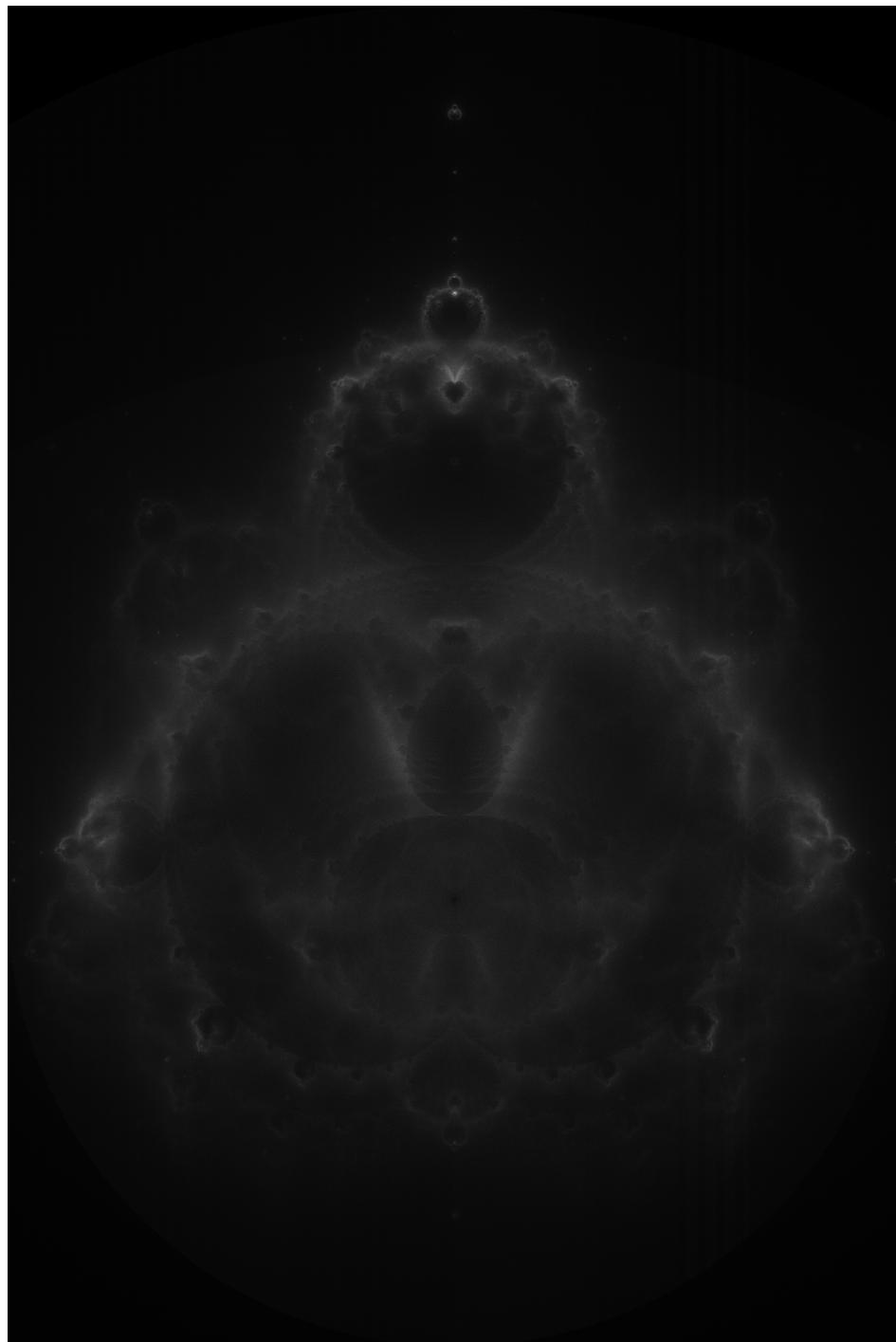


Zoom in das Buddhabrot

Ambros D. Anrig
5. 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 Video. 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 bei meiner Betreuungsperson Fabio Thöny, der mir bei jeglichen Fragen zur Verfügung stand und mich stets unterstützte. Ebenfalls ist meiner Mutter, meiner Schwester und dem Freund meiner Schwester zu danken, die meine Arbeit gegenlassen 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.

