Rupprecht - Gymnasim München Abiturjahrgang 2012

S E M I N A R A R B E I T

Rahmenthema des Wissenschaftspropädeutischen Seminars:
Mathematik löst praktische Probleme des Alltags
Leitfach: Mathematik

Titel der Arbeit:

Möglichkeiten der Darstellung von Julia-Mengen und Apfelmännchen

Verfasser:
Lukas von Stumberg

Kursleiterin:
Fr. Linder

Abgabetermin: 8. November 2011

Bewertung	Note	Notenstufe in Worten	Punkte		Punkte
schriftliche Arbeit				x 3	
Abschlusspräsentation				x 1	
				Summe:	
				Gesamtleistung nach § 61 (7) GSO = Summe:2 (gerundet):	

Datum und Unterschrift der Kursleiterin bzw. des Kursleiters

Inhaltsverzeichnis

1 Einleitung

Schulfächer und die klare Trennung von Fachbereichen haben eine lange Tradition, doch heutzutage wird die Verknüpfung mehrerer Gebiete immer wichtiger. Da Programmieren seit vielen Jahren mein Hobby ist, war ich besonders an einer Verknüpfung der Mathematik mit der Informatik interessiert. Bei kaum einem anderen mathematischen Thema hat die Computertechnik zu einem so plötzlichen Fortschritt beigetragen, wie bei der Darstellung und Analyse von Julia-Mengen.

Schon 1918 beschrieben die Franzosen Gaston Julia und Pierre Fatou die Julia-Menge, doch wegen fehlender Möglichkeiten der graphischen Darstellung wurden ihre Arbeiten zunächst nicht von anderen fortgeführt. Das änderte sich erst 1979 mit Benoit Mandelbrot, dem die modernen Computer eine genauere Analyse ermöglichten. Von der Vereinfachung der Arbeit mit dem Thema abgesehen, waren die Programme zur Erstellung „schöner“ Bilder auch Voraussetzung für die Popularität von Julia-Mengen und Apfelmännchen.[1]

Mittlerweile sind zahlreiche Applikationen zur Erstellung dieser Bilder im Internet zu finden, die auch Fortgeschrittenes wie dreidimensionale Julia-Mengen ermöglichen. In dieser Seminararbeit sollen traditionelle wie neue Möglichkeiten der Darstellung von Julia-Mengen aufgezeigt werden. Um sämtliche Ideen umsetzen zu können, habe ich ein eigenes Programm zur Darstellung geschrieben, das sich auch für experimentelle Visualisierungen eignet.

Zunächst werden einige mathematische Grundlagen - nämlich komplexe Zahlen und Quaternionen - erläutert, die im nachfolgenden Teil der Arbeit benötigt werden. Im Anschluss wird die traditionelle Darstellungsweise der Julia-Mengen erklärt, bei der diese als einfache Bilder interpretiert werden. Hierbei wird neben dem Zusammenhang zwischen dem Apfelmännchen und den Julia-Mengen und deren Berechnung auch das Verfahren der Interpolation erkärt. Darüber hinaus werden Fragen nach der Performance des geschriebenen Programms und nach den Grenzen der Berechnung mit dem Computer gestellt. Nach dieser ersten Darstellungsform werden zwei Möglichkeiten thematisiert, dreidimensionale Julia-Mengen darzustellen. Als weitere Art der Visualisierung werden Filme behandelt. In diesem Teil der Arbeit wird gezeigt, wie Animationen mit dem Keyframe-Verfahren unter Verwendung der zuvor beschriebenen Interpolation flexibel erstellt werden können, was die im Anschluss beschriebenen Animationsvarianten ermöglicht. Abgerundet wird die Seminararbeit durch eine kurze Zusammenfassung der bisherigen Ergebnisse.

[1] vgl. (DRU98, S. 121)

2 Mathematische Grundlagen

2.1 Körper und Vektorräume

Zunächst sollen zwei grundsätzliche Prinzipien der Mathematik erläutert werden, nämlich Körper und Vektorräume, die im Folgenden dazu beitragen werden, die Theorie hinter den Julia-Mengen zu erklären.

Beispiele für Körper sind $\mathbb{Q}$ und $\mathbb{R}$. Ein Körper besteht prinzipiell aus einer Menge von Elementen sowie einer Definition der Addition und Multiplikation, für die die bekannten Gesetze gelten müssen (Kommutativgesetz usw.).[2]

Im Unterschied dazu hat ein Vektorraum eine Addition und eine Multiplikation mit einem Skalar, die beide komponentenweise definiert sind, hier am Beispiel des Vektorraums $\mathbb{R}^2$ veranschaulicht:[3]

$$\begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} a' \\ b' \end{pmatrix} = \begin{pmatrix} a+a' \\ b+b' \end{pmatrix} \text{ und } s \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} sa \\ sb \end{pmatrix}$$

Vektoren kann man als mehrdimensionale Zahlen betrachten, die man nur addieren, nicht aber miteinander multiplizieren kann. Der Dimension ist hierbei keine mathematische Grenze gesetzt.

Vergleicht man die Definitionen von Körper und Vektorraum stellt man fest, dass man aus einem Vektorraum einen Körper machen könnte, wenn man eine Multiplikation zweier Vektoren für ihn definiert. Dies hätte den Vorteil, dass man mit dieser „mehrdimensionalen Zahl“ so wie mit den reellen Zahlen rechnen könnte. Tatsächlich lässt sich eine Multiplikation für Vektorräume, die den Gesetzen des Körpers genügt, jedoch nicht ohne weiteres finden. Nur für zwei spezielle Vektorräume, nämlich $\mathbb{R}^2$ und (eingeschränkt) für $\mathbb{R}^4$, wurde eine gültige Multiplikationsvorschrift gefunden; das Ergebnis sind der Körper der komplexen Zahlen und der Schiefkörper der Quaternionen.[4]

2.2 Komplexe Zahlen

Der Körper der Komplexen Zahlen wurde erstmals im 16. Jahrhundert von Hieronimo Cardano bei der Suche nach der Wurzel von negativen Zahlen gefunden bzw. definiert[5]. Eine komplexe Zahl z besteht aus einem Zahlenpaar $z = (a;b)$, wobei a Realteil und b Imaginärteil genannt werden.[6] Die ebenfalls übliche Darstellung $z = a + bi$ enthält die imaginäre Einheit i, die als die *imaginäre* Wurzel aus -1 definiert ist, wobei für i die üblichen Gesetze der Mathematik gelten. Die Addition

[2] vgl. (Beu01, S. 24)
[3] vgl. (Beu01, S. 48-50)
[4] vgl. (Beu01, S. 53)
[5] vgl. (Beu01, S. 30)
[6] vgl. (Dit76, S. 22)

komplexer Zahlen funktioniert analog zu der Addition zweier Vektoren komponentenweise, was sich auch folgendermaßen ergibt: $z+z' = (a+bi)+(a'+b'i) = a+a'+bi+b'i = (a+a')+(b+b')i$. Die Multiplikation funktioniert auf folgende Weise: $z \cdot z' = (a+bi) \cdot (a'+b'i) = aa'+ab'i+a'bi+bb'i^2 = aa'-bb'+(ab'+a'b)i$.[7] Auf die Beweise, dass diese Rechenregeln dem Kommutativ-, dem Assoziativ- und dem Distributivgesetz genügen, wird hier verzichtet.

Betrachtet man die zwei Rechenregeln, stellt man fest, dass sich, wenn man den Imaginärteil einer Zahl auf 0 setzt, die komplexe Zahl genau so wie eine reelle Zahl verhält. Das heißt, dass jede reelle Zahl a gleichzeitig auch eine komplexe Zahl $z = (a;0)$ ist, wegen $(a;0)+(a';0) = (a+a';0)$ und $(a;0) \cdot (a';0) = (a \cdot a';0)$.[8]

2.3 Quaternionen

Außer $\mathbb{R}^1$ und $\mathbb{R}^2$ ist die einzige Dimension, für die eine Multiplikation gefunden wurde, die zumindest den meisten Gesetzen der reellen Zahlen genügt, $\mathbb{R}^4$. Nach einer 13 Jahre dauernden Suche fand der Mathematiker William Rowan Hamilton heraus, wie man diese heute Quaternionen genannten Quadrupel multiplizieren kann, um den Schiefkörper $\mathbb{H}$ zu erhalten. Er führte zusätzlich zu i zwei weitere imaginäre Einheiten namens j und k ein mit $i^2 = j^2 = k^2 = ijk = -1$, sodass ein Quaternion die Form $h = a+bi+cj+dk$ hat. Die Addition funktioniert genau wie bei den komplexen Zahlen: $h+h' = a+a'+(b+b')i+(c+c')j+(d+d')k$.

Wegen $ijk = -1$ und $i^2 = -1$ ergibt sich $i = jk$ und analog dazu $k = ij$ und $j = ki$. Wegen $j \cdot i = j(jk) = (j \cdot j)k = -k \neq k = ij$ kann das Kommutativgesetz bei der Multiplikation nicht gelten. Mit den Regeln für i, j und k lässt sich die Multiplikation folgendermaßen definieren: $h \cdot h' = (a+bi+cj+dk) \cdot (a'+b'i+c'j+d'k) = aa'+ab'i+ac'j+ad'k+bia'+bb'i^2+bic'j+bid'k+cja'+cjb'i+cc'j^2+cjd'k+dka'+dkb'i+dkc'j+dd'k^2 = aa'-bb'-cc'-dd'+(ab'+a'b+cd'-dc')i+(ac'-bd'+ca'+db')j+(ad'+bc'-cb'+da')k$. Die Quaternionen genügen allen Voraussetzungen des Körpers bis auf die Kommutativität der Multiplikation, weshalb sie zwar keinen Körper, aber immerhin einen Schiefkörper bilden[9].

Jede komplexe Zahl $a+bi$ kann auch als Quaternion $a+bi+0j+0k$ dargestellt werden[10] und jede reelle Zahl kann auch als komplexe Zahl dargestellt werden, das heißt $\mathbb{R} \in \mathbb{C} \in \mathbb{H}$. Daher arbeitet das vorgestellte Programm zu Julia-Mengen immer mit Quaternionen. Will man die Berechnung auf komplexe oder reelle Zahlen beschränken, so kann man die entsprechenden Teile des Quaternions auf 0 setzen.

[7] vgl. (Dit76, S. 21)
[8] vl. (Dit76, S. 23)
[9] vgl. (Beu01, S. 30-31)
[10] vgl. (Beu01, S. 33)

3 Julia-Menge und Apfelmännchen als Bild

3.1 Iteration und die Julia-Menge

Iteration ist ein wesentliches Werkzeug in der Mathematik, unter anderem zur Erzeugung von Julia-Mengen und Apfelmännchen. Der Ausgangspunkt des Iterationsverfahrens ist eine Funktion $f(x)$. Zunächst wird ein Startwert x_0 eingesetzt, woraufhin man das Ergebnis $x_1 = f(x_0)$ erhält. Der Vorgang wird wiederholt, indem nun x_1 eingesetzt wird, das heißt $x_2 = f(x_1)$. Dies wird fortgesetzt, bis die gewünschte Anzahl an Iterationen erreicht ist - im Idealfall unendlich - wobei die aufwändige Aufgabe des Rechnens bei höheren gewünschten Iterationswerten oftmals einem Computer überlassen wird. Eine häufig gebrauchte Schreibweise ist auch $x_{n+1} = f(x_n)$.[11]

Funktionen können sich bei iterierter Anwendung auf verschiedene Arten verhalten. Betrachtet man die Funktion $f(x) = \frac{1}{2}x$ lässt sich feststellen, dass der Wert mit jeder Iteration näher gegen 0 geht, die Folge *konvergiert* also unabhängig vom Startwert gegen 0. Die Funktion $f(x) = 2x$ wird dagegen für $x_0 > 0$ gegen $+\infty$ und für $x_0 < 0$ gegen $-\infty$ gehen. Manche iterierte Funktionen besitzen Fixpunkte, das heißt bei Einsetzen dieses Wertes gilt $x_{n+1} = x_n$. Beispiele sind 0 und 1 bei der Funktion $f(x) = x^2$.[12]

Auch wenn die Grenzwerte für die Iteration bei den bisher betrachteten Werten gut vorhersehbar sind, existieren Funktionen, bei denen dies nicht der Fall ist. Ein populäres Beispiel, das bei der Julia-Menge meistens Anwendung findet, ist die Funktion $z_{n+1} = z_n^2 + c$.[13] Die Parameterwerte dieser Funktion sind im komplexen Bereich, wodurch ihr Verhalten trotz ihrer Einfachheit nicht vorhersehbar ist. c stellt einen ebenfalls komplexen Parameter dar, der beliebig gewählt werden kann.

Die *ausgefüllte Julia-Menge* ergibt sich aus dieser Formel mit der Definition $J_c = \left\{ z \in \mathbb{C} \mid \lim_{n\to\infty} |z_n| \neq \infty \right\}$. Oder anders ausgedrückt: Alle z_0, bei denen der Betrag von z_n bei unendlicher Iteration nicht gegen unendlich strebt, gehören zur ausgefüllten Julia-Menge. Laut der korrekten Definition der *Julia-Menge* ist diese der Rand der *ausgefüllten Julia-Menge*, wobei oft auch die gesamte ausgefüllte Julia-Menge gemeint ist.[14]

Da der Grenzwert der vorgestellten Funktion nicht analytisch berechnet werden kann, bleibt zur Berechnung, ob eine Zahl zur (ausgefüllten) Julia-Menge gehört, nur ein numerisches Verfahren. Dieses bietet aber keine Möglichkeit bis unendlich zu rechnen, stattdessen müssen entsprechend hohe Werte gewählt werden, die

[11] vgl. (ZN94, S. 7-8)
[12] vgl. (ZN94, S. 8)
[13] (BD89, S. 118)
[14] vgl. (ZN94, S. 198)

als unendlich betrachtet werden. Diese veränderte Definition der Julia-Menge sieht folgendermaßen aus: $J_c = \{z \in \mathbb{C} \mid |z_N| < A\}$, wobei N die maximale Anzahl an Iterationen und A der Wert ist, bei dessen Überschreitung die Folge als divergierend angesehen wird.[15] Der Escape Time Algorithmus benutzt diese Definition der Julia-Menge, um die Julia-Menge zu berechnen und darzustellen. Seine Funktionsweise wird im folgenden Unterkapitel erklärt.

3.2 Von der Formel zum Bild - Der Escape Time Algorithmus

Mit Julia-Mengen werden im Allgemeinen farbige, „schöne“ Bilder in Verbindung gebracht. Mit dem Fluchtzeitalgorithmus werden diese Bilder aus einfachen Formeln wie $z_{n+1} = z_n^2 + c$ berechnet. Zunächst wird jedem Pixel des zu erstellenden Bildes eine Koordinate in Form einer komplexen Zahl zugeordnet (Abbildung 1 Schritt 1). Dazu benötigt man Informationen über die Größe des Bildes sowie über den anzuzeigenden Ausschnitt der Julia-Menge in komplexen Koordinaten. In Abbildung 1 ist das zu erzeugende Bild 4×2 Pixel groß und der anzuzeigende Ausschnitt erstreckt sich von $0+0i$ bis $0,3+0,1i$. Für einen Pixel mit den Koordinaten $a,b \in \mathbb{N}$ (startend bei 0) eines $w \times h$ großem Bilds, das den Ausschnitt von $\rho \in \mathbb{C}$ bis $\zeta \in \mathbb{C}$ anzeigen soll, lässt sich der entsprechende Startwert wie folgt berechnen: $z = \frac{a}{w-1} \cdot \left(Re_\zeta - Re_\rho\right) + \frac{b}{h-1} \cdot \left(Im_\zeta - Im_\rho\right) \cdot i + \rho$. Diese Formel entspricht der Newton-Interpolation[16], man kann sie jedoch ohnehin leicht nachvollziehen.

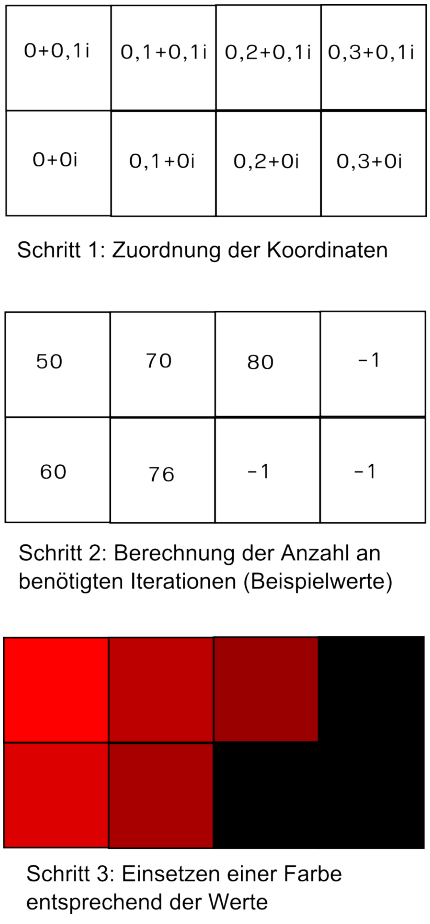


Abbildung 1:
Der Escape Time Algorithmus

Für jeden Pixel wird die soeben berechnete Koordinate als z_0 in die Formel $z_{n+1} = z_n^2 + c$ eingesetzt, der Wert für c kann beliebig gewählt werden und ist charakteristisch für jede Julia-Menge. Die Funktion wird iteriert, bis $n > N$ oder $|z_n| > A$ ist, wobei die Ergebnisse in eine Matrix gespeichert werden, die einen ganzzahligen Wert für jeden Pixel speichert. Erreicht an einer Stelle $|z_n| > A$, schreibt man die n Iterationen, die dafür benötigt wurden, in die

[15] vgl. (ZN94, S. 198)
[16] siehe 3.3

Matrix; wurde vorher die zulässige Zahl an maximalen Iterationen überschritten, so wird der Wert -1 eingetragen (Abbildung 1 Schritt 2.

Aus diesem Raster aus Ganzzahlen wird nun eine Graphik erzeugt. Dazu erhalten alle Pixel mit dem Wert -1 (also die Punkte, die gegen Unendlich streben) eine Farbe, in diesem Fall schwarz und alle anderen Punkte eine Farbe in Abhängigkeit der Anzahl an nötigen Iterationen (Abbildung 1 Schritt 3).[17] Wie der dazu nötige Prozess, aus einer Zahl n mit $0 < n <= N$ eine Farbe zu erzeugen, genau funktioniert, soll im folgenden Kapitel beschrieben werden.

3.3 Farbinterpolation

Um wirklich schöne Bilder zu erhalten, kann die Farbe eines Pixels davon abhängig gemacht werden, wie viele Iterationen für diesen Pixel benötigt wurden. Man benötigt eine Funktion, die aus den ganzzahligen Werten der Matrix jeweils eine Farbe berechnet.

Eine Interpolationsfunktion P hat die Aufgabe mehrere Stützstellen oder Knoten zu verbinden, bei gegebenen Stützstellen (x_i, f_i) für $i = 0, 1, ..., n$ soll gelten $P(x_i) = f_i$.[18] Die mathematisch einfachste Interpolation ist die lineare Interpolation, bei der die Punkte durch einen *Polygonzug*, also durch gerade Linien verbunden werden (siehe Abbildung 2). Dazu wird der zu interpolierende Bereich zunächst in Splines aufgeteilt, das heißt die Intervalle zwischen den Knoten werden einzeln interpoliert.[19]. Wird zu einer Position x der Wert $P(x)$ der Interpolationsfunktion gesucht, muss das Programm zunächst errechnen, zwischen welchen zwei Stützstellen x liegt. Anschließend wird mithilfe der Newton-Interpolation zwischen den zwei Knoten (x_0, f_0) und (x_1, f_1) interpoliert: $P(x) = f_0 + (x - x_0) \cdot \frac{f_1 - f_0}{x_1 - x_0}$.[20] Zwischen der Newton-Interpolation und der Formel zur Berechnung der Startwerte in Kapitel 3.2 lässt sich eine gewisse Ähnlichkeit feststellen, weil die Umwandlung von Pixeln in komplexe Koordinaten auch eine Art der linearen Interpolation ist.

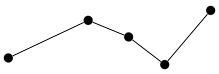


Abbildung 2:
Ein Polygonzug

Zur Umwandlung von Ganzzahlen der Matrix in Farben benötigt man eine Funktion, die für jede Zahl x im Intervall $[1; N]$ eine Farbe zurückgibt. Das vorgestellte Programm lässt den Benutzer dazu Farben für beliebige Stützpunkte angeben, zwischen denen interpoliert wird, was für einen schönen Farbübergang sorgt. Der Einfachheit halber wurde auf die lineare Interpolation zurückgegriffen, die für diesen Anwendungszweck völlig ausreichend ist. Der Computer speichert Farben als Zahlentriplets (r, g, b), wobei die Werte für den Rot-, den Grün- und den Blauan-

[17] vgl. (PR86, S. 189-190)
[18] vgl. (BM04, S. 65)
[19] vgl. (Wer92, S. 160)
[20] vgl. (BM04, S. 68)

teil stehen. Die Interpolation zwischen zwei Farben erfolgt analog zu der zwischen zwei Zahlen, die drei Farbanteile werden gesondert in die Interpolationsfunktion eingesetzt.

Neben der linearen Interpolation gibt es weitere Formen der Interpolation, wie zum Beispiel die Hermite-Interpolation, bei der für jeden Knoten zusätzlich eine Tangente angegeben wird.[21] Bei der Anwendung zur farblichen Darstellung von Julia-Mengen wäre der Vorteil allerdings höchstens marginal, weshalb in meinem Programm auf die einfachere Form zurückgegriffen wurde.

3.4 Das Apfelmännchen und sein Zusammenhang mit der Julia-Menge

Eines der bekanntesten Fraktale ist die auch als Apfelmännchen bezeichnete Mandelbrot-Menge, die in engem Zusammenhang zu den Julia-Mengen steht. Das Apfelmännchen wird genauso wie die Julia-Mengen mit der Formel $z_{n+1} = z_n^2 + c$ unter Verwendung des Fluchtzeitalgorithmus berechnet (siehe 3.2). Der einzige Unterschied besteht in der Wahl der Parameter. Bei den Julia-Mengen werden die in komplexe Zahlen umgewandelten Koordinaten der einzelnen Pixel in z eingesetzt und c ist frei wählbar. Zur Berechnung der Mandelbrot-Menge dagegen wird fest $z = 0 + 0i$ gesetzt, während c den Wert der Koordinate erhält.[22] Das bedeutet, dass jeder Pixel der Mandelbrot-Menge die gleiche Farbe hat, wie der Mittelpunkt der Julia-Menge, die man erhält, wenn man c auf die Koordinaten des Pixels setzt. Dieser Zusammenhang wird besonders deutlich, wenn man den c-Parameter der Julia-Menge – wie in manchen Programmen üblich – durch einen Klick auf die entsprechende Stelle der Mandelbrot-Menge auswählt. So kann man je nachdem welche Stelle man auswählt vorhersagen, welche Farbe der Pixel in der Mitte der Julia-Menge haben wird. Darüber hinaus hat der Mathematiker Gaston Julia bewiesen, dass eine Julia-Menge genau dann zusammenhängend ist, wenn die Iterationsfolge für $z = 0$ zu ihr gehört, sie also nicht gegen unendlich strebt.[23] Das heißt, dass man eine zusammenhängende Julia-Menge erhält, wenn man eine schwarze Stelle der Mandelbrot-Menge als c wählt. Man kann das Apfelmännchen auch als eine Art „Karte“ aller Julia-Mengen auffassen.

3.5 Performance und Grenzen des Computers

Der Computer und die damit verbundene Möglichkeit, Julia-Mengen und das Apfelmännchen zu berechnen und graphisch darzustellen sorgten für einen plötzlichen

[21] vgl. (Sch06, S. 805)
[22] vgl. (CEJ91, S. 35-36)
[23] vgl. (LK92, S. 29)

Fortschritt in diesem Bereich. Ein Ausrechnen der erforderlichen Werte ohne Rechner hätte Jahrzehnte gedauert und selbst ein Computer benötigte damals noch eine so lange Zeit für ein Bild, dass Lerbinger und Kuchenbuch je nach Rechner vor einem „erhöhten Kaffeekonsum“ als Folge der Beschäftigung mit Fraktalen warnen (LK92, S. 43). Auf einem modernen Computer benötigt eine Julia-Menge in üblicher Größe ($600x600$) selbst ohne größere Optimierungen im Quellcode zur Berechnung nur noch etwa eine Sekunde. Dennoch lohnt es sich wegen der Möglichkeit der Erstellung von Filmen (siehe 5) über Performanceverbesserungen nachzudenken. Die wohl größte Optimierung, die hier Anwendung finden könnte, wäre die Aufteilung der Arbeitsleistung auf mehrere CPU-Cores, wodurch sich die Zeit auf dem Testsystem auf $\frac{1}{4}$ reduzieren würde. Da die Zeit ohnehin auch für Filme gering genug ist, wurde darauf im geschriebenen Programm bisher verzichtet.

Neben der zur Berechnung benötigten Zeit fällt vor allem auf, dass man nicht unendlich weit in die Bilder hineinzoomen kann. Nach einigen Zoomstufen entsteht statt der erwarteten Bilder ein Raster aus grobkörnigen Quadraten (siehe Abbildung 3). Dies liegt daran, dass der genaueste Typ von Fließkommazahlen beim Compiler Visual C++ nur eine Genauigkeit von 15 Stellen hat, wie über einen Programmbefehl herausgefunden werden kann.[24] Dadurch unterscheiden sich die Eingangswerte der Funktion ab einem gewissen Zoomfaktor nicht mehr, wodurch auch kein Unterschied in den Farben mehr festzustellen ist. Eine größere Genauigkeit und damit eine höhere Zoomtiefe kann über die Verwendung einer externen Library für Fließkommzahlen erreicht werden, was allerdings mit programmiertechnischen Schwierigkeiten sowie einer starken Verschlechterung der Rechengeschwindigkeit verbunden wäre.[25] Abgesehen von einer geringeren Zoomtiefe entstehen durch die begrenzte Anzahl an Nachkommastellen auch Ungenauigkeiten in der Berechnung, da sich die zwangsläufig entstehenden Rundungsfehler bei iterierter Anwendung aufsummieren.[26]

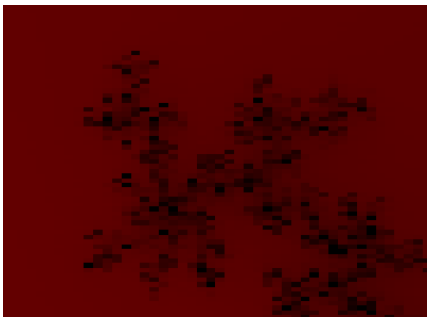


Abbildung 3: Grenze der Zoomfähigkeit

3.6 Weitere iterierte Formeln

Auch wenn die bekannteste Formel im Zusammenhang mit Julia-Mengen $z_{n+1} = z_n^2 + c$ ist, ist sie bei weitem nicht die einzige, die interessante Ergebnisse oder schöne Bilder liefert. Schon durch eine Erhöhung der Potenz von z erhält man

[24] vgl. (Str00, S. 76)
[25] vgl. (Pri10)
[26] vgl. (PJS92, S. 49)

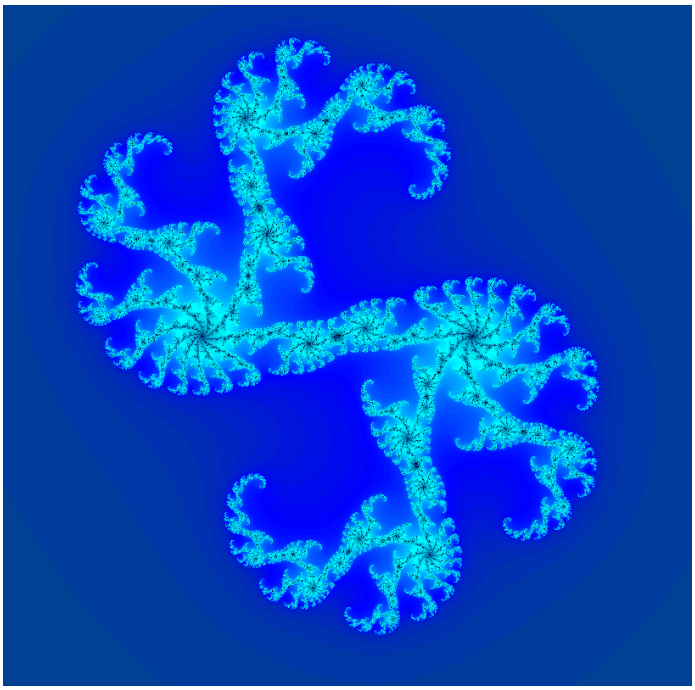


Abbildung 4: Julia-Menge der logistischen Gleichung

Ergebnisse, die komplexer als die der ursprünglichen Formel sind.[27] Eine weitere Möglichkeit ist es, komplett andere Formeln zu verwenden. Hierzu bietet das Buch von Lerbinger und Kuchenbuch eine schöne Zusammenfassung und enthält viele farbige Grafiken (LK92, S. 105-234).

Um dem Benutzer nicht auf vorher festgelegte Formeln zu beschränken, habe ich in meinem Programm die Library muParser eingebunden.[28] Sie übernimmt das Berechnen des Werts einer als Text dargestellten Formel. Auf eine Möglichkeit zur direkten Eingabe von komplexen Zahlen oder Quaternionen wurde verzichtet, um dem Benutzer die freie Wahl der Rechenoperationen zu ermöglichen. Der Benutzer muss für den Real-, den Imaginärteil und – sofern er dies wünscht – für die anderen Teile der Quaternionen separate Formeln eingeben, die durch Kommata getrennt werden. Um die übliche Formel $z_{n+1} = z_n^2 + c$ als Textformel umzusetzen muss man beispielsweise $zr * zr - zi * zi + cr, 2 * zr * zi + ci$ eingeben. Um die Julia-Menge der logistischen Gleichung mit der Formel $z_{n+1} = c \cdot z_n \cdot (1 - z_n)$, die Lerbinger und Kuchenbuch beschreiben (LK92, S. 168), zu erstellen, ist $cr * zr - ci * zi - (cr * (zr * zr - zi * zi) - ci * 2 * zr * zi), cr * zi + ci * zr - (cr * 2 * zr * zi + ci * (zr * zr - zi * zi))$ einzugeben, was sich aus den Rechengesetzen für komplexe Zahlen ergibt.[29] Gibt man diese Formel in das Programm ein, so erhält man ein Ergebnis wie in Abbildung 4.

