)/2)*-1 + i/nn # definitionsbereich = [-1im, 1
im]
38             x = 2*abs(r-2.5)-3 + 2*j/mm # definitionsbereich = [-2, 1]
39             z = x + y*1im
40             if !((abs(z) > 1) & (x >= 0))
41                 c = x + y*1im
42
43                 for q = 1:iteration
44                     if z in f
45                         break
46                     elseif abs(z) >= 2
47                         mandelbrotPart[i, j] += 1
48                         break
49                     end
50                     f[q] = z
51                     z = z^2 + c
52                 end
53             end
54         end
55     end
56     return nothing
57 end
58
59 function bench_mandel!(iteration::Int64, mandelbrotPart, r::Int8)
60     f = CUDA.zeros(ComplexF64, iteration)
61     f .= 3
62
63     nn = CUDA.size(mandelbrotPart, 1)
64     mm = CUDA.size(mandelbrotPart, 2)
65
66     numblocks = ceil(Int, length(mandelbrotPart)/anzahlThreads)
67
68     CUDA.@sync begin
69         @cuda threads=anzahlThreads blocks=numblocks
70         mandelbrotberechnung!(nn, mm, iteration, mandelbrotPart, f, r)
71     end
72 end
73
74 # schaut was die Maximale Iterationzahl war, und speichert in maxValues
wie oft dort ein Punkt ankam
75 function berechnungBuddhaBrot!(nzoom::Int64, mzoom::Int64, nn::Int64, mm
::Int64, iteration::Int64, maxValues, mandelbrot, f::CuDeviceVector{ComplexF64, 1}, r::Int8)
76     indexY = (blockIdx().x - 1) * blockDim().x + threadIdx().x
77     strideY = blockDim().x * gridDim().x
78     indexX = (blockIdx().y - 1) * blockDim().y + threadIdx().y
79     strideX = blockDim().y * gridDim().y
80
81     for i = indexY:strideY:nn
82         for j = indexX:strideX:mm
83             if mandelbrot[i, j] != 0
84                 y = floor((r-1)/2)*-1 + i/nn # definitionsbereich = [-1im
, 1im]
85                 x = 2*abs(r-2.5)-3 + 2*j/mm # definitionsbereich = [-2,
1]

```

```

85     z = x + y*1im
86     c = x + y*1im
87     for q = 1:iteration
88         y = CUDA.floor(Int64, (imag(z)+1)*(nzoom-1)/2+1)
89         x = CUDA.floor(Int64, (real(z)+2)*(mzoom-1)/3+1)
90
91         if z in f # Kontrolle
92             break
93         elseif abs(z) >= 2
94             break
95         end
96         if (1 <= x <= mzoom) & (1 <= y <= nzoom)
97             maxValues[y, x] += 1
98         end
99
100        f[q] = z
101        z = z^2 + c
102    end
103    end
104 end
105
106 return nothing
107 end
108
109 function bench_buddhi!(iteration::Int64, maxValues, mandelbrot, r::Int8)
110     f = CUDA.zeros(ComplexF64, iteration)
111     f .= 3
112
113     nn = CUDA.size(mandelbrot,1)
114     mm = CUDA.size(mandelbrot,2)
115     nzoom = CUDA.size(maxValues,1)
116     mzoom = CUDA.size(maxValues,2)
117
118     numblocks = ceil(Int, length(mandelbrot)/anzahlThreads)
119     CUDA.@sync begin
120         @cuda threads=anzahlThreads blocks=numblocks
121         berechnungBuddhaBrot!(nzoom, mzoom, nn, mm, iteration, maxValues,
122         mandelbrot, f, r)
123     end
124 end
125
126 function zeichnen(n::Int64, m::Int64, horizontal::Int64, vertical::Int64
127 , Values, img, maxLanding::Int128)
128     indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
129     strideX = blockDim().x * gridDim().x
130     indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y
131     strideY = blockDim().y * gridDim().y
132
133     for i = indexX:strideX:n
134         for j = indexY:strideY:m
135             img[i, j] = RGB{Float64}(Values[i+horizontal-1, j+vertical-1]/maxLanding, Values[i+horizontal-1, j+vertical-1]/maxLanding, Values[i+horizontal-1, j+vertical-1]/maxLanding)
136         end
137     end
138 end
139
140 function bench_zeich!(horizontal::Int64, vertical::Int64, Values, img,
141 maxLanding::Int128)

```

