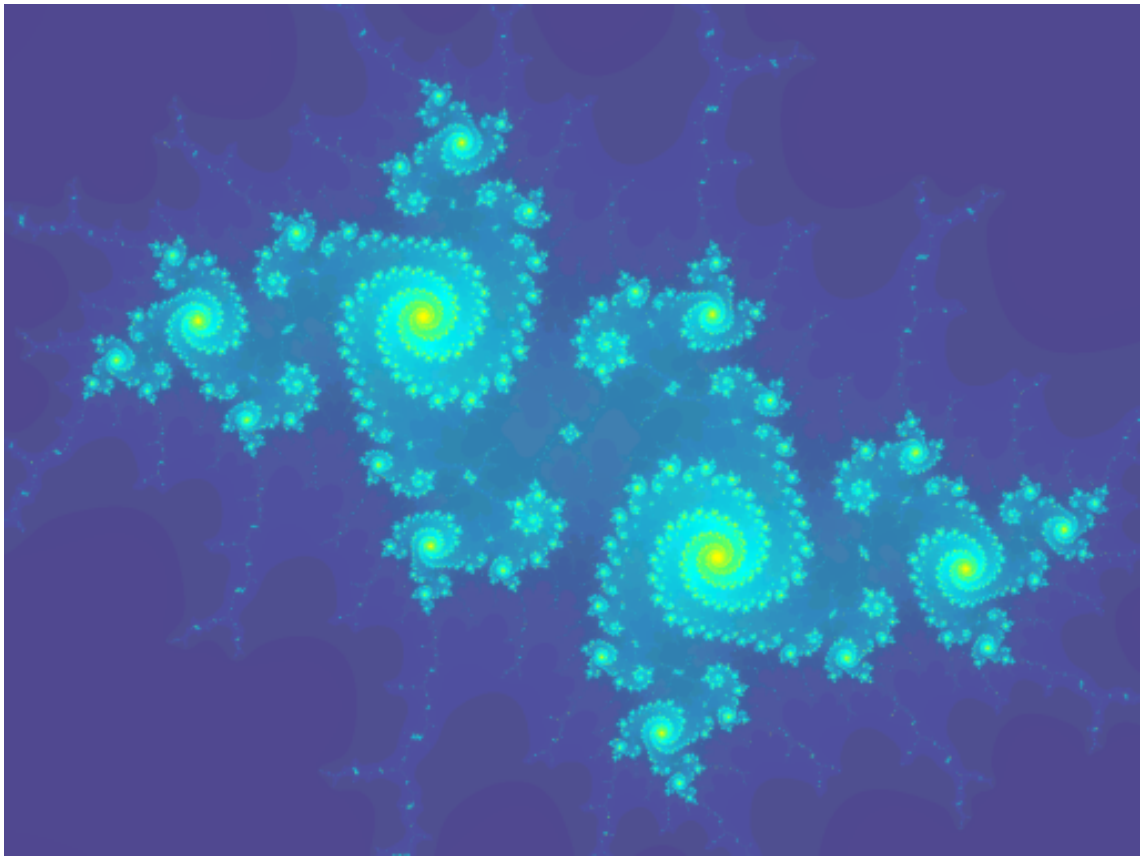
Facharbeit im Leistungskurs Mathematik

Programmierung von Feigenbaumdiagrammen und Mandelbrotmengen

Roland Riegel

Februar 2003



Dieses Dokument ist die in Einzelheiten verbesserte, aber im Wesentlichen unveränderte Facharbeit, welche ich im Jahr 2002 sowie 2003 im Leistungskurs Mathematik erstellt habe.

Für den Leser uninteressante Anhänge wie z. B. die Erläuterungen zu der der Originalarbeit beiliegenden CD-ROM wurden aus dieser Fassung gestrichen.

Das im Rahmen dieser Arbeit entwickelte Programm ChaosExplorer ist wie auch die Facharbeit selbst im Internet auf http://www.roland-riegel.de erhältlich.

Inhaltsverzeichnis

1. Einführung

Fraktale sind spätestens seit der Entdeckung des sogenannten *Apfelmännchens* um 1980 herum in der Öffentlichkeit bekannt. Es ist nur eines von unendlich vielen Gebilden, die „auf jeder Stufe der Vergrößerung eine komplexe und detaillierte Struktur“ besitzen und *selbstähnlich* sind, d. h. dass „jedes kleine Stück [...] die Struktur des Gesamtobjekts hat“.[1]

Doch bereits schon am Anfang des letzten Jahrhunderts forschte ein französischer Mathematiker names Gaston Maurice Julia (1893–1978) auf dem Gebiet der fraktalen Mathematik und entdeckte die nach ihm benannten Juliamengen. Werden diese in der Gaußschen Zahlenebene dargestellt, entstehen Fraktale. Leider hatte er damals mangels Computer noch keine Möglichkeit, seine Entdeckung sichtbar zu machen.

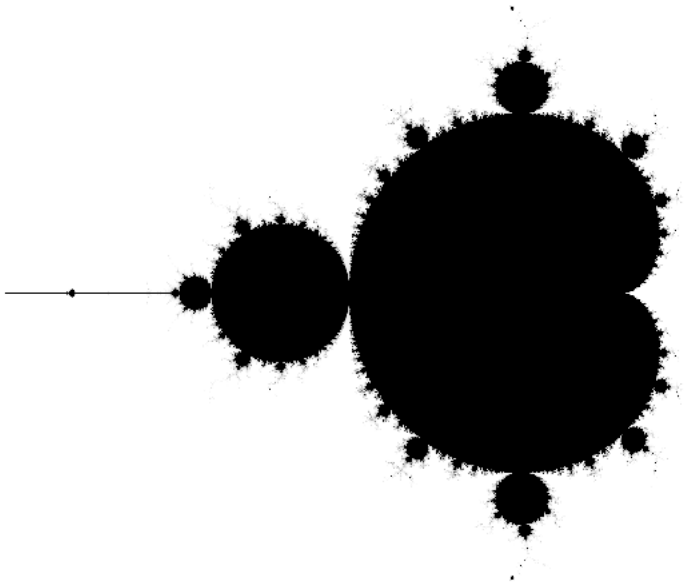


Abbildung 1: Das Apfelmännchen

Auf der Basis seiner Erkenntnisse entdeckte Benoît B. Mandelbrot (*1924), ebenfalls ein Franzose, die Mandelbrotmengen, deren Darstellung in der Gaußschen Zahlenebene die Mandelbrotfraktale entstehen lässt. Das Apfelmännchen (s. Abbildung 1) ist eines dieser Mandelbrotfraktale.

Als einer der ersten entwickelte er ein Computerprogramm, welches er auf einem schnellen Computer laufen ließ. Erst jetzt konnte er die Mathematik, mit welcher er sich zuvor befasst hatte, sehen und die entstandenen Fraktale betrachten.

Mit der Thematik auf erstaunliche Weise verwandt ist das Feigenbaumdiagramm, ebenfalls nach einem Mathematiker namens Mitchell J. Feigenbaum (*1945) benannt. Er suchte einen mathematischen Hintergrund für Wachstumsvorgänge in der Natur, z. B.

[1] [6]

die explosionsartige Vermehrung innerhalb einer Bakterienpopulation. Die Schwierigkeit bestand darin, eine mathematische Beschreibung für all die äußeren Einflüsse wie Platzmangel oder Krankheiten zu finden, die das Wachstum hemmen und schließlich zu einer Obergrenze der Populationsgröße führen.

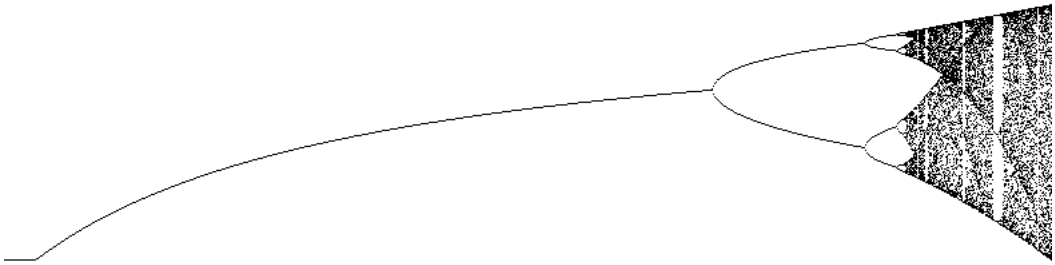


Abbildung 2: Das Feigenbaumdiagramm

Das sog. Feigenbaumdiagramm, dargestellt in Abbildung 2, entsteht durch die Darstellung dieser Obergrenze in Abhängigkeit von der Wachstumsgeschwindigkeit. Es gibt viele Ausführungen des Diagramms, welche z. T. mit einigen Merkmalen verblüffende Gemeinsamkeiten zum Apfelmännchen aufweisen. Dies wird uns später auf Grund der Ähnlichkeit der zur Entstehung der beiden Grafiktypen nötigen mathematischen Beziehungen klar werden.

Das Ziel dieser Arbeit wird sein, in Abschnitt 2 zunächst die mathematischen Grundlagen für die Berechnung dieser Fraktale zu erarbeiten. Wegen der heutigen leistungsfähigen Computer wird es uns dann möglich sein, mit den gewonnenen Kenntnissen ein Computerprogramm zu erstellen (Abschnitt 3), mit dem man einfach und komfortabel diese graphisch interessanten Gebilde berechnen kann. In Abschnitt 4 wollen wir mit diesem Programm experimentieren und untersuchen, welchen Einfluss die Berechnungsparameter auf das Erscheinungsbild der Figuren haben. Daneben lernen wir typische Muster innerhalb der Grafiken kennen.

2. Mathematische Grundlagen

Um später (Abschnitt 3) ein Programm erstellen zu können, welches Feigenbaumdiagramme und Mandelbrotmengen berechnet und grafisch darstellt, wollen wir uns, wie schon in der Einleitung in Abschnitt 1 erwähnt, im Folgenden die hierfür erforderlichen mathematischen Grundlagen erarbeiten.

2.1. Mandelbrotmengen

2.1.1. Definition

Iteriert man

$$z_{n+1} = {z_n}^p + c \qquad n \in \mathbb{N}_0; z_n, c \in \mathbb{C}; p \in \mathbb{R}$$

wobei $z_0 = 0$, so gehört c zur Mandelbrotmenge $\mathbb{M}$, wenn der Betrag von z_n nicht gegen ∞ divergiert. Anders geschrieben:

$$\mathbb{M} = \{c \in \mathbb{C} \mid \lim_{n\to\infty} |z_n| \neq \infty\}$$

Es sei angemerkt, dass man den Exponenten in der Iterationsformel auch als $p \in \mathbb{C}$ definieren kann. Darauf will ich in dieser Facharbeit aber nicht näher eingehen.

2.1.2. Quadrierung der Zahlen

Wie auch in [1] beginnen wir zunächst mit der einfachen Quadrierung von reellen Zahlen:

$$a_{n+1} = {a_n}^2 \qquad a_n \in \mathbb{R}; n \in \mathbb{N}_0$$

Es wird also die Anfangszahl fortlaufend quadriert. Man wählt ein beliebiges $a_0 \in \mathbb{R}$ und erhält durch Quadrierung a_1. Anschließend geht man von a_1 aus und quadriert erneut, um a_2 zu erhalten. Dies wiederholt man für a_3, a_4 usw. bis a_n. Die ständige Wiederverwendung des Ergebnisses einer Rechnung für die selbe Rechnung nennt man *Iteration*.

Ein Beispiel (Zahlenwerte auf vier Stellen gerundet):

$$\begin{aligned}
a_0 &= 1.24 \\
a_1 &= {a_0}^2 = 1.24^2 = 1.5376 \\
a_2 &= {a_1}^2 = 2.3642 \\
a_3 &= 5.5895 \\
a_4 &= 31.2426 \\
a_5 &= 976.0991 \\
a_6 &= 952769.5094
\end{aligned}$$

Man erkennt, dass die Ergebnisse der einzelnen Quadrierungen immer schneller größer werden und gegen unendlich streben.

$$\lim_{n\to\infty} a_n = \infty \text{ für } a_0 = 1.24$$

Dies gilt jedoch, wie leicht festzustellen ist, nicht für alle a_0:

$$\begin{aligned} a_0 &= 0.86 \\ a_1 &= 0.7396 \\ a_2 &= 0.5470 \\ a_3 &= 0.2992 \\ a_4 &= 0.0895 \\ a_5 &= 0.0080 \\ a_6 &= 0.0001 \end{aligned}$$

Ist a_0 negativ, dann hat dies auf die Iteration keinen Einfluss, da schon a_1 durch die Quadrierung wieder positiv wird.

Wenn $-1 < a_0 < 1$ gilt, geht die Folge gegen 0, für $|a_0| > 1$ gegen ∞. Im Fall $a_0 = 1$ gilt natürlich $a_n = 1$ für ein beliebiges n.