[27] vgl. (Voß94, S. 158f)
[28] siehe (Ber11)
[29] siehe C

4 3D Julia-Mengen

4.1 3D mit Quaternionen

Die zweidimensionalen Bilder von Julia-Mengen werden mithilfe einer zweidimensionalen Matrix aus Ganzzahlen erzeugt, die berechnet werden, indem für jeden Wert der Matrix die Position als komplexe Zahl in die Formel eingesetzt wurde. Die naheliegendste Möglichkeit, das Verfahren in den dreidimensionalen Raum zu verschieben, ist statt einer zweidimensionalen eine dreidimensionale Matrix zu verwenden, wobei die hinzukommende Dimension auch die dritte Dimension im Raum darstellt. Hierzu braucht man jedoch statt der „zweidimensionalen“ komplexen Zahlen „dreidimensionale“ Zahlen, die eine Multiplikationsvorschrift und eine Additionsvorschrift benötigen, womit der Vektorraum $\mathbb{R}^3$ nicht in Frage kommt, weil ihm die Multiplikation fehlt.[30] Da kein dreidimensionaler Körper existiert, wird stattdessen auf den vierdimensionalen Schiefkörper der Quaternionen zurückgegriffen.[31] Wird der vierte Teil des Quaternions fest auf einen konstanten Wert und die ersten drei Teile auf die x, y und z-Koordinate gesetzt, so erhält man die gewünschte dreidimensionale Matrix, die interpretiert werden kann. Hierbei wird der Teil der Matrix, für den die Iterationsfolge nicht gegen unendlich geht – also die Julia-Menge – als Festkörper dargestellt (siehe Abbildung 5).[32]

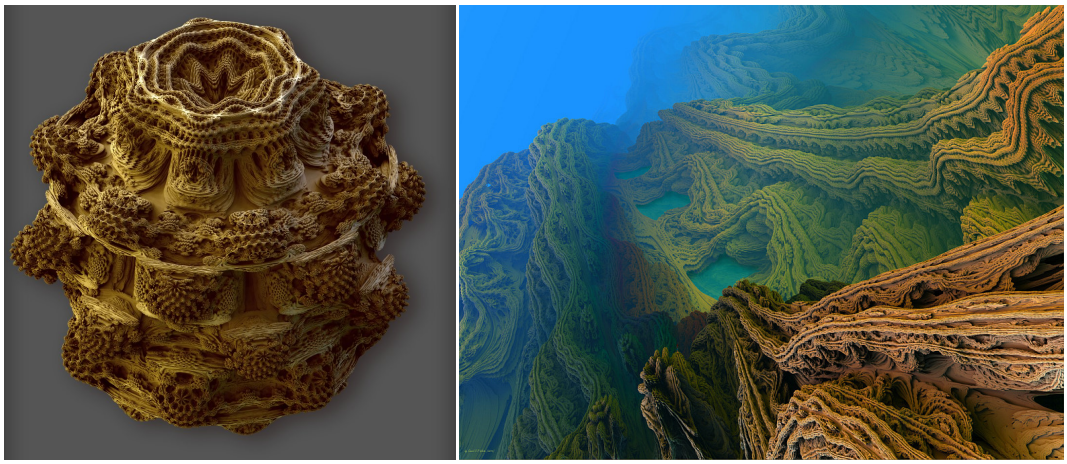
Abbildung 5: Eine 3D Julia-Menge mit Quaternionen; Quelle: (Ram11)

Tatsächlich sehen die dreidimensionalen Julia-Mengen mit Quaternionen zwar interessant aus, den ästhetischen Charakter der 2D Julia-Mengen erreichen sie jedoch nicht. Daher gab es weitere Versuche schöne 3D Julia-Mengen oder Apfelmännchen zu erstellen, von denen einer im folgenden Kapitel beschrieben wird.

4.2 Das Mandelbulb-Fraktal

Das Mandelbulb-Fraktal wurde von verschiedenen Mathematikern vor wenigen Jahren entdeckt.[33] Die Berechnung des Mandelbulbs erfolgt auf die gleiche Weise wie die der 3D-Julia-Mengen mit Quaternionen, wobei die Startwerte genauso wie bei der Mandelbrot-Menge gesetzt werden, die Koordinate jedes Pixel wird also in c

[30] siehe 2.1
[31] siehe 2.3
[32] vgl. (Ram11)
[33] siehe (Whi09a)



(a) Das gesamte Fraktal (b) Im Inneren des Fraktals

Abbildung 6: Das Mandelbulb-Fraktal von der Potenz 4; Quelle: (Whi09a)

eingesetzt, während z konstant auf 0 bleibt. Der einzige Unterschied ist, dass statt vierdimensionalen Quaternionen dreidimensionale Zahlen mit einer eigenen Rechenvorschrift benutzt werden. Die Addition erfolgt genauso wie bei Quaternionen komponentenweise: $(x,y,z)+(x',y',z')=(x+x',y+y',z+z')$. Die Potenz wurde dagegen auf komplett andere Weise definiert: $(x,y,z)^n = r^n \cdot (sin(\theta \cdot n) \cdot cos(\phi \cdot n), sin(\theta \cdot n) \cdot sin(\phi \cdot n), cos(\theta \cdot n))$, mit $r = \sqrt{x^2+y^2+z^2}$, $\theta = \arctan 2(\sqrt{x^2+y^2}, z)$ und $\phi = \arctan(y,x)$.[34] Diese komplizierte Berechnungsvorschrift basiert darauf, dass die komplexe Multiplikation, die bei den zweidimensionalen Julia-Mengen angewandt wird, prinzipiell eine Rotation um den Ursprung mit der Multiplikation der Beträge ist.[35] Im Mandelbulb wird entsprechend eine Rotation um die zwei Winkel ϕ und θ im dreidimensionalen Raum durchgeführt.[36]

Wendet man die oben beschriebenen Rechenvorschriften auf die Formel $z_{n+1} = z_n^2 + c$ an, so erhält man ein Objekt, das zwar der zweidimensionalen Mandelbrot-Menge ähnelt, dem es jedoch beim Heranzoomen an Details mangelt. Erhöht man jedoch die Potenz, so erhält man die gewünschten Details. Die Formel $z_{n+1} = z_n^8 + c$, erzeugt Welten, die beim Zoomen eine erstaunliche Detailtiefe zeigen, wenngleich es auch einige Regionen gibt, die eher an die detailärmeren 3D-Julia-Mengen auf Basis der Quaternionen erinnern.[37]

[34] vgl. (Whi09b)
[35] vgl. (PJS92, S. 779)
[36] vgl. (Whi09a)
[37] vgl. (Whi09b)

5 Darstellung als Film

5.1 Interpolation der Eingangsparameter

Neben der Möglichkeit, die Darstellung der Julia-Mengen in die dritte Dimension zu verlagern, kann man auch mit bewegten Bildern arbeiten, wofür es verschiedene Varianten gibt. Die größte Flexibilität erhält man hierbei mit dem Verfahren der Keyframe-Interpolation. Beim Keyframe-Verfahren, das unter anderem bei Charakteranimationen in der Computergraphik Anwendung findet,[38] wird die Animation in Momentaufnahmen zerlegt, wobei jeder dieser Keyframes ein Bild zu einer bestimmten Zeit beschreibt. Für jedes Schlüsselbild wird ein Zeitpunkt und alle Parameter, die zur Erstellung des Bildes nötig sind, gespeichert. Im beigefügten Programm sind alle diese Parameter, wie zum Beispiel c, z_0, N und A für den Benutzer sichtbar. Für die restlichen Bilder des Films wird zwischen den Keyframes interpoliert,[39] sodass sich weiche Übergänge ergeben.

Die Erstellung der Filme mit diesem Verfahren hat verschiedene Vorteile. Dadurch dass der Benutzer beliebig viele Keyframes einfügen kann, hat er die Wahl, aus nur zwei Julia-Mengen einen Film zu erzeugen, oder viele Zwischenschritte selbst einzufügen und so den zeitlichen Ablauf der Animation genau zu bestimmen. Außerdem lassen sich problemlos verschiedene Parameter gleichzeitig interpolieren, das heißt man kann die verschiedenen Animationsarten, die im Folgenden diskutiert werden, kombinieren und so sehr komplexe Filme produzieren. Da das Verfahren der Interpolation auch bei der Erzeugung von Farbverläufen benutzt wird, bietet sich zudem der programmiertechnische Vorteil, dass diese Programmkomponenten bei geschicktem Design wiederverwendet werden können. Während bei der Erstellung der Farben für die Julia-Mengen mittels Interpolation die lineare Interpolation völlig ausreichend war, zeigt sich, dass die Übergänge bei Verwendung mehrerer Keyframes bei der Produktion von Filmen etwas „ruckartig“ wirken. Dies könnte sich mit Hilfe eines der in Kapitel 3.3 beschriebenen weiteren Interpolationsverfahren verhindern lassen.

5.2 Zooming

Eine besondere Eigenschaft von Julia-Mengen und Apfelmännchen ist die Selbstähnlichkeit, das heißt sie „sind aus Teilmengen aufgebaut, die zu ihnen geometrisch ähnlich sind“.[40] Da dies erst beim Hineinzoomen in das Fraktal hervortritt, liegt es nahe, diesen Vorgang in einem Film darzustellen. Im beigefügten Programm wurde

[38] vgl. (Sch06, S. 179)
[39] siehe 3.3
[40] (SS92, S. 36)

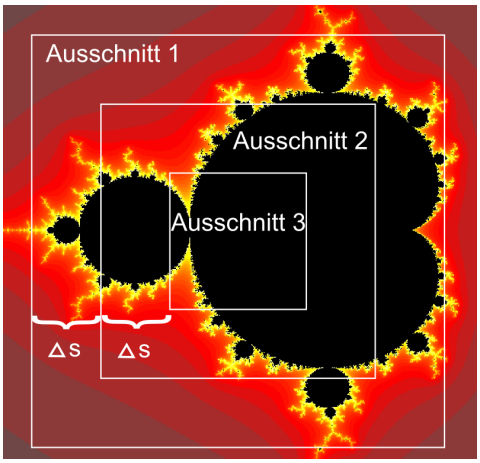


Abbildung 7: Verlauf der Zoomgeschwindigkeit

die in Kapitel 5.1 beschriebenen Methode umgesetzt, sodass die Eingangsparameter, in diesem Fall der anzuzeigende Ausschnitt des Bildes, automatisch interpoliert werden. Dadurch dass bei diesem Ansatz der Ausschnitt gleichmäßig verkleinert wird, wirkt es allerdings so, als würde die Zoomgeschwindigkeit mit der Zeit stark ansteigen. Abbildung 7 verdeutlicht das Problem: Die Schrittweite Δs ist beim Übergang von Ausschnitt 1 zu Ausschnitt 2 genauso groß wie von Ausschnitt 2 zu Ausschnitt 3, weshalb die beiden Übergänge durch die lineare Interpolation in der gleichen Zeit durchgeführt werden. Andererseits wird das angezeigte Bild beim zweiten Übergang halbiert, während es beim ersten Übergang nur um ein Viertel verringert wird, wodurch es scheint, als ob die Zoomgeschwindigkeit zunimmt. Dieses Verhalten nimmt bei hohen Zoomtiefen zu. Möchte der Benutzer eine andere Geschwindigkeit erreichen, kann er dies durch das Einfügen zusätzlicher Keyframes bewirken.

5.3 Kontinuierliche Verschiebung der Eingangsparameter

Die c-Variable ist charakteristisch für jede Julia-Menge. Wählt man sehr nah beieinanderliegende Werte, so lässt sich feststellen, dass sich die zugehörigen Julia-Mengen sehr ähneln und sich zum Teil erst beim Heranzoomen unterscheiden lassen. Durch eine kontinuierliche Verschiebung des c-Wertes in einer Animation erhält man einen weichen Übergang zwischen verschiedenen Julia-Mengen.[41]

Entsprechend kann man bei der Darstellung der Mandelbrot-Menge die z-Koordinate analog verschieben. Dann verformt sich das Apfelmännchen bis es schließlich ganz verschwindet.[42] Die Veränderung von N und A ist ebenfalls möglich.[43] Neben der Erzeugung von schön anzusehenden Filmen, eignet sie sich gut um zu

[41] siehe Abbildung 9
[42] siehe (BD89, S. 158-159)
[43] siehe Abbildung 11

beobachten, wie groß diese Parameter sein müssen, damit die Julia-Menge korrekt dargestellt wird.

5.4 Filmartige Darstellung von 3D-Fraktalen

3D-Fraktale interpretieren eine dreidimensionale Matrix aus Ganzzahlen, die mithilfe des Fluchtzeitalgorithmus berechnet wird, wobei x, y, und z-Koordinate als jeweils eine Dimension der Matrix betrachtet werden. Statt die Matrix in einer dreidimensionalen Welt darzustellen, kann man allerdings auch zwei Dimensionen als Bildschirmkoordinaten und die dritte Dimension als Zeit t interpretieren. Um die Filme mit Quaternionen in das Verfahren der Interpolation der Eingangsparameter einzugliedern mussten für c und z die zwei zusätzlichen imaginären Einheiten j und k als Parameter eingefügt werden. Der gewünschte Film kann mit dem vorgestellten Programm erstellt werden, indem zwischen zwei Julia-Mengen mit unterschiedlichen j-Werten interpoliert wird. Das Ergebnis ist tatsächlich ansprechend und übertrifft sogar die dreidimensionale Darstellung, die in Kapitel 4.1 vorgestellt wurde (siehe Abbildung 8).

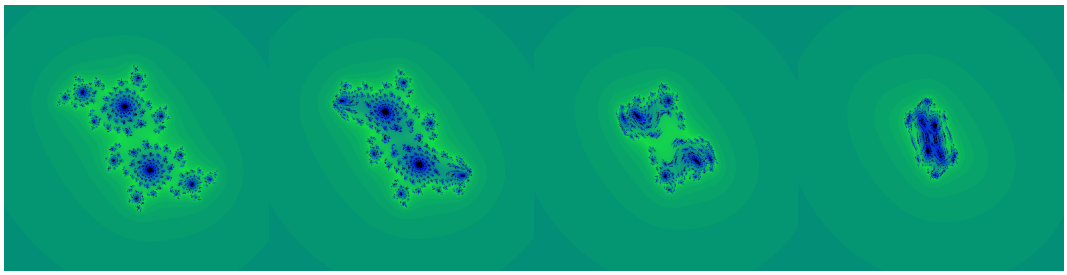


Abbildung 8: Film mit kontinuierlicher Veränderung des Quaternionenanteils

Auch das Mandelbulb-Fraktal kann mit meinem Programm als Film dargestellt werden, obwohl es mit anderen Gesetzen der Multiplikation als denen der Quaternionen erstellt wird. Da man die Rechenoperationen für die einzelnen Teile der Quaternionen selbst eingeben muss, wenn man eigene Formeln mit dem Programm darstellen lassen möchte (siehe Kapitel 3.6), lautet die Formel

$(zr^2+zi^2+zj^2)^4 * sin(8*atan2(sqrt(zr^2+zi^2),zj)) * cos(8*atan2(zi,zr))+cr$, $zr^2+zi^2+zj^2)^4 * sin(8*atan2(sqrt(zr^2+zi^2),zj)) * sin(8*atan2(zi,zr))+ci$,$(zr^2+zi^2+zj^2)^4 * cos(8*atan2(sqrt(zr^2+zi^2),zj))+cj$.[44]

Da die Formel nicht für die farbige Darstellung im zweidimensionalen Raum entworfen wurde, sind die Farben nicht so abwechslungsreich wie beim Apfelmännchen oder bei den Filmen mit Quaternionen. Dennoch ist eine sehr interessante Entwicklung der Formen zu beobachten.[45]

[44] vgl. (Whi09b)

[45] siehe Abbildung 10

6 Zusammenfassung der Ergebnisse und Ausblick

Julia-Mengen und das Apfelmännchen faszinieren die Menschen schon in der einfachen Darstellungsform als zweidimensionales Bild, doch diese Arbeit hat gezeigt, dass damit die Möglichkeiten noch lange nicht ausgeschöpft sind. Neben der Verwendung anderer Formeln, von denen es beliebig viele gibt, kann die Darstellung auf verschiedene Arten in den dreidimensionalen Raum verschoben werden. Auch die Möglichkeit, statt statischer bewegte Bilder zu benutzen, erzeugt schöne Ergebnisse, besonders wenn die Zeitachse die dritte Dimension der 3D-Julia-Mengen ersetzt.

Allgemein betrachtet ist es die Aufgabe jeder Darstellung, eine n-dimensionale Matrix aus Ganzzahlen in einer festgelegten Art und Weise zu interpretieren, wobei die Dimension prinzipiell nur davon abhängt, dass eine geeignete Multiplikationsvorschrift gefunden wird. Der Fantasie sind hierbei keine Grenzen gesetzt und es ist sicher, dass noch nicht alle Möglichkeiten entdeckt wurden. Beispielsweise wäre eine Interpretion der Zahlen als Töne denkbar, wobei das Ergebnis kein Bild, sondern eine Musik wäre.

Dadurch dass ich für diese Arbeit ein eigenes Programm geschrieben habe, konnte ich auf der Suche nach traditionellen und neuen Darstellungsmöglichkeiten Verschiedenes ausprobieren, insbesondere im Bereich der Visualisierung als Film. Überraschend war hierbei, dass alle meine Versuche zur Darstellung von Julia-Mengen als Film ästhetisch ansprechende Resultate hervorgebracht haben. Insbesondere die, Idee 3D-Filme mit Quaternionen zu erstellen, habe ich nirgendwo im Internet gefunden, obwohl sich gezeigt hat, dass die Resultate beeindruckend sind. Ich habe mein Programm auf eine Weise geschrieben, die dem Benutzer viele Alternativen lässt, sodass die Möglichkeiten des Programms noch lange nicht ausgeschöpft sind. So musste ich nur eine winzige Änderung an meinem ursprünglichen Programm vornehmen, um die zunächst nicht vorgesehene Darstellung des Mandelbulb-Fraktals als zweidimensionalen Film umzusetzen.

Insgesamt bieten sich in der Welt der Julia-Mengen noch unzählige Möglichkeiten und mit der beigefügten Software kann man sehr viele von ihnen verwirklichen. Zudem ist mein Programm so flexibel gestaltet, dass es von Programmierern weiterentwickelt werden kann, um weitere Darstellungsformen von Fraktalen umzusetzen.

Literatur

[BD89] BECKER, Karl-Heinz ; DÖRFLER, Michael: *Dynamische Systeme und Fraktale*. 3. Auflage. Braunschweig : Vieweg, 1989

[Ber11] BERG, Ingo: *muParser - a fast math libary, Features*. `http://muparser.sourceforge.net/`, 2011. – zuletzt abgerufen am 07.11.2011

[Beu01] BEUTELSPACHER, Albrecht: *Lineare Algebra*. 5. Auflage. Braunschweig; Wiesbaden : Vieweg, 2001

[BM04] BOLLHÖFER, Mathias ; MEHRMANN, Volker: *Numerische Mathematik*. 1. Auflage. Wiesbaden : Vieweg, 2004

[CEJ91] CRILLY, A.J. ; EARNSHAW, R.A. ; JONES, H.: *Fractals and Chaos*. 1. Auflage. New York : Springer-Verlag, 1991

[Dit76] DITTMANN, Helmut: *Komplexe Zahlen*. 2. Auflage. München : Bayerischer Schulbuch-Verlag, 1976

[DRU98] DUFNER, Julius ; ROSER, Andreas ; UNSELD, Frank: *Fraktale und Julia-Mengen*. 1. Auflage. Thun; Frankfurt am Main : Verlag Harri Deutsch, 1998

[LK92] LERBINGER, Klaus ; KUCHENBUCH, Michael: *Faszination Fraktale*. 1. Auflage. München : Systhema Verlag, 1992

[PJS92] PEINTGEN, Heinz-Otto ; JÜRGENS, Hartmut ; SAUPE, Dietmar: *Chaos and fractals*. 1. Auflage. New York : Springer Verlag, 1992

[PR86] PEINTGEN, Heinz-Otto ; RICHTER, Peter H.: *The Beauty of Fractals*. 1. Auflage. Berlin; Heidelberg : Springer Verlag, 1986

[Pri10] PRIMI, Ivano: *High Precision Arithmetic Library*. `http://www.nongnu.org/hpalib/`, 2010. – zuletzt abgerufen am 07.11.2011

[Ram11] RAMMELT, Patrick: *3D-Fraktale*. `http://patricks-seite.de/fractal/index.html`, 2011. – zuletzt abgerufen am 07.11.2011

[Sch06] SCHERFGEN, David: *3D-Spieleprogrammierung*. 3. Auflage. München; Wien : Hanser, 2006

[SS92] STOYAN, Dietrich ; STOYAN, Helga: *Fraktale Formen Punktfelder*. 1. Auflage. Berlin : Akademie Verlag, 1992

[Str00] STROUSTRUP, Bjarne: *The C++ Programming Language*. 3. Auflage. Indianapolis : Pearson Education, 2000

[Voß94] VOSS, Herbert: *Chaos und Fraktale selbst programmieren*. 1. Auflage. Poing : Franzis-Verlag GmbH, 1994

[Wer92] WERNER, Jochen: *Numerische Mathematik 1*. 1. Auflage. Braunschweig; Wiesbaden : Vieweg, 1992

[Whi09a] WHITE, Daniel: *The Unravelling of the Real 3D Mandelbulb*. `http://www.skytopia.com/project/fractal/mandelbulb.html`, 2009. – zuletzt abgerufen am 07.11.2011

[Whi09b] WHITE, Daniel: *The Unravelling of the Real 3D Mandelbulb*. `http://www.skytopia.com/project/fractal/2mandelbulb.html`, 2009. – zuletzt abgerufen am 07.11.2011

[ZN94] ZEITLER, Herbert ; NEIDHARDT, Wolfgang: *Fraktale und Chaos*. 2. Auflage. Darmstadt : Wissenschaftliche Buchgesellschaft, 1994