```

138     n = CUDA.size(img, 1)
139     m = CUDA.size(img, 2)
140
141     numblocks = ceil(Int, length(Values)/anzahlThreads)
142     CUDA.@sync begin
143         @cuda threads=anzahlThreads blocks=numblocks zeichnen(n, m,
144         horizontal, vertical, Values, img, maxLanding)
145     end
146 end
147
148 # mandelbrot
149 mandelbrot = CUDA.Array([CUDA.ones(Int8, 2, 2) for _ in 1:4])
150 maxValues = CUDA.zeros(Int128, floor(Int, n*zoom), floor(Int, m*zoom))
151
152 # Kontrolle ob Zoom vereinfachung moeglich
153 if zoom > 2
154     if (-horizontal2 > (28.7*n)*vertical2/(4*m) - m*9.5/4) #4. Quadrant
155         mandelbrot[1] = zeros(Int8, round(Int, n/2*zoom), round(Int, m*
156         zoom/3))
157         end
158         if (((vertical2-4340)^2+(horizontal2-n/2)^2>(2.5/3*m)^2) &
159             (horizontal2 < n/2-50)) || (horizontal > 9 * n/(2*m^2)*(vertical-m)^2+n/2)
160         ) #3. Quadrant
161         mandelbrot[2] = zeros(Int8, round(Int, n/2*zoom), round(Int, 2*m
162         *zoom/3))
163         end
164         if (((vertical2-4340)^2+(horizontal-n/2)^2>(2.5/3*m)^2) &
165             (horizontal > n/2+50)) || (horizontal2 < -9 * n/(2*m^2)*(vertical-m)^2+n/2)
166         ) #2. Quadrant
167         mandelbrot[3] = zeros(Int8, round(Int, n/2*zoom), round(Int, 2*m
168         *zoom/3))
169         end
170         if (horizontal > (28.7*n)*vertical2/(3.8*m) - n*13/3.8) #1. Quadrant
171             mandelbrot[4] = zeros(Int8, round(Int, n/2*zoom), round(Int, m*
172             zoom/3))
173             end
174         else
175             for i = 1:2
176                 mandelbrot[i^2] = zeros(Int8, round(Int, n/2*zoom), round(Int, m
177                 *zoom/3))
178                 mandelbrot[i+1] = zeros(Int8, round(Int, n/2*zoom), round(Int,
179                 2*m*zoom/3))
180             end
181         end
182         println("Startet MainProgramm")
183         # Hauptprogramm
184         for r::Int8 = 1:4
185             if CUDA.size(mandelbrot[r]) != CUDA.size(CUDA.zeros(Int8, 2, 2))
186                 bench_mandel!(iteration, mandelbrot[r], r)
187                 bench_buddhi!(iteration, maxValues, mandelbrot[r], r)
188             end
189         end
190
191         # loeschen und neuerstellen von Arrays aufgrund Speicher-Handling
192         mandelbrot = nothing
193         img = CUDA.zeros(RGB{Float64}, n, m)
194         println("Startet Drawing")
195         bench_zeich!(horizontal, vertical, maxValues, img, maximum(maxValues))
196         println(maximum(maxValues))

```

```
186  
187 maxValues = nothing  
188 img_cpu = zeros(RGB{Float64}, n, m)  
189 img_cpu .= img  
190 save(string(@_DIR_) * "/Pictures/gpu/analyse/  
AnalyBuddhabrotmengeWithZoomGPU$(zoom)ToPoint$(zoomPoint)WithIteration$(  
iteration)withResolution$(m)x$(n).png", img_cpu)  
191 end
```

Bilder auf die sich der Text Bezieht



Abbildung 6: Zoompunkt=-1.25, Zoom=12, Iteration=150, Zoomversion 1

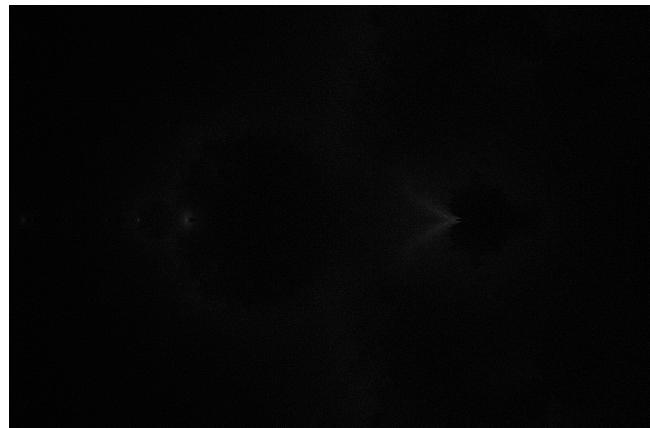


Abbildung 7: Zoompunkt=-1.25, Zoom=6.25, Iteration=150, Zoomversion 2

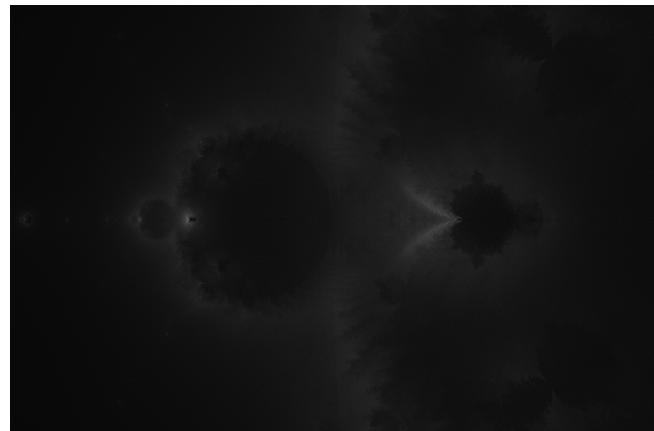


Abbildung 8: Zoompunkt=-1.25, Zoom=6.25, Iteration=150, Zoomversion 3

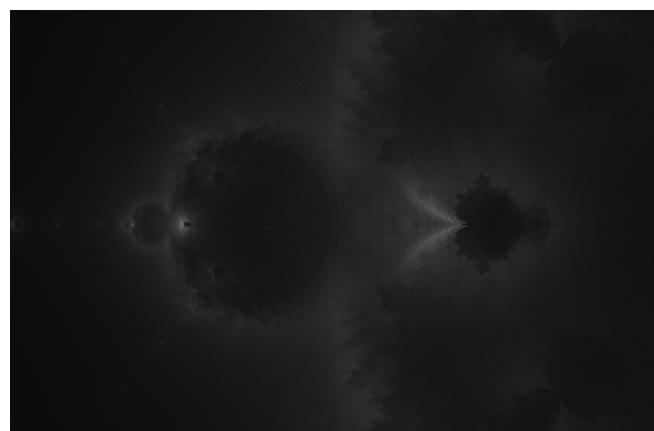


Abbildung 9: Zoompunkt=-1.25, Zoom=6.48, Iteration=150, Zoomversion 3