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

Inhaltsverzeichnis

1 Einleitung	4
2 Präliminarien	5
2.1 Komplexe Zahlen	5
2.2 Fraktale Geometrie	5
2.2.1 Allgemein	5
2.2.2 Mandelbrotmenge	6
2.2.3 Buddhabrot	7
3 Methode	8
3.1 Allgemeines Rechnen	8
3.2 Erstellen eines ersten Zooms	8
3.3 CUDA-Optimierung	8
3.4 Analyse	8
3.5 Implementierung der Ergebnisse	9
4 Ergebnisse	10
4.1 Zahlen ohne Optimierung	10
4.2 Analyse der Ergebnisse	10
4.3 Implementierungszahlen	11
5 Diskussion	12
6 Fazit	13
7 Selbständigkeitserklärung	14
8 Quellenverzeichnis	15
8.1 Literaturverzeichnis	15
8.2 Abbildungsverzeichnis	15
9 Anhang	16

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 einen 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.

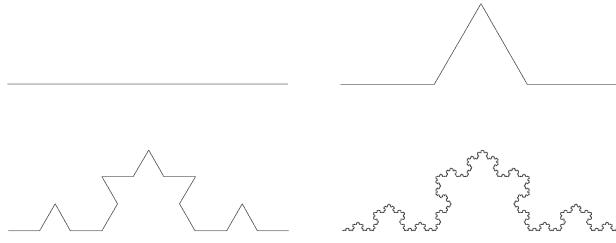


Abbildung 1: Die Entstehung der Kochkurve

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 ∞ divergiert für die rekursive Folge (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 ∞ 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.

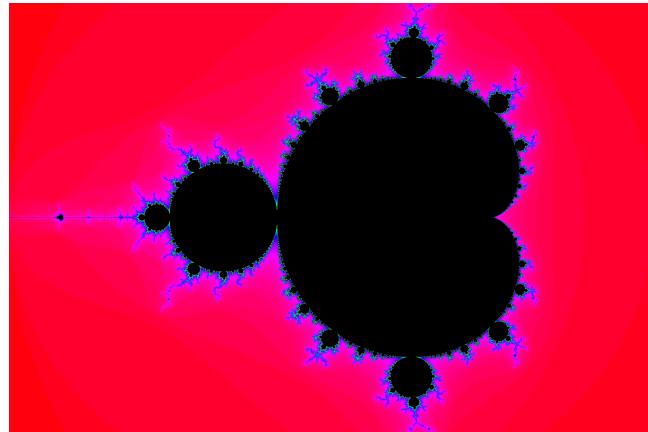


Abbildung 2: Das Mandelbrot, die \mathbb{M} wird hier schwarz dargestellt

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 ∞ divergieren. Nun wird auch nicht mehr geschaut, nach wie vielen Schritten der Punkt c ins ∞ 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).

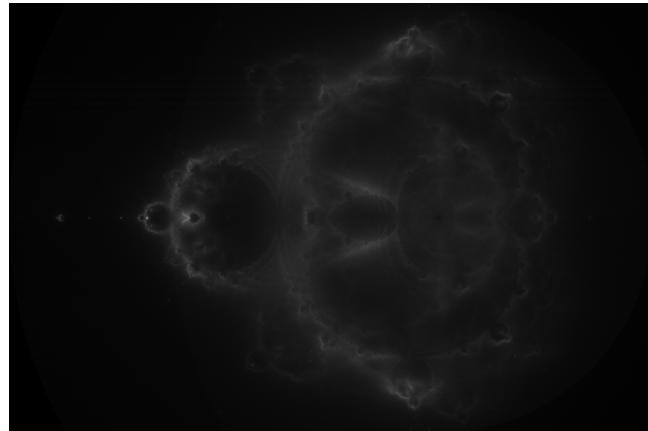


Abbildung 3: Das Buddhabrot

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 ∞ , 0 gehört der Mandelbrotmenge an. Alle Punkte, die gegen ∞ 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 enthaltenden nötigen Werten, gezoomt wurde. Die Grösse der Matrix war die Zoomtiefe im Quadrat, wie das vom erwarteten Bild. Dann wurde ein Ausschnitt gewählt und diesen zeichnen gelassen. 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 ∞ 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 wird 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.