A Ausgewählte Bilder einiger Filmsequenzen

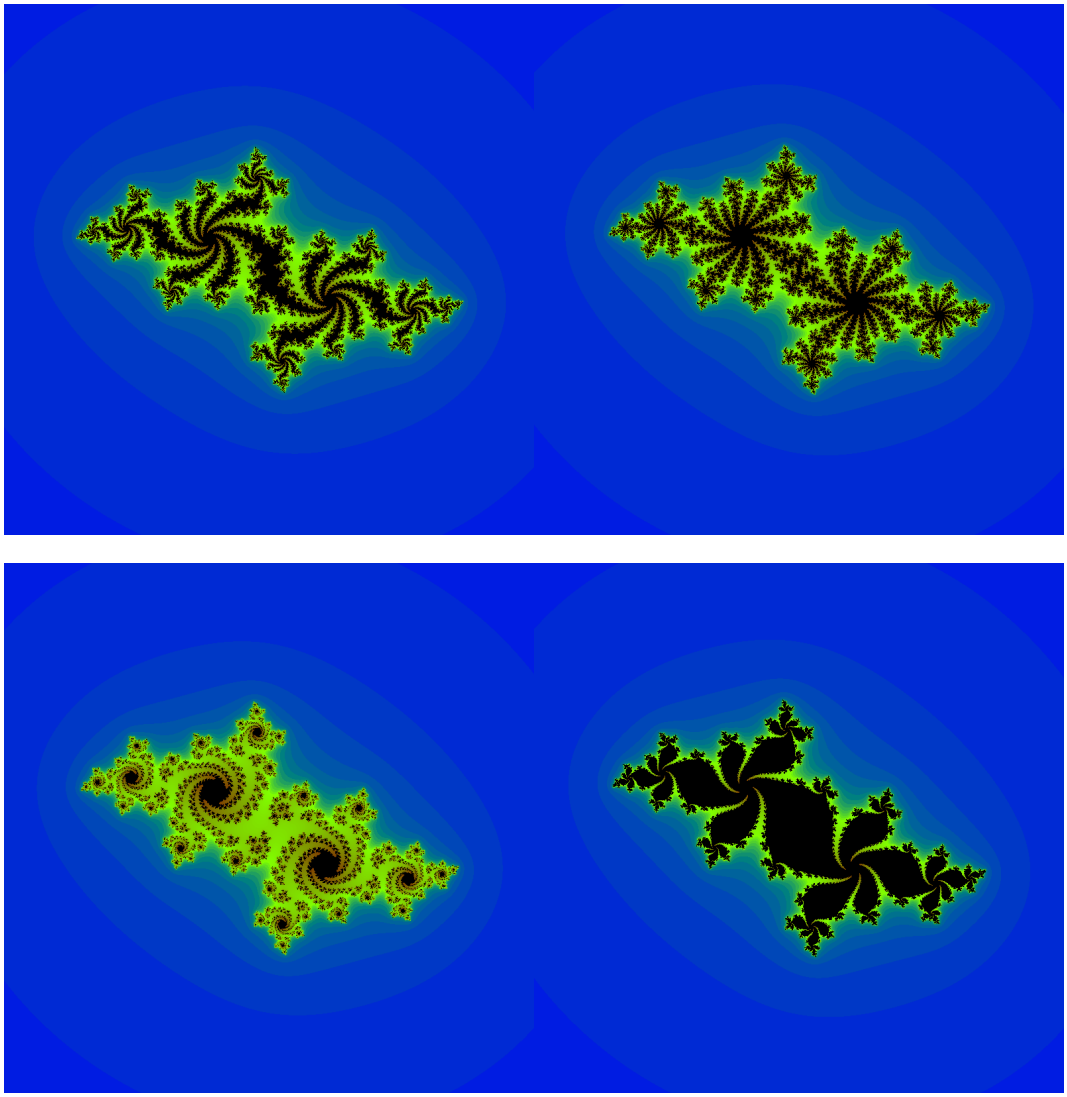


Abbildung 9: Verschiebung des c-Parameters

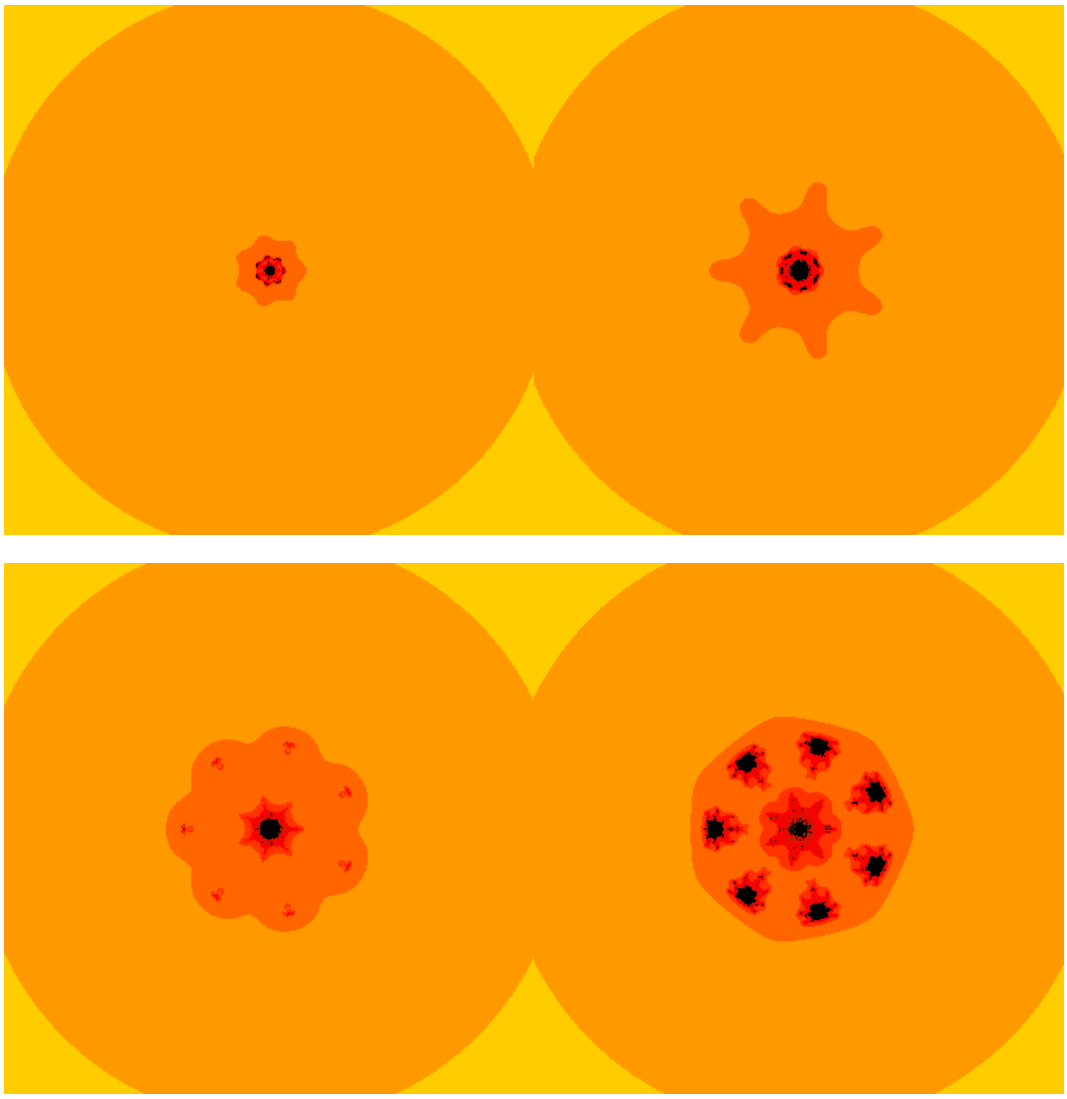


Abbildung 10: Das Mandelbulb-Fraktal als Film

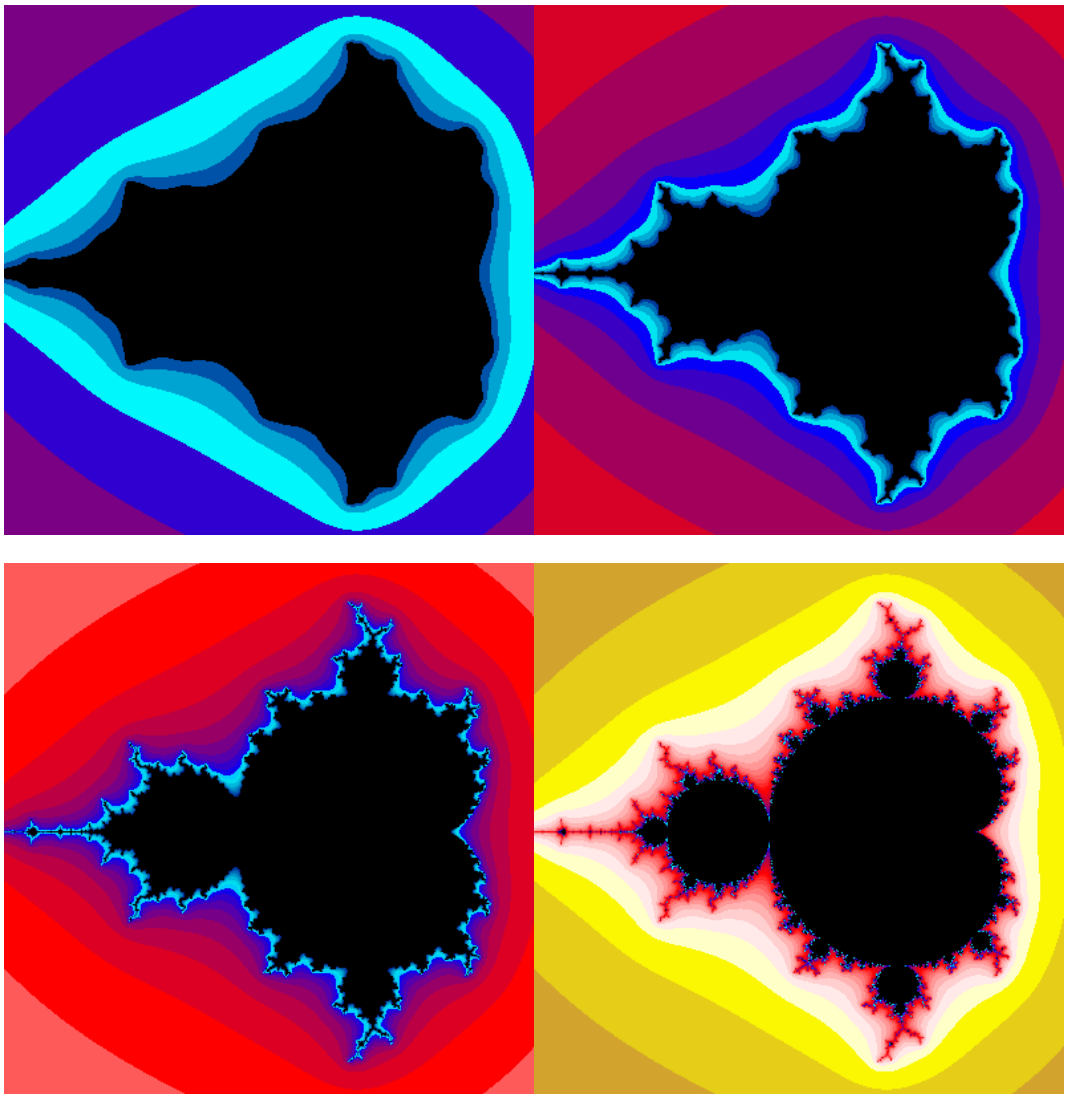


Abbildung 11: Veränderung von numIterations

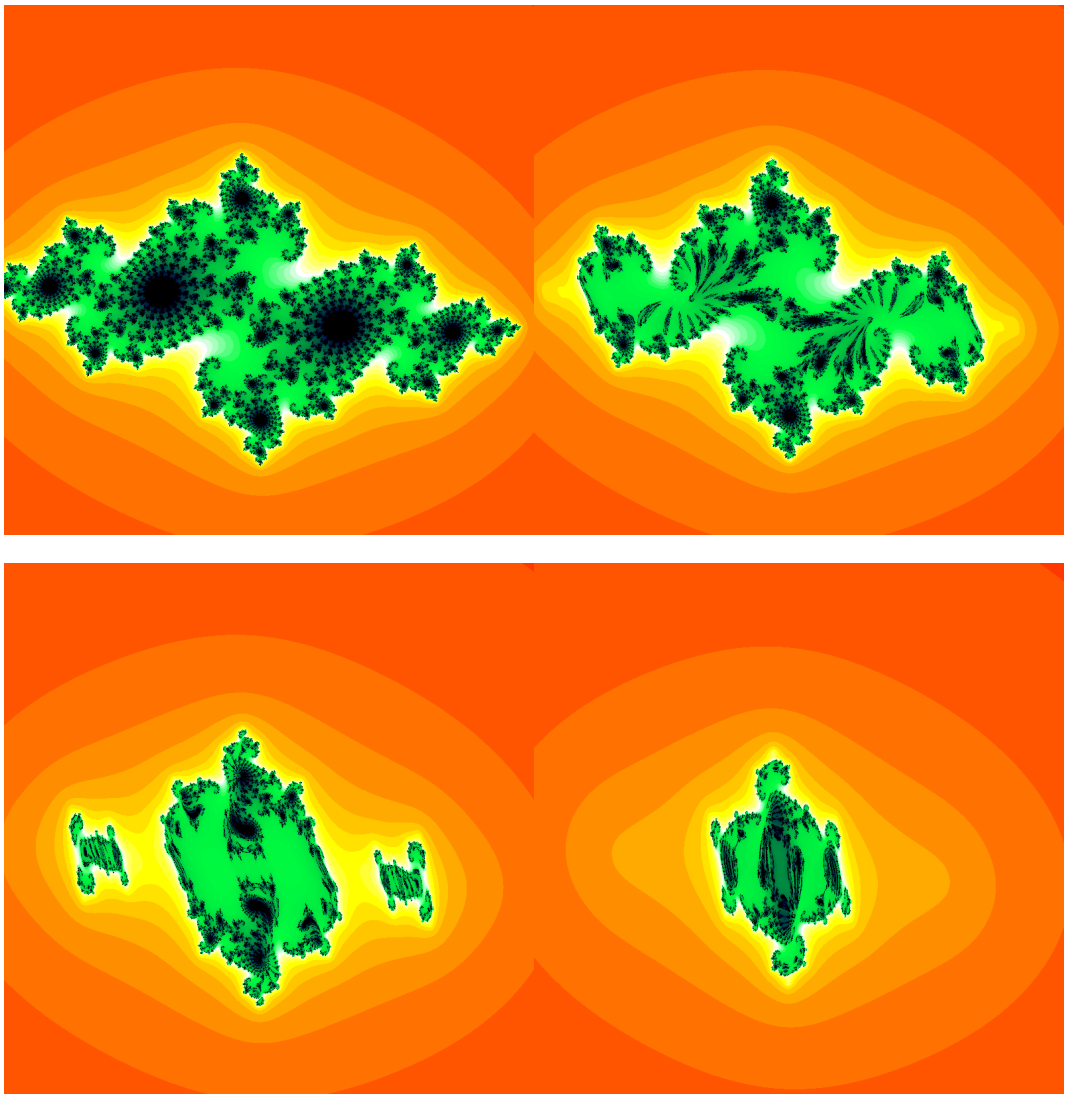


Abbildung 12: Veränderung des Quaternionenteils

B Inhalt der CD

Hier soll der Inhalt der beigefügten CD kurz erläutert werden. Im Ordner „Programm“ befindet sich die beigefügte Software. Eine Einführung in die Benutzung und Installation ist unter Anhang C zu finden. Im Ordner „Quellcode“ steht der Quelltext des Programms in der Programmiersprache C++.

Der Ordner „Filme“ enthält einige Animationen, die mit dem beigefügten Programm und dem Windows Live Movie Maker erstellt wurden. In der Datei Zusammenfassung.txt ist erklärt, wie die Filme jeweils generiert wurden.

Die Seminararbeit selbst ist ebenfalls auf der CD unter dem Namen „Seminararbeit Julia-Mengen.pdf“ zu finden.

C Gebrauchsanweisung Julia-Mengen-Programm

Installation und Starten des Programms

Dies ist eine Einführung in die Benutzung des beigefügten Programms zur Erstellung von Julia-Mengen. Bevor man das Programm starten kann, muss man zunächst das Programm Visual C++ Redistributable von Microsoft installieren, indem die Datei vcredist_x86.exe ausgeführt wird. Danach kann das eigentliche Programm einfach per Doppelklick auf die Datei Juliamengen.exe gestartet werden, woraufhin sich die Benutzeroberfläche des Programms präsentiert. Bei einem Klick auf die Icons in der Toolbar (siehe Abbildung 13) öffnet sich jeweils ein inneres Fenster, das man in dem Hauptfenster verschieben, skalieren und minimieren kann. Wenn man eines dieser Fenster schließt wird auch sein Inhalt gelöscht und er kann von keinem anderen Fenster mehr aufgerufen werden.

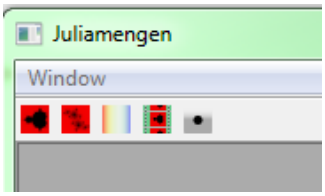


Abbildung 13: Die Toolbar

Erstellung der Mandelbrot-Menge

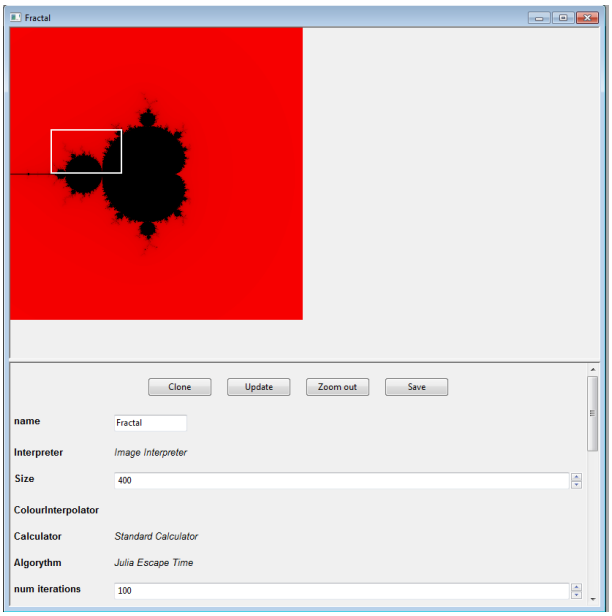


Abbildung 14: Heranzoomen durch Klicken und Ziehen

Zum Erstellen einer Mandelbrot-Menge muss man zunächst über das ganz links befindliche Icon in der Toolbar fahren. Daraufhin erscheint ein Tooltip mit der Aufschrift „New Mandelbrot-Set“. Wenn man nun auf dieses Icon drückt, öffnet sich ein inneres Fenster zur Erstellung einer Mandelbrot-Menge. Im oberen Bereich des Fensters befindet sich ein Bild des erstellten Fraktals, im unteren Abschnitt kann man sämtliche Parameter sehen, die man anpassen kann. Der Benutzer kann die

Platzaufteilung auch selbst ändern, indem er den Trennbalken zwischen dem Bildbereich und dem Parameterbereich per Drag and Drop verschiebt.

Einige der Parameter – wie zum Beispiel num iterations und maximum value – wurden in der Arbeit behandelt. Änderungen an den meisten dieser Parameter werden erst sichtbar, wenn das Fraktal durch einen Klick auf Update neu berechnet wird. Mit einem Klick auf Clone wird ein neues Fenster geöffnet, in das alle Parameter aus dem zu klonenden Fenster übernommen werden, was besonders bei der Erstellung von Filmen hilfreich sein kann.

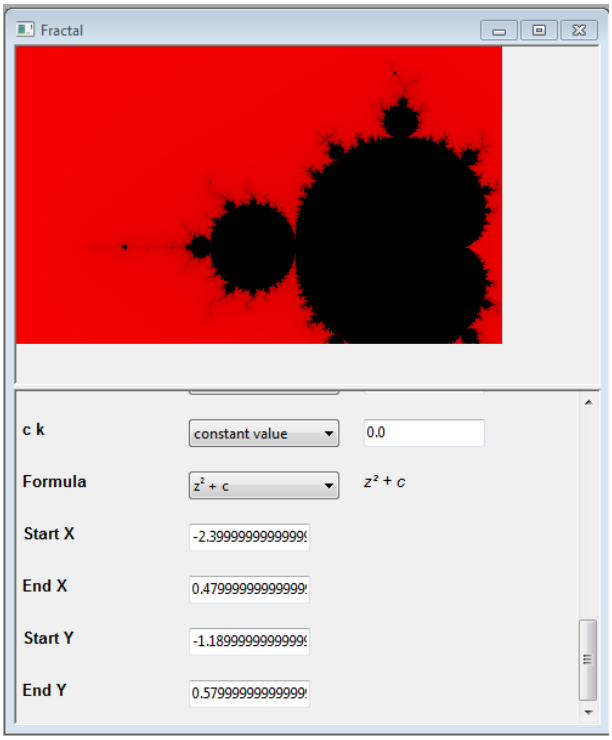


Abbildung 15: Nach dem Heranzoomen

Man kann in das Fraktal hineinzoomen, indem man im Bildbereich des Fensters durch Klicken und Ziehen ein Rechteck erstellt (siehe Abbildung 14). Wenn man im Parameterbereich nach unten scrollt, sieht man, dass sich durch den Zoomvorgang die Werte der Parameter Start X, End X, Start Y und End Y verändert haben. Diese entsprechen dem Ausschnitt des Bildes, der angezeigt wird. Man kann auch manuell durch Eingeben entsprechender Werte in diese Textfelder und einen anschließenden Klick auf Update hinein- und hinauszoomen. Das Seitenverhältnis des resultierenden Bildes passt sich dem Seitenverhältnis des Rechtecks an, das zum Hineinzoomen gezogen wurde. Der Wert des Parameters Size bestimmt die Länge der größeren Seite. Das Bild im Beispiel (siehe Abbildung 15) ist $400x300$ Pixel groß, der Parameter Size beträgt also 400.

Will man wieder einen größeren Teil des Fraktals sehen, kann man einmal oder mehrmals auf den Button mit der Beschriftung „Zoom out“ klicken, wodurch sich

die Werte Start X, End X usw. verändern. Erst bei einem anschließenden Klick auf „Update“ wird der gewünschte Ausschnitt sichtbar.

Auswahl von Punkten

Mit einem Klick auf den ganz rechten Knopf in der Toolbar öffnet sich ein Fenster zur Erstellung eines Punktes. Man kann die Koordinaten des Punktes entweder per Hand eintragen oder auf das Zeigersymbol klicken, das daraufhin gelb hinterlegt wird. Dann wird der Punkt in allen geöffneten Fraktalen angezeigt. Klickt man auf eine Stelle eines beliebigen Fraktals, werden die Koordinaten des Punktes auf diese Stelle gesetzt. Möchte man den Punkt nicht in einem Fraktal neu auswählen, kann man erneut auf das Zeigersymbol klicken, sodass es nicht mehr gelb hinterlegt ist.

Erstellen einer Julia-Menge

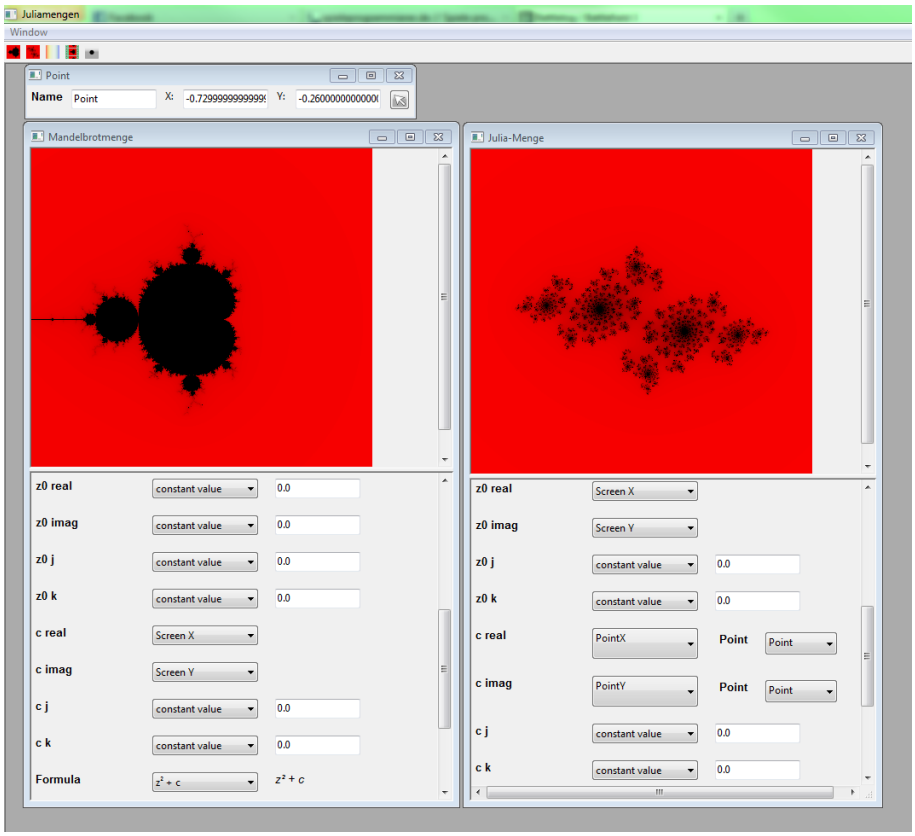


Abbildung 16: Parameter der Julia-Menge und des Apfelmännchens im Vergleich

Eine Julia-Menge kann man am einfachsten erstellen, indem man auf das zweite Symbol von links in der Toolbar klicken. Dann kann man einen Punkt auswählen, den man zuvor erstellt haben sollte. Hierbei ist es hilfreich, wenn man den Punkten unterschiedliche Namen gibt. Wenn man mit „Ok“ bestätigt, wird ein Fenster mit der Julia-Menge angezeigt. Wenn man die Koordinaten des Punktes ändert, zu dem

die Julia-Menge erstellt wird, braucht man in der Julia-Menge nur auf Update zu klicken und das Bild zu den veränderten Koordinaten wird angezeigt.

Betrachtet man die Parameter eines Apfelmännchens und einer Julia-Menge (siehe Abbildung 16), so lässt sich feststellen, dass sie sich nur in den Teilen der Parameter z_0 und c unterscheiden, was auch dem enstpricht, was in Kapitel 3.4 beschrieben wurde. Alle diese Parameter können auf verschiedene Weisen festgelegt werden. Wählt man „Screen X“ oder „Screen Y“ aus, so wird an jedem Pixel der entsprechende Teil der Koordinate des Pixels eingesetzt. Bei „constant value“ kann man selbst einen Wert festlegen, der für alle Pixel gleich ist. Bei „Point X“ und „Point Y“ kann man einen vorher erstellten Punkt auswählen, dessen x- bzw. y-Koordinate eingesetzt wird, wobei diese automatisch aktualisiert wird, wenn sich die Koordinaten des Punktes ändern.

So kann man durch entsprechende Anpassung der Parameter aus einer Julia-Menge das Apfelmännchen machen und umgekehrt.

Anpassung der Farben

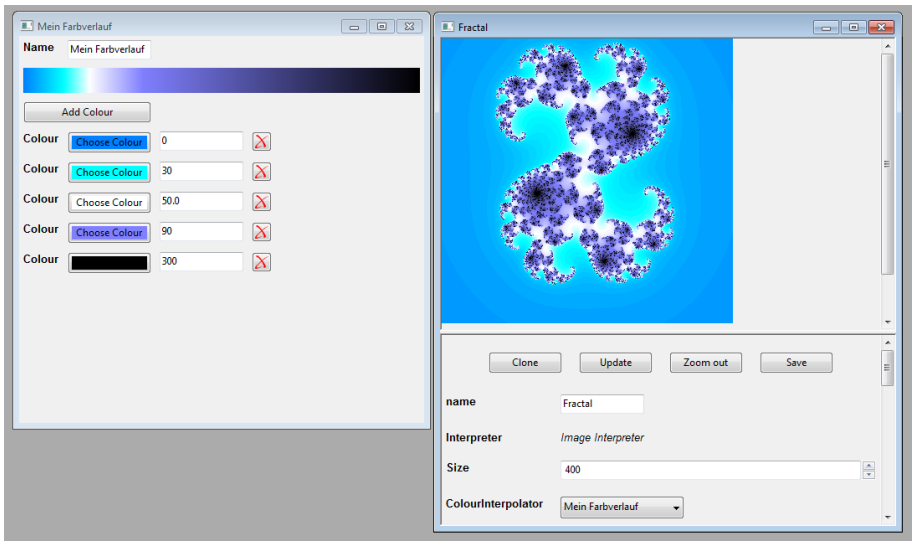


Abbildung 17: Eine Julia-Menge mit eigenem Farbverlauf

Um die Fraktale zu verschönern, bietet sich die Nutzung eines eigenen Farbverlaufs an. Dazu muss man auf das dritte Icon von links in der Toolbar klicken. Im daraufhin erscheinenden Fenster kann man durch Klicks auf „Add Colour“ einen neuen Zwischenschritt einfügen. Für jeden Zwischenschritt kann man mit Klick auf die Schaltfläche mit der Beschriftung „Choose Colour“ eine Farbe auswählen. Daneben muss man die Position des Zwischenschritts angeben. Klickt man auf das rote „X“ wird der Zwischenschritt gelöscht. Über den Zwischenschritten wird der aktuelle Farbverlauf angezeigt. Möchte man ihn auf ein Fraktal anwenden, so kann man ihn im Parameterteil des Fraktalfensters unter Punkt „ColourInterpolator“ auswählen.

Der Farbverlauf wird immer relativ zu den ausgewählten Stützstellen mit der höchsten und niedrigsten Position erstellt. Deshalb ist es bei nur zwei Stützstellen unwichtig, welche Position sie haben, solange diese nicht bei beiden gleich ist.

Erstellung von Filmen

Filme werden mit dem Programm erstellt, indem zwischen mehreren Fraktalen interpoliert wird. Dazu muss man auf den zweiten Knopf von rechts in der Toolbar drücken, woraufhin sich ein Fenster öffnet, das sehr dem Fenster zur Erstellung von Farbverläufen ähnelt. Zunächst müssen mit „Add Keyframe“ Keyframes hinzugefügt werden. Für jeden Keyframe muss man die Julia-Menge angeben, sowie die Position, die der Zeit im Film entspricht, zu der dieser Keyframe angezeigt wird. „Num frames“ entspricht der Anzahl an Bildern, aus denen der Film bestehen wird und „Width“ und „Height“ geben die Abmessungen der Bilder an. Drückt man auf „Create Movie“ öffnet sich ein Dialog, in dem man das Verzeichnis angeben muss, in das die Bilder (durchnummeriert) gespeichert werden.

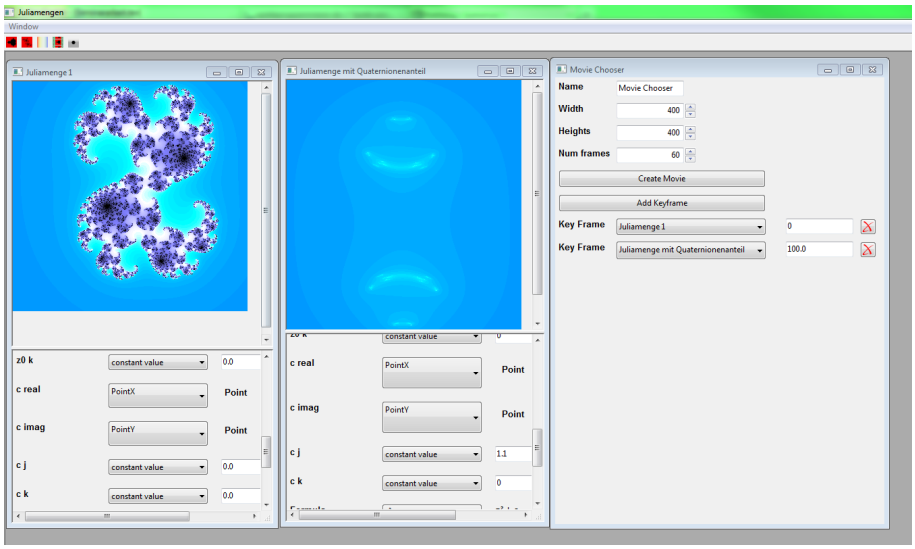


Abbildung 18: Erstellung eines Films

Es ist sinnvoll, aus den Bildern, die vom Programm generiert wurden, eine Filmdatei zu erstellen, was man mit einem beliebigen Filmbearbeitungsprogramm tun kann. Ich kann hierbei das kostenlos verfügbare Programm Windows Live Movie Maker empfehlen.

Eingeben eigener Formeln

Neben der Standardformel $z_{n+1} = z_n^2 + c$ besteht noch die Möglichkeit eigene Formeln einzugeben. Dazu muss man unter dem Parameter „Formula“ den Eintrag „Custom Formula“ auswählen. Hier kann man eine eigene Formel eingeben, wobei

für die vier Teile (Realteil, Imaginärteile usw.) des Quaternions eigene Formeln mit Komma getrennt angegeben werden müssen. Die Parameter kann man in der Form „zr“ für den Realteil von z, „ci“ für den Imaginärteil von c, etc angeben. Beispiele für derartige Formeln sind in der Arbeit im Kapitel 3.6 zu finden. Diese Funktionalität wurde mithilfe der Library muParser umgesetzt, die unterstützten Formeln (wie zum Beispiel sin und cos) stehen unter (Ber11). Neben den dort aufgelisteten Funktionen wurde noch „atan2“ eingebaut. Sollten weitere mathematische Funktionen benötigt werden, kann ich sie auf Wunsch ebenfalls implementieren.