Zusammenfassend lässt sich sagen[2]:

$$\begin{aligned} \lim_{n\to\infty} a_n &= 0 \text{ für } |a_0| < 1 \\ \lim_{n\to\infty} a_n &= 1 \text{ für } |a_0| = 1 \\ \lim_{n\to\infty} a_n &= \infty \text{ für } |a_0| > 1 \end{aligned}$$

-1 und 1 sind *Scheidepunkte*, da auf dem Zahlenstrahl „beliebig benachbarte Zahlen auf verschiedenen Seiten von ihnen unterschiedliches Grenzverhalten zeigen“[3].

0 und 1 sind *Fixpunkte*, da gilt:

$$a_n = a_0 \text{ für beliebige } n$$

-1 ist kein Fixpunkt, jedoch geht dieser schon nach der ersten Iteration in einen solchen über, nämlich 1. -1 ist ein *Vorfixpunkt*[4]. Vorfixpunkte gehen allgemein nach ein oder mehreren Iterationen in einen Fixpunkt über.

$$\begin{aligned} \lim_{n\to\infty} a_n &= 1 \text{ für } a_0 = 1 \\ \lim_{n\to\infty} a_n &= 0 \text{ für } a_0 = 0 \\ \lim_{n\to\infty} a_n &= 1 \text{ für } a_0 = -1 \end{aligned}$$

[2] [1], S. 9
[3] [1], S. 14
[4] [1], S. 18

2.1.3. Hinzufügen eines Summanden

Nun wird die Iteration durch Addieren einer Konstante verändert:

$$a_{n+1} = {a_n}^2 + c \qquad a_n, c \in \mathbb{R}; n \in \mathbb{N}_0$$

Wir führen zunächst wieder eine Beispielrechnung mit $c = -1.2$ und $a_0 = 0.3$ durch:

$$\begin{aligned} a_0 &= 0.3 \\ a_1 &= -1.11 \\ a_2 &= 0.0321 \\ a_3 &= -1.1990 \\ a_4 &= 0.2375 \\ a_5 &= -1.1436 \\ a_6 &= 0.1078 \end{aligned}$$

Wie es scheint, pendeln die Ergebnisse sowohl mit ungeradem als auch mit geradem n um jeweils einen Grenzwert. Diese beiden Grenzwerte heißen *Attraktoren*[5].

Bei dieser Iteration sind die Fixpunkte nicht so selbstverständlich wie bei der rein quadratischen Form von oben. Nennt man sie x_f, so gilt:

$${x_f}^2 + c = x_f$$

Formt man dies für das obige Beispiel um, erhält man eine Gleichung zweiten Grades und kann die Fixpunkte ausrechnen. Aus der obigen Bedingung folgt:

$${x_f}^2 - x_f + c = 0$$

Mit der Formel zum Lösen quadratischer Gleichungen der Form $ax^2 + bx + c = 0$,

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

lässt sich formulieren:

$$x_{f_{1,2}} = \frac{1 \pm \sqrt{1 - 4c}}{2}$$

und somit:

$$\begin{aligned} x_{f_{1,2}} &= \frac{1 \pm \sqrt{5.8}}{2} \\ x_{f_1} &\approx 1.7042 \\ x_{f_2} &\approx -0.7042 \end{aligned}$$

Das $\approx$-Zeichen ist hier nicht unberechtigt. Berechnet man x_{f_1} bzw. x_{f_2} mit dem Taschenrechner und führt dann eine Iteration durch, ändert sich der Wert tatsächlich nicht.

[5] [1], S. 10

Dies ist jedoch abhängig von der internen Rechengenauigkeit des Taschenrechners oder Computers, da $\sqrt{5.8}$ eine irrationale Zahl ist[6] und daher nur als Näherungswert be- und verrechnet werden kann. So sind also alle Rechnungen in der Praxis nur Näherungen, so dass daraus folgt, dass bei Iterationen beliebig nahe beieinander liegende Ausgangswerte a_0 völlig unterschiedliche Ergebnisse für $n \to \infty$ haben können.

Testen wir dies mit den nur um 0.0001 von x_{f_1} abweichenden Ausgangswerten a_0:

a_0	=	1.7041	a_0	=	1.7043
a_1	=	1.7040	a_1	=	1.7046
a_2	=	1.7035	a_2	=	1.7058
a_3	=	1.7018	a_3	=	1.7097
a_4	=	1.6961	a_4	=	1.7232
a_5	=	1.6769	a_5	=	1.7693
a_6	=	1.6120	a_6	=	1.9305
a_7	=	1.3986	a_7	=	2.5267
a_8	=	0.7560	a_8	=	5.1841
a_9	=	−0.6285	a_9	=	25.6745
a_{10}	=	−0.8050	a_{10}	=	657.9783

Dies zeigt, dass sich nahe an x_{f_1} liegende a_0 in der Iteration ganz unterschiedlich verhalten. x_{f_1} ist ein *Scheidepunkt* (vgl. Abschnitt 2.1.2).

Überprüfen wir nun x_{f_2} in gleicher Form:

a_0	=	−0.7043	a_0	=	−0.7041
a_1	=	−0.7040	a_1	=	−0.7042
a_2	=	−0.7044	a_2	=	−0.7040
a_3	=	−0.7038	a_3	=	−0.7043
a_4	=	−0.7047	a_4	=	−0.7039
a_5	=	−0.7034	a_5	=	−0.7045
a_6	=	−0.7053	a_6	=	−0.7037
a_7	=	−0.7026	a_7	=	−0.7048
a_8	=	−0.7063	a_8	=	−0.7032
a_9	=	−0.7011	a_9	=	−0.7055
a_{10}	=	−0.7085	a_{10}	=	−0.7023

Dieses Ergebnis überrascht: Im Gegensatz zu x_{f_1} ist x_{f_2} kein Scheidepunkt. Fixpunkte sind also nicht immer auch gleichzeitig Scheidepunkte.

Nun wollen wir analog zu der Vorgehensweise in Abschnitt 2.1.2 Vorfixpunkte bestimmen. Leicht zu finden, da sofort ersichtlich, sind die Fixpunkte mit umgekehrtem Vorzeichen. Schon bei der ersten Iteration verschwindet das negative Vorzeichen durch die Quadrierung und entspricht somit wieder dem jeweiligen Fixpunkt. Hier ist diese Rechnung am Beispiel von x_{f_1} durchgeführt, ohne den Taschenrechner und dessen unge-

[6] [1], S. 10

naue Rundung zu benutzen:

$$\begin{aligned} a_0 &= -x_{f_1} = -\frac{1+\sqrt{5.8}}{2} \\ a_1 &= \left(-\frac{1+\sqrt{5.8}}{2}\right)^2 - 1.2 = \\ &= \frac{\left(-1-\sqrt{5.8}\right)^2}{4} - 1.2 = \\ &= \frac{2\sqrt{5.8}+6.8}{4} - 1.2 = \\ &= \frac{1+\sqrt{5.8}}{2} = \\ &= x_{f_1} \end{aligned}$$

Natürlich existieren auch „Vorvorfixpunkte“, die erst nach mehreren Iterationen in einen Fixpunkt übergehen, wozu sie aber zuerst die zugehörigen Vorfixpunkte erreichen müssen. Ich möchte dies hier jedoch nicht näher erläutern, da dies den Rahmen dieser Facharbeit übersteigen würde.

2.1.4. Ausweitung auf die komplexen Zahlen

Erweitern wir die Iteration aus Abschnitt 2.1.3 dahingehend, dass wir statt reeller Zahlen aus $\mathbb{R}$ komplexe Zahlen aus $\mathbb{C}$ verwenden, dann entspricht sie schon der Definition der Mandelbrotmenge (s. Abschnitt 2.1.1).

Für $p = 2$:

$$z_{n+1} = {z_n}^2 + c \qquad n \in \mathbb{N}_0; z_n, c \in \mathbb{C}$$

Natürlich lassen sich die eben gemachten Beobachtungen auch auf die komplexen Zahlen übertragen. Untersuchen wir dazu wieder die Fixpunkte, zunächst mit $c = 0$. Beispiel für $z_0 = 1.5 + 0.6i$:

$$\begin{array}{lcllcl} z_0 &=& 1.5 + 0.6i & |z_0| &=& 1.6155 \\ z_1 &=& 1.89 + 1.8i & |z_1| &=& 2.61 \\ z_2 &=& 0.3321 + 6.804i & |z_2| &=& 6.8121 \\ z_3 &=& -46.1841 + 4.5192i & |z_3| &=& 46.4047 \end{array}$$

Betrachtet man die Beträge der Iterationsergebnisse, so fällt auf, dass die Entfernung vom Ursprung, also der Betrag, immer weiter zunimmt.

Eine andere Iteration:

$$\begin{array}{lcllcl} z_0 &=& 0.8 + 0.3i & |z_0| &=& 0.8544 \\ z_1 &=& 0.55 + 0.48i & |z_1| &=& 0.73 \\ z_2 &=& 0.0721 + 0.528i & |z_2| &=& 0.5329 \\ z_3 &=& -0.2736 + 0.0761i & |z_3| &=& 0.2840 \\ z_4 &=& 0.0691 - 0.0416i & |z_4| &=& 0.0807 \end{array}$$

Wie man leicht sieht, nehmen hier die Beträge ab. Es liegt die Vermutung nahe, dass für die Fixpunkte $|z_0| = 1$ gelten muss.

$$\begin{array}{lcllcl} z_0 & = & 0.6 + 0.8i & |z_0| & = & 1 \\ z_1 & = & -0.28 + 0.96i & |z_1| & = & 1 \\ z_2 & = & -0.8432 - 0.5376i & |z_2| & = & 1 \end{array}$$

Der Begriff Fix*punkt* bzw. Scheide*punkt* ist hier allerdings nicht mehr ganz zutreffend, da sich z nach jeder Iteration ändert, nur der Betrag bleibt gleich. Daher ist es zweckmäßig, den Begriff der *Scheidekurve* einzuführen, welche die Rolle der Scheidepunkte des reellen Zahlenbereichs übernimmt[7]. Hier, in dem Sonderfall $c = 0$, ist die Scheidekurve ein Kreis mit dem Radius 1 um den Ursprung (Einheitskreis).

Beim Quadrieren besteht eine Analogie zwischen den reellen und den komplexen Zahlen. Diese kann man sich an dem folgenden Zusammenhang verdeutlichen:

$$|z^2| = |z|^2 \qquad z \in \mathbb{C}$$

Der Beweis ist relativ einfach:

$$\begin{aligned} |z^2| &= |(a+bi)^2| = |a^2 - b^2 + 2abi| = \sqrt{(a^2-b^2)^2 + 4a^2b^2} = \\ &= \sqrt{a^4 + b^4 + 2a^2b^2} = \sqrt{(a^2+b^2)^2} = \left(\sqrt{a^2+b^2}\right)^2 = \\ &= |z|^2 \end{aligned}$$

Man erkennt, dass sich die Beträge der komplexen Zahlen bei der Quadrierung nach den in Abschnitt 2.1.2 besprochenen Regeln verhalten, da der Betrag einer komplexen Zahl ja selbst reell ist. Daraus folgt auch, dass die Punkte der Scheidekurve den Betrag 1 haben müssen, da 1, wie festgestellt, Fixpunkt ist.

Bis hierhin haben wir nur die Iteration für $c = 0$ betrachtet. Man erahnt jedoch, wie komplex die Rechnungen werden, wenn man nun noch bei jeder Iteration eine Konstante zu z_n addiert. Die Scheidekurve ist dann kein einfacher Kreis mehr, sondern eine kompliziert verlaufende Grenzlinie, welche die Punkte abgrenzt, die ein unterschiedliches Verhalten während der Iteration zeigen. Die Menge der von dieser Linie eingeschlossenen Punkte nennt man *Juliamenge*, benannt nach Gaston Julia (1893–1978), einem französischen Mathematiker. Am Computer lassen sich Juliamengen leicht darstellen. Dazu wird jeder Punkt z_0, welcher in dem darzustellenden Ausschnitt der Gaußschen Zahlenebene liegt, mit einem zuvor festgelegten c iteriert. Geht die Zahlenfolge der Iteration nicht gegen unendlich, wird der entsprechende Punkt des Bildes schwarz eingefärbt. c bestimmt somit den Inhalt bzw. die Figur der Juliamenge. Ein mögliches Ergebnis sieht man in Abbildung 3.

Der einzige, aber bedeutende Schritt zur Mandelbrotmenge besteht nun darin, nicht z_0 mit dem jeweils zu berechnenden Bildpunkt zu initialisieren, sondern c. z_0 wird auf den Wert 0 festgelegt. Die Mandelbrotmenge enthält also das c jeder einzelnen Juliamenge, deren Ursprung bei der Iteration nicht gegen unendlich divergiert. Berechnet man die Mandelbrotmenge, erhält man als Ergebnis die typische Darstellung wie in Abbildung 1 auf Seite 5.