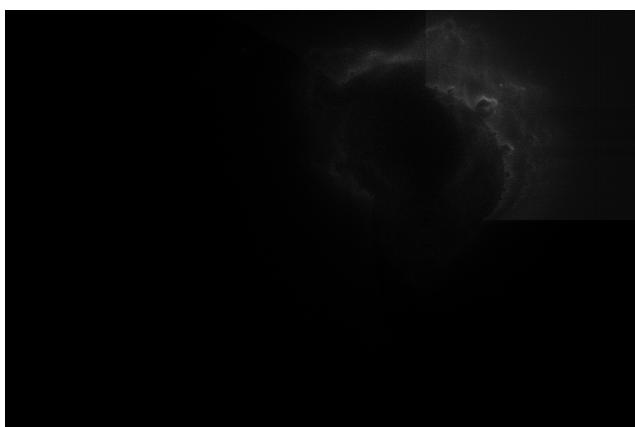
4 Ergebnisse

4.1 Zahlen ohne Optimierung

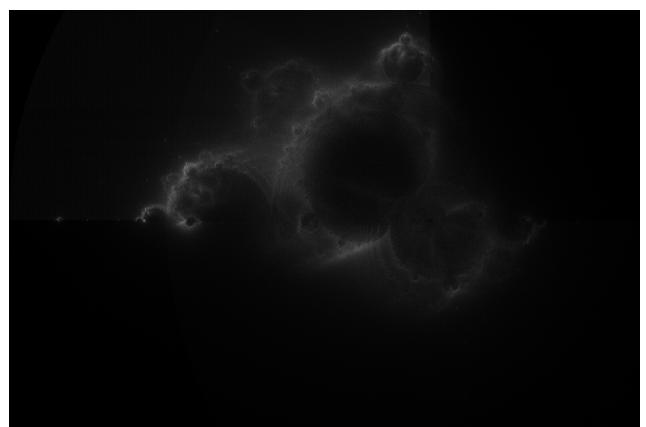
Beim ersten Zoom ist ein 16-facher Zoom nicht mehr möglich, da die Matrix so gross wird, dass der Computer nicht genügend RAM freiräumen kann. Ausserdem dauert es dann schnell mal eine lange Zeit. Einen 12-fachen Zoom mit der Auflösung 4'001 auf 2'667 Pixel zum Punkt -1.25 und mit 150 Iterationen dauert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

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.

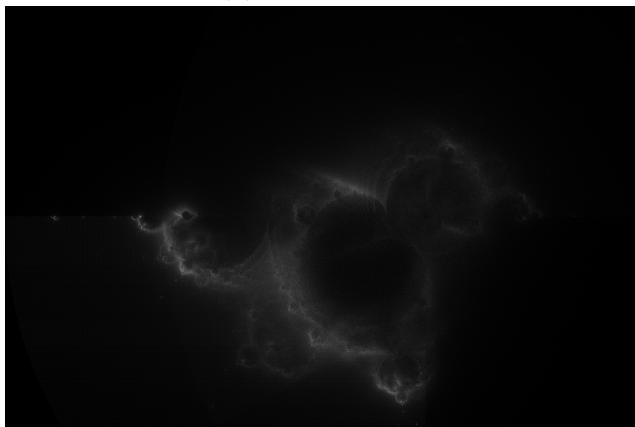
4.2 Analyse der Ergebnisse



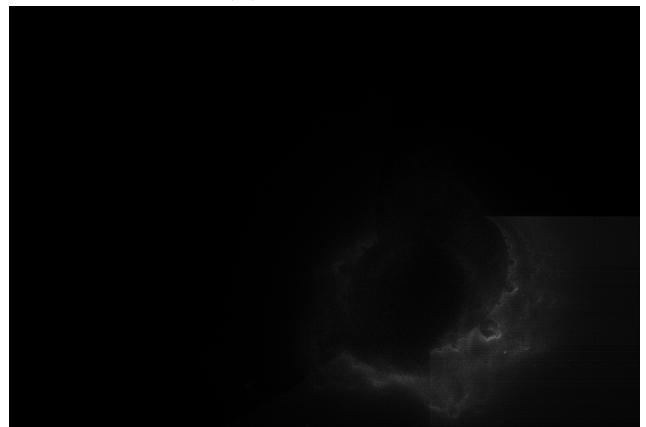
(a) 1. Quadrant



(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 vier.