D Quellcode des beigefügten Programms

Datei Juliamengen.h

```
#ifndef JULIAMENGEN_H
#define JULIAMENGEN_H

#include "wx/wx.h"
#include "Calculators.h"
#include "Parameters.h"
#include "wx/splitter.h"

namespace Gui
{
  class JuliamengenApp : public wxApp
  {
  public:
    virtual bool OnInit();
  };

  class FractalPanel;
  class ColourInterpolator;

  template<class T> class ComponentStorage
  {
  public:
    typedef T Component;
    /*typedef std::list<Component*> ComponentList;*/
    typedef std::list< Parameters::NonPanelParameterChooser<Component*>*> ComponentList;
    typedef Parameters::ParameterChooserObserver<Component*> ComponentObserver;
    typedef std::list<ComponentObserver*> ObserverList;

    ComponentStorage()
      : components(),
      observers()
    {

    }

    void addComponent(Parameters::NonPanelParameterChooser<Component*>* component)
    {
      components.push_back(component);
      for(ObserverList::iterator it = observers.begin(); it != observers.end(); ++it)
      {
        (*it)->addChooser(new Parameters::PassingChooser<Component*> ((*it)->getParent(), component, "") );
      }
    }

    void removeComponent(Parameters::NonPanelParameterChooser<Component*>* component)
    {
      components.remove(component);
      for(ObserverList::iterator it = observers.begin(); it != observers.end(); ++it)
      {
        std::vector<Parameters::ParameterChooser<Component*>*>* choosers = (*it)->getChooserSet();
        for(std::vector<Parameters::ParameterChooser<Component*>*>::iterator it2 = choosers->begin(); it2 !=
            choosers->end(); ++it2)
        {
          Parameters::PassingChooser<Component*>* casted = dynamic_cast< Parameters::PassingChooser<Component
              *>*> ( (*it2) );
          if(casted && casted->getPassedTo() == component)
          {
            (*it)->removeChooser(casted);
            break;
          }
        }
      }
    }

    ComponentList* getComponents()
    {
      return &components;
    }

    void addObserver(ComponentObserver* observer)
    {
      observers.push_back(observer);
      for( ComponentList::iterator it = components.begin(); it != components.end(); ++it)
      {
        observer->addChooser( new Parameters::PassingChooser<Component*>(observer->getParent(), (*it), ""));
      }
    }

    void removeObserver(ComponentObserver* observer)
    {
      observers.remove(observer);
    }

  private:
    ComponentList components;
    ObserverList observers;

  };

  class ColourInterpolator : public Parameters::Interpolator<wxColour>, public Parameters::
      NonPanelParameterChooser<ColourInterpolator*>
```

```
89  {
    public:
91    ColourInterpolator(std::string name);

93    virtual wxColour interpolate(wxColour one, double factor1, wxColour two, double factor2) const;

95    virtual void onInterpolationUpdate(const InterpolationData& data);

97    virtual ColourInterpolator* getValue();

99    void setName(std::string name);

101   /**
      * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
103   **/
      virtual void selected();
105
      /**
107   * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
      **/
109   virtual void unselected();
    };
111
    class ColourInterpolatorInterpolator : public ColourInterpolator
113 {
    public:
115   ColourInterpolatorInterpolator(ColourInterpolator *param1, const double *factor1, ColourInterpolator *
          param2, const double *factor2);
      /*~ColourInterpolatorInterpolator();*/
117
      virtual wxColour getInterpolation(double position);
119
    private:
121   ColourInterpolator *param1, *param2;
      const double *factor1, *factor2;
123 };

125 class Point;

127 class PointSelectionObserver
    {
129 public:
      virtual void setPointToSelect(Point* point) = 0;
131 };

133 class ImageComputor;

135 class MainFrame : public wxMDIParentFrame, public PointSelectionObserver
    {
137 public:

139   typedef std::vector<FractalPanel*> FractalPanelList;

141   MainFrame(const wxString& title, const wxPoint& pos, const wxSize& size);

143   virtual ~MainFrame();

145   FractalPanel* newFractalPanel(Point* point = 0);

147   void onNewClick(wxCommandEvent& ev);

149   void onNewJuliaClick(wxCommandEvent& ev);

151   void newColourChooser();

153   void onNewColourChooserClick(wxCommandEvent& ev);

155   void newMovie();

157   void onNewMovieClick(wxCommandEvent& ev);

159   void newPoint(bool invisible = false);

161   void onNewPointClick(wxCommandEvent& evt);

163   virtual void setPointToSelect(Point* point);

165   ComponentStorage<ColourInterpolator> getInterpolatorStorage();

167 private:
      ComponentStorage<ColourInterpolator> interpolatorStorage;
169   // For Interpolators
      ComponentStorage<ColourInterpolator>* interpolatorPointer;
171
      ComponentStorage<ImageComputor> fractalStorage;
173   ComponentStorage<ImageComputor>* fractalPointer;

175   ComponentStorage<Point> pointStorage;
      ComponentStorage<Point>* pointPointer;
177
      /*FractalPanelList fractalPanels;*/
179   Parameters::Interpolator<wxColour>::InterpolationData standardInterpolationData;
      ColourInterpolator standardInterpolator;
181
      Point* activePoint;
183
      /*Calculators::CalculateJuliaEscapeTime juliaCalculator;
185   Calculators::CalculateAppleEscapeTime appleCalculator;*/
    };
```

```
187
    class ImageComputor
189 {
    public:
191   ImageComputor(Parameters::Parameter<Calculators::AbstractCalculator*>* calculatorParam, Parameters::
          Parameter<ColourInterpolator*>* interpolatorParam, Parameters::Parameter<double>* startX, Parameters
          ::Parameter<double>* endX, Parameters::Parameter<double>* startY, Parameters::Parameter<double>* endY
          , Parameters::ScreenChooserSupplier* screenX, Parameters::ScreenChooserSupplier* screenY);
      virtual ~ImageComputor();
193   wxBitmap& computeImage(int width, int height);
      wxBitmap& updateOnlyColour();
195
      /*std::auto_ptr<Parameters::Parameter<ImageComputor*> > getInterpolator(const double* factor1Pointer,
          ImageComputor* other, const double* factor2Pointer);*/
197   Parameters::Parameter<ImageComputor*>* getInterpolator(const double* factor1Pointer, ImageComputor* other,
          const double* factor2Pointer);

199 private:
      Parameters::Parameter<Calculators::AbstractCalculator*>* calculatorParam;
201   Parameters::Parameter<ColourInterpolator*>* interpolatorParam;
      wxBitmap bitmap;
203   Calculators::AlgorythmReturning* numIterations;
      int maxIterations;
205   int width, height;

207   Parameters::Parameter<double>* startX;
      Parameters::Parameter<double>* endX;
209   Parameters::Parameter<double>* startY;
      Parameters::Parameter<double>* endY;
211   Parameters::ScreenChooserSupplier* screenX;
      Parameters::ScreenChooserSupplier* screenY;
213
    };
215
    class ImageInterpreter : public Calculators::Interpreter, public Parameters::NonPanelParameterChooser<
        Calculators::Interpreter*>, public Parameters::ParameterObserver<ColourInterpolator*>, public
        PointSelectionObserver, public Parameters::ChooserSupplier<double>
217 {
    public:
219
      // typedef Parameters::Interpolator<wxColour> ColourInterpolator;
221
      ImageInterpreter(wxScrolledWindow* parent, Parameters::ParametersPanel& parametersPanel, ComponentStorage<
          ColourInterpolator>** interpolatorStorage, ComponentStorage<Point>** pointStorage, Point*
          juliaForPoint);
223   virtual ~ImageInterpreter();

225   // virtual void SetSize(int size);

227   virtual void update();

229   void OnPaint(wxPaintEvent& event);

231   void OnLeftDown(wxMouseEvent& event);
      void ImageInterpreter::OnLeftUp(wxMouseEvent& event);
233   void OnMouseMove(wxMouseEvent& event);
      void OnEraseBackground(wxEraseEvent& event);
235
      void zoomOut();
237
      void save();
239
      void setPointToSelect(Point* point);
241
      // Inherited from NonPanelParameterChooser
243   virtual Interpreter* getValue();

245   ImageComputor* getComputor();

247   virtual void addChoosersToGui(Parameters::ParameterGui<double>* gui, wxWindow* parent);

249   /**
      * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
251   */
      virtual void selected();
253   /**
      * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
255   **/
      virtual void unselected();
257
      virtual void onChange(ParameterReturning newValue, Parameter<ParameterReturning>* sender);
259
    private:
261   Parameters::ParametersPanel& parametersPanel;
      std::auto_ptr<Parameters::ParameterGui<int> > sizeParam/*, startXParam, startYParam, endXParam, endYParam
          */;
263   std::auto_ptr<Parameters::ParameterGui<ColourInterpolator*> > colourParam;
      std::auto_ptr<Parameters::ParameterGui<Calculators::AbstractCalculator*> > calculatorParam;
265   std::auto_ptr<Parameters::ParameterGui<double> > startXGui, endXGui, startYGui, endYGui;
      wxBitmap* bitmap;
267   wxRect selectedRect;
      bool rectSelected;
269   ComponentStorage<ColourInterpolator>** interpolatorStorage;
      Parameters::ScreenChooserSupplier screenX, screenY;
271   double startX, endX, startY, endY;

273   std::auto_ptr<ImageComputor> computor;

275   std::vector<ComponentStorage<Point>::ComponentObserver*> observersToBeRemoved;
```

```
277     ComponentStorage<Point>** pointStorage;
        Point* selectPoint;
279     Point* juliaForPoint;
    };
281
    class FractalPanel : public wxMDIChildFrame, public Parameters::NonPanelParameterChooser<ImageComputor*>,
          public Parameters::ParameterObserver<wxString>, public PointSelectionObserver
283 {
    public:
285     FractalPanel(MainFrame* parent, std::string name, ComponentStorage<ColourInterpolator>**
            interpolatorStorage, ComponentStorage<ImageComputor>** fractalStorage, ComponentStorage<Point>**
            pointStorage, Point* juliaForPoint = 0);
        virtual ~FractalPanel();
287
        Parameters::ParametersPanel* getParametersPanel();
289
        void setPointToSelect(Point* point);
291
        virtual void onChange(wxString newValue, Parameter<wxString>* sender);
293     void onUpdateClick(wxCommandEvent& ev);
        void onRightClick(wxCommandEvent& evt);
295     void onCloneClick(wxCommandEvent& evt);
        void onZoomOutClick(wxCommandEvent& evt);
297     void onSaveClick(wxCommandEvent& evt);

299     virtual Returning getValue();

301     void updateImageInterpreter();

303     /**
        * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
305     */
        virtual void selected();
307
        /**
309     * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
        **/
311     virtual void unselected();

313 private:
        MainFrame* mainFrame;
315     wxSplitterWindow* splitter;
        wxScrolledWindow *topScroll, *bottomScroll;
317     ImageInterpreter* imageInterpreter;
        Parameters::ParametersPanel* parametersPanel;
319     Parameters::ParameterGui<Calculators::Interpreter*> parameterGui;
        Parameters::ParameterGui<int> gui2;
321     Parameters::ParameterGui<wxString> nameGui;

323     ComponentStorage<ImageComputor>** fractalStorage;

325     /*std::auto_ptr<Calculators::Formula> formula;
        Calculators::ConverterSet converters;
327     Calculators::Calculator calculator;
        Calculators::Interpreter* interpreter;*/
329 };

331

333 class ColourParameterChooser : public Parameters::ParameterChooser<wxColour>
    {
335 public:
      public:
337
        ColourParameterChooser(wxWindow* parent, std::string theName, wxColour startColour);
339
        void onButtonPress(wxCommandEvent& ev);
341
        // void onSpin(wxSpinEvent& ev);
343
        virtual Returning getValue();
345
        /**
347     * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
        */
349     virtual void selected();
        /**
351     * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
        **/
353     virtual void unselected();

355 private:
        wxWindow* parent;
357     wxSizer* sizer;
        wxButton* chooseButton;
359     wxColour value;
    };
361
    class ColourChooser : public wxMDIChildFrame, public Parameters::ChooserSupplier<wxColour>, public
          Parameters::InterpolationObserver<wxColour>
363 {
    public:
365     ColourChooser(wxMDIParentFrame* parent, std::string name, ComponentStorage<ColourInterpolator>**
            interpolatorStorage);
        virtual ~ColourChooser();
367
        void onNameChange(wxCommandEvent& evt);
369
```

```
    virtual void addChoosersToGui(Parameters::ParameterGui<wxColour>* gui, wxWindow* parent);

    virtual void onInterpolationUpdate(const InterpolationData& data);

    void updatePreviewImage();

    void onSizeChange(wxSizeEvent& evt);

    ColourInterpolator* getInterpolator();

  private:
    ColourChooser(const ColourChooser&);

    static const int PREVIEW_HEIGHT = 30;
    wxScrolledWindow* panel;
    wxGridBagSizer* sizer;
    wxBitmap bitmap;
    wxStaticBitmap* preview;
    Parameters::InterpolationChooser<wxColour> interpolationChooser;
    ColourInterpolator interpolator;
    ComponentStorage<ColourInterpolator>** interpolatorStorage;
  };

  class FractalInterpolator : public Parameters::Interpolator<ImageComputor*>
  {
  public:
    FractalInterpolator();

    virtual ~FractalInterpolator();

    virtual ImageComputor* interpolate(ImageComputor* one, double factor1, ImageComputor* two, double factor2)
        const;
    void prepareInterpolation(ImageComputor* one, ImageComputor* two) const;
    void reset();

  private:
    mutable std::auto_ptr<Parameters::Parameter<ImageComputor*> > computor;
    mutable double factor1, factor2;
    mutable ImageComputor *two;

  };

  class MovieChooser : public wxMDIChildFrame, public Parameters::ChooserSupplier<ImageComputor*>, public
      Parameters::InterpolationObserver<ImageComputor*>
  {
  public:
    MovieChooser(wxMDIParentFrame* parent, std::string name, ComponentStorage<ImageComputor>** fractalStorage)
        ;
    virtual ~MovieChooser();

    virtual void addChoosersToGui(Parameters::ParameterGui<ImageComputor*>* gui, wxWindow* parent);
    virtual void removeChooser(Parameters::ParameterGui<ImageComputor*>* gui, wxWindow* parent);

    virtual void onInterpolationUpdate(const InterpolationData& data);

    void onNameChange(wxCommandEvent& evt);

    void onCreateMovie(wxCommandEvent& evt);

  private:
    wxScrolledWindow* panel;
    wxGridBagSizer* sizer;
    Parameters::InterpolationChooser<ImageComputor*> interpolationChooser;
    ComponentStorage<ImageComputor>** fractalStorage;
    wxSpinCtrl *width, *height, *numFrames;
    FractalInterpolator interpolator;
  };

  /*class SelectionCallback
  {
  public:
    virtual void rectSelected(wxRect rect);
    virtual void pointSelected(wxPoint point);
  }*/

  class Point :  public wxMDIChildFrame, public Parameters::NonPanelParameterChooser<Point*>
  {
  public:
    Point(wxMDIParentFrame* parent, std::string name, ComponentStorage<Point>** pointStorage,
        PointSelectionObserver* observer, Point** activePoint);
    virtual ~Point();

    void onNameChange(wxCommandEvent& evt);
    void onSelectPointClick(wxCommandEvent& evt);

    virtual Returning getValue();

    double getX();
    double getY();

    void setPoint(double x, double y);

    void startSelection();
    void stopSelection(bool newSelectionAfterwards = false);

    /**
    * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
    */
    virtual void selected();
```

```
465       /**
          * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
467       **/
          virtual void unselected();
469
      private:
471       // Point::Point(const Point& point);
          wxBoxSizer* sizer;
473       Parameters::TextfieldChooser<double> *positionX, *positionY;
          ComponentStorage<Point>** pointStorage;
475       Point** activePoint;
          wxBitmap normalBitmap, selectedBitmap;
477       wxBitmapButton* button;
          bool selecting;
479       PointSelectionObserver* observer;
      };
481
      class PointChooser : public Parameters::ParameterChooser<double>
483   {
      public:
485       PointChooser(wxWindow* parent, bool xElseY);

487       Parameters::ParameterGui<Point*>* getGui();

489       virtual Returning getValue();

491       /**
          * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
493       */
          virtual void selected();
495
          /**
497       * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
          **/
499       virtual void unselected();

501       virtual std::string getMyName() const;

503   private:
          bool xElseY;
505       Parameters::ParameterSizer* sizer;
          Parameters::ParameterGui<Point*> gui;
507   };

509 }

511 #endif
```

Quellcode/Juliamengen.h

Datei Parameters.h

```
1   #ifndef PARAMETERS_H
    #define PARAMETERS_H
3
    #include "wx/wx.h"
5   #include "wx/spinctrl.h"
    #include "wx/gbsizer.h"
7   #include "wx/event.h"
    // #include "wx/valgen.h"
9   #include <vector>
    #include <string>
11  #include <list>
    #include <string>
13  #include <fstream>
    #include <algorithm>
15

17  namespace Parameters
    {
19    class Copyable
      {
21    public:
        virtual void copyTo(Copyable* copy){};
23    };

25    template<class R> class Parameter;
      template<class R> class ParameterObserver
27    {
      public:
29      typedef R ParameterReturning;

31      virtual void onChange(ParameterReturning newValue, Parameter<ParameterReturning>* sender) = 0;
        /*virtual void onDelete(){}*/
33
      };
35
      /**
37    * Dies ist ein Parameter, der im Programm oder vom Benutzer ausgewaehlt werden kann.
      **/
39    template<class R> class Parameter : public Copyable
      {
41    public:
```

```
    typedef R Returning;
43  typedef ParameterObserver<Returning> Observer;
    typedef std::list<Observer*> Observerlist;
45
    virtual ~Parameter()
47  {
    /*  for(Observerlist::iterator it = observers.begin(); it != observers.end(); ++it)
49    {
        (*it)->onDelete();
51    }*/
    }
53
    virtual Returning getValue() = 0;
55
    // voluntary
57  virtual void setValue(R value){}
    virtual void prepareMultiGetValue() {}
59
    virtual void addObserver(Observer* observer)
61  {
      observers.push_back(observer);
63  }

65  virtual void removeObserver(Observer* observer)
    {
67    observers.remove(observer);
    }
69
    void notifyObservers(Returning newValue)
71  {
      for(Observerlist::iterator it = observers.begin(); it != observers.end(); ++it)
73    {
        (*it)->onChange(newValue, this);
75    }
    }
77
    virtual void copyTo(Copyable* copy)
79  {
      Parameter<Returning>* other = dynamic_cast<Parameter<Returning>* >(copy);
81    if(other)
      {
83      other->setValue(getValue());
      }
85  }

87 private:
    Observerlist observers;
89
    };
91
    typedef wxGridBagSizer ParameterSizer;
93
    template<class T> class NumberValidator : public wxValidator
95  {
    public:
97    typedef T NumberType;
      /*NumberValidator()
99      : value(0),
        value2()
101   {}*/

103   NumberValidator(NumberType* value)
        : wxValidator(),
105     value(value),
        value2()
107   {}

109   NumberValidator(const NumberValidator& copy)
        : value(copy.value),
111     value2(copy.value2)
      {}
113
      /*NumberValidator()
115     : wxTextValidator(wxFILTER_NONE)
      {}*/
117
      virtual bool TransferFromWindow()
119   {
        wxTextCtrl* ctrl = dynamic_cast<wxTextCtrl*>(GetWindow());
121     if(!ctrl)
        {
123       return false;
        }
125     std::stringstream stream;
        stream.precision(100);
127     std::string value = ctrl->GetValue();
        if(value == "" || value=="-")
129     {
          getUsedValue() = 0;
131       return true;
        }
133     stream << value;
        if( (stream >> getUsedValue()).fail() || !stream.eof() )
135     {
          return false;
137     }

139     return true;
      }
```

```
141
      virtual bool TransferToWindow()
143   {
        wxTextCtrl* ctrl = dynamic_cast<wxTextCtrl*>(GetWindow());
145     if(!ctrl)
        {
147       return false;
        }
149
        std::stringstream stream;
151     stream.precision(100);
        stream << getUsedValue();
153     ctrl->SetValue(stream.str());
        return true;
155   }

157   /*virtual bool TransferFromWindow()
      {
159     return wxTextValidator::TransferFromWindow();
      }
161
      virtual bool TransferToWindow()
163   {
        return wxTextValidator::TransferToWindow();
165   }*/

167   virtual bool Validate(wxWindow* parent)
      {
169     wxTextCtrl* ctrl = dynamic_cast<wxTextCtrl*>(parent);
        if(!ctrl)
171     {
          return false;
173     }

175     std::stringstream stream;
        stream.precision(100);
177     NumberType t = 0;
        std::string value = ctrl->GetValue();
179     if(value == "" || value=="-")
        {
181       return true;
        }
183     stream << value;
        stream >> t;
185     bool fail = stream.fail() || !stream.eof();
        if(fail)
187     {
          wxMessageBox(wxT("Sie müssen eine Zahl eingeben!"), "Juliamengen");
189     }
        return !stream.fail();
191   }

193   virtual wxObject* Clone() const
      {
195     return static_cast<wxObject*>(new NumberValidator<NumberType>(*this));
      }
197
    private:
199
      NumberType& getUsedValue()
201   {
        if(value)
203     {
          return *value;
205     }
        return value2;
207   }

209   NumberType* value;
      NumberType value2;
211
    };
213
    // typedef wxPanel AbstractParameterGui;
215 class AbstractParameterGui : public Copyable
    {
217 public:

219   static const wxFont PARAMNAME_FONT;
      static const wxFont FONT_2;
221   // static NumberValidator<double> DOUBLE_VALIDATOR;

223   /**
      * Fügt alle Komponenten an der angegebenen Position hinzu.
225   **/
      // virtual void addToPanel(wxPanel* panel, ParameterSizer* sizer, int row) = 0;
227   virtual void addToPanel(int row) = 0;

229   virtual void removeFromPanel(wxPanel*, ParameterSizer* sizer) = 0;
    };
231
    template<class R> class NonPanelParameterChooser : public Parameter<R>
233 {
    public:
235
      NonPanelParameterChooser()
237   {}

239
```

```
    virtual std::string getMyName() const
    {
      return myName;
    }

    /**
    * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
    **/
    virtual void selected() = 0;

    /**
    * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
    **/
    virtual void unselected() = 0;

  protected:
    std::string myName;
  };

  /**
  * Diese Klasse waehlt den eigentlichen Parameter aus. Manche Parametertypen koennen von mehreren
  * verschiedenen ParameterChoosern ausgewaehlt werden.
  **/
  template<class R> class ParameterChooser : public NonPanelParameterChooser<R>, public wxPanel
  {
  public:
    ParameterChooser(wxWindow* parent)
      : wxPanel(parent)
    {}

  };

  template<class R> class ParameterChooserObserver
  {
  public:
    typedef R ParameterReturning;
    virtual void addChooser( ParameterChooser<ParameterReturning>* chooser) = 0;
    virtual void removeChooser(ParameterChooser<ParameterReturning>* chooser) = 0;
    virtual std::vector<ParameterChooser<ParameterReturning>*>* getChooserSet() = 0;
    virtual wxPanel* getParent() = 0;
  };

  template<class R> class ParameterGui : public AbstractParameterGui, public Parameter<R>, public wxEvtHandler
        , public ParameterChooserObserver<R>, public ParameterObserver<R>
  {
  public:

    typedef ParameterChooser<Returning> Chooser;
    // typedef std::tr1::shared_ptr<Chooser> ChooserPtr;
    typedef Chooser* ChooserPtr;
    typedef std::vector<ChooserPtr> ChooserSet;

    ParameterGui(std::string name, wxPanel* parent, ParameterSizer* sizer)
      : parent(parent),
      choosers(),
      sizer(sizer),
      box(new wxChoice(parent, -1)),
      ownRow(),
      selected(0),
      active(false)
    {
      heading = new wxStaticText(parent, -1, name);
      heading->SetFont(AbstractParameterGui::PARAMNAME_FONT);
      heading->Show(false),
      box->SetSelection(0);

      box->Show(false);
      // box->SetEditable(false);

      box->PushEventHandler(this);
      this->Connect(wxEVT_COMMAND_CHOICE_SELECTED, wxCommandEventHandler(ParameterGui::onBoxSelection));
      // box->SetWindowStyle(wxCB_READONLY);
    }

    void onBoxSelection(wxCommandEvent& event)
    {
      if(event.GetEventObject() != box || selected == box->GetSelection())
      {
        return;
      }
      choosers[selected]->Show(false);
      choosers[selected]->unselected();
      sizer->Detach(choosers[selected]);
      selected = box->GetSelection();
      choosers[selected]->Show(true);
      choosers[selected]->selected();
      sizer->Add(choosers[selected], wxGBPosition(ownRow, 2), wxDefaultSpan, wxEXPAND | wxALL, 10);
      parent->SendSizeEventToParent();
      notifyObservers(choosers[selected]->getValue());
    }

    /*ParameterGui(ChooserSet choosers)
      : wxPanel(parent, -1),
      choosers(choosers)
    {}*/

    virtual void addChooser(ChooserPtr chooser)
    {
      choosers.push_back(chooser);
```

```
      chooser->addObserver(this);
339   box->Append(chooser->getMyName());
      chooser->Show(false);
341
      if(choosers.size() == 2 && active)
343   {
        sizer->Detach(choosers[0]);
345     box->Show(true);
        box->SetSelection(0);
347     selected = 0;
        sizer->Add(box, wxGBPosition(ownRow, 1), wxDefaultSpan, wxEXPAND | wxALL, 5);
349     sizer->Add(choosers[selected], wxGBPosition(ownRow, 2), wxDefaultSpan, wxEXPAND | wxALL, 5);
        parent->SendSizeEventToParent();
351   }
    }
353
    virtual void removeChooser(ChooserPtr chooser)
355 {
      int i = 0;
357   for(ChooserSet::iterator it = choosers.begin(); it != choosers.end(); ++it, ++i)
      {
359     if( (*it) == chooser )
        {
361       if(active && (i == selected || choosers.size() >= 2))
          {
363         removeFromPanel(parent, sizer);
            chooser->Destroy();
365         choosers.erase(it);
            box->Delete(i);
367         if(i == selected)
            {
369           selected = 0;
            }else if(i < selected)
371         {
              selected--;
373         }
            // selected = box->GetSelection();
375         addToPanel(ownRow);
            return;
377       }
          chooser->Destroy();
379       choosers.erase(it);
          selected = box->GetSelection();
381       return;
        }
383   };
    }
385
    virtual std::vector<ChooserPtr>* getChooserSet()
387 {
      return &choosers;
389 }

391 virtual void onChange(Returning newValue, Parameter<Returning>* sender)
    {
393   if(sender == choosers[selected])
      {
395     Parameter<Returning>::notifyObservers(newValue);
      }
397   NonPanelParameterChooser<Returning>* casted = static_cast<NonPanelParameterChooser<Returning>* >(sender)
          ;
      int i = 0;
399   for( ChooserSet::iterator it = choosers.begin(); it != choosers.end(); ++it, ++i)
      {
401     if( (*it) == casted )
        {
403       box->SetString(i, casted->getMyName());
          box->SetSelection(selected);
405       box->SendSizeEvent();
          break;
407     }
      }
409   // box->setname
    }
411
    virtual Returning getValue()
413 {
      /*if(choosers.size() == 1)
415   {
        return choosers[0]->getValue();
417   }
      return choosers[box->GetSelection()]->getValue();*/
419   return choosers[selected]->getValue();
    }
421
    virtual void setValue(Returning value)
423 {
      choosers[selected]->setValue(value);
425 }

427 virtual void removeFromPanel(wxPanel* panel, ParameterSizer* sizer)
    {
429   heading->Show(false);
      sizer->Detach(heading);
431   choosers[selected]->Show(false);
      sizer->Detach(choosers[selected]);
433   if(choosers.size() > 1)
      {
435     box->Show(false);
```

```
        sizer->Detach(box);
437     }
        active = false;
439     // Parameters::NumberValidator<double> testValidator;
    }
441
    virtual void addToPanel(int row)
443 {
        active = true;
445     ownRow = row;
        heading->Show(true);
447     sizer->Add(heading, wxGBPosition(ownRow, 0), wxGBSpan(1, 1), wxALL, 5);

449     if(!choosers.size())
        {
451       return;
        }
453
        choosers[selected]->selected();
455
        if(choosers.size() == 1)
457     {
          choosers[selected]->Show();
459       sizer->Add(choosers[selected], wxGBPosition(ownRow, 1), wxGBSpan(1, 2), wxALL | wxEXPAND, 5);
          return;
461       }

463     box->Show(true);
        box->SetSelection(selected);
465     sizer->Add(box, wxGBPosition(ownRow, 1), wxDefaultSpan, wxEXPAND | wxALL, 5);
        choosers[selected]->Show();
467     sizer->Add(choosers[selected], wxGBPosition(ownRow, 2), wxDefaultSpan, wxEXPAND | wxALL, 5);
    }
469
    int getOwnRow()
471 {
      return ownRow;
473 }

475 virtual wxPanel* getParent()
    {
477   return parent;
    }
479
    virtual void copyTo(Copyable* copy)
481 {
      ParameterGui<Returning>* other = dynamic_cast< ParameterGui<Returning>*>(copy);
483   if(other)
      {
485     other->box->SetSelection(selected);
        wxCommandEvent event;
487     event.SetEventObject(other->box);
        other->onBoxSelection(event);
489     other->setValue(getValue());
      }
491 }

493 std::string getName()
    {
495   return heading->GetLabel();
    }
497
    void setSelection(int num, bool afterAdding = false)
499 {
      // box->SetSelection(num);
501   if(afterAdding)
      {
503     box->SetSelection(num);
        wxCommandEvent event;
505     event.SetEventObject(box);
        onBoxSelection(event);
507   }else
      {
509     selected = num;
      }
511 }

513 //void setSelectedChooser(Chooser* chooser)
    //{
515 //  for(int i = 0; i < choosers.size(); ++i)
    //  {
517 //    if(choosers[i] == chooser)
    //    {
519 //      selected = i;
    //      return;
521 //    }
    //  }
523 //}

525 int getSelection()
    {
527   return selected;
    }
529
    int getNumChoosers()
531 {
      return choosers.size();
533 }
```

```
535     Chooser* getChooser(int num)
        {
537       return choosers[num];
        }
539
        virtual ~ParameterGui()
541     {
          // TODO: Maybe the superclass destructors must be declared virtual, too.
543       box->RemoveEventHandler(this);
        }
545
      private:
547     wxStaticText* heading;
        wxPanel* parent;
549     ChooserSet choosers;
        ParameterSizer* sizer;
551     int ownRow;
        wxChoice* box;
553     int selected;
        bool active;
555   };

557   class ParametersPanel : public wxPanel
      {
559   public:
        ParametersPanel(wxWindow* parent);
561
        void addParameter(AbstractParameterGui* param);
563     void removeParameter(AbstractParameterGui* param);
        int getSize() const;
565     ParameterSizer* getSizer();
        void copyTo(ParametersPanel* copy);
567
      private:
569     std::list<AbstractParameterGui*> parameters;
        ParameterSizer* sizer;
571   };

573   template<class T> class NumberChooser : public ParameterChooser<T>
      {
575   public:
        typedef T NumberType;
577
        NumberChooser(wxWindow* parent, std::string theName, NumberType startValue, int minValue, int maxValue)
579       : ParameterChooser<T>(parent),
          spinner(new wxSpinCtrl(this, -1, wxEmptyString, wxDefaultPosition, wxDefaultSize, wxSP_ARROW_KEYS,
                minValue, maxValue, startValue)),
581       sizer(new wxBoxSizer(wxHORIZONTAL)),
          value(startValue)
583     {
          ParameterChooser::myName = theName;
585       SetSizer(sizer);
          sizer->Add(spinner, 1);
587       // this->Connect(wxEVT_COMMAND_SPINCTRL_UPDATED, wxSpinEventHandler(NumberChooser::onSpin));
          this->Connect(wxEVT_COMMAND_TEXT_UPDATED, wxCommandEventHandler(NumberChooser::onChange));
589     }

591     void onChange(wxCommandEvent& ev)
        {
593       value = spinner->GetValue();
          notifyObservers(value);
595     }

597     void onSpin(wxSpinEvent& ev)
        {
599

601     }

603     virtual Returning getValue()
        {
605       return value;
        }
607
        virtual void setValue(NumberType newValue)
609     {
          spinner->SetValue(newValue);
611       value = newValue;
        }
613
        /**
615     * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
        */
617     virtual void selected()
        {
619
        }
621
        /**
623     * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
        **/
625     virtual void unselected()
        {
627
        }
629
      private:
631     wxSpinCtrl* spinner;
        wxBoxSizer* sizer;
```

```
633     NumberType value;
    };
635
    template<class T> class TextfieldChooser : public ParameterChooser<T>
637 {
    public:
639   typedef T TextType;

641   TextfieldChooser(wxWindow* parent, std::string theName, const std::string defaultValue, wxValidator&
          validator, std::auto_ptr<TextType> valuePointer)
        : ParameterChooser<TextType>(parent),
643     sizer(new wxBoxSizer(wxVERTICAL)),
      text(new wxTextCtrl(this, -1, defaultValue, wxDefaultPosition, wxDefaultSize, 0L, validator)),
645   myValidator(text->GetValidator()),
      value(valuePointer)
647   {
      // TODO: The validator is cloned internally and can therefore be a usual reference.
649     myName = theName;
      SetSizer(sizer);
651   sizer->Add(text);
      wxEvtHandler::Connect(wxEVT_COMMAND_TEXT_UPDATED, wxCommandEventHandler(TextfieldChooser::onTextUpdate))
          ;
653   myValidator->TransferFromWindow();
    }
655
    void onTextUpdate(wxCommandEvent& ev)
657 {
      // wxMessageBox("hey");
659   if(!myValidator->TransferFromWindow())
      {
661     myValidator->Validate(text);
        myValidator->TransferToWindow();
663   }
      notifyObservers(*(value.get()));
665 }

667   virtual TextType getValue()
    {
669   return *(value.get());
      // return static_cast<TextType>(text->GetValue());
671 }

673   virtual void setValue(TextType newValue)
    {
675   *(value.get()) = newValue;
      myValidator->TransferToWindow();
677 }

679   virtual void prepareMultiGetValue()
    {
681
    }
683
    /**
685 * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
    */
687 virtual void selected()
    {
689
    }
691
    /**
693 * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
    **/
695 virtual void unselected()
    {
697
    }
699
    private:
701   wxBoxSizer* sizer;
      std::auto_ptr<TextType> value;
703   wxTextCtrl* text;
      wxValidator* myValidator;
705 };

707 template<class T> class BasicScreenChooserSupplier
    {
709 public:
      typedef T NumberType;
711
      BasicScreenChooserSupplier(bool xElseY, int numPixels, NumberType start, NumberType end)
713     : x(xElseY),
        size(size),
715     currentPixel(0),
        start(start),
717     end(end),
        realSize(end - start),
719     shareWith(0)
      {
721     // myName = "Screen " + xElseY ? "X" : "Y";
      }
723
      NumberType getValue() const
725   {
        return (static_cast<NumberType>(currentPixel) / static_cast<NumberType>(size)) * realSize + start;
727   }

729   int getCurrentPixel() const
```

```
    {
731   return currentPixel;
    }
733
    void setCurrentPixel(int currentPixel)
735 {
      this->currentPixel = currentPixel;
737   if(shareWith)
      {
739     shareWith->setCurrentPixel(currentPixel);
      }
741 }

743 void setShareWith(BasicScreenChooserSupplier<T>* shareWith)
    {
745   this->shareWith = shareWith;
    }
747
    bool isXElseY() const
749 {
      return x;
751 }

753 void setNumPixels(int size)
    {
755   this->size = size;
      if(shareWith)
757   {
        shareWith->setNumPixels(size);
759   }
    }
761
    void setBounds(NumberType start, NumberType end)
763 {
      this->start = start;
765   this->end = end;
      realSize = end - start;
767   if(shareWith)
      {
769     shareWith->setBounds(start, end);
      }
771 }

773 private:
      bool x;
775   int size, currentPixel;
      NumberType start, end, realSize;
777   BasicScreenChooserSupplier<T>* shareWith;
    };
779
    template<class T> class BasicScreenChooser : public ParameterChooser<T>
781 {
    public:
783   typedef T NumberType;

785   BasicScreenChooser(wxWindow* parent, BasicScreenChooserSupplier<NumberType>* supplier)
        : ParameterChooser<NumberType>(parent),
787     supplier(supplier)
      {
789     std::stringstream stream;
        stream << "Screen " << (supplier->isXElseY() ? "X" : "Y");
791     myName = stream.str();
        /*myName = "Screen " + supplier->isXElseY() ? "X" : "Y";*/
793   }

795   virtual NumberType getValue()
      {
797     return supplier->getValue();
      }
799
      /**
801   * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
      */
803   virtual void selected()
      {
805
      }
807
      /**
809   * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
      **/
811   virtual void unselected()
      {
813
      }
815
    private:
817   BasicScreenChooserSupplier<T>* supplier;
    };
819
    typedef BasicScreenChooserSupplier<double> ScreenChooserSupplier;
821 typedef BasicScreenChooser<double> ScreenChooser;

823
    /**
825 * Reicht alles an einen uebergebenen NonPanelParameterChooser weiter, der den Vorteil hat, das sein
        wxPaneldasein für etwas anderes benutzt werden kann.
    **/
827 template<class R> class PassingChooser : public ParameterChooser<R>, public ParameterObserver<R>
```

```
  {
829 public:
    typedef R ReturnType;
831
    PassingChooser(wxWindow* parent, NonPanelParameterChooser<ReturnType>* passedTo, std::string textFieldText
        )
833   : ParameterChooser<ReturnType>(parent),
      passedTo(passedTo)
835   {
      wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);
837   wxPanel::SetSizer(sizer);
      wxStaticText* text = new wxStaticText(this, -1, textFieldText);
839   text->SetFont(AbstractParameterGui::FONT_2);
      sizer->Add(text);
841   passedTo->addObserver(this);
    }
843
    virtual ~PassingChooser()
845 {
      /*passedTo->removeObserver(this);*/
847 }

849 virtual Returning getValue()
    {
851   return passedTo->getValue();
    }
853
    virtual void setValue(Returning value)
855 {
      passedTo->setValue(value);
857 }

859 /**
    * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
861 */
    virtual void selected()
863 {
      passedTo->selected();
865 }

867 /**
    * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
869 **/
    virtual void unselected()
871 {
      passedTo->unselected();
873 }

875 virtual std::string getMyName() const
    {
877   return passedTo->getMyName();
    }
879
    NonPanelParameterChooser<ReturnType>* getPassedTo()
881 {
      return passedTo;
883 }

885 virtual void onChange(ParameterReturning newValue, Parameter<ParameterReturning>* sender)
    {
887   notifyObservers(newValue);
    }
889
    /*virtual void addObserver(Observer* observer)
891 {
      passedTo->addObserver(observer);
893 }

895 virtual void removeObserver(Observer* observer)
    {
897   passedTo->removeObserver(observer);
    }*/
899
  private:
901 NonPanelParameterChooser<ReturnType>* passedTo;
  };
903
  template<class T, class R, class B> bool sortPositionValuePair( std::pair< std::pair<T, R>, B> pair1, std::
      pair< std::pair<T, R>, B> pair2)
905 {
    return pair1.first.first->getValue() < pair2.first.first->getValue();
907 }

909 template<class R> class InterpolationObserver
  {
911 public:
    typedef R ReturnType;
913 typedef std::pair<double, ReturnType> InterpolationPair;
    typedef std::vector<InterpolationPair> InterpolationData;
915
    virtual void onInterpolationUpdate(const InterpolationData& data) = 0;
917
    virtual void onInterpolationDelete(){}
919 };

921 template<class R> class Interpolator : public InterpolationObserver<R>
  {
923 public:
    typedef R ReturnType;
```

```
925
    Interpolator()
927   : data(0),
      minValue(0),
929   multiplicator(0)
    {}
931
    virtual ReturnType interpolate(ReturnType one, double factor1, ReturnType two, double factor2) const = 0;
933
    // position must be between 0.0 and 1.0
935 virtual ReturnType getInterpolation(double position)
    {
937   double realPosition = position * multiplicator + minValue;
      if(!multiplicator)
939   {
        if(!data || !data->size())
941     {
          return ReturnType();
943     }
      }
945
      unsigned int i;
947   for(i = 0; i < data->size(); ++i)
      {
949     InterpolationPair pair = (*data)[i];
        if(pair.first >= realPosition)
951     {
          // maybe put out
953       if(pair.first == realPosition)
          {
955         return pair.second;
          }
957       i--;
          break;
959     }
      }
961   InterpolationPair pair1 = (*data)[i];
      InterpolationPair pair2 = (*data)[i + 1];
963   double distance = pair2.first - pair1.first;
      double factor1 = (realPosition - pair1.first) / distance;
965   return interpolate(pair1.second, 1 - factor1, pair2.second, factor1);
    }
967
    virtual void onInterpolationUpdate(const InterpolationData& data)
969 {
      this->data = &data;
971   if(data.size() == 0)
      {
973     multiplicator = minValue = 0.0;
        return;
975   }
      minValue = data[0].first;
977   double maxValue = data[data.size() - 1].first;
      multiplicator = maxValue - minValue;
979 }

981 protected:
      const InterpolationData* data;
983   double minValue, multiplicator;
    };
985
    template<class N> class NumberInterpolator : public Interpolator<N>
987 {
    public:
989   typedef N NumberType;

991   virtual ReturnType interpolate(ReturnType one, double factor1, ReturnType two, double factor2) const
      {
993     return one*factor1 + two*factor2;
      }
995 };

997 template<class R, class T> class InterpolatingParam : public Parameter<R>
    {
999 public:
      typedef R ReturnType;
1001  typedef T InterpolatorType;

1003  InterpolatingParam(Parameter<ReturnType>* param1, const double* factor1Pointer, Parameter<ReturnType>*
          param2, const double* factor2Pointer, InterpolatorType interpolator)
        : param1(param1),
1005    param2(param2),
        factor1Pointer(factor1Pointer),
1007    factor2Pointer(factor2Pointer),
        interpolator(interpolator)
1009  {}

1011  virtual R getValue()
      {
1013    return interpolator.interpolate(param1->getValue(), *factor1Pointer, param2->getValue(), *factor2Pointer
            );
      }
1015
    private:
1017  Parameter<ReturnType>* param1;
      Parameter<ReturnType>* param2;
1019  const double *factor1Pointer, *factor2Pointer;
      InterpolatorType interpolator;
1021  };
```

```
1023
    template<class R> class Interpolatable
1025 {
    public:
1027   typedef R ReturnType;

1029   virtual Parameter<ReturnType>* getInterpolator(const double* factor1Pointer, ReturnType* other, const
           double* factor2Pointer) = 0;
       /*virtual R getInterpolatedValue(const double* factor1Pointer, ReturnType* other, const double*
           factor2Pointer) = 0;*/
1031 };

1033 template <class R> class ChooserSupplier
     {
1035 public:
       typedef R ReturnType;
1037
       virtual void addChoosersToGui(ParameterGui<ReturnType>* gui, wxWindow* parent) = 0;
1039   virtual void removeChooser(ParameterGui<ReturnType>* gui, wxWindow* parent){}
     };
1041
     template<class R> class InterpolationChooser : public wxEvtHandler, public ParameterObserver<double>, public
         ParameterObserver<R>
1043 {
     public:
1045   typedef R ReturnType;

1047   // Chooses the interpolationposition.
       typedef TextfieldChooser<double> PositionChooser;
1049   typedef ParameterGui<ReturnType> ValueChooser;
       typedef std::pair<PositionChooser*, ValueChooser*> PositionValuePair;
1051   typedef wxBitmapButton DeleteButton;
       typedef std::pair<PositionValuePair, DeleteButton*> PositionValuePairWithDeleteButton;
1053
       typedef std::vector<PositionValuePairWithDeleteButton> InterpolationList;
1055
       typedef InterpolationObserver<ReturnType> Observer;
1057   typedef std::list<Observer *> ObserverList;


1059
       // sizer sollte ein Sizer des panels sein.
1061   InterpolationChooser(wxPanel* panel, wxGridBagSizer* sizer, std::string addButtonDescription, std::string
           rowDescription, int startRow, ChooserSupplier<ReturnType>* chooserSupplier)
         : panel(panel),
1063       sizer(sizer),
           add(new wxButton(panel, wxID_ADD, addButtonDescription)),
1065       startRow(startRow),
           chooserSupplier(chooserSupplier),
1067       data(),
           rowDescription(rowDescription)
1069   {
         sizer->Add(add, wxGBPosition(startRow, 0), wxGBSpan(1, 2), wxALL | wxEXPAND, 5);
1071
         panel->PushEventHandler(this);
1073     Connect(wxID_ADD, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(InterpolationChooser::
             addButtonClicked));
       }
1075
       void onChange(double newValue, Parameter<double>* sender)
1077   {
         std::sort(list.begin(), list.end(), sortPositionValuePair<PositionChooser*, ValueChooser*, DeleteButton
             *>);
1079
         data.clear();
1081     maxPosition = 0;

1083     int row = startRow + 1;
         for(InterpolationList::iterator it = list.begin(); it != list.end(); ++it)
1085     {
           ValueChooser* chooser = it->first.second;
1087       if(chooser->getOwnRow() != row)
           {
1089         chooser->removeFromPanel(panel, sizer);
             sizer->Detach(it->first.first);
1091         sizer->Detach(it->second);
           }
1093
           data.push_back(Observer::InterpolationPair(it->first.first->getValue(), chooser->getValue()));
1095
           ++row;
1097     }

1099     row = startRow + 1;
         for(InterpolationList::iterator it = list.begin(); it != list.end(); ++it)
1101     {
           ValueChooser* chooser = it->first.second;
1103       if(chooser->getOwnRow() != row)
           {
1105         // chooser->removeFromPanel(panel, sizer);
             addGuiToPanel(*it, row);
1107
             // Change number of DeleteButton
1109         DeleteButton* deleteButton = it->second;
             int i = row - startRow - 1;
1111         std::stringstream stream;
             stream << i;
1113         deleteButton->SetName(stream.str());
           }
```

```
1115            ++row;
                }
1117
                panel->SendSizeEvent();
1119            panel->Refresh(true);
                // panel->SendSizeEventToParent();
1121

1123            notifyObservers();
            }
1125
            virtual void onChange(ReturnType newValue, Parameter<ReturnType>* sender)
1127        {
                // parameter unimportant
1129            onChange(0.0, 0);
            }
1131
            void addGui()
1133        {
                int row = startRow + list.size() + 1;
1135            double* doubleValue = new double;
                PositionChooser* position = new PositionChooser(panel, "Textfield", "100.0", NumberValidator<double>(
                    doubleValue), std::auto_ptr<double>(doubleValue));
1137            position->addObserver(this);
                // position->PushEventHandler(this);
1139
                ValueChooser* value = new ValueChooser(rowDescription, panel, sizer);
1141            chooserSupplier->addChoosersToGui(value, panel);

1143            std::stringstream stream;
                stream << list.size();
1145            DeleteButton* deleteButton = new DeleteButton(panel, wxID_DELETE, wxBitmap("delete.png",
                    wxBITMAP_TYPE_PNG));
                deleteButton->SetToolTip("Delete Colour");
1147            deleteButton->SetName(stream.str());

1149            Connect(wxID_DELETE, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(InterpolationChooser::
                    onDeleteClicked));

1151            PositionValuePairWithDeleteButton pair( PositionValuePair(position, value), deleteButton);
                addGuiToPanel(pair, row);
1153
                list.push_back(pair);
1155            panel->SendSizeEvent();
            }
1157
            void deleteAllValues()
1159        {
                for(InterpolationList::iterator it = list.begin(); it != list.end(); ++it)
1161            {
                    chooserSupplier->removeChooser((*it).first.second, panel);
1163            }
            }
1165
            void onDeleteClicked(wxCommandEvent& evt)
1167        {
                wxButton* button = dynamic_cast<wxButton*>(evt.GetEventObject());
1169            if(!button)
                {
1171                return;
                }
1173            std::stringstream stream;
                stream << button->GetName();
1175            int row = 0;
                stream >> row;
1177            if(stream.fail() || !stream.eof())
                {
1179                // Wrong name, doesn't convert to int
                    return;
1181            }
                chooserSupplier->removeChooser(list[row].first.second, panel);
1183            removeGui(list[row], row);
                // Arguments unimportant
1185            onChange(0.0, 0);
            }
1187
            void removeGui(PositionValuePairWithDeleteButton& pair, int position)
1189        {
                // list.remove(pair);
1191            pair.first.second->removeFromPanel(panel, sizer);

1193            sizer->Detach(pair.first.first);
                pair.first.first->Destroy();
1195
                cleanupGui(pair.first);
1197
                /*sizer->Detach(pair.second);
1199            panel->RemoveChild(pair.second);*/

1201            // DeleteButtons ARE switched together with their pairs...

1203            // Swap element with end
                /*PositionValuePairWithDeleteButton& endButton = list.back();
1205            PositionValuePair buffer = endButton.first;
                endButton.first = list[*/
1207            /*std::swap(pair.first, list.back().first);

1209            wxButton* endButton = list.back().second;*/
                sizer->Detach(pair.second);
```

```
1211        pair.second->Destroy();
            list.erase(list.begin() + position);
1213      }

1215      void addButtonClicked(wxCommandEvent& evt)
          {
1217        addGui();
          }
1219
          void addInterpolationObserver(InterpolationObserver<ReturnType>* observer)
1221      {
            observerList.push_back(observer);
1223      }

1225      void removeInterpolationObserver(InterpolationObserver<ReturnType>* observer)
          {
1227        observerList.remove(observer);
          }
1229
          void notifyObservers()
1231      {
            for(ObserverList::iterator it = observerList.begin(); it != observerList.end(); ++it)
1233        {
              (*it)->onInterpolationUpdate(data);
1235        }
          }
1237
          virtual ~InterpolationChooser()
1239      {
            // TODO: Maybe the superclass destructors must be declared virtual, too.
1241        panel->RemoveEventHandler(this);
            for(InterpolationList::iterator it = list.begin(); it != list.end(); ++it)
1243        {
              cleanupGui(it->first);
1245        }
            for(ObserverList::iterator it = observerList.begin(); it != observerList.end(); ++it)
1247        {
              // (*it)->onInterpolationDelete();
1249        }
          }
1251
          /*
1253        Compares two PositionValuePairs

1255      bool operator() (PositionValuePair& pair1, PositionValuePair& pair2)
          {
1257        return pair1.first->getValue() < pair2.first->getValue();
          }*/
1259
        private:
1261
          void cleanupGui(PositionValuePair& pair)
1263      {
            // pair.first->RemoveEventHandler(this);
1265        // pair.second->removeFromPanel(panel, sizer);
            delete pair.second;
1267      }

1269      void addGuiToPanel(PositionValuePairWithDeleteButton& pair, int row)
          {
1271        pair.first.second->addToPanel(row);
            sizer->Add(pair.first.first, wxGBPosition(row, 3), wxDefaultSpan, wxALL, 5);
1273        sizer->Add(pair.second, wxGBPosition(row, 4), wxDefaultSpan, wxALL, 5);
          }
1275
          wxPanel* panel;
1277      wxGridBagSizer* sizer;
          InterpolationList list;
1279      wxButton* add;
          int startRow;
1281      ChooserSupplier<ReturnType>* chooserSupplier;
          ObserverList observerList;
1283      typename Observer::InterpolationData data;
          double maxPosition;
1285      std::string rowDescription;
        };
1287
        //template<class T> class ChooserSupplier
1289    //{
        //public:
1291    //  typedef T Type;

1293    //  virtual void addChoosersToGui(ParameterGui<Type> gui, wxPanel* parent) = 0;
        //};
1295
      }
1297
      #endif
```