[7] [1], S. 24

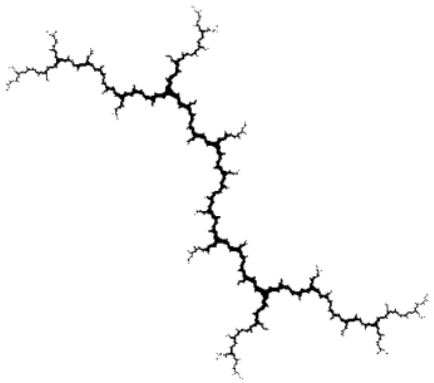


Abbildung 3: Beispiel für eine Julia-Menge

2.1.5. Eigenschaften des Mandelbrot-Fraktals

Im Folgenden werde ich einige mathematische Eigenschaften der Mandelbrotmenge erläutern, die vor allem auch für die spätere Berechnung innerhalb des Computerprogramms entscheidende Vorteile bezüglich der Berechnungsgeschwindigkeit haben.

2.1.5.1. Selbstähnlichkeit

Die Mandelbrotmenge ist *selbstähnlich*[8], da sie „ähnlich einem Teil von sich ist“[9]. Vergrößert man also einen Ausschnitt der Mandelbrotmenge, so kann man einzelne Formen entdecken, die genauso oder ähnlich wie die Mandelbrotmenge selbst aussehen.

Abbildung 4 zeigt hierzu eine Folge von Ausschnittsvergrößerungen der Mandelbrotmenge. Wie man sehen kann, enthält jedes Bild eine Figur, welche nahezu exakt die selbe Form hat wie die ganze Mandelbrotmenge.

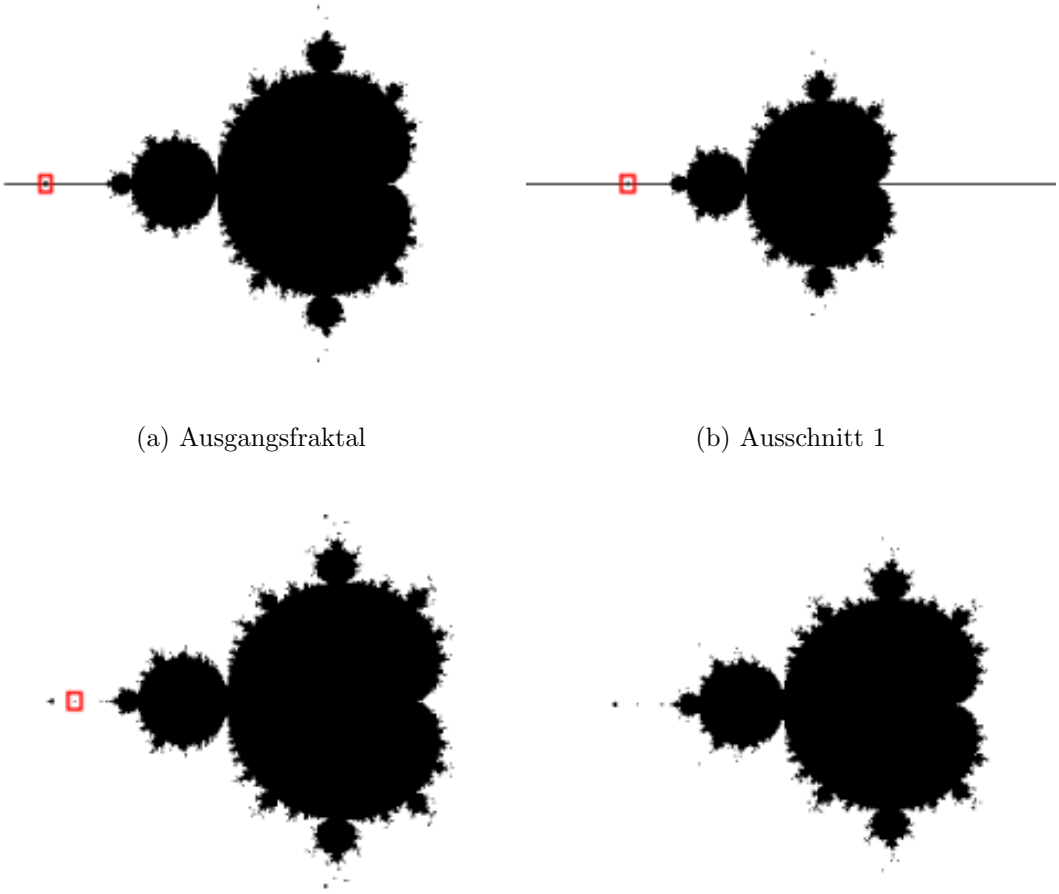
2.1.5.2. Symmetrie

Mandelbrotmengen haben mindestens eine Symmetrieachse, die x-Achse. Wie viele es tatsächlich sind, hängt vom Exponent p (siehe Abschnitt 2.1.1) ab.

Um die Symmetrie zur reellen Achse z. B. für $p = 2$ zu beweisen[10], schreiben wir die Mandelbrot-Iteration nun als Funktion:

$$f_c(z) = z^2 + c \qquad z, c \in \mathbb{C}$$

[8] [1], S. 58
[9] [1], S. 47
[10] [9]



(a) Ausgangsfraktal

(b) Ausschnitt 1

(c) Ausschnitt 2

(d) Ausschnitt 3

Abbildung 4: Die Selbstähnlichkeit der Mandelbrotmenge demonstriert an drei Ausschnittsvergrößerungen

Damit können wir die Iterationswerte wie folgt ausdrücken:

$$\begin{aligned}
z_0 &= 0 \\
z_1 &= f_c(z_0) = c \\
z_2 &= f_c(z_1) = {f_c}^2(z_0) = c^2 + c \\
z_3 &= {f_c}^3(z_0) = c^4 + 2c^3 + c^2 + c \\
z_4 &= {f_c}^4(z_0) = c^8 + 4c^7 + 6c^6 + 6c^5 + 5c^4 + 2c^3 + c^2 + c \\
&\vdots \\
z_n &= {f_c}^n(z_0) = c^k + a_{k-1}c^{k-1} + a_{k-2}c^{k-2} + \cdots + a_2c^2 + c \quad n, k, a_k \in \mathbb{N}_0; k = 2^{n-1}
\end{aligned}$$

Nun muss rechnerisch bewiesen werden, dass $\overline{{f_c}^n(z_0)} = {f_{\overline{c}}}^n(z_0)$ gilt, wobei $\overline{c}$ dem konjugierten (gespiegelten) c entspricht.

$$\begin{aligned}
\overline{{f_c}^n(z_0)} &= \overline{c^k + a_{k-1}c^{k-1} + a_{k-2}c^{k-2} + \cdots + a_2c^2 + c} = \\
&= \overline{c^k} + \overline{a_{k-1}c^{k-1}} + \overline{a_{k-2}c^{k-2}} + \cdots + \overline{a_2c^2} + \overline{c} = \\
&= \overline{c}^k + a_{k-1}\overline{c}^{k-1} + a_{k-2}\overline{c}^{k-2} + \cdots + a_2\overline{c}^2 + \overline{c} = \\
&= {f_{\overline{c}}}^n(z_0)
\end{aligned}$$

Zusammen mit der selbstverständlichen Tatsache, dass

$$\begin{aligned}
|z| &= |\overline{z}| \qquad z \in \mathbb{C} \\
|z| &= |a + bi| = \\
&= \sqrt{a^2 + b^2} = \\
&= \sqrt{a^2 + (-b)^2} = \\
&= |a - bi| = \\
&= |\overline{z}|
\end{aligned}$$

folgt daraus

$$|{f_c}^n(z_0)| = \left|\overline{{f_c}^n(z_0)}\right| = |{f_{\overline{c}}}^n(z_0)|$$

und somit $c \in \mathbb{M} \Leftrightarrow \overline{c} \in \mathbb{M}$, was zu zeigen war.

2.1.5.3. Zusammenhang

Mandelbrotmengen sind zusammenhängend[11], d. h. es gibt beim Mandelbrotfraktal keine „Inseln“. Jeder Punkt der Menge kann von jedem anderen Punkt der Menge aus erreicht werden, ohne die Menge zu verlassen. Dies zu beweisen ist mathematisch nicht einfach, weshalb ich dies hier nicht durchführe.

Sollte es in einem Mandelbrotfraktal doch einmal so aussehen, als wären hier einzelne, abgetrennte Punkte vorhanden, so ist dies auf die beschränkte Rechengenauigkeit sowie Bildauflösung zurückzuführen.

[11] [7], Frage 6h; [1], S. 54

2.1.6. Die Berechnung in der Praxis

Die Definition der Mandelbrotmenge in Abschnitt 2.1.1 ist für eine Berechnung am Computer aus einem einfachen Grund untauglich: Man müsste unendlich viele Iterationsschritte durchführen. Daraus folgt auch, dass man unendlich lange auf die Vervollständigng des Bildes warten müsste. In der Praxis bedient man sich daher einer Annäherung an die Mandelbrotmenge:

$$\mathbb{M} = \{c \in \mathbb{C} \mid |z_n| < A, n \leq N\}$$

Wir iterieren also maximal nur N-mal, und brechen nach N Iterationsschritten ab. N wird *Iterationstiefe* genannt und legt die Genauigkeit fest, mit der sich die berechnete Mandelbrotmenge der mathematisch korrekten Mandelbrotmenge annähert. Zusätzlich überprüfen wir nach jedem Iterationsschritt, ob bereits eindeutig feststeht, dass $|z_n|$ unendlich groß wird. Dies ist in der obigen Definition mit $|z_n| < A$ verdeutlicht. Wird also $|z_n|$ größer als A, nehmen wir an, dass $|z_n|$ gegen unendlich strebt. In der Literatur wird für A meistens der Wert 2 verwendet, da dann feststeht, dass $|z_n|$ gegen unendlich divergiert.

Um dies zu beweisen[12], nimmt man $|z| > 2$ sowie $|z| \geq |c|$ an und folgert:

$$\begin{aligned} |z| > 2 \quad \Rightarrow \quad & |z^2 + c| \geq |z^2| - |c| > 2|z| - |c| > |z| \\ & |z^2 + c| > |z| \end{aligned}$$

Daraus ergibt sich, dass der Wert von $|z|$ bei jeder Iteration steigt und somit unendlich groß wird.

2.1.7. Darstellung in Farbe

Die Mandelbrotmenge sieht man so wie sie in Abbildung 1 auf Seite 5 zu sehen ist eigentlich eher selten. Dies liegt einerseits daran, dass für gewöhnlich lediglich kleine Ausschnitte der Mandelbrotmenge gezeigt werden, aber auch an dem Umstand, dass die Bilder meist farbig sind und so oft einen besonderen visuellen Reiz ausstrahlen.

Um Bilder der Mandelbrotmenge einzufärben, gibt es viele Methoden[13]. Die gebräuchlichste[14] richtet sich nach der Anzahl der ausgeführten Iterationen bis zum Erreichen einer der zwei Abbruchsbedingungen (siehe Abschnitt 2.1.6).

Zum Umrechnen der erreichten Iterationszahl, welche zwischen 0 und der Iterationstiefe liegt, dient ein Farbverlauf (siehe Abbildung 5). Der linke Rand des Farbverlaufs entspricht 0 % der Iterationstiefe, der rechte entspricht 100 % und somit der Farbe der Mandelbrotmenge. Beträgt also z. B. die Iterationstiefe 100, so wird der Bildpunkt, dessen Berechnung nach dem 53. Iterationsschritt abgebrochen wurde, mittels des in Abbildung 5 dargestellten Farbverlaufs türkis eingefärbt. In Bild 6 ist solch ein farbiger Mandelbrotausschnitt dargestellt.