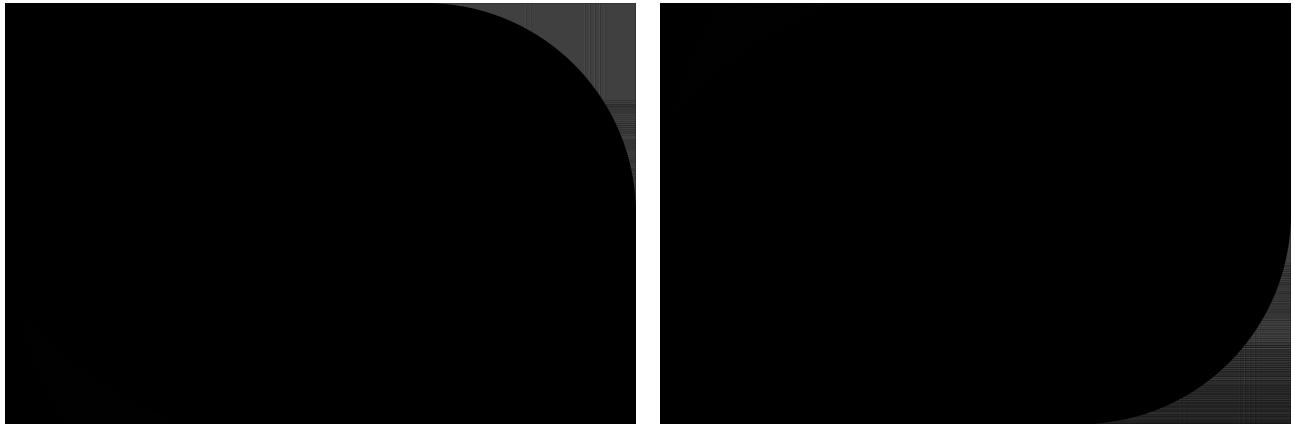


Abbildung 5: Das Buddhabrot für $\{z \in \mathbb{C} \mid |z| > 1\}$ in den Quadranten 1 & 4

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-fachen 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

8.1 Literaturverzeichnis

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

8.2 Abbildungsverzeichnis

Abbildungsverzeichnis

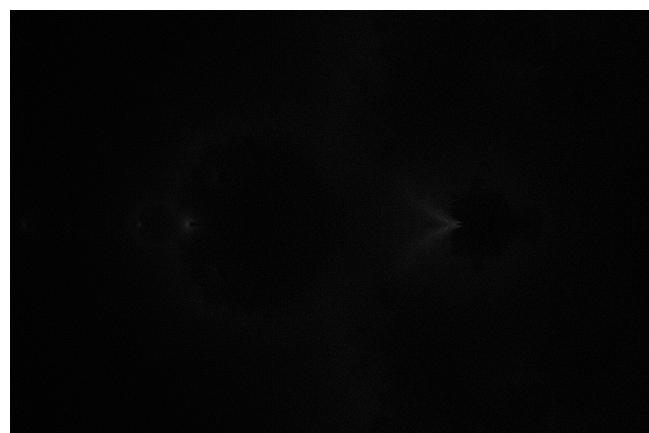
1	Die Entstehung der Kochkurve	6
2	Das Mandelbrot, die \mathbb{M} wird hier schwarz dargestellt	7
3	Das Buddhabrot	7
4	Das Buddhabrot der einzelnen Quadranten	10
5	Das Buddhabrot für $\{z \in \mathbb{C} \mid z > 1\}$ in den Quadranten 1 & 4	11

9 Anhang

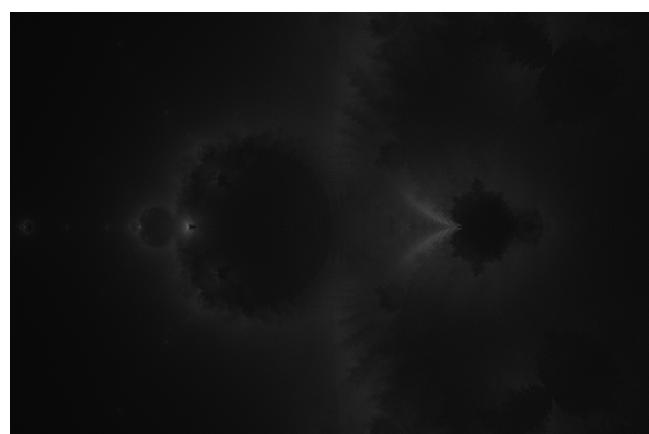
Zoompunkt=-1.25, Zoom=12, Iteration=150, Zoomversion 1



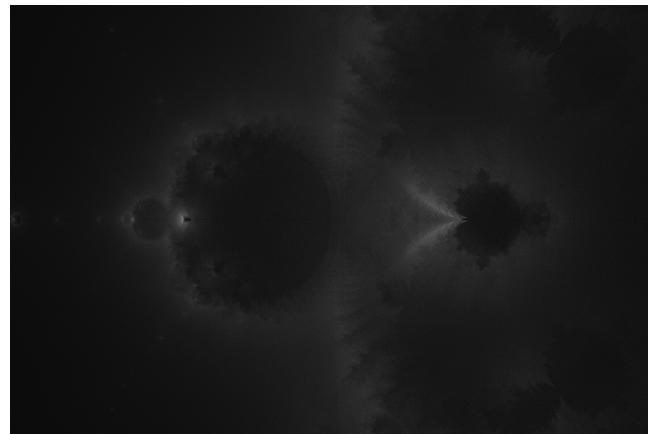
Zoompunkt=-1.25, Zoom=6.25, Iteration=150, Zoomversion 2