Quellcode/Parameters.h

Datei Calculators.h

```
#ifndef IMAGE_CALCULATOR_H
#define IMAGE_CALCULATOR_H

#include <complex>
#include <vector>
#include <list>
#include "wx/wx.h"
#include "Parameters.h"
#include "muParser/muParser.h"

namespace Calculators
{

  typedef double FormulaParam;
  typedef int AlgorythmReturning;

  // typedef double FormulaReturning;

  class Formula : public Parameters::Parameter<Formula*>
  {
  public:
    virtual void operator() (FormulaParam* real, FormulaParam* imag, FormulaParam* j, FormulaParam* k) = 0;
    virtual void prepare() = 0;

    virtual std::auto_ptr<Parameters::Parameter<Formula*> > getInterpolator(const double* factor1Pointer,
        Formula* other, const double* factor2Pointer) = 0;
  };

  class Algorythm : public Parameters::Parameter<Algorythm*>
  {
  public:
    virtual void prepare() = 0;
    virtual AlgorythmReturning operator() () const = 0 ;
    virtual int getMaxReturning() const = 0;

    virtual std::auto_ptr<Parameters::Parameter<Algorythm*> > getInterpolator(const double* factor1Pointer,
        Algorythm* other, const double* factor2Pointer) = 0;
  };

  class JuliaEscapeTime : public Algorythm
  {
  public:
    typedef Parameters::InterpolatingParam<FormulaParam, Parameters::NumberInterpolator<FormulaParam> >
        FormulaParamInterpolator;

    JuliaEscapeTime(Parameters::Parameter<Formula*>* formula, Parameters::Parameter<FormulaParam>* zRe,
        Parameters::Parameter<FormulaParam>* zIm, Parameters::Parameter<FormulaParam>* zJ, Parameters::
        Parameter<FormulaParam>* zK, Parameters::Parameter<AlgorythmReturning>* maxIterations, Parameters::
        Parameter<FormulaParam>* maxValue);

    virtual void prepare();
    virtual AlgorythmReturning operator() () const;
    virtual int getMaxReturning() const;

    virtual std::auto_ptr<Parameters::Parameter<Algorythm*> > getInterpolator(const double* factor1Pointer,
        Algorythm* other, const double* factor2Pointer);
    virtual JuliaEscapeTime* getValue();

  private:
    Formula* theFormula;
    Parameters::Parameter<Formula*>* formula;
    Parameters::Parameter<FormulaParam>* zRe;
    Parameters::Parameter<FormulaParam>* zIm;
    Parameters::Parameter<FormulaParam>* zJ;
    Parameters::Parameter<FormulaParam>* zK;
    Parameters::Parameter<AlgorythmReturning>* maxIterations;
    Parameters::Parameter<FormulaParam>* maxValue;
  };

  class JuliaEscapeTimeChooser : public Parameters::ParameterChooser<Algorythm*>
  {
  public:
    JuliaEscapeTimeChooser(Parameters::ParametersPanel* parametersPanel, Parameters::ScreenChooserSupplier*
        screenX, Parameters::ScreenChooserSupplier* screenY, Parameters::ChooserSupplier<double>*
        chooserSupplier);

    /**
    * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
    **/
    virtual void selected();

    /**
    * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
    **/
    virtual void unselected();

    virtual Calculators::Algorythm* getValue(void);
  private:

    void addParamChoosersForFormula(Parameters::ParameterGui<FormulaParam>* gui);

    Parameters::ParametersPanel& parametersPanel;
    std::auto_ptr<Parameters::ParameterGui<Formula*> > formulaGui;
    std::auto_ptr<Parameters::ParameterGui<int> > numIterationsGui;
    std::auto_ptr<Parameters::ParameterGui<FormulaParam> > maxValueGui;
    std::auto_ptr<Parameters::ParameterGui<FormulaParam> > zReGui, zImGui, zJGui, zKGui, cReGui, cImGui, cJGui
        , cKGui;
    Parameters::ScreenChooserSupplier *screenX, *screenY;
    std::auto_ptr<JuliaEscapeTime> escapeTime;
```

```
90      Parameters::ChooserSupplier<double>* chooserSupplier;
    };
92
    class AbstractCalculator : public Parameters::Parameter<AbstractCalculator*>
94  {
    public:
96    // returning must be a 2-dimension array with width and height as its first and second dimension.
      /**
98    * @returns the maximum value in returning
      **/
100   virtual int calculate(AlgorythmReturning** returning, int width, int height) const = 0;
      virtual std::auto_ptr<Parameters::Parameter<AbstractCalculator*> > getInterpolator(const double*
          factor1Pointer, AbstractCalculator* other, const double* factor2Pointer) = 0;
102 };

104 class Calculator : public AbstractCalculator
    {
106 public:

108   Calculator(Parameters::Parameter<Algorythm*>* algorythm, Parameters::ScreenChooserSupplier* screenX,
          Parameters::ScreenChooserSupplier* screenY);

110   // returning must be a 2-dimension array with width and height as its first and second dimension.
      /**
112   * @returns the maximum value in returning
      **/
114   virtual int calculate(AlgorythmReturning** returning, int width, int height) const;

116   virtual std::auto_ptr<Parameters::Parameter<AbstractCalculator*> > getInterpolator(const double*
          factor1Pointer, AbstractCalculator* other, const double* factor2Pointer);

118   virtual Calculator* getValue();

120 private:
      Parameters::Parameter<Algorythm*>* algorythm;
122   Parameters::ScreenChooserSupplier *screenX, *screenY;
    };
124
    class CalculatorChooser : public Parameters::ParameterChooser<AbstractCalculator*>
126 {
    public:
128   CalculatorChooser(Parameters::ParametersPanel* parametersPanel, Parameters::ScreenChooserSupplier* screenX
          , Parameters::ScreenChooserSupplier* screenY, Parameters::ChooserSupplier<double>* chooserSupplier);

130   virtual Returning getValue();

132   /**
      * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
134   */
      virtual void selected();
136
      /**
138   * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
      **/
140   virtual void unselected();

142 private:
      Parameters::ScreenChooserSupplier *screenX, *screenY;
144   Parameters::ParametersPanel& parametersPanel;
      std::auto_ptr<Parameters::ParameterGui<Algorythm*> > algorythmParam;
146   std::auto_ptr<Calculator> calculator;
      Parameters::ChooserSupplier<double>* chooserSupplier;
148 };

150 class Interpreter : public wxPanel
    {
152 public:

154   Interpreter(wxWindow* parent);

156   virtual void update() = 0;

158   // virtual wxPanel* getPreviewPanel() = 0;
      // void saveToFile(std::string file);
160 };

162 class StandardFormula : public Formula
    {
164 public:
      StandardFormula(Parameters::Parameter<FormulaParam>* cRe, Parameters::Parameter<FormulaParam>* cIm,
          Parameters::Parameter<FormulaParam>* cJ, Parameters::Parameter<FormulaParam>* cK);
166
      virtual void operator() (FormulaParam* real, FormulaParam* imag, FormulaParam* j, FormulaParam* k);
168

170   virtual void prepare();

172   virtual std::auto_ptr<Parameters::Parameter<Formula*> > getInterpolator(const double* factor1Pointer,
          Formula* other, const double* factor2Pointer);

174   virtual Formula* getValue();

176 private:
      Parameters::Parameter<FormulaParam>* cRe;
178   Parameters::Parameter<FormulaParam>* cIm;
      Parameters::Parameter<FormulaParam>* cJ;
180   Parameters::Parameter<FormulaParam>* cK;
    };
182
```

```
  class StandardFormulaChooser : public Parameters::ParameterChooser<Formula*>
184 {
  public:
186
    StandardFormulaChooser(Parameters::ParametersPanel* parametersPanel, Parameters::ParameterGui<FormulaParam
        >* cReGui, Parameters::ParameterGui<FormulaParam>* cImGui, Parameters::ParameterGui<FormulaParam>*
        cJGui, Parameters::ParameterGui<FormulaParam>* cKGui);
188

190   /**
    * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
192 **/
    virtual void selected();
194
    /**
196 * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
    **/
198 virtual void unselected();

200   virtual Formula* getValue();

202 private:
    Parameters::ParametersPanel& parametersPanel;
204 Parameters::ParameterGui<FormulaParam> *cReGui, *cImGui, *cJGui, *cKGui;
    // FormulaParam cRe, cIm, cJ, cK;
206 std::auto_ptr<StandardFormula> formula;
  };
208
  class CustomFormula : public Formula
210 {
  public:
212   CustomFormula(wxString formulaText, Parameters::Parameter<FormulaParam>* cRe, Parameters::Parameter<
        FormulaParam>* cIm, Parameters::Parameter<FormulaParam>* cJ, Parameters::Parameter<FormulaParam>* cK)
        ;

214   virtual void operator() (FormulaParam* real, FormulaParam* imag, FormulaParam* j, FormulaParam* k);

216   virtual void prepare();

218   virtual std::auto_ptr<Parameters::Parameter<Formula*> > getInterpolator(const double* factor1Pointer,
        Formula* other, const double* factor2Pointer);

220   virtual Formula* getValue();

222   virtual wxString getFormulaText();

224 private:
    mu::Parser parser;
226 wxString formulaText;
    Parameters::Parameter<FormulaParam>* cRe;
228 Parameters::Parameter<FormulaParam>* cIm;
    Parameters::Parameter<FormulaParam>* cJ;
230 Parameters::Parameter<FormulaParam>* cK;

232   double zr, zi, zj, zk, cr, ci, cj, ck;

234 };

236 class CustomFormulaChooser : public Parameters::ParameterChooser<Formula*>
  {
238 public:

240   CustomFormulaChooser(Parameters::ParametersPanel* parametersPanel, Parameters::ParameterGui<FormulaParam>*
        cReGui, Parameters::ParameterGui<FormulaParam>* cImGui, Parameters::ParameterGui<FormulaParam>*
        cJGui, Parameters::ParameterGui<FormulaParam>* cKGui);


242
    /**
244 * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
    **/
246 virtual void selected();

248   /**
    * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
250 **/
    virtual void unselected();
252
    virtual Formula* getValue();
254
    virtual void setValue(Formula* value);
256
  private:
258   Parameters::ParametersPanel& parametersPanel;
    Parameters::ParameterGui<FormulaParam> *cReGui, *cImGui, *cJGui, *cKGui;
260 // FormulaParam cRe, cIm, cJ, cK;
    std::auto_ptr<CustomFormula> formula;
262 wxTextCtrl* text;
  };
264
}
266
#endif
```