[12] [7], Frage 6d
[13] [5], Abschnitt „Coloring Schemes“
[14] [5], Abschnitt „Coloring Schemes“, Unterabschnitt „Normal“

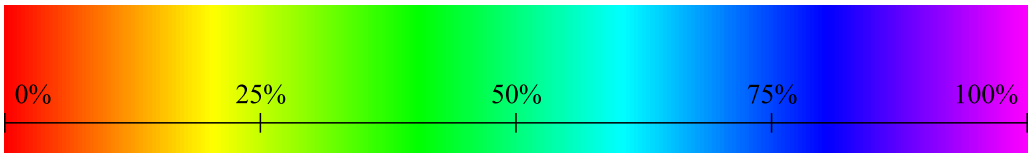


Abbildung 5: Der Farbverlauf bestimmt an Hand des erreichten Bruchteils der Iterationstiefe die Farbe des entsprechenden Bildpunktes

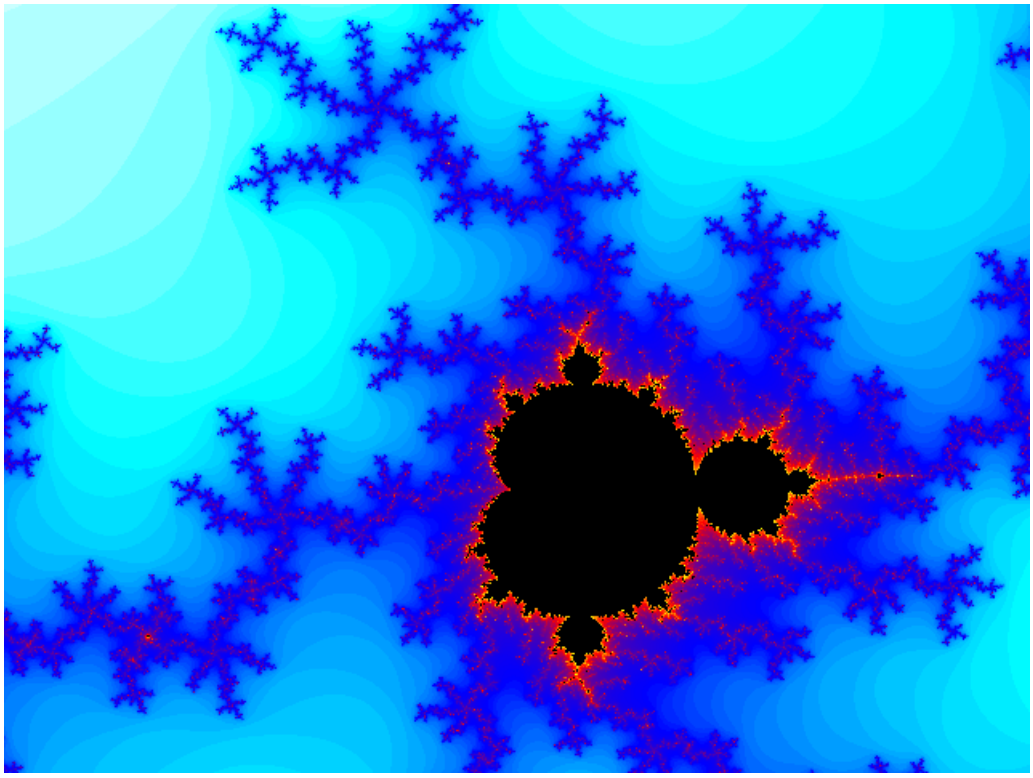


Abbildung 6: Ein Ausschnitt des Mandelbrotfraktals mit eingefärbter Umgebung

2.2. Feigenbaumdiagramm

2.2.1. Nichtlineares Wachstum

Beginnen wir unsere Überlegungen[15] an einem Beispiel aus der Wirtschaft. Legt man z. B. bei einer Bank einen gewissen Geldbetrag K_0 von 10000 Euro zu einem Zinssatz von 6 % für ein Jahr lang an, erhält man nach einem Jahr das $1\frac{6}{100}$-fache seines ursprünglichen Anlagebetrages, also $K_1 = 1\frac{6}{100}K_0$ zurück. In der Regel will man das Geld aber nicht nur für ein Jahr anlegen, so dass man auch den durch die Zinsen verursachten Gewinn anlegt:

$$\begin{aligned} K_1 &= 1\frac{6}{100}K_0 \\ K_2 &= 1\frac{6}{100}K_1 = \left(1\frac{6}{100}\right)^2 K_0 \end{aligned}$$

Nach n Jahren besitzt man also das Kapital

$$K_n = \left(1\frac{6}{100}\right)^n K_0$$

Nennt man den Zinssatz nun verallgemeinert *Wachstumsrate* w und das Anfangskapital *Ausgangsmenge* m_0, erhält man schließlich

$$m_n = (1+w)^n m_0$$

Überträgt man diese Erkenntnis nun auf das Populationswachstum in der Natur, stellt man ein unrealistisches Verhalten fest: Die Populationsdichte wächst in der Praxis für $n \to \infty$ nicht ins Unendliche, da äußere Einflüsse wie Platzmangel, Feinde oder Krankheiten das Wachstum hemmen. Man benötigt also ein mathematisches Modell, welches dies berücksichtigt und für $n \to \infty$ gegen einen bestimmten Wert M geht. Dies kann nur durch eine variable Wachstumsrate w_n erreicht werden, welche mit immer größerem m_n kleiner wird, z. B.

$$w_n = \mu\,(M - m_n)$$

mit μ als konstantem Proportionalitätsfaktor. Durch die nicht mehr konstante Wachstumsrate heißt der Wachstumsvorgang *nichtlinear*. Hier liegt auch *Rückkopplung* vor, da die Wachstumsrate immer auf dem zuvor berechneten Wachstum basiert.

Den obigen mathematischen Ansatz zur Beschreibung natürlicher Wachstumsvorgänge von Populationen machte auch schon Pièrre François Verhulst (1804–1849) in der Mitte des 19. Jahrhunderts. „Entwickelt sich eine Größe m_n in einer Iterationsfolge mit einer Wachstumsrate w_n, so daß also

$$m_{n+1} = (1+w_n)\,m_n,$$

[15] [1], S. 63 ff.

so spricht man von einem *Verhulst-Prozeß*, wenn w_n proportional dem Abstand der Größe m_n von einem angestrebten Grenwert ist.“[16] Drückt man m_{n+1} mit m_n und $w_n = \mu\,(M - m_n)$ aus, so ergibt sich:

$$\begin{aligned} m_{n+1} &= (1 + w_n)\, m_n \\ &= (1 + \mu\,(M - m_n))\, m_n \\ &= (1 + \mu M - \mu m_n)\, m_n \\ &= m_n + \mu M m_n - \mu {m_n}^2 \end{aligned}$$

Möchte man nun nur noch den Zuwachs (oder Gewinn), also $w_n m_n$ betrachten, erhält man (mit $M = 1$):

$$m_{n+1} = \mu m_n - \mu {m_n}^2$$

Diese Formel hat auch Verhulst gefunden.

2.2.2. Das Feigenbaumdiagramm

Es ist möglich, das Verhulst-Wachstum in Abhängigkeit von μ in einem Koordinatensystem darzustellen. Dazu trägt man μ an der x-Achse und $lim_{n\to\infty} m_n$ an der y-Achse an.

Ähnlich wie bei der Berechnung der Mandelbrotmenge in Abschnitt 2.1.6 muss der Grenzwert $lim_{n\to\infty} m_n$ für eine praxistaugliche Berechnung angenähert werden. Für jedes μ führt man dazu eine bestimmte Anzahl n von Wachstumsschritten durch, von denen man aber nur i Schritte ($i < n$) in das Koordinatensystem einzeichnet, und zwar die zuletzt durchgeführten (m_{n-i+1} bis m_n). Diese Einschränkung bewirkt ein sauberes Einschwingen der Iterationswerte (siehe Abschnitt 4.2.2), bevor sie in das Diagramm eingetragen werden.

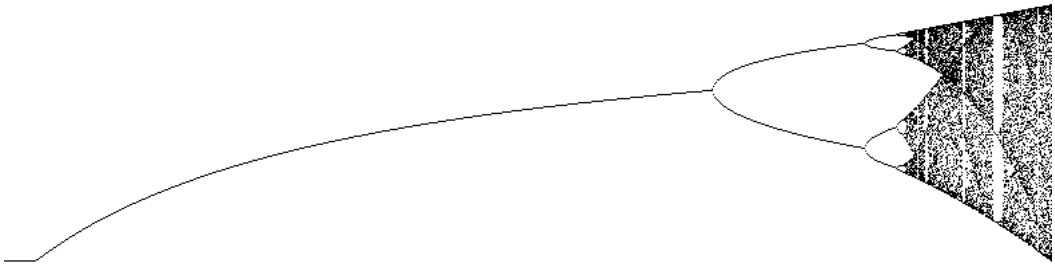


Abbildung 7: Das Feigenbaumdiagramm

Als Ergebnis erhält man das sogenannte Feigenbaumdiagramm, wie es in Abbildung 7 dargestellt ist. Den Namen verdankt es seinem Entdecker, dem Mathematiker Mitchell J. Feigenbaum (*1945).

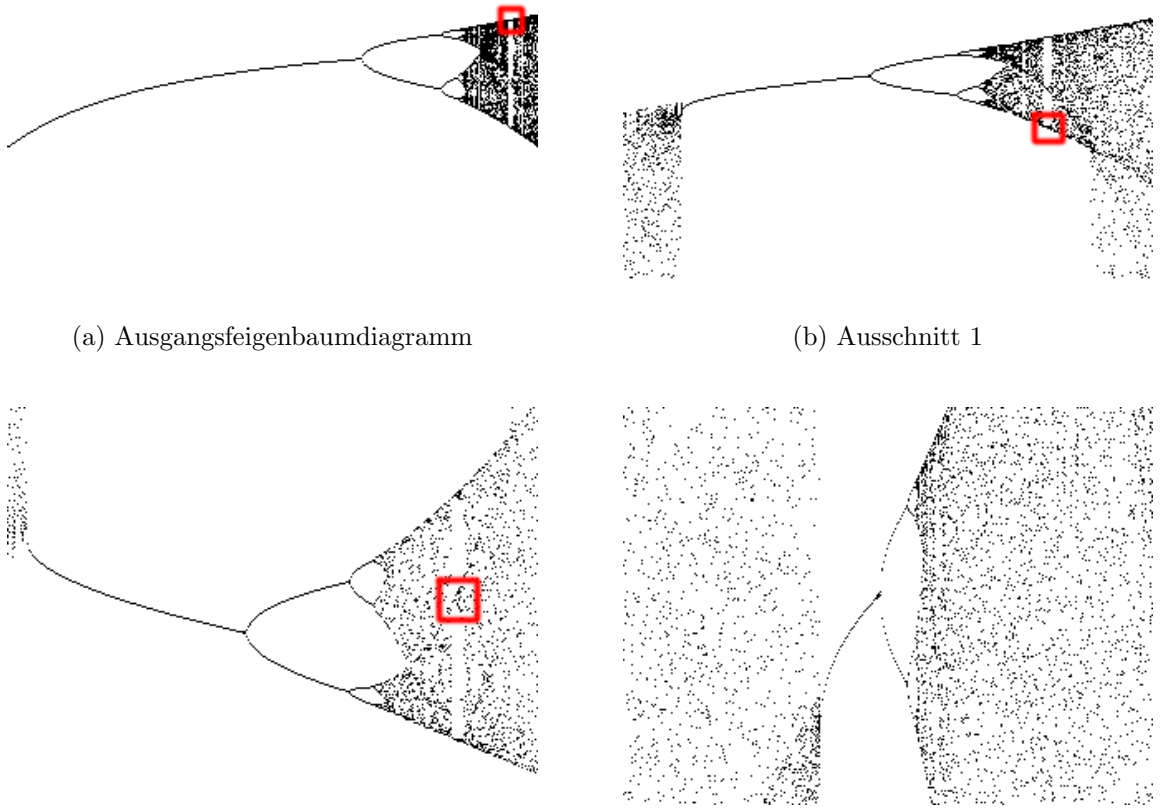
In dem Diagramm sind deutlich diejenigen μ zu erkennen, bei denen das periodische Schwanken von m_n einsetzt bzw. sich die Anzahl der Grenzwerte verdoppelt. Das diffuse

[16] [1], S. 64

„Rauschen“ im rechten Teil des Diagramms rührt vom chaotischen Verhalten her, welches beim Wachstum mit ca. $\mu = 3.6$ eintritt (siehe Abschnitt 4.2.2).

2.2.3. Selbstähnlichkeit

Genau wie das Mandelbrotfraktal auch (siehe Abschnitt 2.1.5.1) zeigt das Feigenbaumdiagramm das Merkmal der Selbstähnlichkeit. Zu beobachten ist dies z. B. innerhalb des „Fensters im Chaos“, wie auch Abbildung 8 zeigt.



(a) Ausgangsfeigenbaumdiagramm

(b) Ausschnitt 1

(c) Ausschnitt 2

(d) Ausschnitt 3

Abbildung 8: Auch das Feigenbaumdiagramm zeigt Selbstähnlichkeit.

2.2.4. Die Feigenbaumkonstante

Erwähnenswert ist eine besondere Eigenschaft des Feigenbaumdiagramms. Das Verhältnis der Abstände der Verzweigungen (siehe Abbildung 9) strebt für $n \to \infty$ gegen einen

besonderen Wert, die sogenannte Feigenbaumkonstante δ.

$$\delta = \lim_{n\to\infty} \frac{l_n}{l_{n+1}}$$

Sie hat ungefähr den Wert $\delta \approx 4.66920$ und ist (ähnlich wie die Kreiszahl π oder die Eulersche Zahl e) eine transzendente Zahl.