Zoompunkt=-1.25, Zoom=6.25, Iteration=150, Zoomversion 3



Zoompunkt=-1.25, Zoom=6.48, Iteration=150, Zoomversion 3



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 rechte
24    const vertical = Int(floor(zoomPointAsMatrixPoint[2] - m/2))           #
25    zuerst die auf der Komplexenebene hoechtere
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

```

```

35     function mandelbrotmenge()
36         for i = 1:(n*zoom)
37             for j = 1:(m*zoom)
38                 y = (2*i - n)/n * 1im # definitionsbereich = [-1im, 1im]
39                 x = (3*j - 2*m)/m # definitionsbereich = [-2, 1]
40                 z = x + y
41                 c = x + y
42                 f = []
43                 for _ = 1:iteration
44                     if z in f
45                         break
46                     elseif abs(z) >= 2
47                         mandelbrot[i, j] = 1
48                         break
49                     end
50                     append!(f, z)
51                     z = z^2 + c
52                 end
53             end
54         end
55     end
56
57     # schaut was die Maximale Iterationzahl war, und speichert in maxValues
58     # wie oft dort ein Punkt ankam
59     function berechnungBuddhaBrot(i, j)
60         global maxLanding
61
62         y = (2*i - n)/n * 1im # definitionsbereich = [-1im, 1im]
63         x = (3*j - 2*m)/m # definitionsbereich = [-2, 1]
64         z = x + y
65         c = x + y
66         f = []
67         for _ = 1:iteration
68             y = PointImagToIndex(z)
69             x = PointRealToIndex(z)
70
71             if z in f
72                 break
73             elseif abs(z) >= 2
74                 break
75             elseif true in (x > (m*zoom) , x < 1 , y > (n*zoom) , y < 1)
76                 break
77             end
78
79             maxValues[y, x] += 1
80             if maxValues[y, x] > maxLanding
81                 maxLanding = maxValues[y, x]
82             end
83             append!(f, z)
84             z = z^2 + c
85         end
86     end
87
88     function zeichnen(y, x)
89         return RGB{Float64}(maxValues[y, x]/maxLanding, maxValues[y, x]/
90         maxLanding, maxValues[y, x]/maxLanding)
91     end
92
93     PointImagToIndex(c) = floor(Int64, ((imag(c) + 1) * (n*zoom) / 2))

```

```

92 PointRealToIndex(c) = floor(Int64, ((real(c) + 2) * (m*zoom) / 3))
93
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
103 mandelbrotmenge()
104
105 for i = 1:(n*zoom)
106     for j = 1:(m*zoom)
107         if mandelbrot[i, j] != 0
108             berechnungBuddhaBrot(i, j)
109         end
110     end
111 end
112 for i = 1:n
113     for j = 1:m
114         img[i, j] = zeichnen(i + horizontal - 1, j + vertical - 1)
115     end
116 end
117
118 # Bildstellung
119 save(string(@__DIR__) * "/Pictures/BuddhabrotmengeWithZoom$(zoom)"
120      ToPoint$(zoomPoint)WithIteration$(iteration)withResolution$(m)x$(n).png",
121      img)
122 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

```