Quellcode/Calculators.h

Datei Juliamengen.cpp

```
#include "Juliamengen.h"
#include <wx/rawbmp.h>
#include <wx/dcbuffer.h>
#include <Windows.h>
#include <wx/colordlg.h>
#include <wx/statbmp.h>

using Gui::JuliamengenApp;
using Gui::MainFrame;
using Gui::ImageComputor;
using Gui::ImageInterpreter;
using Gui::FractalPanel;
using Gui::ColourParameterChooser;
using Gui::ColourChooser;
using Gui::ColourInterpolator;
using Gui::ColourInterpolatorInterpolator;
using Gui::ComponentStorage;
using Gui::MovieChooser;
using Gui::FractalInterpolator;
using Gui::Point;
using Gui::PointSelectionObserver;
using Gui::PointChooser;

const int ID_NEW_COLOR_CHOOSER = 15;
const int ID_NEW_MOVIE = 16;
const int ID_NEW_POINT = 17;
const int ID_NAME_CHANGE = 18;
const int ID_SELECT_POINT = 19;
const int ID_NEW_JULIA = 20;

bool JuliamengenApp::OnInit()
{
  MainFrame* frame = new MainFrame(_("Juliamengen"), wxPoint(100, 100), wxSize(600, 600));
  frame->Show(true);
  SetTopWindow(frame);

  // Calculators::calculateJuliaEscapeTime(0, 0, 0, std::complex<double>(0.23, 1.03), std::complex<double
      >(0.26, 0.34));

  return true;
}


MainFrame::MainFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
  : wxMDIParentFrame(NULL, -1, title, pos, size),
  interpolatorStorage(),
  standardInterpolationData(),
  interpolatorPointer(&interpolatorStorage),
  standardInterpolator("Standard Interpolator"),
  fractalStorage(),
  fractalPointer(&fractalStorage),
  pointStorage(),
  pointPointer(&pointStorage),
  activePoint(0)
  /*juliaCalculator(std::complex<double>(-0.7805907172995781, 0.13247011952191223)),
  appleCalculator()*/
{
  // wxPanel* panel = new wxPanel(this, -1);
  wxInitAllImageHandlers();
  // SetIcon(wxIcon("icon.png", wxBITMAP_TYPE_PNG));
  wxMenuBar* menu = new wxMenuBar;
  SetMenuBar(menu);

  // wxImage::InsertHandler(new wxPNGHandler);
  wxToolBar* bar = CreateToolBar();
  bar->AddTool(wxID_NEW, "new window", wxBitmap("mandelbroticon.png", wxBITMAP_TYPE_PNG), "New Mandelbrot-Set"
      );
  bar->AddTool(ID_NEW_JULIA, "new julia", wxBitmap("juliaicon.png", wxBITMAP_TYPE_PNG), "New Julia-Set");
  bar->AddTool(ID_NEW_COLOR_CHOOSER, "new colourchooser", wxBitmap("color.png", wxBITMAP_TYPE_PNG), "New
      colour chooser");
  bar->AddTool(ID_NEW_MOVIE, "new movie", wxBitmap("movies2.png", wxBITMAP_TYPE_PNG), "New movie");
  bar->AddTool(ID_NEW_POINT, "new point", wxBitmap("point.png", wxBITMAP_TYPE_PNG), "New point");

  bar->Realize();

  standardInterpolationData.push_back(ColourInterpolator::InterpolationPair(0.0, wxColour(255, 0, 0)));
  standardInterpolationData.push_back(ColourInterpolator::InterpolationPair(10.0, wxColour(0.0, 0.0, 0.0)));
  standardInterpolator.onInterpolationUpdate(standardInterpolationData);
  interpolatorStorage.addComponent(&standardInterpolator);

  Connect(wxID_NEW, wxEVT_COMMAND_TOOL_CLICKED, wxCommandEventHandler(MainFrame::onNewClick));
  Connect(ID_NEW_JULIA, wxEVT_COMMAND_TOOL_CLICKED, wxCommandEventHandler(MainFrame::onNewJuliaClick));
  Connect(ID_NEW_COLOR_CHOOSER, wxEVT_COMMAND_TOOL_CLICKED, wxCommandEventHandler(MainFrame::
      onNewColourChooserClick));
  Connect(ID_NEW_MOVIE, wxEVT_COMMAND_TOOL_CLICKED, wxCommandEventHandler(MainFrame::onNewMovieClick));
  Connect(ID_NEW_POINT, wxEVT_COMMAND_TOOL_CLICKED, wxCommandEventHandler(MainFrame::onNewPointClick));

  newPoint(true);

  /*wxBoxSizer* sizer = new wxBoxSizer(wxHORIZONTAL);
  panel->SetSizer(sizer);

  std::complex<double> c(-0.7805907172995781, 0.13247011952191223);*/

  /*FILETIME now;
  GetSystemTimeAsFileTime(&now);
```

```
93    long time = now.dwLowDateTime;
      Calculators::generateBitmap(&bitmap, 1200, 1200, Calculators::CalculateJuliaEscapeTime(c));
95    GetSystemTimeAsFileTime(&now);
      time = now.dwLowDateTime - time;
97    std::stringstream stream;
      stream << "Time needed: ";
99    stream << time;
      wxMessageBox(stream.str(), _("Hallo"));*/
101
      /*wxScrolledWindow* scroll = new wxScrolledWindow(panel);
103   ImageInterpreter* displayer = new ImageInterpreter(scroll, &appleCalculator, 600);
      sizer->Add(scroll, 1, wxEXPAND | wxALL, 10);*/
105 }

107 MainFrame::~MainFrame()
    {
109   interpolatorPointer = 0;
      fractalPointer = 0;
111   pointPointer = 0;
    }
113
    FractalPanel* MainFrame::newFractalPanel(Point* point)
115 {
      FractalPanel* panel = new FractalPanel(this, "Fractal", &interpolatorPointer, &fractalPointer, &pointPointer
          , point);
117   /*fractalPanels.push_back(panel);*/
      fractalStorage.addComponent(panel);
119   panel->SetSize(500, 600);
      panel->updateImageInterpreter();
121   return panel;
    }
123
    void MainFrame::onNewClick(wxCommandEvent& ev)
125 {
      newFractalPanel();
127 }

129 void MainFrame::onNewJuliaClick(wxCommandEvent& ev)
    {
131   wxArrayString choices;
      ComponentStorage<Point>::ComponentList* points = pointStorage.getComponents();
133   for(ComponentStorage<Point>::ComponentList::iterator it = points->begin(); it != points->end(); ++it)
      {
135     choices.Add((*it)->getMyName());
      }
137   wxSingleChoiceDialog* dialog = new wxSingleChoiceDialog(this, "Please select the point, for which the Julia-
          Set shall be created", "Juliamengen", choices);
      if(dialog->ShowModal() != wxID_OK)
139   {
        dialog->Destroy();
141     return;
      }
143
      int selected = dialog->GetSelection();
145   ComponentStorage<Point>::ComponentList::iterator it = points->begin();
      for(int i = 0; i < selected; ++it, ++i){}
147   dialog->Destroy();

149   newFractalPanel((*it)->getValue());

151 }

153 void MainFrame::newColourChooser()
    {
155   ColourChooser* chooser = new ColourChooser(this, "Colour Chooser", &interpolatorPointer);
      chooser->SetSize(500, 500);
157
      interpolatorStorage.addComponent(chooser->getInterpolator());
159   /*interpolators.push_back(chooser->getInterpolator());
      for(InterpolatorObserverList::iterator it = interpolatorObservers.begin(); it != interpolatorObservers.end()
          ; ++it)
161   {
        (*it)->addChooser(new Parameters::PassingChooser<ColourInterpolator*> ((*it)->getParent(), chooser->
            getInterpolator()) );
163   }*/
    }
165
    void MainFrame::onNewColourChooserClick(wxCommandEvent& ev)
167 {
      newColourChooser();
169 }

171 void MainFrame::newMovie()
    {
173   MovieChooser* chooser = new MovieChooser(this, "Movie Chooser", &fractalPointer);
      chooser->SetSize(500, 500);
175 }

177 void MainFrame::onNewMovieClick(wxCommandEvent& ev)
    {
179   newMovie();
    }
181
    void MainFrame::newPoint(bool invisible)
183 {
      Point* point = new Point(this, invisible ? "Ursprung" : "Point", &pointPointer, this, &activePoint);
185   // point->SetSize(434, 65);
      point->SetSize(460, 65);
187   pointStorage.addComponent(point);
```

```
    if(invisible)
189 {
      point->Show(false);
191 }
    /*point->SetMaxSize(wxSize(435, 74));
193 point->SetMinSize(wxSize(434, 73));*/
}
195
void MainFrame::onNewPointClick(wxCommandEvent& evt)
197 {
    newPoint();
199 }

201 void MainFrame::setPointToSelect(Point* point)
{
203 ComponentStorage<ImageComputor>::ComponentList* components = fractalStorage.getComponents();
    for(ComponentStorage<ImageComputor>::ComponentList::iterator it = components->begin(); it != components->end
        (); ++it)
205 {
      PointSelectionObserver* panel = dynamic_cast<PointSelectionObserver*>(*it);
207 if(panel)
      {
209 panel->setPointToSelect(point);
      }
211 }
}
213
ComponentStorage<ColourInterpolator> MainFrame::getInterpolatorStorage()
215 {
    return interpolatorStorage;
217 }

219 FractalPanel::FractalPanel(MainFrame* parent, std::string name, ComponentStorage<ColourInterpolator>**
      interpolatorStorage, ComponentStorage<ImageComputor>** fractalStorage, ComponentStorage<Point>**
      pointStorage, Point* juliaForPoint)
    : wxMDIChildFrame(parent, -1, name),
221 /*Algorythm(new Calculators::JuliaEscapeTime()),
    converters(),
223 calculator(Algorythm.get(), &converters),*/
    splitter(new wxSplitterWindow(this, -1, wxDefaultPosition, wxDefaultSize, wxSP_NO_XP_THEME | wxSP_3D |
        wxSP_LIVE_UPDATE )),
225 topScroll(new wxScrolledWindow(splitter)),
    bottomScroll(new wxScrolledWindow(splitter)),
227 parametersPanel(new Parameters::ParametersPanel(bottomScroll)),
    parameterGui("Interpreter", parametersPanel, parametersPanel->getSizer()),
229 gui2("c", parametersPanel, parametersPanel->getSizer()),
    nameGui("name", parametersPanel, parametersPanel->getSizer()),
231 mainFrame(parent),
    fractalStorage(fractalStorage)
233 {

235 wxBoxSizer* sizer = new wxBoxSizer(wxHORIZONTAL);
    SetSizer(sizer);
237
    wxString* namePointer = new wxString;
239 nameGui.addChooser(new Parameters::TextfieldChooser<wxString>(parametersPanel, "test", name, wxTextValidator
        (0, namePointer), std::auto_ptr<wxString>(namePointer)));
    nameGui.addObserver(this);
241 parametersPanel->addParameter(&nameGui);

243 imageInterpreter = new ImageInterpreter(topScroll, *parametersPanel, interpolatorStorage, pointStorage,
        juliaForPoint);

245 // parameterGui.addChooser(Parameters::ParameterGui<int>::ChooserPtr(new Parameters::NumberChooser<int>(
        parametersPanel, "c", 10, 0, 50)));
    parameterGui.addChooser(Parameters::ParameterGui<Calculators::Interpreter*>::ChooserPtr(new Parameters::
        PassingChooser<Calculators::Interpreter*>(parametersPanel, imageInterpreter, imageInterpreter->
        getMyName())));
247 parametersPanel->addParameter(&parameterGui);

249 gui2.addChooser(Parameters::ParameterGui<int>::ChooserPtr(new Parameters::NumberChooser<int>(
        parametersPanel, "rfff", 10, 0, 50)));
    gui2.addChooser(Parameters::ParameterGui<int>::ChooserPtr(new Parameters::NumberChooser<int>(
        parametersPanel, "isdf", 5, -20, 50)));
251 // parametersPanel->addParameter(&gui2);

253 wxBoxSizer* bottomScrollSizer = new wxBoxSizer(wxVERTICAL);

255 wxBoxSizer* buttonSizer = new wxBoxSizer(wxHORIZONTAL);
    wxButton* update = new wxButton(bottomScroll, wxID_OK, "Update");
257 wxButton* clone = new wxButton(bottomScroll, wxID_COPY, "Clone");
    wxButton* zoomOut = new wxButton(bottomScroll, wxID_UP, "Zoom out");
259 wxButton* save = new wxButton(bottomScroll, wxID_SAVE, "Save");
    buttonSizer->Add(clone, 1, wxCENTER | wxALL, 10);
261 buttonSizer->Add(update, 1, wxCENTER | wxALL, 10);
    buttonSizer->Add(zoomOut, 1, wxCENTER | wxALL, 10);
263 buttonSizer->Add(save, 1, wxCENTER | wxALL, 10);

265 bottomScrollSizer->Add(buttonSizer, 0, wxCENTER | wxALL, 10);
    bottomScrollSizer->Add(parametersPanel, 1, wxEXPAND);
267 bottomScroll->SetSizer(bottomScrollSizer);
    bottomScroll->SetScrollRate(5, 5);
269 splitter->SplitHorizontally(topScroll, bottomScroll);
    // splitter->SplitVertically(topScroll, bottomScroll);
271 sizer->Add(splitter, 1, wxEXPAND);
    // sizer->Add(parametersPanel, 1, wxEXPAND);
273 splitter->SetMinimumPaneSize(20);
    splitter->SetSashGravity(0.7);
275
```

```
    Connect(wxID_OK, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(FractalPanel::onUpdateClick));
277 Connect(wxID_COPY, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(FractalPanel::onCloneClick));
    Connect(wxID_UP, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(FractalPanel::onZoomOutClick));
279 Connect(wxID_SAVE, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(FractalPanel::onSaveClick));

281 // imageInterpreter->update();
}
283
FractalPanel::~FractalPanel()
285 {
    if(*fractalStorage)
287 {
      (*fractalStorage)->removeComponent(this);
289 }
}
291
void FractalPanel::onChange(wxString newValue, Parameter<wxString>* sender)
293 {
    SetTitle(newValue);
295 myName = newValue;
    notifyObservers(getValue());
297 }

299 void FractalPanel::onUpdateClick(wxCommandEvent& ev)
{
301 /*SetTitle(nameGui.getValue());
    myName = nameGui.getValue();*/
303 parameterGui.getValue()->update();
    notifyObservers(getValue());
305 }

307 Parameters::ParametersPanel* FractalPanel::getParametersPanel()
{
309 return parametersPanel;
}
311
void FractalPanel::onCloneClick(wxCommandEvent& evt)
313 {
    FractalPanel* copy = mainFrame->newFractalPanel();
315 Parameters::ParametersPanel* otherParameters = copy->getParametersPanel();
    parametersPanel->copyTo(otherParameters);
317 copy->imageInterpreter->update();
}
319
void FractalPanel::onZoomOutClick(wxCommandEvent& evt)
321 {
    imageInterpreter->zoomOut();
323 }

325 void FractalPanel::onSaveClick(wxCommandEvent& evt)
{
327 imageInterpreter->save();
}
329
ImageComputor* FractalPanel::getValue()
331 {
    return imageInterpreter->getComputor();
333 }

335 void FractalPanel::updateImageInterpreter()
{
337 imageInterpreter->update();
}
339
void FractalPanel::setPointToSelect(Point* point)
341 {
    imageInterpreter->setPointToSelect(point);
343 }

345 /**
* Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
347 */
void FractalPanel::selected()
349 {

351 }

353 /**
* Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
355 **/
void FractalPanel::unselected()
357 {

359 }

361 ImageComputor::ImageComputor(Parameters::Parameter<Calculators::AbstractCalculator*>* calculatorParam,
      Parameters::Parameter<ColourInterpolator*>* interpolatorParam, Parameters::Parameter<double>* startX,
      Parameters::Parameter<double>* endX, Parameters::Parameter<double>* startY, Parameters::Parameter<double
      >* endY, Parameters::ScreenChooserSupplier* screenX, Parameters::ScreenChooserSupplier* screenY)
  : calculatorParam(calculatorParam),
363 interpolatorParam(interpolatorParam),
    width(0),
365 height(0),
    numIterations(0),
367 bitmap(1, 1),
    startX(startX),
369 startY(startY),
    endX(endX),
371 endY(endY),
```

```
    screenX(screenX),
373 screenY(screenY)
{
375
}
377
ImageComputor::~ImageComputor()
379 {
  if(numIterations)
381 {
    delete[] numIterations;
383 }
}
385
/**
387 Height shall be -1 if it shall be determined by the ratio
*/
389 wxBitmap& ImageComputor::computeImage(int theWidth, int theHeight)
{
391 int width;
  int height;
393
  double startX = this->startX->getValue();
395 double startY = this->startY->getValue();
  double endX = this->endX->getValue();
397 double endY = this->endY->getValue();

399 if(theHeight == -1)
  {
401 int size = theWidth;
    if(endX - startX > endY - startY)
403 {
      width = size;
405   height = size * ((endY - startY) / (endX - startX));
    }else
407 {
      height = size;
409   width = size * ((endX - startX) / (endY - startY));
    }
411 }else
  {
413 width = theWidth;
    height = theHeight;
415
    double wantedRatio = (double) width / (double) height;
417
    // double realRatio = (endX - startX) / (endY - startY);
419 double neededEndXMinusStartX = wantedRatio * (endY - startY);
    double differencePer2 = (neededEndXMinusStartX - (endX - startX)) / 2;
421 startX -= differencePer2;
    endX += differencePer2;
423
    /*if(endX - startX > endY - startY)
425 {
      double something = (endX - startX) / theWidth;
427   endY = endY + something * theHeight;
    }else
429 {
      double something = (endY - startY) / theHeight;
431   endX = endX + something * theWidth;
    }*/
433 }

435 screenX->setBounds(startX, endX);
  screenY->setBounds(startY, endY);
437 screenX->setNumPixels(width);
  screenY->setNumPixels(height);
439
  this->width = width;
441 this->height = height;

443 if(numIterations)
  {
445 delete[] numIterations;
  }
447 numIterations = new Calculators::AlgorythmReturning[width * height];
  Calculators::AbstractCalculator* calculator = calculatorParam->getValue();
449 maxIterations = calculator->calculate(&numIterations, width, height);

451 updateOnlyColour();

453 return bitmap;
}
455
wxBitmap& ImageComputor::updateOnlyColour()
457 {
  if(!numIterations)
459 {
    return bitmap;
461 }

463 wxImage image(width, height);
  unsigned char* data = static_cast<unsigned char*> (malloc(width * height * 3));
465
  ColourInterpolator* interpolator = interpolatorParam->getValue();
467 int index = 0;
  for(int y = 0; y < height; ++y)
469 {
    for(int x = 0; x < width; ++x)
```

```
471     {
          unsigned char* r = &data[index++];
473       unsigned char* g = &data[index++];
          unsigned char* b = &data[index++];
475       if(numIterations[y * width + x] > -1)
          {
477         wxColour colour(interpolator->getInterpolation( (double) numIterations[y * width + x] / (double)
                maxIterations));
            *r = colour.Red();
479         *g = colour.Green();
            *b = colour.Blue();
481       }else
          {
483         *r = *g = *b = 0;
          }
485       /**r = *g = *b = 0;
          if(numIterations[y * width + x] > -1)
487       {
            *r = (maxIterations - numIterations[y * width + x]) * (255 / maxIterations);
489       }else
          {
491         *r = 0;
          }*/
493     }
    }
495
    // delete[] numIterations;
497 image.SetData(data);
    bitmap = wxBitmap(image);
499 image.Destroy();
    return bitmap;
501 }

503 class ImageComputorParameter : public Parameters::Parameter<ImageComputor*>
    {
505 public:
      ImageComputorParameter(std::auto_ptr<ImageComputor> computor, std::auto_ptr<ColourInterpolator> interpolator
            , std::auto_ptr< Parameters::Parameter<Calculators::AbstractCalculator*> > calculator, std::auto_ptr<
            Parameters::Parameter<double> > startX, std::auto_ptr<Parameters::Parameter<double> > endX, std::
            auto_ptr<Parameters::Parameter<double> > startY, std::auto_ptr<Parameters::Parameter<double> > endY)
507     : computor(computor),
        interpolator(interpolator),
509     calculator(calculator),
        startX(startX),
511     endX(endX),
        startY(startY),
513     endY(endY)
      {
515     int i = 0;
      }
517
      virtual ~ImageComputorParameter()
519   {
        int i = 0;
521   }

523   virtual ImageComputor* getValue()
      {
525     return computor.get();
      }
527
    private:
529   std::auto_ptr<ImageComputor> computor;
      std::auto_ptr<ColourInterpolator> interpolator;
531   std::auto_ptr< Parameters::Parameter<Calculators::AbstractCalculator*> > calculator;
      std::auto_ptr<Parameters::Parameter<double> > startX, endX, startY, endY;
533 };

535 Parameters::Parameter<ImageComputor*>* ImageComputor::getInterpolator(const double* factor1Pointer,
        ImageComputor* other, const double* factor2Pointer)
    {
537   screenX->setShareWith(other->screenX);
      screenY->setShareWith(other->screenY);
539   ColourInterpolatorInterpolator* interpolator = new ColourInterpolatorInterpolator(interpolatorParam->
            getValue(), factor1Pointer, other->interpolatorParam->getValue(), factor2Pointer);
      std::auto_ptr< Parameters::Parameter<Calculators::AbstractCalculator*> > calculator(this->calculatorParam->
            getValue()->getInterpolator(factor1Pointer, other->calculatorParam->getValue(), factor2Pointer));
541   Parameters::Parameter<double>* startX = new Parameters::InterpolatingParam<double, Parameters::
            NumberInterpolator<double> >(this->startX, factor1Pointer, other->startX, factor2Pointer, Parameters::
            NumberInterpolator<double>());
      Parameters::Parameter<double>* endX = new Parameters::InterpolatingParam<double, Parameters::
            NumberInterpolator<double> >(this->endX, factor1Pointer, other->endX, factor2Pointer, Parameters::
            NumberInterpolator<double>());
543   Parameters::Parameter<double>* startY = new Parameters::InterpolatingParam<double, Parameters::
            NumberInterpolator<double> >(this->startY, factor1Pointer, other->startY, factor2Pointer, Parameters::
            NumberInterpolator<double>());
      Parameters::Parameter<double>* endY = new Parameters::InterpolatingParam<double, Parameters::
            NumberInterpolator<double> >(this->endY, factor1Pointer, other->endY, factor2Pointer, Parameters::
            NumberInterpolator<double>());
545   return new ImageComputorParameter(std::auto_ptr<ImageComputor>(new ImageComputor(calculator.get(),
            interpolator, startX, endX, startY, endY, this->screenX, this->screenY)), std::auto_ptr<
            ColourInterpolator>(interpolator), calculator, std::auto_ptr<Parameters::Parameter<double> >(startX),
            std::auto_ptr<Parameters::Parameter<double> >(endX), std::auto_ptr<Parameters::Parameter<double> >(
            startY), std::auto_ptr<Parameters::Parameter<double> >(endY));
      return 0;
547 }

549 //std::auto_ptr<ImageComputor> ImageComputor::getInterpolator(const double* factor1Pointer, ImageComputor*
        other, const double* factor2Pointer)
```

```
//{
//  return std::auto_ptr<ImageComputor>(new ImageComputor(calculatorParam, ColourInterpolatorInterpolator(
      interpolatorParam->getValue(), factor1Pointer, other->interpolatorParam->getValue(), factor2Pointer)));
//}

/**

Erstellt einen ImageInterpreter.
@param parent Das ParentWindow
@param calculator Der Calculator, der das anzuzeigende Bild berechnet.
@param convert Ein ScreenConverter, in dem gespeichert wird
**/
ImageInterpreter::ImageInterpreter(wxScrolledWindow* parent, Parameters::ParametersPanel& parametersPanel,
    ComponentStorage<ColourInterpolator>** interpolatorStorage, ComponentStorage<Point>** pointStorage, Point
    * juliaForPoint)
  : Interpreter(parent),
  parametersPanel(parametersPanel),
  sizeParam(0),
  colourParam(0),
  calculatorParam(0),
  selectedRect(),
  rectSelected(false),
  startX(-2.0),
  endX(2.0),
  startY(-2.0),
  endY(2.0),
  screenX(true, 600, -2.0, 2.0),
  screenY(false, 600, -2.0, 2.0),
  bitmap(0),
  computor(0),
  interpolatorStorage(interpolatorStorage),
  pointStorage(pointStorage),
  selectPoint(0),
  observersToBeRemoved(),
  juliaForPoint(juliaForPoint)
{
  myName = "Image Interpreter";
  this->Show(false);
  // update();
  this->Connect(wxEVT_PAINT, wxPaintEventHandler(ImageInterpreter::OnPaint));
  this->Connect(wxEVT_LEFT_DOWN, wxMouseEventHandler(ImageInterpreter::OnLeftDown));
  this->Connect(wxEVT_LEFT_UP, wxMouseEventHandler(ImageInterpreter::OnLeftUp));
  this->Connect(wxEVT_ERASE_BACKGROUND, wxEraseEventHandler(ImageInterpreter::OnEraseBackground));
}

ImageInterpreter::~ImageInterpreter()
{
  if(*interpolatorStorage)
  {
    (*interpolatorStorage)->removeObserver(colourParam.get());
  }
  if(*pointStorage)
  {
    for(int i = 0; i < observersToBeRemoved.size(); ++i)
    {
      (*pointStorage)->removeObserver(observersToBeRemoved[i]);
    }
  }
}

void ImageInterpreter::update()
{
  /*
  // Calculate needed Time
  FILETIME now;f
  GetSystemTimeAsFileTime(&now);
  long time = now.dwLowDateTime;
  Calculators::generateBitmap(&bitmap, size, startX, startY, endX, endY, *calculator);
  GetSystemTimeAsFileTime(&now);
  time = now.dwLowDateTime - time;
  std::stringstream stream;
  stream << "Time needed: ";
  stream << time;
  wxMessageBox(stream.str(), _("Hallo"));*/
  // Calculators::generateBitmap(&bitmap, size, startX, startY, endX, endY, *calculator);
  if(!sizeParam.get())
  {
    return;
  }
  int size = sizeParam->getValue();

  bitmap = &computor->computeImage(size, -1);

  wxWindow::SetSize(wxSize(bitmap->GetWidth(), bitmap->GetHeight()));

  static_cast<wxScrolledWindow*>(this->GetParent())->SetScrollbars(10, 10, this->GetSize().GetWidth() / 10,
      this->GetSize().GetHeight() / 10);

  this->Refresh();
  this->Update();
}

void ImageInterpreter::OnPaint(wxPaintEvent& event)
{
  wxPaintDC paintDC(this);
  wxBufferedDC dc(&paintDC);
  if(bitmap)
  {
    dc.DrawBitmap(*bitmap, wxPoint(0, 0));
```

```
645     }
647     if(rectSelected)
        {
649       dc.SetBrush(wxBrush(wxColour(255, 255, 255), wxTRANSPARENT));
          dc.SetPen(wxPen(wxColour(255, 255, 255), 2));
651       dc.DrawRectangle(selectedRect);
        }
653
        if(selectPoint)
655     {
          double x = selectPoint->getX();
657       double y = selectPoint->getY();
          double realX = (x-startX)*( (double) bitmap->GetSize().GetWidth() / (endX-startX));
659       double realY = (y-startY)*(bitmap->GetSize().GetHeight() / (endY-startY));
          dc.SetBrush(wxBrush(wxColour(248, 166, 0)));
661       dc.SetPen(wxPen(wxColour(0, 0, 0), 1));
          dc.DrawCircle((int) realX, (int) realY, 6);
663     }
665 }
667 void ImageInterpreter::OnLeftDown(wxMouseEvent& event)
    {
669   if(!selectPoint)
      {
671     this->Connect(wxEVT_MOTION, wxMouseEventHandler(ImageInterpreter::OnMouseMove));
        rectSelected = true;
673     selectedRect.SetTopLeft(event.GetPosition());
      }
675 }
677 void ImageInterpreter::OnLeftUp(wxMouseEvent& event)
    {
679   if(!bitmap)
      {
681     return;
      }
683   if(!selectPoint)
      {
685     startX = startXGui->getValue();
        startY = startYGui->getValue();
687     endX = endXGui->getValue();
        endY = endYGui->getValue();
689
        this->Disconnect(wxEVT_MOTION, wxMouseEventHandler(ImageInterpreter::OnMouseMove));
691     rectSelected = false;
        int newStartX = selectedRect.GetTopLeft().x, newStartY = selectedRect.GetTopLeft().y, newEndX =
            selectedRect.GetBottomRight().x, newEndY = selectedRect.GetBottomRight().y;
693     if(newStartX == newEndX || newStartY == newEndY || newStartX == -1 || newStartY == -1 || newEndX == -1 ||
            newEndY == -1)
        {
695       this->Refresh();
          this->Update();
697       return;
        }
699     if(newStartX > newEndX)
        {
701       std::swap(newStartX, newEndX);
        }
703     if(newStartY > newEndY)
        {
705       std::swap(newStartY, newEndY);
        }
707
        double absWidth = endX - startX;
709     double absHeight = endY - startY;
        int screenWidth = bitmap->GetSize().GetWidth();
711     int screenHeight = bitmap->GetSize().GetHeight();

713     endX = newEndX * (absWidth / screenWidth) + startX;
        startX = newStartX * (absWidth / screenWidth) + startX;
715
        endY = newEndY * (absHeight / screenHeight) + startY;
717     startY = newStartY * (absHeight / screenHeight) + startY;

719     startXGui->setValue(startX);
        startYGui->setValue(startY);
721     endXGui->setValue(endX);
        endYGui->setValue(endY);
723
        update();
725   }else
      {
727     wxPoint point = event.GetPosition();
        // double realX = (x-startX)*(bitmap->GetSize().GetWidth() / (endX-startX));
729     double x = startX + (endX-startX) * ( (double) point.x / (double) bitmap->GetSize().GetWidth());
        double y = startY + (endY-startY) * ( (double) point.y / (double) bitmap->GetSize().GetHeight());
731     selectPoint->setPoint(x, y);
        selectPoint->stopSelection();
733   }
    }
735
    void ImageInterpreter::OnMouseMove(wxMouseEvent& event)
737 {
      wxPoint point = event.GetPosition();
739   wxPoint start = selectedRect.GetTopLeft();
      int size = point.x - start.x;
741   selectedRect.SetBottomRight(event.GetPosition());
```

```
    this->Refresh();
743 this->Update();
}
745
void ImageInterpreter::OnEraseBackground(wxEraseEvent& event)
747 {}

749 void ImageInterpreter::zoomOut()
{
751 double startX = startXGui->getValue();
  double startY = startYGui->getValue();
753 double endX = endXGui->getValue();
  double endY = endYGui->getValue();
755
  this->startX =  startX - (endX - startX);
757 startXGui->setValue(this->startX);
  this->endX = endX + (endX - startX);
759 endXGui->setValue(this->endX);

761 this->startY = startY - (endY - startY);
  startYGui->setValue(this->startY);
763 this->endY = endY + (endY - startY);
  endYGui->setValue(this->endY);
765 }

767 void ImageInterpreter::save()
{
769 wxFileDialog* dialog = new wxFileDialog(this, "Choose a file", "", "", "*.png");
  if(dialog->ShowModal() != wxID_OK || dialog->GetFilename() == "")
771 {
    dialog->Destroy();
773 return;
  }
775 std::string path = dialog->GetPath();
  dialog->Destroy();
777
  bitmap->SaveFile(path, wxBITMAP_TYPE_PNG);
779 }

781 Calculators::Interpreter* ImageInterpreter::getValue()
{
783 return this;
}
785
void ImageInterpreter::setPointToSelect(Point* point)
787 {
  selectPoint = point;
789
  this->Refresh();
791 this->Update();
}
793
/**
795 * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
*/
797 void ImageInterpreter::selected()
{
799 if(!sizeParam.get())
  {
801   sizeParam.reset(new Parameters::ParameterGui<int>("Size", &parametersPanel, parametersPanel.getSizer()));
    sizeParam->addChooser(Parameters::ParameterGui<int>::ChooserPtr(new Parameters::NumberChooser<int>(&
        parametersPanel, "Sizes", 400, 10, 2000)));
803 }
  parametersPanel.addParameter(sizeParam.get());
805
  if(!colourParam.get())
807 {
    colourParam.reset(new Parameters::ParameterGui<ColourInterpolator*>("ColourInterpolator", &parametersPanel
        , parametersPanel.getSizer()));
809 (*interpolatorStorage)->addObserver(colourParam.get());
    colourParam->addObserver(this);
811 }
  parametersPanel.addParameter(colourParam.get());
813
  if(!calculatorParam.get())
815 {
    calculatorParam.reset(new Parameters::ParameterGui<Calculators::AbstractCalculator*>("Calculator", &
        parametersPanel, parametersPanel.getSizer()));
817 calculatorParam->addChooser(Parameters::ParameterGui<Calculators::AbstractCalculator*>::ChooserPtr(new
        Calculators::CalculatorChooser(&parametersPanel, &screenX, &screenY, this)));
  }
819 parametersPanel.addParameter(calculatorParam.get());

821 if(!startXGui.get())
  {
823   double* startX = new double;
    startXGui.reset(new Parameters::ParameterGui<double>("Start X", &parametersPanel, parametersPanel.getSizer
        ()));
825 startXGui->addChooser(new Parameters::TextfieldChooser<double>(&parametersPanel, "Number", "-2.0",
        Parameters::NumberValidator<double>(startX), std::auto_ptr<double>(startX)));
  }
827 parametersPanel.addParameter(startXGui.get());

829 if(!endXGui.get())
  {
831   double* endX = new double;
    endXGui.reset(new Parameters::ParameterGui<double>("End X", &parametersPanel, parametersPanel.getSizer()))
        ;
```

```
833     endXGui->addChooser(new Parameters::TextfieldChooser<double>(&parametersPanel, "Number", "2.0", Parameters
            ::NumberValidator<double>(endX), std::auto_ptr<double>(endX)));
    }
835   parametersPanel.addParameter(endXGui.get());

837   if(!startYGui.get())
    {
839     double* startY = new double;
      startYGui.reset(new Parameters::ParameterGui<double>("Start Y", &parametersPanel, parametersPanel.getSizer
            ()));
841     startYGui->addChooser(new Parameters::TextfieldChooser<double>(&parametersPanel, "Number", "-2.0",
            Parameters::NumberValidator<double>(startY), std::auto_ptr<double>(startY)));
    }
843   parametersPanel.addParameter(startYGui.get());

845   if(!endYGui.get())
    {
847     double* endY = new double;
      endYGui.reset(new Parameters::ParameterGui<double>("End Y", &parametersPanel, parametersPanel.getSizer()))
            ;
849     endYGui->addChooser(new Parameters::TextfieldChooser<double>(&parametersPanel, "Number", "2.0", Parameters
            ::NumberValidator<double>(endY), std::auto_ptr<double>(endY)));

851     // Computor
      computor.reset(new ImageComputor(calculatorParam.get(), colourParam.get(), startXGui.get(), endXGui.get(),
             startYGui.get(), endYGui.get(), &screenX, &screenY));
853   }
    parametersPanel.addParameter(endYGui.get());
855
    startX = startXGui->getValue();
857   startY = startYGui->getValue();
    endX = endXGui->getValue();
859   endY = endYGui->getValue();

861   this->Show(true);
    // update();
863 }

865 /**
  * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
867 **/
  void ImageInterpreter::unselected()
869 {
    parametersPanel.removeParameter(sizeParam.get());
871   // sizeParam.reset();

873   parametersPanel.removeParameter(colourParam.get());

875   parametersPanel.removeParameter(calculatorParam.get());
    // calculatorParam.reset();
877
    parametersPanel.removeParameter(startXGui.get());
879   parametersPanel.removeParameter(endXGui.get());
    parametersPanel.removeParameter(startYGui.get());
881   parametersPanel.removeParameter(endYGui.get());

883   this->Show(false);
  }
885
  void ImageInterpreter::onChange(ParameterReturning newValue, Parameter<ParameterReturning>* sender)
887 {
    bitmap = &computor->updateOnlyColour();
889
    this->Refresh();
891   this->Update();
  }
893
  ImageComputor* ImageInterpreter::getComputor()
895 {
    return computor.get();
897 }

899 void ImageInterpreter::addChoosersToGui(Parameters::ParameterGui<double>* gui, wxWindow* parent)
  {
901   PointChooser* chooser = new PointChooser(parent, true);
    (*pointStorage)->addObserver(chooser->getGui());
903   observersToBeRemoved.push_back(chooser->getGui());
    gui->addChooser(chooser);
905
    chooser = new PointChooser(parent, false);
907   (*pointStorage)->addObserver(chooser->getGui());
    observersToBeRemoved.push_back(chooser->getGui());
909   gui->addChooser(chooser);

911   std::string name = gui->getName();
    if(!juliaForPoint)
913   {
      if(name == "c real")
915     {
        gui->setSelection(1);
917     }else if(name == "c imag")
      {
919       gui->setSelection(2);
      }
921   }else
    {
923     bool changeSelectedPoint = false;
      if(name == "z0 real")
925     {
```

```
        gui->setSelection(1);
927     }else if(name == "z0 imag")
        {
929       gui->setSelection(2);
        }else if(name == "c real")
931     {
          gui->setSelection(3);
933       changeSelectedPoint = true;
        }else if(name == "c imag")
935     {
          gui->setSelection(4);
937       changeSelectedPoint = true;
        }
939     if(changeSelectedPoint)
        {
941       Parameters::ParameterChooser<double>* chooser = gui->getChooser(gui->getSelection());
          PointChooser* casted = dynamic_cast<PointChooser*>(chooser);
943       if(casted)
          {
945         Parameters::ParameterGui<Point*>* gui = casted->getGui();
            for(int i = 0; i < gui->getNumChoosers(); ++i)
947         {
              if(gui->getChooser(i)->getValue() == juliaForPoint)
949           {
                gui->setSelection(i, true);
951           }
            }
953       }
        }
955   }

957 }

959 ColourParameterChooser::ColourParameterChooser(wxWindow* parent, std::string theName, wxColour startColour)
      : ParameterChooser<wxColour>(parent),
961   // sizer(new wxBoxSizer(wxVERTICAL)),
      parent(parent),
963   chooseButton(new wxButton(this, -1, "Choose Colour")),
      value(startColour)
965 {
      ParameterChooser::myName = theName;
967   // SetSizer(sizer);
      // sizer->Add(chooseButton, 1);
969   chooseButton->SetBackgroundColour(value);
      // this->Connect(wxEVT_COMMAND_SPINCTRL_UPDATED, wxSpinEventHandler(NumberChooser::onSpin));
971   this->Connect(wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(ColourParameterChooser::onButtonPress));
  }
973
  void ColourParameterChooser::onButtonPress(wxCommandEvent& ev)
975 {
      // value = spinner->GetValue();
977   wxColourData data;
      data.SetColour(value);
979   wxColourDialog* dialog = new wxColourDialog(parent, &data);
      dialog->ShowModal();
981   dialog->Destroy();
      value = dialog->GetColourData().GetColour();
983   chooseButton->SetBackgroundColour(value);
      notifyObservers(value);
985 }

987 //void ColourParameterChooser::onSpin(wxSpinEvent& ev)
  //{
989 //
  //
991 //}

993 wxColour ColourParameterChooser::getValue()
  {
995   return value;
  }
997
  /**
999 * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
  */
1001 void ColourParameterChooser::selected()
  {
1003
  }
1005
  /**
1007 * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
  **/
1009 void ColourParameterChooser::unselected()
  {
1011
  }
1013
  ColourInterpolator::ColourInterpolator(std::string name)
1015 {
    myName = name;
1017 }

1019
  wxColour ColourInterpolator::interpolate(wxColour one, double factor1, wxColour two, double factor2) const
1021 {
    return wxColour(one.Red() * factor1 + two.Red() * factor2, one.Green() * factor1 + two.Green() * factor2,
          one.Blue() * factor1 + two.Blue() * factor2);
1023 }
```

```
1025 ColourInterpolator* ColourInterpolator::getValue()
1027 {
       return this;
     }
1029
     void ColourInterpolator::setName(std::string name)
1031 {
       myName = name;
1033   notifyObservers(this);
     }
1035
     void ColourInterpolator::onInterpolationUpdate(const InterpolationData& data)
1037 {
       Parameters::Interpolator<wxColour>::onInterpolationUpdate(data);
1039   notifyObservers(this);
     }
1041
     /**
1043 * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
     **/
1045 void ColourInterpolator::selected()
     {
1047
     }
1049
     /**
1051 * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
     **/
1053 void ColourInterpolator::unselected()
     {
1055
     }
1057
     ColourInterpolatorInterpolator::ColourInterpolatorInterpolator(ColourInterpolator *param1, const double *
         factor1, ColourInterpolator *param2, const double *factor2)
1059   : ColourInterpolator("ddd"),
       param1(param1),
1061   param2(param2),
       factor1(factor1),
1063   factor2(factor2)
     {
1065
     }
1067
     //ColourInterpolatorInterpolator::~ColourInterpolatorInterpolator()
1069 //{
     //  int i = 0;
1071 //}

1073 wxColour ColourInterpolatorInterpolator::getInterpolation(double position)
     {
1075   return interpolate(param1->getInterpolation(position), *factor1, param2->getInterpolation(position), *
           factor2);
     }
1077
     ColourChooser::ColourChooser(wxMDIParentFrame* parent, std::string name, ComponentStorage<ColourInterpolator
         >** interpolatorStorage)
1079   : wxMDIChildFrame(parent, -1, name),
       panel(new wxScrolledWindow(this)),
1081   sizer(new wxGridBagSizer),
       bitmap(0, 0),
1083   preview(new wxStaticBitmap(panel, -1, bitmap)),
       interpolationChooser(panel, sizer, "Add Colour", "Colour", 2, this),
1085   interpolator(name),
       interpolatorStorage(interpolatorStorage)
1087 {
       interpolationChooser.addInterpolationObserver(&interpolator);
1089   interpolationChooser.addInterpolationObserver(this);

1091   wxStaticText* nameL = new wxStaticText(panel, -1, "Name");
       nameL->SetFont(Parameters::AbstractParameterGui::PARAMNAME_FONT);
1093   sizer->Add(nameL, wxGBPosition(0, 0), wxGBSpan(1, 1), wxALL, 5);

1095   wxTextCtrl* nameT = new wxTextCtrl(panel, -1, name, wxDefaultPosition);

1097   sizer->Add(nameT, wxGBPosition(0, 1), wxGBSpan(1, 1), wxALL, 5);

1099   preview->SetMinSize(wxSize(0, PREVIEW_HEIGHT));
       sizer->Add(preview, wxGBPosition(1, 0), wxGBSpan(1, 5), wxALL | wxEXPAND, 5);
1101
       sizer->AddGrowableCol(4);
1103   panel->SetSizer(sizer);
       panel->SetScrollRate(5, 5);
1105
       Connect(wxEVT_COMMAND_TEXT_UPDATED, wxCommandEventHandler(ColourChooser::onNameChange));
1107   Connect(wxEVT_SIZE, wxSizeEventHandler(ColourChooser::onSizeChange));

1109   // onInterpolationUpdate();
     }
1111
     void ColourChooser::onSizeChange(wxSizeEvent& evt)
1113 {
       updatePreviewImage();
1115   evt.Skip();
     }
1117
     void ColourChooser::onNameChange(wxCommandEvent& evt)
1119 {
```

```
    SetTitle(evt.GetString());
1121  interpolator.setName(static_cast<std::string>(evt.GetString()));
    }
1123
    void ColourChooser::addChoosersToGui(Parameters::ParameterGui<wxColour>* gui, wxWindow* parent)
1125 {
      ColourParameterChooser* chooser = new ColourParameterChooser(parent, "ColourChooser", wxColour(255, 0, 0));
1127  gui->addChooser(chooser);
      chooser->addObserver(&interpolationChooser);
1129 }

1131 void ColourChooser::updatePreviewImage()
    {
1133  wxSize size = preview->GetSize();
      int width = size.GetWidth();
1135  int height = size.GetHeight();
      if(!width || !height)
1137  {
        return;
1139  }
      wxImage image(width, height);
1141  unsigned char* data = static_cast<unsigned char*>(malloc(width * height * 3));

1143  int index = 0;
      for(int x = 0; x < width; ++x)
1145  {
        // wxColour colour = wxColour( (int) (255.0 * ( (double) x / (double) width)), 0, 0);
1147    wxColour colour = interpolator.getInterpolation( (double) x / (double) width);
        for(int y = 0; y < height; ++y)
1149    {
          index = (x + y * width) * 3;
1151      unsigned char* r = &data[index++];
          unsigned char* g = &data[index++];
1153      unsigned char* b = &data[index++];
          *r = colour.Red();
1155      *g = colour.Green();
          *b = colour.Blue();
1157    }
      }
1159
      /*Calculators::AlgorythmReturning* numIterations = new Calculators::AlgorythmReturning[width * height];
1161  Calculators::AbstractCalculator* calculator = calculatorParam->getValue();
      calculator->calculate(&numIterations, width, height);
1163
      int maxIterations = 150;
1165  int index = 0;
      for(int y = 0; y < height; ++y)
1167  {
        for(int x = 0; x < width; ++x)
1169    {
          unsigned char* r = &data[index++];
1171      unsigned char* g = &data[index++];
          unsigned char* b = &data[index++];
1173      *r = *g = *b = 0;
          if(numIterations[y * width + x] > -1)
1175      {
            *r = (maxIterations - numIterations[y * width + x]) * (255 / maxIterations);
1177      }else
          {
1179        *r = 0;
          }
1181    }
      }
1183
      delete[] numIterations;*/
1185  image.SetData(data);
      bitmap = wxBitmap(image);
1187  preview->SetBitmap(bitmap);
      image.Destroy();
1189 }

1191 void ColourChooser::onInterpolationUpdate(const InterpolationData& data)
    {
1193  updatePreviewImage();
    }
1195
    ColourInterpolator* ColourChooser::getInterpolator()
1197 {
      return &interpolator;
1199 }

1201 ColourChooser::~ColourChooser()
    {
1203  if(*interpolatorStorage)
      {
1205    (*interpolatorStorage)->removeComponent(&interpolator);
      }
1207 }

1209 FractalInterpolator::FractalInterpolator()
      : factor1(0.0),
1211    factor2(0.0),
      two(0),
1213    computor()
    {
1215 }

1217 FractalInterpolator::~FractalInterpolator()
    {
```

```
1219    int i = 0;
    }
1221
1223  ImageComputor* FractalInterpolator::interpolate(ImageComputor* one, double factor1, ImageComputor* two, double
          factor2) const
    {
1225    this->factor1 = factor1;
      this->factor2 = factor2;
1227    if(this->two != two)
      {
1229      this->two = two;
        prepareInterpolation(one, two);
1231    }
      return computor->getValue();
1233  }
1235  void FractalInterpolator::reset()
    {
1237    two = 0;
    }
1239
    void FractalInterpolator::prepareInterpolation(ImageComputor* one, ImageComputor* two) const
1241  {
      computor.reset(one->getInterpolator(&factor1, two, &factor2));
1243  }
1245  MovieChooser::MovieChooser(wxMDIParentFrame* parent, std::string name, ComponentStorage<ImageComputor>**
        fractalStorage)
      : wxMDIChildFrame(parent, -1, name),
1247    panel(new wxScrolledWindow(this)),
      sizer(new wxGridBagSizer),
1249    interpolationChooser(panel, sizer, "Add Keyframe", "Key Frame", 5, this),
      fractalStorage(fractalStorage),
1251    interpolator()
    {
1253    interpolationChooser.addInterpolationObserver(&interpolator);
      interpolationChooser.addInterpolationObserver(this);
1255
      wxStaticText* nameL = new wxStaticText(panel, -1, "Name");
1257    nameL->SetFont(Parameters::AbstractParameterGui::PARAMNAME_FONT);
      sizer->Add(nameL, wxGBPosition(0, 0), wxGBSpan(1, 1), wxALL, 5);
1259
      wxTextCtrl* nameT = new wxTextCtrl(panel, ID_NAME_CHANGE, name, wxDefaultPosition);
1261
      sizer->Add(nameT, wxGBPosition(0, 1), wxGBSpan(1, 1), wxALL, 5);
1263
      wxStaticText* widthL = new wxStaticText(panel, -1, "Width");
1265    widthL->SetFont(Parameters::AbstractParameterGui::PARAMNAME_FONT);
      sizer->Add(widthL, wxGBPosition(1, 0), wxDefaultSpan, wxALL, 5);
1267
      width = new wxSpinCtrl(panel, -1, wxEmptyString, wxDefaultPosition, wxDefaultSize, 4608L, 0, 20000, 400);
1269    sizer->Add(width, wxGBPosition(1, 1), wxDefaultSpan, wxALL, 5);
1271
      wxStaticText* heightL = new wxStaticText(panel, -1, "Heights");
1273    heightL->SetFont(Parameters::AbstractParameterGui::PARAMNAME_FONT);
      sizer->Add(heightL, wxGBPosition(2, 0), wxDefaultSpan, wxALL, 5);
1275
      height = new wxSpinCtrl(panel, -1, wxEmptyString, wxDefaultPosition, wxDefaultSize, 4608L, 0, 20000, 400);
1277    sizer->Add(height, wxGBPosition(2, 1), wxDefaultSpan, wxALL, 5);
1279    wxStaticText* numFramesL = new wxStaticText(panel, -1, "Num frames");
      numFramesL->SetFont(Parameters::AbstractParameterGui::PARAMNAME_FONT);
1281    sizer->Add(numFramesL, wxGBPosition(3, 0), wxDefaultSpan, wxALL, 5);
1283    numFrames = new wxSpinCtrl(panel, -1, wxEmptyString, wxDefaultPosition, wxDefaultSize, 4608L, 0, 100000, 60)
          ;
      sizer->Add(numFrames, wxGBPosition(3, 1), wxDefaultSpan, wxALL, 5);
1285
      wxButton* createMovie = new wxButton(panel, wxID_OK, "Create Movie");
1287    sizer->Add(createMovie, wxGBPosition(4, 0), wxGBSpan(1, 2), wxEXPAND | wxALL, 5);
1289    /*preview->SetMinSize(wxSize(0, PREVIEW_HEIGHT));
      sizer->Add(preview, wxGBPosition(1, 0), wxGBSpan(1, 5), wxALL | wxEXPAND, 5);*/
1291
      /*sizer->AddGrowableCol(4);*/
1293    panel->SetSizer(sizer);
      panel->SetScrollRate(5, 5);
1295
      Connect(ID_NAME_CHANGE, wxEVT_COMMAND_TEXT_UPDATED, wxCommandEventHandler(MovieChooser::onNameChange));
1297    Connect(wxID_OK, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(MovieChooser::onCreateMovie));
      /*Connect(wxEVT_SIZE, wxSizeEventHandler(ColourChooser::onSizeChange));*/
1299  }
1301  MovieChooser::~MovieChooser()
    {
1303    if(*fractalStorage)
      {
1305      interpolationChooser.deleteAllValues();
      }
1307  }
1309  void MovieChooser::addChoosersToGui(Parameters::ParameterGui<ImageComputor*>* gui, wxWindow* parent)
    {
1311    (*fractalStorage)->addObserver(gui);
    }
1313
    void MovieChooser::removeChooser(Parameters::ParameterGui<ImageComputor*>* gui, wxWindow* parent)
```

```
{
  (*fractalStorage)->removeObserver(gui);
}

void MovieChooser::onInterpolationUpdate(const InterpolationData& data)
{

}

void MovieChooser::onCreateMovie(wxCommandEvent& evt)
{
  interpolator.reset();
  wxDirDialog* dirDialog = new wxDirDialog(this);
  if(dirDialog->ShowModal() != wxID_OK)
  {
    dirDialog->Destroy();
    return;
  }
  std::string path = dirDialog->GetPath();
  dirDialog->Destroy();

  int myWidth = width->GetValue();
  int myHeight = height->GetValue();
  double num = (double) numFrames->GetValue();
  for(int i = 0; i < num; ++i)
  {
    ImageComputor* computor = interpolator.getInterpolation((double) i / (num - 1));
    wxBitmap bitmap = computor->computeImage(myWidth, myHeight);
    std::stringstream stream;
    stream << path;
    stream << "\\";
    stream << i+1;
    stream << ".png";
    bitmap.SaveFile(stream.str(), wxBITMAP_TYPE_PNG);
  }
}

void MovieChooser::onNameChange(wxCommandEvent& evt)
{
  SetTitle(evt.GetString());
  /*interpolator.setName(static_cast<std::string>(evt.GetString()));*/
}

Point::Point(wxMDIParentFrame* parent, std::string name, ComponentStorage<Point>** pointStorage,
      PointSelectionObserver* observer, Point** activePoint)
  : wxMDIChildFrame(parent, -1, name, wxDefaultPosition, wxDefaultSize, wxMINIMIZE_BOX | wxSYSTEM_MENU |
      wxCAPTION),
  sizer(new wxBoxSizer(wxHORIZONTAL)),
  pointStorage(pointStorage),
  activePoint(activePoint),
  normalBitmap("selectpoint.png", wxBITMAP_TYPE_PNG),
  selectedBitmap("selectingpoint.png", wxBITMAP_TYPE_PNG),
  selecting(false),
  observer(observer)
{
  wxPanel* panel = new wxPanel(this);
  wxStaticText* nameL = new wxStaticText(panel, -1, "Name");
  nameL->SetFont(Parameters::AbstractParameterGui::PARAMNAME_FONT);
  sizer->Add(nameL, 1, wxALL, 5);

  wxTextCtrl* nameT = new wxTextCtrl(panel, ID_NAME_CHANGE, name, wxDefaultPosition);
  sizer->Add(nameT, 1, wxALL, 5);
  myName = name;

  wxStaticText* xL = new wxStaticText(panel, -1, "X:");
  sizer->Add(xL, 1, wxALL, 5);

  double* pointer = new double;
  positionX = new Parameters::TextfieldChooser<double>(panel, "x", "0.0", Parameters::NumberValidator<double>(
      pointer), std::auto_ptr<double>(pointer));
  sizer->Add(positionX, 1, wxALL, 5);

  wxStaticText* yL = new wxStaticText(panel, -1, "Y:");
  sizer->Add(yL, 1, wxALL, 5);

  pointer = new double;
  positionY = new Parameters::TextfieldChooser<double>(panel, "y", "0.0", Parameters::NumberValidator<double>(
      pointer), std::auto_ptr<double>(pointer));
  sizer->Add(positionY, 1, wxALL, 5);

  button = new wxBitmapButton(panel, wxID_SETUP, normalBitmap);
  button->SetToolTip("Select the point in a fractal");
  sizer->Add(button, 1, wxALL, 5);

  panel->SetSizer(sizer);

  Connect(ID_NAME_CHANGE, wxEVT_COMMAND_TEXT_UPDATED, wxCommandEventHandler(Point::onNameChange));
  Connect(wxID_SETUP, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(Point::onSelectPointClick));
}

Point::~Point()
{
  if(*pointStorage)
  {
    (*pointStorage)->removeComponent(this);
  }
}

void Point::onNameChange(wxCommandEvent& evt)
```

```
{
  SetTitle(evt.GetString());
  myName = evt.GetString();
  notifyObservers(this);
}

void Point::startSelection()
{
  if(selecting)
  {
    return;
  }

  selecting = true;
  button->SetBitmap(selectedBitmap);

  if(*activePoint)
  {
    (*activePoint)->stopSelection(true);
  }
  *activePoint = this;

  observer->setPointToSelect(this);

}

void Point::stopSelection(bool newSelectionAfterwards)
{
  if(!selecting)
  {
    return;
  }

  selecting = false;
  button->SetBitmap(normalBitmap);
  *activePoint = 0;
  if(!newSelectionAfterwards)
  {
    observer->setPointToSelect(0);
  }
}

void Point::onSelectPointClick(wxCommandEvent& evt)
{
  if(selecting)
  {
    stopSelection();
  }else
  {
    startSelection();
  }
}

double Point::getX()
{
  return positionX->getValue();
}

double Point::getY()
{
  return positionY->getValue();
}

void Point::setPoint(double x, double y)
{
  positionX->setValue(x);
  positionY->setValue(y);
}

Point::Returning Point::getValue()
{
  return this;
}

/**
* Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
*/
void Point::selected()
{

}

/**
* Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
**/
void Point::unselected()
{

}

PointChooser::PointChooser(wxWindow* parent, bool xElseY)
  : Parameters::ParameterChooser<double>(parent),
  xElseY(xElseY),
  sizer(new Parameters::ParameterSizer(10, 10)),
  gui("Point", this, sizer)
{
  sizer->SetCols(3);
  this->SetSizer(sizer);
  gui.addToPanel(0);
```

```
1509 }

1511 Parameters::ParameterGui<Point*>* PointChooser::getGui()
{
1513   return &gui;
}
1515
double PointChooser::getValue()
1517 {
   return xElseY ? gui.getValue()->getX() : gui.getValue()->getY();
1519 }

1521 /**
* Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
1523 */
void PointChooser::selected()
1525 {

1527 }

1529 /**
* Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
1531 **/
void PointChooser::unselected()
1533 {

1535 }

1537 std::string PointChooser::getMyName() const
{
1539   return xElseY ? "PointX" : "PointY";
}
1541
IMPLEMENT_APP(JuliamengenApp)
```