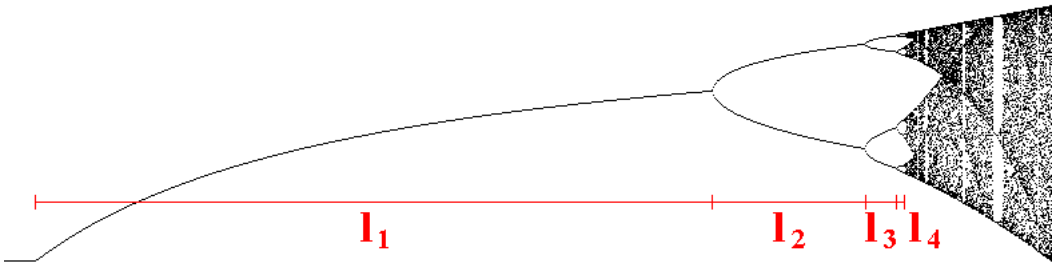


Abbildung 9: Das Verhältnis der Abstände der Periodenverdoppelungen nähert sich der Feigenbaumkonstanten δ an

2.2.5. Ähnlichkeiten mit dem Mandelbrotfraktal

Neben dem oben besprochenen Verhulst-Prozess gibt es noch viele andere Iterationsformeln[17], die ein ähnliches Verhalten zeigen. Geometrisch interessant ist die Variante

$$m_{n+1} = {m_n}^2 + \mu$$

des Feigenbaumdiagramms, welche in Abbildung 10 dargestellt ist. Im Hintergrund eingeblendet erkennt man das Mandelbrotfraktal, welches für den selben Bildausschnitt wie das Feigenbaumdiagramm berechnet wurde.

Auffallend ist, dass sich (von rechts nach links) bei jeder Verzweigung des Feigenbaumdiagramms eine neue „Scheibe“ im Mandelbrotfraktal bildet. Dies kann bis zum Übergang ins Chaos beim Feigenbaumdiagramm beobachtet werden. Auch das Mini-Mandelbrotfraktal an der „Antenne“ des Mandelbrotfraktals findet sich im Feigenbaumdiagramm wieder, dort entdeckt man eine Lücke, ein kleines Fenster im Chaos.

Die Gemeinsamkeiten der beiden Figuren sind nicht verwunderlich, wenn man die Iterationsterme vergleicht:

$$\begin{aligned} m_{n+1} &= {m_n}^2 + \mu \qquad m_n, \mu \in \mathbb{R} \\ x_{n+1} &= {x_n}^2 + c \qquad x_n, c \in \mathbb{C} \end{aligned}$$

Der einzige Unterschied besteht in der Definitionsmenge (natürlich ist auch die Art der Darstellung der Iterationen im Koordinatensystem eine andere). Während bei den Mandelbrotmengen komplexe Zahlen verwendet werden, rechnet man beim Feigenbaumdiagramm mit reellen Zahlen. Wir wissen, dass $\mathbb{R} \subset \mathbb{C}$. Komplexe Zahlen haben ein

[17] [8], Abschnitt „Weitere Feigenbaumszenarios“

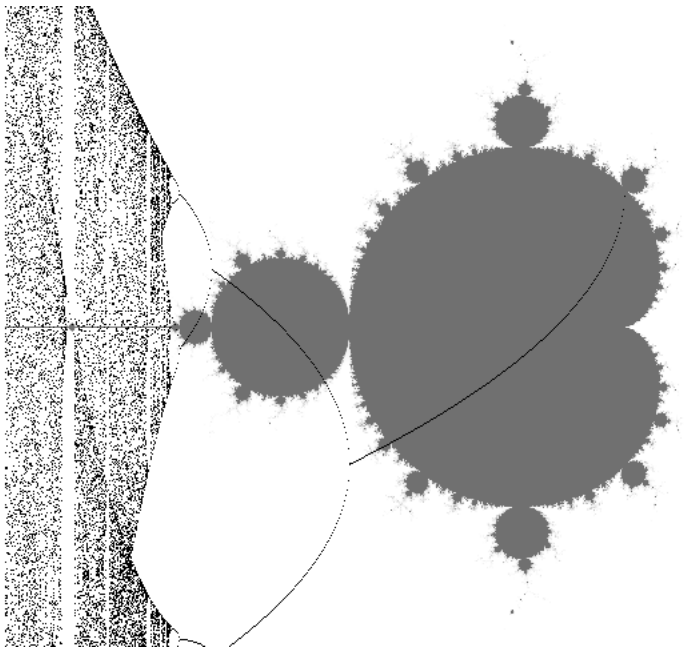


Abbildung 10: Das Mandelbrotfraktal und das geänderte Feigenbaumdiagramm im selben Koordinatensystem

reelles Pendant, wenn ihr imaginärer Teil den Wert 0 hat. So gesehen ist also die obige Feigenbaum-Iteration ein Spezialfall der Mandelbrot-Iteration und taucht im Mandelbrotfraktal wieder auf; sie entspricht der x-Achse.

3. Das Programm ChaosExplorer

Nachdem wir den mathematischen Hintergrund des Feigenbaumdiagramms und vor allem auch der Mandelbrotmengen im letzten Abschnitt kennengelernt haben, ist es uns nun möglich, ein Computerprogramm zu schreiben, mit dem man Mandelbrotfraktale und Feigenbaumdiagramme leicht erstellen und deren Darstellungsparameter verändern kann.

Im Folgenden wird das Programm „ChaosExplorer“ vorgestellt. Neben der Bedienung wird auch die möglichst schnelle und effiziente Berechnung der Computergraphiken erläutert.

3.1. Die Wahl der Programmiersprache

Ich habe mich entschlossen, das Programm mit Visual C++ in Verbindung mit dem Qt-Toolkit[18] umzusetzen. Dies hat zugleich mehrere Vorteile:

- C++ ist in der Ausführung des fertigen Programms relativ schnell, vergleicht man es mit anderen Programmiersprachen wie z. B. Visual Basic. Dies schlägt sich später auch in der Berechnungsgeschwindigkeit der fraktalen Grafiken nieder.
- Das erstellte Programm ist durch die Benutzung des Qt-Toolkit leicht auf andere Rechnersysteme wie z. B. Linux übertragbar. Qt stellt eine Art Schnittstelle zwischen Programm und Bildschirmdarstellung zur Verfügung. Für den Programmcode selbst ist es nicht wichtig, auf welcher Plattform er gerade läuft; Qt sorgt für eine systemspezifische Umsetzung der Befehle. Freundlicherweise stellt Trolltech Qt für nichtkommerzielle Zwecke kostenlos zur Verfügung.
- Der Quellcode ist relativ leicht verständlich und kann auch von Außenstehenden gut beurteilt werden.

Natürlich sind auch die Nachteile dieser Entscheidung zu beachten:

- Die zu entwickelnde Programmoberfläche muss mit Programmcode generiert werden und kann nicht intuitiv mit der Maus zusammengestellt werden. Qt bietet mit dem Qt-Designer zwar einen Ansatz in diese Richtung, ist jedoch bei komplizierteren Programmoberflächen schnell überfordert.
- Bei Qt und natürlich vor allem bei C++, der eigentlichen Programmiersprache, bedarf es einiger Einarbeitungszeit, um mit dem System schnell und effizient umgehen zu können. Dies wird jedoch später mit einer großen Funktionsvielfalt belohnt.

[18] Qt-Toolkit der Firma Trolltech, http://www.trolltech.com

3.2. Die Programmoberfläche

An dieser Stelle führe ich kurz in die Bedienung von ChaosExplorer ein. Dies erleichtert auch das spätere Verständnis des Quellcodes.

ChaosExplorer ist als *MDI-Anwendung* (*Multiple Document Interface*) ausgelegt, d. h. es können mehrere Dokumente gleichzeitig bearbeitet werden. Sogar die Berechnung erfolgt gleichzeitig. Im Wesentlichen sind alle auch von anderen Windows-Programmen bekannten Funktionen vorhanden:

Abspeichern und Öffnen von Dokumenten: Auf Wunsch speichert ChaosExplorer Dokumente für die spätere Betrachtung oder Abänderung in Dateien mit der Endung *.che*. Lediglich die Dokument*einstellungen* werden in diesem Dateityp gespeichert, bei einem Mandelbrotdokument also z. B. Iterationstiefe, Bildausschnitt und Farbpalette. Das berechnete Bild wird nicht in die Datei geschrieben.

Ausdruck: Die berechneten Bilder können auf jedem bei Windows angemeldeten Drucker ausgegeben werden.

Exportieren der berechneten Grafik: Um die berechneten Grafiken auch in anderen Programmen nutzen zu können, ist es möglich, diese in einer Bitmap-Datei der Formate *.bmp* oder *.png* abzuspeichern.

Rückgängig und Wiederherstellen: Alle an einem Dokument vorgenommenen Änderungen können ohne Begrenzung rückgängig gemacht oder wiederhergestellt werden.

ChaosExplorer unterscheidet drei unterschiedliche Mausmodi, *Werkzeuge* genannt, die dem Benutzer des Programms die Wahl des Bildausschnittes erleichtern. Für jedes Dokument sind sie einzeln wählbar.

Bereichsauswahl: Dieses Werkzeug dient der Ausschnittsvergrößerung. Innerhalb des berechneten Bildes wird mit der Maus ein gestrichelter Rahmen aufgezogen. Dieser Rahmen entspricht dem neuen Bildausschnitt und wird gegebenenfalls bezüglich der Proportionen an die aktuelle Bildgröße angepasst.

Zentrieren: Der Bildpunkt, welcher mit diesem Werkzeug angeklickt wird, wird im neuen Bildausschnitt zentriert, d. h. er wird in die Mitte des Bildes verschoben.

Verschieben: Mit diesem Werkzeug „greift“ man den Bildausschnitt und verschiebt ihn. Die Verschiebung kann man an der Bewegung eines bereits berechneten Bildes beobachten.

Die Anwendung dieser Werkzeuge kann mit einem Rechtsklick abgebrochen werden. Sollte man also z. B. während des Aufziehens des Zoomrahmens bemerken, dass man das falsche Werkzeug anwendet, kann der Zoomvorgang durch Drücken der rechten Maustaste beendet werden.

Zusätzlich zu diesen Ausschnittswahlmöglichkeiten gibt es noch zwei Icons in der Symbolleiste: Die Funktionen „Vergrößern“ und „Verkleinern“ zoomen um den Bildmittelpunkt, verkleinern bzw. vergrößern also die Breite und Höhe des Bildausschnitts um jeweils die Hälfte.

Für jeden Dokumenttyp existiert weiterhin ein Dialog zum Einstellen der typspezifischen Parameter.

3.2.1. Mandelbrot-Einstellungen

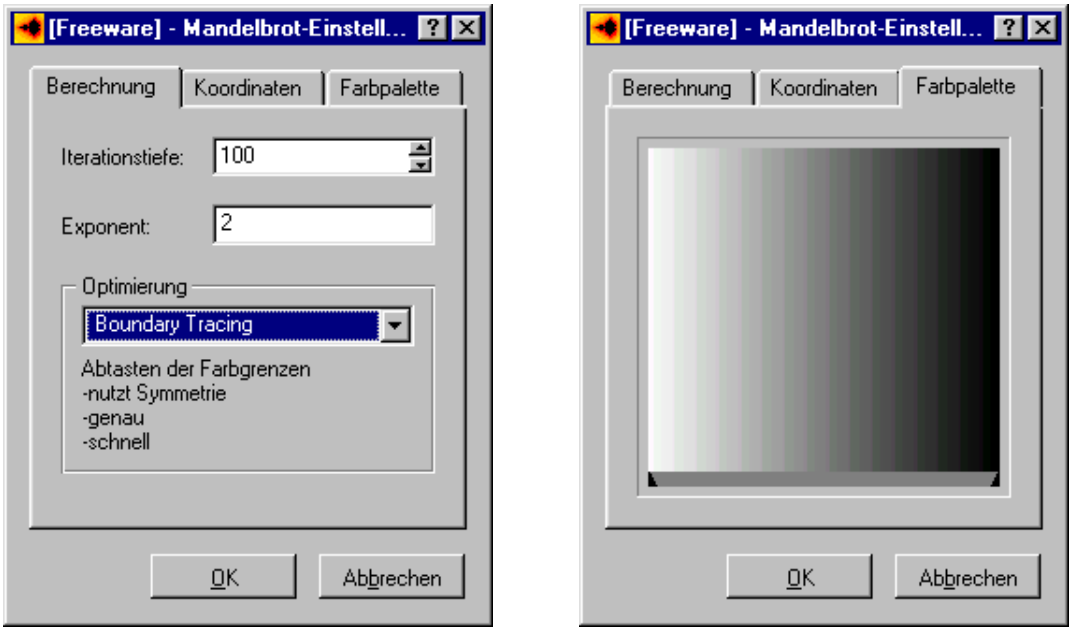


Abbildung 11: Die Mandelbrot-Einstellungen sind aufgeteilt in drei Register, hier sind „Berechnung“ und „Farbpalette“ abgebildet

Der Dialog für die Mandelbroteinstellungen (Abbildung 11) ist in drei Register eingeteilt:

Berechnung: Hier kann die Iterationstiefe sowie der Exponent für die Iterationen während der Berechnung festgelegt werden (siehe auch Abschnitt 2.1.4). Zusätzlich ist die Wahl der Berechnungsmethode möglich. Die einzelnen Einstellungen werden in Abschnitt 3.3.1.4 besprochen.

Koordinaten: In diesem Register wird der Bildausschnitt festgelegt. Angegeben werden die linke obere sowie die rechte untere Ecke in Koordinaten der Gaußschen Zahlenebene.