```

23    println(string(horizontal) * " " * string(vertical))
24
25    # erstellen der array
26    mandelbrot = CUDA.zeros(Int8, round(Int, n*zoom), round(Int, m*zoom))
27
28    # erstellen der Funktionen
29
30    # bearbeitet das mandelbrot array so das nunroch 1 und 0 gibt, 1 fuer
31    # drausen und 0 fuer in der Menge
32    function mandelbrotmenge!(nzoom::Int64, mzoom::Int64, n::Int64, m::Int64
33        , iteration::Int64, mandelbrot, f::CuDeviceVector{ComplexF64, 1})
34        indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
35        strideX = blockDim().x * gridDim().x
36        indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y
37        strideY = blockDim().y * gridDim().y
38
39        for i = indexX:strideX:nzoom
40            for j = indexY:strideY:mzoom
41                y = (2*i - n)/n * 1im # definitionsbereich = [-1im, 1im]
42                x = (3*j - 2*m)/m # definitionsbereich = [-2, 1]
43                z = x + y
44                c = x + y
45
46                for r = 1:iteration
47                    if z in f
48                        break
49                    elseif abs(z) >= 2
50                        mandelbrot[i, j] += 1
51                        break
52                    end
53                    f[r] = z
54                    z = z^2 + c
55                end
56            end
57        end
58        return nothing
59    end
60
61    function bench_mandel!(nzoom::Int64, mzoom::Int64, n::Int64, m::Int64,
62        iteration::Int64, mandelbrot)
63        f = CUDA.zeros(ComplexF64, iteration)
64        f .= 3
65        numblocks = ceil(Int, length(mandelbrot)/anzahlThreads)
66        CUDA.@sync begin
67            @cuda threads=anzahlThreads blocks=numblocks mandelbrotmenge!(
68                nzoom, mzoom, n, m, iteration, mandelbrot, f)
69        end
70    end
71
72    # schaut was die Maximale Iterationzahl war, und speichert in maxValues
73    # wie oft dort ein Punkt ankam
74    function berechnungBuddhaBrot!(nzoom::Int64, mzoom::Int64, n::Int64, m::
75        Int64, iteration::Int64, maxValues, mandelbrot, f::CuDeviceVector{
76        ComplexF64, 1})
77        indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
78        strideX = blockDim().x * gridDim().x
79        indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y
80        strideY = blockDim().y * gridDim().y

```

```

75     for i = indexX:strideX:nzoom
76         for j = indexY:strideY:mzoom
77             if mandelbrot[i, j] != 0
78                 y = (2*i - n) / n * 1im # definitionsbereich = [-1im, 1im]
79                 x = (3*j - 2*m) / m # definitionsbereich = [-2, 1]
80                 z = x + y
81                 if !((abs(z) > 1) & (x >= 0))
82                     c = x + y
83                     for r = 1:iteration
84                         y = CUDA.floor(Int64, (imag(z)+1)*nzoom/2)
85                         x = CUDA.floor(Int64, (real(z)+2)*mzoom/3)
86
87                         if z in f
88                             break
89                         elseif abs(z) >= 2
90                             break
91                         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        end
102    end
103    return nothing
104 end

105
106 function bench_buddhi!(nzoom::Int64, mzoom::Int64, n::Int64, m::Int64,
107 iteration::Int64, maxValues, mandelbrot)
108     f = CUDA.zeros(ComplexF64, iteration)
109     f .= 3
110     numblocks = ceil(Int, length(mandelbrot)/anzahlThreads)
111     CUDA.@sync begin
112         @cuda threads=anzahlThreads blocks=numblocks
113         berechnungBuddhaBrot!(nzoom, mzoom, n, m, iteration, maxValues,
114         mandelbrot, f)
115     end
116 end

117
118 function zeichnen(n::Int64, m::Int64, horizontal::Int64, vertical::Int64
119 , maxValues, img, maxLanding::Int128)
120     indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
121     strideX = blockDim().x * gridDim().x
122     indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y
123     strideY = blockDim().y * gridDim().y
124
125     for i = indexX:strideX:n
126         for j = indexY:strideY:m
127             img[i, j] = RGB{Float64}(maxValues[i+horizontal-1, j+
vertical-1]/maxLanding, maxValues[i+horizontal-1, j+vertical-1]/
maxLanding, maxValues[i+horizontal-1, j+vertical-1]/maxLanding)
128         end
129     end
130 end

```

```

128     function bench_zeich!(n::Int64, m::Int64, horizontal::Int64, vertical::
129     Int64, maxValues, img, maxLanding::Int128)
130         numblocks = ceil(Int, length(maxValues)/anzahlThreads)
131         CUDA.@sync begin
132             @cuda threads=anzahlThreads blocks=numblocks zeichnen(n, m,
133             horizontal, vertical, maxValues, img, maxLanding)
134         end
135     end
136
137     # Hauptprogramm
138     if (real(zoomPoint) - 3/(zoom*2) < -2) || (real(zoomPoint) + 3/(zoom*2)
139     > 1) || (imag(zoomPoint) - 1/(zoom) < -1) || (imag(zoomPoint) + 1/(zoom)
140     > 1)
141         println("Zoom auserhalb der Vordefinierte Bildreichweite")
142         exit()
143     end
144
145     bench_mandel!(round(Int64, n*zoom), round(Int64, m*zoom), n, m,
146     iteration, mandelbrot)
147     maxValues = CUDA.zeros(Int128, round(Int, n*zoom), round(Int, m*zoom))
148     bench_buddhi!(round(Int64, n*zoom), round(Int64, m*zoom), n, m,
149     iteration, maxValues, mandelbrot)
150     mandelbrot = nothing
151
152     img = CUDA.zeros(RGB{Float64}, n, m)
153     println(maximum(maxValues))
154     bench_zeich!(n, m, horizontal, vertical, maxValues, img, maximum(
155     maxValues))
156
157     # Bildstellung
158     img_cpu = zeros(RGB{Float64}, n, m)
159     img_cpu .= img
160     save(string(@__DIR__) * "/Pictures/gpu/BuddhabrotmengeWithZoomGPU$(zoom)
161     ToPoint$(zoomPoint)WithIteration$(iteration)withResolution$(m)x$(n)high.
162     png", img_cpu)
163 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 variabeln
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 variabeln
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

```