Quellcode/Juliamengen.cpp

Datei Parameters.cpp

```
2  #include "Parameters.h"

4  using Parameters::ParametersPanel;
   using Parameters::AbstractParameterGui;
6  using Parameters::ParameterSizer;

8  ParametersPanel::ParametersPanel(wxWindow* parent)
     : wxPanel(parent, -1),
10   parameters(),
     sizer(new ParameterSizer(10, 10))
12 {
     this->SetSizer(sizer);
14   sizer->SetCols(3);
     sizer->AddGrowableCol(2);
16 }

18 void ParametersPanel::addParameter(AbstractParameterGui* param)
   {
20   parameters.push_back(param);
     // sizer->Add(param, 0);
22   param->addToPanel(this->getSize() - 1);
   }
24
   void ParametersPanel::removeParameter(AbstractParameterGui* param)
26 {
     parameters.remove(param);
28   param->removeFromPanel(this, sizer);
   }
30
   int ParametersPanel::getSize() const
32 {
     return parameters.size();
34 }

36 ParameterSizer* ParametersPanel::getSizer()
   {
38   return sizer;
   }
40
   void ParametersPanel::copyTo(ParametersPanel* copy)
42 {
     int i = 0;
44   for( std::list<AbstractParameterGui*>::iterator it = parameters.begin(); it != parameters.end(); ++it, ++i)
     {
46     std::list<AbstractParameterGui*>::iterator it2 = copy->parameters.begin();
       for(int i2 = 0; i2 < i; ++i2, ++it2)
48     {
         if(it2 == copy->parameters.end())
50       {
           return;
52       }
       }
54     (*it)->copyTo((*it2));
```

```
  }
}

const wxFont AbstractParameterGui::PARAMNAME_FONT = wxFont(10, wxDEFAULT, wxNORMAL, wxBOLD);
const wxFont AbstractParameterGui::FONT_2 = wxFont(10, wxDEFAULT, wxITALIC, wxNORMAL);;
// Parameters::NumberValidator<double> AbstractParameterGui::DOUBLE_VALIDATOR = Parameters::NumberValidator<
    double>();
```