Farbpalette: Der in diesem Register dargestellte Farbverlauf wird für das Einfärben der Bildpunkte genutzt (s. Abschnitt 3.3.1.3). Er symbolisiert die Farbverteilung:

Links die Farbe für Punkte, welche nur eine niedrige Iterationszahl erreicht haben, rechts die Farbe für hohe Iterationszahlen, und ganz am rechten Rand die Farbe für die Mandelbrotmenge selbst (entspricht 100 % der Iterationstiefe).

3.2.2. Feigenbaum-Einstellungen

Die Einstellungen für Feigenbaumdokumente sind ähnlich wie für Mandelbrotdokumente gegliedert:

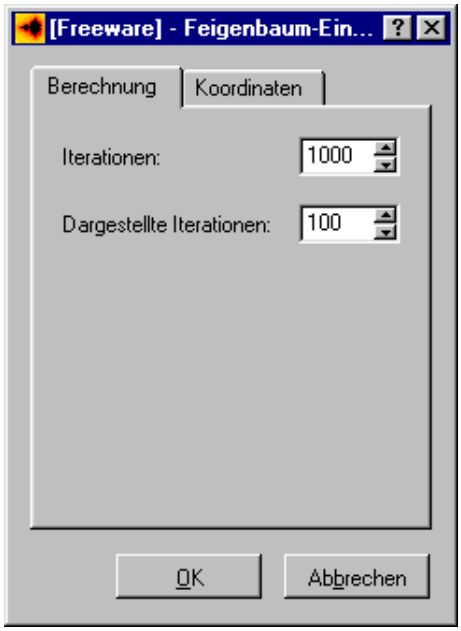


Abbildung 12: Das Register „Berechnung“ in den Feigenbaum-Einstellungen

Berechnung: In diesem Register wird, zu sehen in Abbildung 12, die Iterationstiefe sowie die Anzahl der in der Grafik dargestellten Iterationen angegeben.

Koordinaten: Dieses Register entspricht dem Register „Koordinaten“ der Mandelbroteinstellungen (Abschnitt 3.2.1).

3.3. Die Implementierung der Grafikroutinen

Der Quellcode zum Berechnen der Mandelbrotmenge und des Feigenbaumdiagramms wird im Folgenden besprochen. Um die Übersichtlichkeit zu bewahren, wird im Text lediglich mit Zeilennummern auf den Quelltext verwiesen. Er wird jedoch im Anhang A ab Seite 45 abgedruckt.

3.3.1. Mandelbrot

Im Programm ChaosExplorer benutze ich fertige Funktionen für den Umgang mit komplexen Zahlen. Diese stammen aus der C++-Standardbibliothek *STL* (*Standard Tem-*

plate Library). So ist dort der Typ der komplexen Zahl mit complex<**double**> definiert[19] (in meinem Sourcecode mit **compl** abgekürzt). Zusammen mit der Funktion zum Potenzieren (pow()) und der Betragsfunktion (abs()) wird so der Umgang mit komplexen Zahlen erheblich vereinfacht. Dadurch verursachte Geschwindigkeitseinbußen sind zwar vorhanden, aber in der Regel verhältnismäßig klein.

3.3.1.1. Die Berechnung eines einzelnen Pixels

Betrachten wir zunächst einmal die Funktion iterateComplex() (Listing 1), welche die eigentliche Iteration übernimmt.

Der Funktion wird der zu berechnende Bildpunkt als komplexe Zahl übergeben. Dieser Bildpunkt wird bei der Iteration als Konstante c verwendet (siehe 2.1.4). Die Iterationsvariable z_n, hier cpl genannt, wird wie für Mandelbrotmengen notwendig in Zeile 3 mit 0 initialisiert. Darauf folgt eine **for**-Schleife, die die Iteration maximal so oft durchführt, wie dies der Benutzer des Programms durch die Iterationstiefe festgelegt hat (s. Abschnitt 3.2.1). n dient als Laufvariable und zählt die Anzahl der bereits ausgeführten Iterationen. In Zeile 6 wird der eigentliche Iterationsschritt durchgeführt, wobei als Exponent erneut die Benutzerangabe verwendet wird. Zeile 8 berechnet das Quadrat des Betrages des Iterationsergebnisses. Auf das Wurzelziehen wird verzichtet, da dies hier für die Abbruchbedingung in Zeile 10 nicht unbedingt nötig ist und so Rechenzeit gespart wird. Andernfalls hätte auch die abs()-Funktion der STL verwendet werden können. Die Iteration wird unterbrochen, sobald der Betrag der Iterationsvariable den Wert 2 übersteigt, da dann ein Divergieren gegen ∞ feststeht (siehe Abschnitt 2.1.6). Ist diese Abbruchbedingung oder die max. Iterationstiefe erreicht, wird die **for**-Schleife verlassen und die Funktion liefert die Anzahl der erfolgten Iterationsschritte zurück (Zeile 14).

3.3.1.2. Die Entstehung eines Bildes

Um nun ein ganzes Bild entstehen zu lassen, iteriert man für jedes Pixel des Bildes die entsprechende komplexe Zahl und färbt das Pixel ein.

In der Funktion calcUnOptimized() (Listing 2) laufen zwei **for**-Schleifen in x- bzw. y-Richtung an der Bildkante entlang, so dass $(x|y)$ dem aktuellen Bildpunkt entspricht (Zeilen 10–13). Dieser wird in den Zeilen 14–16 unter Berücksichtigung des aktuellen Bildausschnittes in den entsprechenden Punkt in der Gaußschen Zahlenebene umgerechnet und an die in Abschnitt 3.3.1.1 besprochene Funktion iterateComplex() übergeben. Das Ergebnis, also die erreichte Iterationstiefe, wird in einer Matrix gespeichert. Zeile 18 veranlasst die Darstellung der neu berechneten Zeile auf dem Bildschirm.

Die Iterationsergebnisse werden für alle Pixel in einer Matrix festgehalten. Dies hat den Vorteil, dass bei einer Farbpalettenänderung durch den Benutzer das Bild nicht nochmals berechnet werden muss, sondern sofort die noch in der Matrix erhaltenen Werte genutzt werden können und nur das Bild nochmals ausgegeben werden muss (siehe auch 3.3.1.3).

[19] [2], S. 729ff.

3.3.1.3. Ausgabe des Bildes

Nach der Berechnung müssen die Matrixwerte nur noch in Farben umgerechnet werden und auf dem Bildschirm ausgegeben werden. Die Farbe der Bildpunkte richtet sich nach der erreichten Iterationszahl. Die Farbverteilung von 0 % bis 100 % der maximalen Iterationszahl bestimmt der vom Benutzer angegebene Farbverlauf (siehe Abschnitt 3.2.1).

Um später die zum Bildpunkt zugehörige Farbe einfach bestimmen zu können, berechnen wir zuvor für jede mögliche Iterationszahl die Farbe und legen sie in einem Array ab, wobei als Index die Iterationszahl verwendet wird. Dies mag zunächst ineffizient erscheinen, ist jedoch sinnvoll, wenn man bedenkt, dass ansonsten für jeden Bildpunkt die Farbe extra berechnet werden müsste statt sie einfach nur aus dem Array zu lesen. Selbst wenn 10000 Iterationen durchgeführt werden und daher 10000 Farbwerte berechnet werden müssen, ist dies noch immer schneller erledigt als für mehrere 100000 Bildpunkte einzeln.

Die **for**-Schleife in Zeile 9 des Listings 3 durchläuft mittels der Variable i alle Iterationszahlen. Zeile 11 berechnet den dazugehörigen Bruchteil zwischen 0 % und 100 % und fragt beim Objekt, welches den Farbverlauf repräsentiert, nach der zugehörigen Farbe (Zeile 12). In Zeile 13 wird diese unter dem entsprechenden Index gespeichert.

Nachdem die Farbpalette berechnet wurde, kann nun mit dem Erstellen des eigentlichen Bildes begonnen werden. Dafür überprüft die Funktion rectPainted() (Listing 4) zunächst, ob das Bitmap, in dem das Bild gespeichert wird, eine ausreichende Größe hat. Ist dies nicht der Fall, wird die Größe des Bitmaps entsprechend angepasst. Liegt das neu zu zeichnende Rechteck, welches der Funktion als Argument übergeben wird, innerhalb des Bildes (Z. 11–12), so wird es von zwei **for**-Schleifen Punkt für Punkt durchlaufen (Zeilen 15–17). Für jeden Punkt wird die erreichte Iterationstiefe aus der Matrix abgefragt (Zeile 19) und als Index für die Farbpalette verwendet (Zeile 22). Zeile 23 schließlich zeichnet dann den Punkt in der richtigen Farbe.

3.3.1.4. Geschwindigkeitsoptimierungen

Um die Geschwindigkeit der Bildberechnung zu erhöhen, werden zwei Eigenschaften der Mandelbrotmenge ausgenutzt (siehe auch Abschnitt 2.1.5):

Symmetrie: Mandelbrotmengen sind (nicht nur) bezüglich der x-Achse spiegelsymmetrisch. Im Idealfall halbiert sich hier die Rechenzeit, indem der oberhalb der x-Achse liegende Bildteil nach unten gespiegelt wird.

Zusammenhang: Mandelbrotmengen sind zusammenhängend, d. h. es gibt keine „Farbinseln“. Jeder Punkt der Menge kann von jedem anderen Punkt der Menge aus erreicht werden, ohne die Menge zu verlassen.

Im Programm ChaosExplorer habe ich insgesamt drei Methoden implementiert, mit deren Hilfe die Anzahl der zur Darstellung des Bildes berechneten Bildpunkte minimiert werden kann. Für eine kurze Vorstellung der Methoden siehe auch [5].

3.3.1.4.1. Interleave

Die unoptimierte Berechnungsmethode (siehe 3.3.1.2) geht Zeile für Zeile, Bildpunkt für Bildpunkt vor, so dass das komplette Bild entsteht.

Bei *Interleave* wird ebenfalls zeilenweise vorgegangen, jedoch mit dem Unterschied, dass innerhalb solch einer Zeile versucht wird, nicht jedes Pixel berechnen zu müssen. Statt dessen wird nur jeder fünfte Bildpunkt berechnet. Hat dieser die selbe Farbe wie der zuletzt berechnete, werden alle dazwischen liegenden Punkte in der selben Farbe eingefärbt. Andernfalls wird ein Pixel zurückgegangen und der Vergleich wiederholt.

Diese Technik ist sehr schnell, aber auch ungenau. So ist es z. B. möglich, bei einem „Arm“ der Mandelbrotmenge diesen zu überspringen und auf der „anderen Seite“ auf ein Pixel zu stoßen, welches die selbe Farbe hat wie das zuletzt berechnete. Die Folge ist ein einfarbiger Strich quer durch diesen Arm. Interleave eignet sich daher wohl nur für schnelle Berechnungen, die als Vorschau dienen sollen.

Der Interleave-Modus in ChaosExplorer nutzt die erwähnte Symmetrie der Mandelbrotmengen und spiegelt Bildbereiche an der x-Achse, wenn dies der Bildausschnitt zulässt.

Der Interleave-Algorithmus ist in der Funktion calcInterleaved () (Listing 5) definiert. Um zeilenweise vorzugehen, wird in Zeile 11 wieder eine **for**-Schleife benutzt, die für die y-Koordinate alle möglichen Werte durchläuft und somit jede Bildzeile abarbeitet. In Zeile 18 wird, da die Symmetrie der Mandelbrotmenge ausgenutzt werden soll, die Nummer der Bildzeile berechnet, welche durch Spiegelung an der x-Achse der Gaußschen Zahlenebene entsteht. Mit dieser Nummer wird nun überprüft (Zeile 20), ob die gespiegelte Bildzeile innerhalb des Bildes liegt und bereits berechnet wurde. Falls ja, wird deren Inhalt in die aktuelle Bildzeile übernommen und die Berechnung ist damit abgeschlossen. Andernfalls wird in Zeile 29 die Berechnung des ersten Bildpunkts der Zeile, dem Referenzpixel für den Farbvergleich, durchgeführt. In der darauf folgenden **while**-Schleife werden die fünf dem Referenzpixel folgenden Pixel farblich verglichen. Entspricht die Farbe des fünften Folgepixels nicht der des Referenzpixels, wird das vierte Pixel überprüft. So wird fortgefahren, bis entweder das Referenzpixel erreicht ist oder die Farben übereinstimmen. In diesem Fall (Zeile 48) werden die zwischen dem Referenzpixel und dem farblich gleichen Pixel liegenden Bildpunkte eingefärbt (Zeile 52) und das Referenzpixel wird um fünf Bildpunkte nach rechts verschoben (Zeile 56).

Ist die Bildzeile fertig berechnet, wird schließlich in Zeile 61 des Listings ihre Darstellung auf dem Bildschirm veranlasst.

3.3.1.4.2. Kästchen-Methode

Für die *Kästchen-Methode* wird der Bildausschnitt in zwei Rechtecke unterteilt. Bei jedem dieser Rechtecke werden die Randpixel berechnet und deren Farbe verglichen. Haben alle berechneten Randpunkte die selbe Farbe, wird das ganze Rechteck mit dieser Farbe aufgefüllt, ansonsten wird das Rechteck erneut geteilt und die beiden Teile werden auf die selbe Weise behandelt. Unterschreitet ein Rechteck eine gewisse Größe, wird sein Inhalt Punkt für Punkt berechnet.