```

20     const horizontal = floor(Int, zoomPointAsMatrixPoint[1] - n/2)      #
21     zuerst die auf der Komplexenebene rechte
22     const horizontal2 = floor(Int, zoomPointAsMatrixPoint[1] + n/2)
23     const vertical = floor(Int, zoomPointAsMatrixPoint[2] - m/2)      #
24     zuerst die auf der Komplexenebene hoechere
25     const vertical2 = floor(Int, zoomPointAsMatrixPoint[2] + m/2)
26     zoomPointAsMatrixPoint = nothing
27
28     # erstellen der Funktionen
29
30     # bearbeitet das mandelbrot array so das nunroch 1 und 0 gibt, 1 fuer
31     # drausen und 0 fuer in der Menge
32     function mandelbrotberechnung!(nn::Int64, mm::Int64, iteration::Int64,
33     mandelbrotPart, f::CuDeviceVector{ComplexF64, 1}, r::Int8)
34         indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
35         strideX = blockDim().x * gridDim().x
36         indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y
37         strideY = blockDim().y * gridDim().y
38
39         for i = indexX:strideX:nn
40             for j = indexY:strideY:mm
41                 y = floor((r-1)/2)*-1 + i/nn # definitionsbereich = [-1im, 1
42                 im]
43                 x = 2*abs(r-2.5)-3 + 2*j/mm # definitionsbereich = [-2, 1]
44                 z = x + y*1im
45                 if !((abs(z) > 1) & (x >= 0))
46                     c = x + y*1im
47
48                     for q = 1:iteration
49                         if z in f
50                             break
51                         elseif abs(z) >= 2
52                             mandelbrotPart[i, j] += 1
53                             break
54                         end
55                         f[q] = z
56                         z = z^2 + c
57                     end
58                 end
59             end
60         end
61         return nothing
62     end
63
64     function bench_mandel!(iteration::Int64, mandelbrotPart, r::Int8)
65         f = CUDA.zeros(ComplexF64, iteration)
66         f .= 3
67
68         nn = CUDA.size(mandelbrotPart, 1)
69         mm = CUDA.size(mandelbrotPart, 2)
70
71         numblocks = ceil(Int, length(mandelbrotPart)/anzahlThreads)
72
73         CUDA.@sync begin
74             @cuda threads=anzahlThreads blocks=numblocks
75             mandelbrotberechnung!(nn, mm, iteration, mandelbrotPart, f, r)
76         end
77     end
78 
```

```

73     # schaut was die Maximale Iterationzahl war, und speichert in maxValues
74     wie oft dort ein Punkt ankam
75     function berechnungBuddhaBrot!(nzoom::Int64, mzoom::Int64, nn::Int64, mm
76     ::Int64, iteration::Int64, maxValues, mandelbrot, f::CuDeviceVector{
77     ComplexF64, 1}, r::Int8)
78         indexY = (blockIdx().x - 1) * blockDim().x + threadIdx().x
79         strideY = blockDim().x * gridDim().x
80         indexX = (blockIdx().y - 1) * blockDim().y + threadIdx().y
81         strideX = blockDim().y * gridDim().y
82
83         for i = indexY:strideY:nn
84             for j = indexX:strideX:mm
85                 if mandelbrot[i, j] != 0
86                     y = floor((r-1)/2)*-1 + i/nn # definitionsbereich = [-1im
87 , 1im]
88                     x = 2*abs(r-2.5)-3 + 2*j/mm # definitionsbereich = [-2,
89 1]
90
91                     z = x + y*1im
92                     c = x + y*1im
93                     for q = 1:iteration
94                         y = CUDA.floor(Int64, (imag(z)+1)*(nzoom-1)/2+1)
95                         x = CUDA.floor(Int64, (real(z)+2)*(mzoom-1)/3+1)
96
97                         if z in f # Kontrolle
98                             break
99                         elseif abs(z) >= 2
100                            break
101                        end
102                        if (1 <= x <= mzoom) & (1 <= y <= nzoom)
103                            maxValues[y, x] += 1
104                        end
105
106                     f[q] = z
107                     z = z^2 + c
108                 end
109             end
110         end
111         return nothing
112     end
113
114     function bench_buddhi!(iteration::Int64, maxValues, mandelbrot, r::Int8)
115         f = CUDA.zeros(ComplexF64, iteration)
116         f .= 3
117
118         nn = CUDA.size(mandelbrot,1)
119         mm = CUDA.size(mandelbrot,2)
120         nzoom = CUDA.size(maxValues,1)
121         mzoom = CUDA.size(maxValues,2)
122
123         numblocks = ceil(Int, length(mandelbrot)/anzahlThreads)
124         CUDA.@sync begin
125             @cuda threads=anzahlThreads blocks=numblocks
126             berechnungBuddhaBrot!(nzoom, mzoom, nn, mm, iteration, maxValues,
127             mandelbrot, f, r)
128         end
129     end
130
131     function zeichnen(n::Int64, m::Int64, horizontal::Int64, vertical::Int64

```