Quellcode/Parameters.cpp

Datei Calculators.cpp

```
#include "Calculators.h"
#include <wx/rawbmp.h>
#include <wx/dcbuffer.h>
#include <algorithm>
#include "Juliamengen.h"

using Calculators::AbstractCalculator;
using Calculators::AlgorythmReturning;
using Calculators::Algorythm;
using Calculators::Calculator;
using Calculators::CalculatorChooser;
using Calculators::Interpreter;
using Calculators::JuliaEscapeTime;
using Calculators::JuliaEscapeTimeChooser;
using Calculators::StandardFormula;
using Calculators::StandardFormulaChooser;
using Calculators::Formula;
using Calculators::FormulaParam;
using Calculators::CustomFormula;
using Calculators::CustomFormulaChooser;

Calculator::Calculator(Parameters::Parameter<Algorythm*>* algorythm, Parameters::ScreenChooserSupplier*
    screenX, Parameters::ScreenChooserSupplier* screenY)
  : algorythm(algorythm),
  screenX(screenX),
  screenY(screenY)
{

}

int Calculator::calculate(AlgorythmReturning** returning, int width, int height) const
{
  //FILETIME now;
  //GetSystemTimeAsFileTime(&now);
  //long time = now.dwLowDateTime;

  try
  {
    Algorythm* algorythm = this->algorythm->getValue();
    algorythm->prepare();
    for(int x = 0; x < width; ++x)
    {
      screenX->setCurrentPixel(x);
      for(int y = 0; y < height; ++y)
      {
        screenY->setCurrentPixel(y);
        (*returning)[width * y + x] = (*algorythm)();
      }
    }
    return algorythm->getMaxReturning();
  }catch(mu::Parser::exception_type& exception)
  {
    return 10;
  }

  /*GetSystemTimeAsFileTime(&now);
  time = now.dwLowDateTime - time;
  std::stringstream stream;
  stream << "Time needed: ";
  stream << time;
  wxMessageBox(stream.str(), _("Hallo"));*/


  /*int convertersSize = converters->size();
  int AlgorythmSize = Algorythm->getNumParameters();
  ParameterSet parameters(AlgorythmSize);
  if(convertersSize < AlgorythmSize)
  {
    for(int i = convertersSize; i < AlgorythmSize; ++i)
    {
      converters->push_back(ParameterConverterPtr(new ConstConverter(Parameter(0.0, 0.0))));
    }
  }
  for(int x = 0; x < width; ++x)
  {
    for(int y = 0; y < height; ++y)
    {
      for(int i = 0; i < convertersSize; ++i)
      {
        parameters.at(i) = converters->at(i)->convert(x, y, width, height, time);
      }
```

```
      (*returning)[width * y + x] = (*Algorythm)(parameters);
82    }
    }*/
84  }

86  class CalculatorParameter : public Parameters::Parameter<AbstractCalculator*>
    {
88  public:
      CalculatorParameter(std::auto_ptr<AbstractCalculator> calculator, std::auto_ptr< Parameters::Parameter<
          Algorythm*> > algorythm)
90      : calculator(calculator),
        algorythm(algorythm)
92    {}

94    virtual AbstractCalculator* getValue()
      {
96      return calculator.get();
      }
98
    private:
100   std::auto_ptr<AbstractCalculator> calculator;
      std::auto_ptr< Parameters::Parameter<Algorythm*> > algorythm;
102 };

104 std::auto_ptr<Parameters::Parameter<AbstractCalculator*> > Calculator::getInterpolator(const double*
        factor1Pointer, AbstractCalculator* other, const double* factor2Pointer)
    {
106   /*ColourInterpolatorInterpolator* interpolator = new ColourInterpolatorInterpolator(interpolatorParam->
          getValue(), factor1Pointer, other->interpolatorParam->getValue(), factor2Pointer);
      return std::auto_ptr<Parameters::Parameter<ImageComputor*> >(new ImageComputorParameter(std::auto_ptr<
          ImageComputor>(new ImageComputor(calculatorParam, interpolator)), std::auto_ptr<ColourInterpolator>(
          interpolator)));*/
108   Calculator* othercalc = dynamic_cast<Calculator*>(other);
      if(othercalc)
110   {
        std::auto_ptr<Parameters::Parameter<Algorythm*> > algorythm(this->algorythm->getValue()->getInterpolator(
            factor1Pointer, othercalc->algorythm->getValue(), factor2Pointer));
112     return std::auto_ptr< Parameters::Parameter<AbstractCalculator*> > (new CalculatorParameter(std::auto_ptr<
            AbstractCalculator>(new Calculator(algorythm.get(), screenX, screenY)), algorythm));
      }
114   // TODO: Calculator inerpolation for different Algorythms.
    }
116
    Calculator* Calculator::getValue()
118 {
      return this;
120 }

122 CalculatorChooser::CalculatorChooser(Parameters::ParametersPanel* parametersPanel, Parameters::
        ScreenChooserSupplier* screenX, Parameters::ScreenChooserSupplier* screenY, Parameters::ChooserSupplier<
        double>* chooserSupplier)
      : ParameterChooser(parametersPanel),
124   algorythmParam(0),
      parametersPanel(*parametersPanel),
126   screenX(screenX),
      screenY(screenY),
128   chooserSupplier(chooserSupplier)
    {
130   wxBoxSizer* sizer = new wxBoxSizer(wxHORIZONTAL);
      wxStaticText* text = new wxStaticText(this, -1, "Standard Calculator");
132   text->SetFont(Parameters::AbstractParameterGui::FONT_2);
      sizer->Add(text);
134   this->SetSizer(sizer);
    }
136
    CalculatorChooser::Returning CalculatorChooser::getValue()
138 {
      calculator.reset(new Calculator(algorythmParam.get(), screenX, screenY));
140   return calculator.get();
    }
142
    /**
144 * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
    */
146 void CalculatorChooser::selected()
    {
148   if(!algorythmParam.get())
      {
150     algorythmParam.reset(new Parameters::ParameterGui<Algorythm*>("Algorythm", &parametersPanel,
            parametersPanel.getSizer()));
        algorythmParam->addChooser(Parameters::ParameterGui<Algorythm*>::ChooserPtr(new JuliaEscapeTimeChooser(&
            parametersPanel, screenX, screenY, chooserSupplier)));
152   }
      parametersPanel.addParameter(algorythmParam.get());
154 }

156 /**
    * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
158 **/
    void CalculatorChooser::unselected()
160 {
      parametersPanel.removeParameter(algorythmParam.get());
162 }

164 Interpreter::Interpreter(wxWindow* parent)
      : wxPanel(parent)
166 {}

168
```

```
JuliaEscapeTimeChooser::JuliaEscapeTimeChooser(Parameters::ParametersPanel* parametersPanel, Parameters::
    ScreenChooserSupplier* screenX, Parameters::ScreenChooserSupplier* screenY, Parameters::ChooserSupplier<
    double>* chooserSupplier)
  : ParameterChooser<Algorythm*>(parametersPanel),
  parametersPanel(*parametersPanel),
  formulaGui(0),
  cReGui(0),
  cImGui(0),
  cJGui(0),
  cKGui(0),
  zReGui(0),
  zImGui(0),
  zJGui(0),
  zKGui(0),
  numIterationsGui(0),
  maxValueGui(0),
  screenX(screenX),
  screenY(screenY),
  chooserSupplier(chooserSupplier)
{

  // TODO: PASSED UNTIL HERE
  myName = "Julia Escape Time";
  wxBoxSizer* sizer = new wxBoxSizer(wxHORIZONTAL);
  wxStaticText* text = new wxStaticText(this, -1, myName);
  text->SetFont(Parameters::AbstractParameterGui::FONT_2);
  sizer->Add(text);
  this->SetSizer(sizer);
}

void JuliaEscapeTimeChooser::addParamChoosersForFormula(Parameters::ParameterGui<FormulaParam>* gui)
{
  std::auto_ptr<double> value(new double);
  *(value.get()) = 0.0;
  double* valueP = value.get();
  gui->addChooser(new Parameters::TextfieldChooser<double>(&parametersPanel, "constant value", "0.0",
      Parameters::NumberValidator<double>(valueP), value));
  gui->addChooser(new Parameters::ScreenChooser(&parametersPanel, screenX));
  gui->addChooser(new Parameters::ScreenChooser(&parametersPanel, screenY));
  chooserSupplier->addChoosersToGui(gui, &parametersPanel);
  /*gui->addChooser(new Gui::PointChooser(&parametersPanel, true));
  gui->addChooser(new Gui::PointChooser(&parametersPanel, false));*/
}

/**
* Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
**/
void JuliaEscapeTimeChooser::selected()
{
  if(!numIterationsGui.get())
  {
    numIterationsGui.reset(new Parameters::ParameterGui<int> ("num iterations", &parametersPanel,
        parametersPanel.getSizer()));
    numIterationsGui->addChooser(new Parameters::NumberChooser<int>(&parametersPanel, "number", 100, 1, 10000)
        );
  }
  parametersPanel.addParameter(numIterationsGui.get());

  if(!maxValueGui.get())
  {
    maxValueGui.reset(new Parameters::ParameterGui<FormulaParam> ("maximum value", &parametersPanel,
        parametersPanel.getSizer()));
    maxValueGui->addChooser(new Parameters::NumberChooser<FormulaParam>(&parametersPanel, "number", 1000, 1,
        100000000));
  }
  parametersPanel.addParameter(maxValueGui.get());

  if(!zReGui.get())
  {
    zReGui.reset(new Parameters::ParameterGui<FormulaParam> ("z0 real", &parametersPanel, parametersPanel.
        getSizer()));
    addParamChoosersForFormula(zReGui.get());
  }

  if(!zImGui.get())
  {
    zImGui.reset(new Parameters::ParameterGui<FormulaParam> ("z0 imag", &parametersPanel, parametersPanel.
        getSizer()));
    addParamChoosersForFormula(zImGui.get());
  }

  if(!zJGui.get())
  {
    zJGui.reset(new Parameters::ParameterGui<FormulaParam> ("z0 j", &parametersPanel, parametersPanel.getSizer
        ()));
    addParamChoosersForFormula(zJGui.get());
  }

  if(!zKGui.get())
  {
    zKGui.reset(new Parameters::ParameterGui<FormulaParam> ("z0 k", &parametersPanel, parametersPanel.getSizer
        ()));
    addParamChoosersForFormula(zKGui.get());
  }

  parametersPanel.addParameter(zReGui.get());
  parametersPanel.addParameter(zImGui.get());
  parametersPanel.addParameter(zJGui.get());
  parametersPanel.addParameter(zKGui.get());
```

```
258   if(!cReGui.get())
      {
260     cReGui.reset(new Parameters::ParameterGui<FormulaParam> ("c real", &parametersPanel, parametersPanel.
            getSizer()));
        addParamChoosersForFormula(cReGui.get());
262   }

264   if(!cImGui.get())
      {
266     cImGui.reset(new Parameters::ParameterGui<FormulaParam> ("c imag", &parametersPanel, parametersPanel.
            getSizer()));
        addParamChoosersForFormula(cImGui.get());
268   }

270   if(!cJGui.get())
      {
272     cJGui.reset(new Parameters::ParameterGui<FormulaParam> ("c j", &parametersPanel, parametersPanel.getSizer
            ()));
        addParamChoosersForFormula(cJGui.get());
274   }

276   if(!cKGui.get())
      {
278     cKGui.reset(new Parameters::ParameterGui<FormulaParam> ("c k", &parametersPanel, parametersPanel.getSizer
            ()));
        addParamChoosersForFormula(cKGui.get());
280   }

282   parametersPanel.addParameter(cReGui.get());
      parametersPanel.addParameter(cImGui.get());
284   parametersPanel.addParameter(cJGui.get());
      parametersPanel.addParameter(cKGui.get());
286
      if(!formulaGui.get())
288   {
        formulaGui.reset(new Parameters::ParameterGui<Formula*>("Formula", &parametersPanel, parametersPanel.
            getSizer()));
290     formulaGui->addChooser(new StandardFormulaChooser(&parametersPanel, cReGui.get(), cImGui.get(), cJGui.get
            (), cKGui.get()));
        formulaGui->addChooser(new CustomFormulaChooser(&parametersPanel, cReGui.get(), cImGui.get(), cJGui.get(),
             cKGui.get()));
292   }
      parametersPanel.addParameter(formulaGui.get());
294 }

296 /**
    * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
298 **/
    void JuliaEscapeTimeChooser::unselected()
300 {
      parametersPanel.removeParameter(formulaGui.get());
302   parametersPanel.removeParameter(zImGui.get());
      parametersPanel.removeParameter(zReGui.get());
304   parametersPanel.removeParameter(zJGui.get());
      parametersPanel.removeParameter(zKGui.get());
306
      parametersPanel.removeParameter(cReGui.get());
308   parametersPanel.removeParameter(cImGui.get());
      parametersPanel.removeParameter(cJGui.get());
310   parametersPanel.removeParameter(cKGui.get());

312   parametersPanel.removeParameter(numIterationsGui.get());
      parametersPanel.removeParameter(maxValueGui.get());
314 }

316 Calculators::Algorythm* JuliaEscapeTimeChooser::getValue()
    {
318   escapeTime.reset(new JuliaEscapeTime(formulaGui.get(), zReGui.get(), zImGui.get(), zJGui.get(), zKGui.get(),
          numIterationsGui.get(), maxValueGui.get()));
      return escapeTime.get();
320 }

322 JuliaEscapeTime::JuliaEscapeTime(Parameters::Parameter<Formula*>* formula, Parameters::Parameter<FormulaParam
        >* zRe, Parameters::Parameter<FormulaParam>* zIm, Parameters::Parameter<FormulaParam>* zJ, Parameters::
        Parameter<FormulaParam>* zK, Parameters::Parameter<AlgorythmReturning>* maxIterations, Parameters::
        Parameter<FormulaParam>* maxValue)
    : formula(formula),
324 zRe(zRe),
    zIm(zIm),
326 zJ(zJ),
    zK(zK),
328 maxIterations(maxIterations),
    maxValue(maxValue)
330 {}

332 void JuliaEscapeTime::prepare()
    {
334   theFormula = formula->getValue();
    }
336
    AlgorythmReturning JuliaEscapeTime::operator() () const
338 {
      /*FILETIME now;
340   GetSystemTimeAsFileTime(&now);
      long time = now.dwLowDateTime;*/
342
      /*Formula* formula = formulaGui->getValue();
344   FormulaParam zRe = zReGui->getValue();
```

```
    FormulaParam zIm = zImGui->getValue();
346 AlgorythmReturning maxIterations = numIterationsGui->getValue();
    FormulaParam maxValue = maxValueGui->getValue();*/
348
    //GetSystemTimeAsFileTime(&now);
350 //time = now.dwLowDateTime - time;
    //std::stringstream stream;
352 //stream << "Time needed: ";
    //stream << time;
354 //wxMessageBox(stream.str(), _("Hallo"));


356
    //GetSystemTimeAsFileTime(&now);
358 //time = now.dwLowDateTime;
    double re = zRe->getValue();
360 double im = zIm->getValue();
    double j = zJ->getValue();
362 double k = zK->getValue();

364 // Formula* formula = this->formula->getValue();
    AlgorythmReturning maxIterations = this->maxIterations->getValue();
366 FormulaParam maxValue = this->maxValue->getValue();

368 for(AlgorythmReturning numIterations = 0; numIterations < maxIterations; ++numIterations)
    {
370   (*theFormula)(&re, &im, &j, &k);
      // if(re > maxValue || im > maxValue)
372   if( std::sqrt( re*re + im*im +j*j + k*k) > maxValue )
      {
374     return numIterations;
      }
376 }
    /*GetSystemTimeAsFileTime(&now);
378 time = now.dwLowDateTime - time;
    std::stringstream stream2;
380 stream2 << "Time needed: ";
    stream2 << time;
382 wxMessageBox(stream2.str(), _("Hallo"));*/

384 /*Parameter z = parameters[0];
    Parameter c = parameters[1];
386
    AlgorythmReturning numIterations = 0;
388 static const int maxIterations = 150;
    static const int maxValue = 10000;
390
    for(; numIterations < maxIterations; ++numIterations)
392 {
      z = std::pow(z, 2);
394   z = z + c;
      if(/*std::abs(z) > maxValue/ z.real() > maxValue || z.imag() > maxValue)
396   // if(std::abs(z) > maxValue)
      {
398     int n = numIterations * (255 / maxIterations);
        // *r = numIterations * (255 / maxIterations);
400     return numIterations;
      }
402 }*/
    return -1;
404 }

406 int JuliaEscapeTime::getMaxReturning() const
    {
408   return this->maxIterations->getValue();
    }
410
    class AlgorythmParameter : public Parameters::Parameter<Algorythm*>
412 {
    public:
414   AlgorythmParameter(std::auto_ptr<Algorythm> algorythm, std::auto_ptr< Parameters::Parameter<Formula*> >
          formula, std::auto_ptr<Parameters::Parameter<int> > maxIterations, std::auto_ptr<Parameters::Parameter<
          FormulaParam> > maxValue, std::auto_ptr<Parameters::Parameter<FormulaParam> > zRe, std::auto_ptr<
          Parameters::Parameter<FormulaParam> > zIm, std::auto_ptr<Parameters::Parameter<FormulaParam> > zJ, std
          ::auto_ptr<Parameters::Parameter<FormulaParam> > zK)
      : algorythm(algorythm),
416   formula(formula),
      maxIterations(maxIterations),
418   maxValue(maxValue),
      zRe(zRe),
420   zIm(zIm),
      zJ(zJ),
422   zK(zK)
    {}
424
    virtual Algorythm* getValue()
426 {
      return algorythm.get();
428 }

430 private:
    std::auto_ptr<Algorythm> algorythm;
432 std::auto_ptr< Parameters::Parameter<Formula*> > formula;
    std::auto_ptr<Parameters::Parameter<int> > maxIterations;
434 std::auto_ptr<Parameters::Parameter<FormulaParam> > maxValue;
    std::auto_ptr<Parameters::Parameter<FormulaParam> > zRe, zIm, zJ, zK;
436

    };
438
```

```
std::auto_ptr<Parameters::Parameter<Algorythm*> > JuliaEscapeTime::getInterpolator(const double*
    factor1Pointer, Algorythm* other, const double* factor2Pointer)
{
  JuliaEscapeTime* otheralg = dynamic_cast<JuliaEscapeTime*>(other);
  if(otheralg)
  {
    /*Algorythm* algorythm = this->algorythm->getValue()->getInterpolator(factor1Pointer, othercalc->algorythm
        ->getValue(), factor2Pointer)->getValue();*/
    std::auto_ptr<Parameters::Parameter<Formula*> > formula(this->formula->getValue()->getInterpolator(
        factor1Pointer, otheralg->formula->getValue(), factor2Pointer));
    Parameters::Parameter<int>* maxIterations = new Parameters::InterpolatingParam<int, Parameters::
        NumberInterpolator<int> >(this->maxIterations, factor1Pointer, otheralg->maxIterations,
        factor2Pointer, Parameters::NumberInterpolator<int>());
    Parameters::Parameter<FormulaParam>* maxValue = new FormulaParamInterpolator(this->maxValue,
        factor1Pointer, otheralg->maxValue, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
    Parameters::Parameter<FormulaParam>* zRe = new FormulaParamInterpolator(this->zRe, factor1Pointer,
        otheralg->zRe, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
    Parameters::Parameter<FormulaParam>* zIm = new FormulaParamInterpolator(this->zIm, factor1Pointer,
        otheralg->zIm, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
    Parameters::Parameter<FormulaParam>* zJ = new FormulaParamInterpolator(this->zJ, factor1Pointer, otheralg
        ->zJ, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
    Parameters::Parameter<FormulaParam>* zK = new FormulaParamInterpolator(this->zK, factor1Pointer, otheralg
        ->zK, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
    return std::auto_ptr< Parameters::Parameter<Algorythm*> > (new AlgorythmParameter(std::auto_ptr<Algorythm
        >(new JuliaEscapeTime(formula.get(), zRe, zIm, zJ, zK, maxIterations, maxValue)), formula, std::
        auto_ptr<Parameters::Parameter<int> >(maxIterations), std::auto_ptr<Parameters::Parameter<
        FormulaParam> >(maxValue), std::auto_ptr<Parameters::Parameter<FormulaParam> >(zRe), std::auto_ptr<
        Parameters::Parameter<FormulaParam> >(zIm), std::auto_ptr<Parameters::Parameter<FormulaParam> >(zJ),
        std::auto_ptr<Parameters::Parameter<FormulaParam> >(zK)));
  }
  // TODO: Interpolation for different Algorythms

  // return std::auto_ptr<Parameters::Parameter<Algorythm*> >(new JuliaEscapeTime(formula, zRe, zIm,
      maxIterations, maxValue));
}

JuliaEscapeTime* JuliaEscapeTime::getValue()
{
  return this;
}

StandardFormula::StandardFormula(Parameters::Parameter<FormulaParam>* cRe, Parameters::Parameter<FormulaParam
    >* cIm, Parameters::Parameter<FormulaParam>* cJ, Parameters::Parameter<FormulaParam>* cK)
  : cRe(cRe),
  cIm(cIm),
  cJ(cJ),
  cK(cK)
{}

void StandardFormula::prepare()
{
  cRe->prepareMultiGetValue();
  cIm->prepareMultiGetValue();
}

void StandardFormula::operator() (FormulaParam* real, FormulaParam* imag, FormulaParam* j, FormulaParam* k)
{
  //std::complex<double> c(cRe->getValue(), cIm->getValue());
  //std::complex<double> value(*real, *imag);

  ///*std::complex<double> c(0.0, 0.0);
  //std::complex<double> value(0.0, 0.0);*/
  //value = std::pow(value, 2);
  //value = value + c;

  double realValue = *real * *real - *imag * *imag - *j * *j - *k * *k   + cRe->getValue();
  double imagValue = 2 * *real * *imag     + cIm->getValue();
  double jValue = 2* *real * *j  + cJ->getValue();
  double kValue = 2* *real * *k  + cK->getValue();

  *real = realValue;
  *imag = imagValue;
  *j = jValue;
  *k = kValue;
}

class FormulaParameter : public Parameters::Parameter<Formula*>
{
public:
  FormulaParameter(std::auto_ptr<Formula> formula, std::auto_ptr<Parameters::Parameter<FormulaParam> > cRe,
      std::auto_ptr<Parameters::Parameter<FormulaParam> > cIm, std::auto_ptr<Parameters::Parameter<
      FormulaParam> > cJ, std::auto_ptr<Parameters::Parameter<FormulaParam> > cK)
    : formula(formula),
    cRe(cRe),
    cIm(cIm),
    cJ(cJ),
    cK(cK)
  {}

  virtual Formula* getValue()
  {
    return formula.get();
  }

private:
  std::auto_ptr<Formula> formula;
  std::auto_ptr<Parameters::Parameter<FormulaParam> > cRe, cIm, cJ, cK;

};
```

```
520 std::auto_ptr<Parameters::Parameter<Formula*> > StandardFormula::getInterpolator(const double* factor1Pointer,
        Formula* other, const double* factor2Pointer)
{
522   StandardFormula* otherform = dynamic_cast<StandardFormula*>(other);
    if(otherform)
524   {
      /*Algorythm* algorythm = this->algorythm->getValue()->getInterpolator(factor1Pointer, othercalc->algorythm
          ->getValue(), factor2Pointer)->getValue();*/
526     Parameters::Parameter<FormulaParam>* re = new JuliaEscapeTime::FormulaParamInterpolator(this->cRe,
          factor1Pointer, otherform->cRe, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
      Parameters::Parameter<FormulaParam>* im = new JuliaEscapeTime::FormulaParamInterpolator(this->cIm,
          factor1Pointer, otherform->cIm, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
528     Parameters::Parameter<FormulaParam>* j = new JuliaEscapeTime::FormulaParamInterpolator(this->cJ,
          factor1Pointer, otherform->cJ, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
      Parameters::Parameter<FormulaParam>* k = new JuliaEscapeTime::FormulaParamInterpolator(this->cK,
          factor1Pointer, otherform->cK, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
530     return std::auto_ptr< Parameters::Parameter<Formula*> > (new FormulaParameter(std::auto_ptr<
          StandardFormula>(new StandardFormula(re, im, j, k)), std::auto_ptr<Parameters::Parameter<FormulaParam
          > >(re), std::auto_ptr<Parameters::Parameter<FormulaParam> >(im), std::auto_ptr<Parameters::Parameter
          <FormulaParam> >(j), std::auto_ptr<Parameters::Parameter<FormulaParam> >(k)));
    }
532   // TODO: Interpolation for different Formula

534   return std::auto_ptr<Parameters::Parameter<Formula*> >();
    // return std::auto_ptr<Parameters::Parameter<Formula*> >(new StandardFormula());
536 }

538 Formula* StandardFormula::getValue()
{
540   return this;
}
542
StandardFormulaChooser::StandardFormulaChooser(Parameters::ParametersPanel* parametersPanel, Parameters::
      ParameterGui<FormulaParam>* cReGui, Parameters::ParameterGui<FormulaParam>* cImGui, Parameters::
      ParameterGui<FormulaParam>* cJGui, Parameters::ParameterGui<FormulaParam>* cKGui)
544   : Parameters::ParameterChooser<Formula*>(parametersPanel),
    parametersPanel(*parametersPanel),
546   cImGui(cImGui),
    cReGui(cReGui),
548   cJGui(cJGui),
    cKGui(cKGui),
550   formula()
{
552   myName = "z + c";
    wxBoxSizer* sizer = new wxBoxSizer(wxHORIZONTAL);
554   wxStaticText* text = new wxStaticText(this, -1, myName);
    text->SetFont(Parameters::AbstractParameterGui::FONT_2);
556   sizer->Add(text);
    this->SetSizer(sizer);
558 }

560 /**
  * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
562 **/
  void StandardFormulaChooser::selected()
564 {
    /*parametersPanel.addParameter(cReGui);
566   parametersPanel.addParameter(cImGui);
    parametersPanel.addParameter(cJGui);
568   parametersPanel.addParameter(cKGui);*/
}
570
572 /**
  * Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
  **/
574 void StandardFormulaChooser::unselected()
{
576   //parametersPanel.removeParameter(cImGui);
    //parametersPanel.removeParameter(cReGui);
578   //parametersPanel.removeParameter(cJGui);
    //parametersPanel.removeParameter(cKGui);
580 }

582 Formula* StandardFormulaChooser::getValue()
{
584   formula.reset(new StandardFormula(cReGui, cImGui, cJGui, cKGui));
    return formula.get();
586 }

588 mu::value_type atan2callback(mu::value_type arg1, mu::value_type arg2)
{
590   return std::atan2(arg1, arg2);
}
592

594 CustomFormula::CustomFormula(wxString formulaText, Parameters::Parameter<FormulaParam>* cRe, Parameters::
      Parameter<FormulaParam>* cIm, Parameters::Parameter<FormulaParam>* cJ, Parameters::Parameter<FormulaParam
      >* cK)
    : formulaText(formulaText),
596   cRe(cRe),
    cIm(cIm),
598   cJ(cJ),
    cK(cK),
600   parser(),
    zr(0),
602   zi(0),
    zj(0),
604   zk(0),
```

```
    cr(0),
606 ci(0),
    cj(0),
608 ck(0)
{
610 parser.SetExpr(std::wstring(formulaText));
    parser.DefineFun(L"atan2", atan2callback, true);
612 parser.DefineVar(L"zr", &zr);
    parser.DefineVar(L"zi", &zi);
614 parser.DefineVar(L"zj", &zj);
    parser.DefineVar(L"zk", &zk);
616 parser.DefineVar(L"cr", &cr);
    parser.DefineVar(L"ci", &ci);
618 parser.DefineVar(L"cj", &cj);
    parser.DefineVar(L"ck", &ck);
620 }

622 void CustomFormula::prepare()
{
624 cRe->prepareMultiGetValue();
    cIm->prepareMultiGetValue();
626 }

628 void CustomFormula::operator() (FormulaParam* real, FormulaParam* imag, FormulaParam* j, FormulaParam* k)
{
630 //std::complex<double> c(cRe->getValue(), cIm->getValue());
    //std::complex<double> value(*real, *imag);
632
    ///*std::complex<double> c(0.0, 0.0);
634 //std::complex<double> value(0.0, 0.0);*/
    //value = std::pow(value, 2);
636 //value = value + c;

638 cr = cRe->getValue();
    ci = cIm->getValue();
640 cj = cJ->getValue();
    ck = cK->getValue();
642
    zr = *real;
644 zi = *imag;
    zj = *j;
646 zk = *k;

648 int num = 4;

650 try
    {
652   double* values = parser.Eval(num);

654   *real = values[0];
      *imag = values[1];
656   *j = values[2];
      *k = values[3];
658 }catch(mu::Parser::exception_type& exception)
    {
660   wxMessageBox(exception.GetMsg());
      throw;
662 }
}
664
std::auto_ptr<Parameters::Parameter<Formula*> > CustomFormula::getInterpolator(const double* factor1Pointer,
      Formula* other, const double* factor2Pointer)
666 {
    CustomFormula* otherform = dynamic_cast<CustomFormula*>(other);
668 if(otherform)
    {
670   /*Algorythm* algorythm = this->algorythm->getValue()->getInterpolator(factor1Pointer, othercalc->algorythm
          ->getValue(), factor2Pointer)->getValue();*/
      Parameters::Parameter<FormulaParam>* re = new JuliaEscapeTime::FormulaParamInterpolator(this->cRe,
          factor1Pointer, otherform->cRe, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
672   Parameters::Parameter<FormulaParam>* im = new JuliaEscapeTime::FormulaParamInterpolator(this->cIm,
          factor1Pointer, otherform->cIm, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
      Parameters::Parameter<FormulaParam>* j = new JuliaEscapeTime::FormulaParamInterpolator(this->cJ,
          factor1Pointer, otherform->cJ, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
674   Parameters::Parameter<FormulaParam>* k = new JuliaEscapeTime::FormulaParamInterpolator(this->cK,
          factor1Pointer, otherform->cK, factor2Pointer, Parameters::NumberInterpolator<FormulaParam>());
      return std::auto_ptr< Parameters::Parameter<Formula*> > (new FormulaParameter(std::auto_ptr<CustomFormula
          >(new CustomFormula(formulaText, re, im, j, k)), std::auto_ptr<Parameters::Parameter<FormulaParam> >(
          re), std::auto_ptr<Parameters::Parameter<FormulaParam> >(im), std::auto_ptr<Parameters::Parameter<
          FormulaParam> >(j), std::auto_ptr<Parameters::Parameter<FormulaParam> >(k)));
676 }
    // TODO: Interpolation for different Formula
678
    return std::auto_ptr<Parameters::Parameter<Formula*> >();
680 // return std::auto_ptr<Parameters::Parameter<Formula*> >(new StandardFormula());
}
682
Formula* CustomFormula::getValue()
684 {
    return this;
686 }

688 wxString CustomFormula::getFormulaText()
{
690 return formulaText;
}
692
CustomFormulaChooser::CustomFormulaChooser(Parameters::ParametersPanel* parametersPanel, Parameters::
      ParameterGui<FormulaParam>* cReGui, Parameters::ParameterGui<FormulaParam>* cImGui, Parameters::
```

```
        ParameterGui<FormulaParam>* cJGui, Parameters::ParameterGui<FormulaParam>* cKGui)
694   : Parameters::ParameterChooser<Formula*>(parametersPanel),
    parametersPanel(*parametersPanel),
696   cImGui(cImGui),
    cReGui(cReGui),
698   cJGui(cJGui),
    cKGui(cKGui),
700   formula(),
    text(new wxTextCtrl(this, -1))
702 {
    myName = "Custom Formula";
704   wxBoxSizer* sizer = new wxBoxSizer(wxHORIZONTAL);
    // text->SetFont(Parameters::AbstractParameterGui::FONT_2);
706   sizer->Add(text);
    this->SetSizer(sizer);
708 }

710 void CustomFormulaChooser::setValue(Formula* value)
{
712   CustomFormula* formula = dynamic_cast<CustomFormula*>(value);
    if(formula)
714   {
      text->SetValue(formula->getFormulaText());
716   }
}
718
/**
720 * Wird ausgefuehrt, wenn der ParameterChooser aktiviert wurde.
**/
722 void CustomFormulaChooser::selected()
{
724   /*parametersPanel.addParameter(cReGui);
    parametersPanel.addParameter(cImGui);
726   parametersPanel.addParameter(cJGui);
    parametersPanel.addParameter(cKGui);*/
728 }

730 /**
* Wird ausgefuehrt, wenn der ParameterChooser deaktiviert wurde.
732 **/
void CustomFormulaChooser::unselected()
734 {
    /*parametersPanel.removeParameter(cImGui);
736   parametersPanel.removeParameter(cReGui);
    parametersPanel.removeParameter(cJGui);
738   parametersPanel.removeParameter(cKGui);*/
}
740
Formula* CustomFormulaChooser::getValue()
742 {
    formula.reset(new CustomFormula(text->GetValue(), cReGui, cImGui, cJGui, cKGui));
744   return formula.get();
}
```

Quellcode/Calculators.cpp