Die Kästchen-Methode ist relativ genau. Es besteht jedoch die Möglichkeit, dass ein

sehr dünner Arm der Mandelbrotmenge auf Grund der begrenzten Auflösung des Bildes durch den Rand eines Rechtecks hindurchschlüpft und in dessen Innern wieder sichtbar wird. Dieser Fall kommt aber in der Praxis nur selten vor.

Im ChaosExplorer wird das Bild gleich zu Anfang geteilt, um zu verhindern, dass durch die einfarbige Umgebung der Mandelbrotmenge (außerhalb eines Kreises mit Radius 2 um den Ursprung) das komplette Bild sogleich einfarbig gefüllt wird.

Das beschriebene Verfahren arbeitet aber nur bei mittleren Iterationstiefen noch ausreichend schnell. Dies liegt daran, dass durch die häufige Nichtübereinstimmung der Farben (also der erreichten Iterationszahlen) die Rechtecke zu klein werden und der Geschwindigkeitsvorteil durch das Füllen eines ganzen Rechtecks verloren geht.

Die Berechnung wird in der Funktion calcSquares() (Listing 6) für ein Rechteck durchgeführt, welches als Argument übergeben wird. Die Funktion ruft sich selbst immer wieder rekursiv auf, falls eine Zweiteilung des Rechtecks nötig wird. Zu Beginn der Funktion wird in Zeile 4 überprüft, ob der Benutzer die Berechnung abgebrochen hat. Dies geschieht ganz am Anfang, da die Funktion wie erwähnt oft aufgerufen wird und im Falle eines Abbruchs das Rechteck einfach nicht berechnet wird. Im Anschluss werden Rechtecke, deren Kante weniger als drei Pixel lang ist, direkt, also Pixel für Pixel berechnet (vergleiche Abschnitt 3.3.1.2), da in diesem Fall die Randpixel bereits das ganze Rechteck ausmachen. Ist das Rechteck größer, werden in den Zeilen 29–59 für eine Art Schnelltest zunächst einmal die vier Ecken des Rechtecks auf Farbgleichheit überprüft. Ist diese bereits bei den Ecken nicht gegeben, wird das Rechteck sofort geteilt („**goto** divide;“). Andernfalls werden nun die Kanten untersucht, zunächst die obere sowie untere. Eine **for**-Schleife läuft ab Zeile 63 wie gewohnt an der x-Achse entlang. Innerhalb der Schleife wird sowohl die oberste als auch die unterste Pixelreihe berechnet und bezüglich ihrer Farbe verglichen. Ist die Farbgleichheit nicht erfüllt, wird das Rechteck geteilt. War dieser Test jedoch erfolgreich, folgt die Überprüfung der linken und rechten Kante auf die selbe Weise, mit dem Unterschied, dass die **for**-Schleife an der y-Achse entlang fährt (Zeile 82). Sind auch diese Kanten gleichfarbig, wird in Zeile 100 das gesamte Rechteck in genau dieser Farbe aufgefüllt. Danach wird das Rechteck in Zeile 104 noch auf dem Bildschirm dargestellt und die Funktion beendet.

Der dem Label divide (Zeile 110) folgende Code führt die eigentliche Teilung und den rekursiven Funktionsaufruf durch. Je nach Längenverhältnis wird das Rechteck horizontal oder vertikal gespalten, so dass erst möglichst spät Rechtecke mit einer kleinen Kantenlänge entstehen. Wie oben gezeigt, kann das Label mit **goto** angesprungen werden. Natürlich hätte dies auch in eine separate Funktion eingefügt werden können.

3.3.1.4.3. Boundary Tracing

Wegen des Zusammenhangs der Mandelbrotmenge gibt es innerhalb eines einfarbigen Bildbereichs keinen anderen Farbbereich. Deshalb ist es möglich, Farbgrenzen abzufahren und die so eingeschlossenen Bildpunkte alle in der selben Farbe einzufärben. Genau diese Technik liegt *Boundary Tracing* zugrunde.

Verwirklicht wird dies in ChaosExplorer durch eine trickreiche Vorgehensweise[20]: Man

[20] [4], [5]

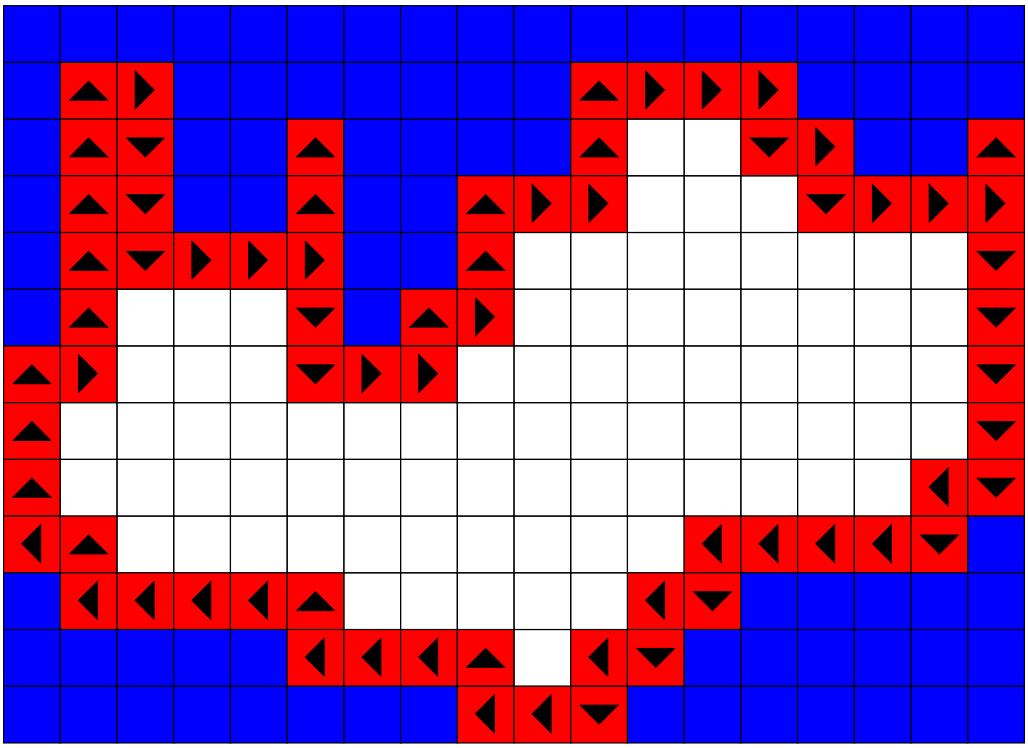


Abbildung 13: Die durch Boundary Tracing erforschte Farbgrenze

definiert vier Richtungen „hoch“, „runter“, „rechts“ und „links“ und ein Pixel als das „aktuelle Pixel“ mit einer diesem Pixel zugewiesenen Richtung. Zu Beginn wählt man nun ein Pixel im Bild als das aktuelle Pixel aus und weist die Richtung rechts zu. Nun werden die das aktuelle Pixel umgebenden Pixel berechnet und farblich mit dem aktuellen Pixel verglichen. Dazu wird als erstes der links vom aktuellen Pixel liegende Bildpunkt untersucht, wobei links bezüglich der dem aktuellen Pixel zugewiesenen Richtung zu betrachten ist. Hat also das aktuelle Pixel die Richtung rechts, ist das linke Pixel das *über* dem aktuellen Pixel liegende Pixel. Sind das linke Pixel und das aktuelle Pixel gleichfarbig, wird das linke Pixel zum aktuellen Pixel, die Blickrichtung auf links gesetzt und die Prozedur beginnt von neuem. Ist die Farbgleichheit nicht erfüllt, wird statt dem linken Pixel das Pixel geradeaus, also das in aktueller Blickrichtung liegende Pixel überprüft und wird, wenn die Farbe übereinstimmt, zum aktuellen Pixel. Kann auch hier die Farbgleichheit nicht festgestellt werden, wird das in Blickrichtung rechte Pixel untersucht und evntl. zum aktuellen Pixel ernannt, wobei auch die Blickrichtung auf rechts gesetzt wird. Sind alle drei untersuchten Pixel farblich unterschiedlich, wird das Pixel, welches entgegengesetzt der Blickrichtung liegt, das aktuelle Pixel und die Blickrichtung wird um 180 Grad gedreht. Sollte ein zu überprüfendes Pixel außerhalb des Bildes liegen, so wird angenommen, dass das Pixel eine andere Farbe als das aktuelle Pixel hat.

Die komplette Farbgrenze wurde abgefahren, wenn das aktuelle Pixel wieder dem Anfangspixel entspricht. Abbildung 13 zeigt diesen Zustand.

Grundsätzlich gibt es aber bei Boundary Tracing ein Problem: Tastet der Algorithmus

das erste Mal die Farbgrenze ab, steht noch nicht fest, bis wohin die Farbfläche reicht. D. h. beim ersten Durchlauf kann die Fläche noch nicht gefüllt werden. In ChaosExplorer wird der Algorithmus dazu ein zweites Mal durchlaufen und bei jeder Abwärtsbewegung des aktuellen Pixels die aktuelle Zeile soweit gefüllt, bis man entweder am Bildrand oder an der Farbgrenze anstößt[21]. Der zweite Lauf verbraucht wesentlich weniger Rechenzeit, da ja bereits alle notwendigen Pixel berechnet und in der Matrix (siehe Abschnitt 3.3.1.2) abgespeichert wurden.

Bezüglich der Genauigkeit gibt es anzumerken, dass es bei Boundary Tracing genauso wie bei der Kästchen-Methode möglich ist, dass ein sehr dünner, die Auflösung unterschreitender Arm der Mandelbrotmenge die abgefahrene Farbgrenze durchbricht. Dies kommt jedoch, wie bereits gesagt, nicht oft vor.

Boundary Tracing ist eine der schnellsten und zugleich genauesten Methoden, Mandelbrotfraktale zu berechnen. Zur nochmaligen Beschleunigung nutzt das in ChaosExplorer implementierte Boundary Tracing wie der Interleave-Modus die Symmetrie der Mandelbrotmengen.

Auch in der Funktion calcBoundaryTraced() wird zeilenweise vorgegangen, deshalb wird in Zeile 13 wieder eine **for**-Schleife benutzt. Analog zur Vorgehensweise in der Funktion calcInterleaved () werden schon berechnete Bildzeilen gespiegelt, um die Symmetrie auszunutzen (Zeilen 16–25). Ist diese Beschleunigungsmöglichkeit nicht nutzbar, wird nun mit dem Boundary Tracing begonnen. Hierfür muss ein Anfangspunkt gewählt werden, von welchem aus dann die Grenzverfolgung durchgeführt wird. Da jedoch nicht bekannt ist, welche Punkte verwendet werden müssen, um alle Farbflächen im Bild zu erreichen, muss ein Boundary Tracing für jeden Bildpunkt durchgeführt werden, der noch nicht berechnet wurde. Deshalb die zweite **for**-Schleife in Zeile 28, welche die x-Achse entlang läuft. In dieser Schleife werden zunächst wie oben beschrieben die vier Richtungen definiert und die aktuelle Richtung auf rechts festgelegt (Zeilen 39–44). Der Anfangspunkt, welcher durch die Laufvariablen x und y der **for**-Schleifen festgelegt ist, wird zum aktuellen Pixel (curr_x und curr_y ab Zeile 46). Weiter wird in Zeile 49 für diesen ersten Durchlauf das Füllen der Farbfläche ausgeschaltet, da die Grenzen der Fläche noch nicht bekannt sind. In der folgenden **while**-Schleife wird zunächst das aktuelle Pixel durch den **switch**-Block einen Schritt weiter in die gespeicherte Richtung bewegt. Das Pixel an der neuen Position wird nun überprüft. Verläuft die x-Achse durch die momentan behandelte Farbfläche (Zeile 73), wird diese „abgeschnitten“, d. h. sie wird nicht über die x-Achse hinaus verfolgt. Das hat den Sinn, zusätzlich Zeit zu sparen, da der „abgeschnittene“ Flächenteil später durch die Spiegelung an der x-Achse ohnehin eingefärbt wird. Der nächste Test in Zeile 80 überprüft, ob die neue Position außerhalb des Bildes liegt. Ist dies zutreffend, wird die Richtung um 90 Grad nach rechts gedreht und durch das Schlüsselwort **continue** die **while**-Schleife neu gestartet, so dass das aktuelle Pixel in die neue Richtung bewegt wird (siehe oben). In Zeile 88 wird überprüft, ob dies bereits der zweite Boundary Tracing-Durchlauf ist, ob also die aktuelle Zeile bis zur Farbgrenze oder bis zum Bildrand gefüllt werden muss (fill == **true**, siehe auch Zeile 49), und erledigt dies. Da das neue aktuelle Pixel noch gar nicht berechnet wurde, wird dies nun

[21] [4]

ab Zeile 102 erledigt. Der Farbvergleich findet direkt im Anschluss statt (Zeile 110). Bei gleicher Farbe wird nach links, sonst nach rechts gedreht. Zuletzt wird nun noch in Zeile 115 überprüft, ob das aktuelle Pixel wieder dem Anfangspixel entspricht. Ist dies der Fall und wurde die fertig abgefahrene Farbfläche noch nicht gefüllt, wird dies veranlasst, indem fill auf **true** gesetzt wird und die Richtung auf den Ausgangswert rechts. Das Boundary Tracing beginnt dadurch von neuem und die nun bekannte Fläche wird gefüllt (siehe oben). Ist das Füllen erledigt, wird das Boundary Tracing beendet (**break** verlässt die **while**-Schleife).