```

    , Values, img, maxLanding::Int128)
125   indexX = (blockIdx().x - 1) * blockDim().x + threadIdx().x
126   strideX = blockDim().x * gridDim().x
127   indexY = (blockIdx().y - 1) * blockDim().y + threadIdx().y
128   strideY = blockDim().y * gridDim().y
129
130   for i = indexX:strideX:n
131     for j = indexY:strideY:m
132       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)
133     end
134   end
135 end
136
137 function bench_zeich!(horizontal::Int64, vertical::Int64, Values, img,
138 maxLanding::Int128)
139   n = CUDA.size(img, 1)
140   m = CUDA.size(img, 2)
141
142   numblocks = ceil(Int, length(Values)/anzahlThreads)
143   CUDA.@sync begin
144     @cuda threads=anzahlThreads blocks=numblocks zeichnen(n, m,
145     horizontal, vertical, Values, img, maxLanding)
146   end
147 end
148
149 # mandelbrot
150 mandelbrot = CUDA.Array([CUDA.ones(Int8, 2, 2) for _ in 1:4])
151 maxValues = CUDA.zeros(Int128, floor(Int, n*zoom), floor(Int, m*zoom))
152
153 # Kontrolle ob Zoom vereinfachung moeglich
154 if zoom > 2
155   if (-horizontal2 > (28.7*n)*vertical2/(4*m) - m*9.5/4) #4. Quadrant
156     mandelbrot[1] = zeros(Int8, round(Int, n/2*zoom), round(Int, m*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) ) #3. Quadrant
160     mandelbrot[2] = zeros(Int8, round(Int, n/2*zoom), round(Int, 2*m*zoom/3))
161   end
162   if (((vertical2-4340)^2+(horizontal-n/2)^2>(2.5/3*m)^2) &
163     (horizontal > n/2+50)) || (horizontal2 < -9 * n/(2*m^2)*(vertical-m)^2+n/2) ) #2. Quadrant
164     mandelbrot[3] = zeros(Int8, round(Int, n/2*zoom), round(Int, 2*m*zoom/3))
165   end
166   if (horizontal > (28.7*n)*vertical2/(3.8*m) - n*13/3.8) #1. Quadrant
167     mandelbrot[4] = zeros(Int8, round(Int, n/2*zoom), round(Int, m*zoom/3))
168   end
169 else
170   for i = 1:2
171     mandelbrot[i^2] = zeros(Int8, round(Int, n/2*zoom), round(Int, m*zoom/3))
172     mandelbrot[i+1] = zeros(Int8, round(Int, n/2*zoom), round(Int, 2*m*zoom/3))
173   end

```

```

169     end
170   end
171   println("Startet MainProgramm")
172   # Hauptprogramm
173   for r::Int8 = 1:4
174     if CUDA.size(mandelbrot[r]) != CUDA.size(CUDA.zeros(Int8,2,2))
175       bench_mandel!(iteration, mandelbrot[r], r)
176       bench_buddhi!(iteration, maxValues, mandelbrot[r], r)
177     end
178   end
179
180   # loeschen und neuerstellen von Arrays aufgrund Speicher-Handling
181   mandelbrot = nothing
182   img = CUDA.zeros(RGB{Float64}, n, m)
183   println("Startet Drawing")
184   bench_zeich!(horizontal, vertical, maxValues, img, maximum(maxValues))
185   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

```