3.3.2. Feigenbaum

Um das Feigenbaumdiagramm auf dem Bildschirm entstehen zu lassen, ist weit weniger Rechenaufwand nötig als bei der Mandelbrotmenge. Die Berechnung konzentriert sich im Wesentlichen in der Funktion run() (siehe Listing 8).

Nachdem das Bitmap, in welches gezeichnet werden soll, an die vom Benutzer eingestellte Größe angepasst worden ist und initialisiert wurde (Zeilen 7–9), startet ähnlich wie bei den Berechnungen der Mandelbrotmenge eine **for**-Schleife (Zeile 11). Diese durchläuft jedoch nicht von oben nach unten die Werte der y-Achse, sondern von links nach rechts die der x-Achse. Da auf der x-Achse die verschiedenen μ aufgetragen sind (siehe Abschnitt 2.2.2), wird die x-Koordinate in den Zeilen 14–16 in das zugehörige μ umgerechnet (Ergebnis in der Variable u). Um die nun folgende Iteration ausführen zu können, wird noch die Variable result, welche m_n darstellt, eingeführt und mit dem vom Benutzer vorgegebenen m_0 initialisiert (Zeile 17). Die daraufhin folgende **for**-Schleife führt nun in Zeile 21 die Wachstumsberechnung nach Abschnitt 2.2.1 aus und überprüft per **if**-Entscheidung in Zeile 22, ob schon genügend Iterationen ausgeführt wurden, um das Iterationsergebnis in das Bitmap einzuzeichnen. Ist dies der Fall, wird das erzielte Iterationsergebnis in die y-Koordinate innerhalb des Bitmaps umgerechnet (Zeilen 26–29) und schließlich als Punkt eingezeichnet (Zeile 31). Am Ende der Berechnungen wird das resultierende Bitmap (bzw. die gerade berechnete Bildspalte) in Zeile 35 auf dem Bildschirm angezeigt.

4. Experimente mit ChaosExplorer

Nach der Entwicklung von ChaosExplorer ist es nun besonders einfach, einige Experimente mit Mandelbrotfraktalen und Feigenbaumdiagrammen durchzuführen. Im Folgenden werden wir besonders den Einfluss der Berechnungsparameter auf die Erscheinung der berechneten Bilder untersuchen und auf einige besondere Ausschnitte des Mandelbrotfraktals eingehen.

4.1. Mandelbrot

4.1.1. Variation der Berechnungsparameter

Die Mandelbrotmenge wird durch mehrere Parameter, welche in die Berechnung einfließen, beeinflusst. Dies ist neben der Iterationstiefe, die die Berechnungsgenauigkeit angibt, vor allem auch der Exponent innerhalb der Iterationsformel. Die Farbpalette wirkt hier nur indirekt, ändert sie doch lediglich die Farbverteilung, also die Darstellung der Mandelbrotmenge.

Betrachten wir zunächst die Änderungen, die sich durch Variationen der Iterationstiefe ergeben. Aus Abschnitt 2.1.6 ist bekannt, dass die Iterationstiefe die Genauigkeit der Berechnung festlegt. Dies wird auch bei Betrachten von Abbildung 14 deutlich. Bei 10 Iterationen ist die Approximation der Mandelbrotmenge noch sehr mangelhaft, Details sind kaum zu erkennen. Wesentlich deutlicher ist die Darstellung schon bei 25 Iterationen. Steigert man dann eine bereits relativ hohe Iterationstiefe nochmals deutlich, ist am Fraktal kaum ein Unterschied mehr zu erkennen. In diesem Fall hat sich die berechnete Mandelbrotmenge so weit an die Definition angenähert, dass eventuell vorhandene Unterschiede mit dem Auge nicht mehr erkennbar sind.

Interessante Formänderungen ergeben sich durch Ändern des Grades der Iterationsformel aus Abschnitt 2.1.1. An Hand von Abbildung 15 sieht man, dass je höher der Grad wird, desto mehr Symmetrieachsen bekommt das Fraktal. Der Hauptkörper wird dominierender.

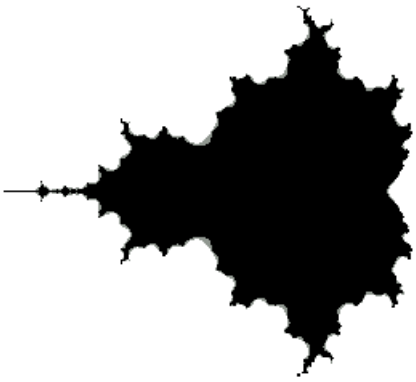
4.1.2. Typische Muster

Zoomt man mit ChaosExplorer an den Rändern der Mandelbrotmenge in das Bild hinein, gibt es einige besondere Formen zu entdecken.[22] Am Knospenansatz entwickeln sich Spiralen, wie sie in Abbildung 16 zu sehen sind. Die Anzahl der Arme nimmt zur Symmetrieachse hin zu.

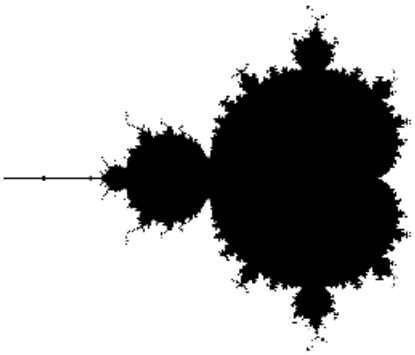
Auch Sterne, wie z. B. in Abbildung 17, beobachtet man oft. Interessanterweise liegen sich Sterne und Spiralen häufig in trichterförmigen Einkerbungen gegenüber.

Aus diesen Grundelementen sind etwas kompliziertere Gebilde aufgebaut, die oft an Tiere erinnern. Beispiele hierfür sind Elefanten oder auch Seepferdchen in Abbildung 18.

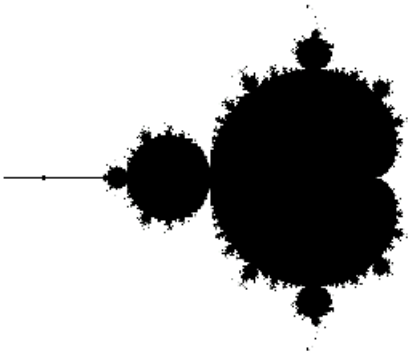
[22] [3]



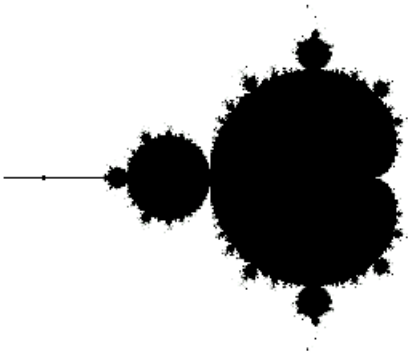
(a) 10 Iterationen



(b) 25 Iterationen

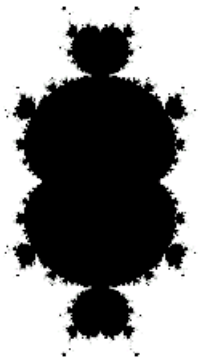


(c) 50 Iterationen

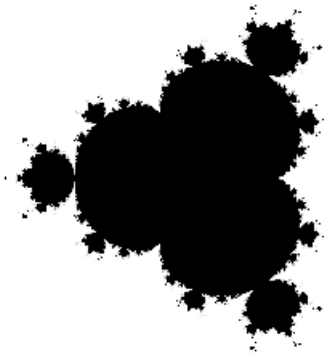


(d) 100 Iterationen

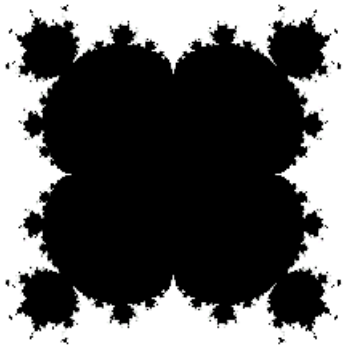
Abbildung 14: Apfelmännchen, jeweils mit einer anderen Genauigkeit berechnet



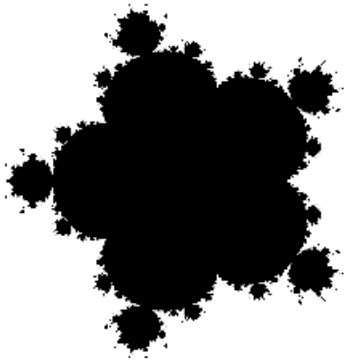
(a) $p = 3$



(b) $p = 4$

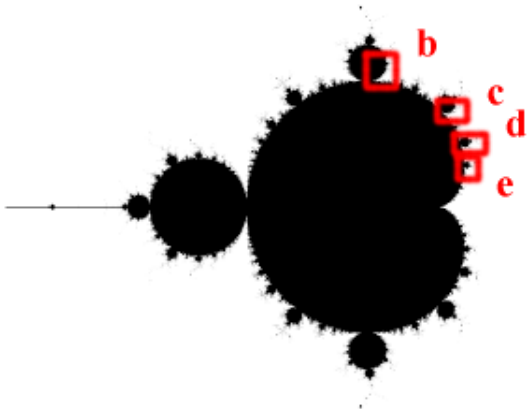


(c) $p = 5$

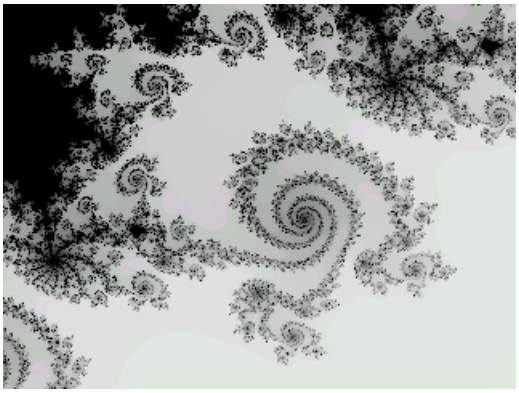


(d) $p = 6$

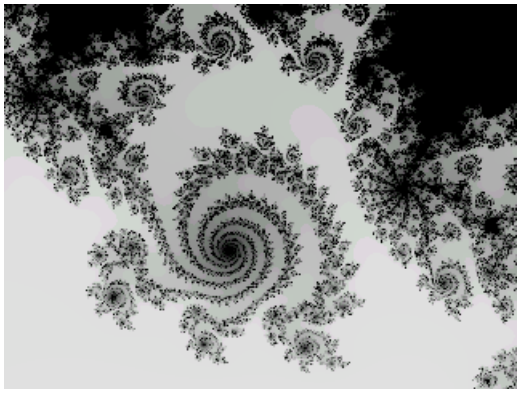
Abbildung 15: Mandelbrotfraktale mit unterschiedlichem Grad der Iterationsformel



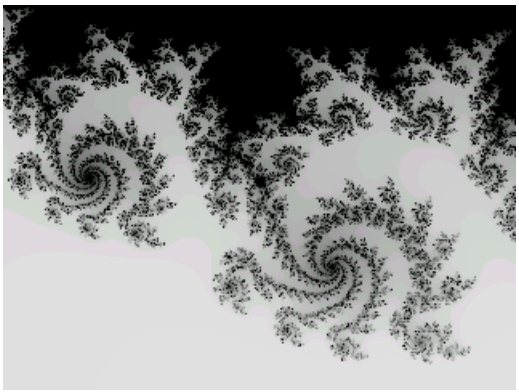
(a) Übersicht



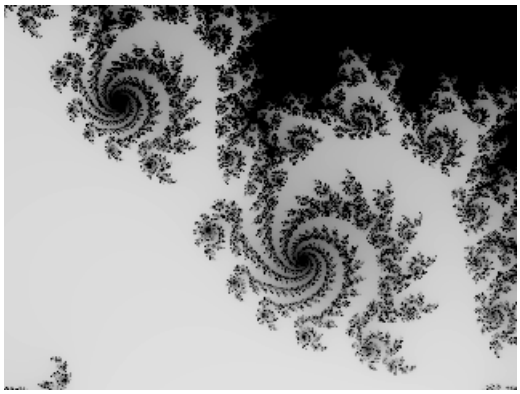
(b) dreiarmig



(c) vierarmig



(d) fünfarmig



(e) sechsarmig

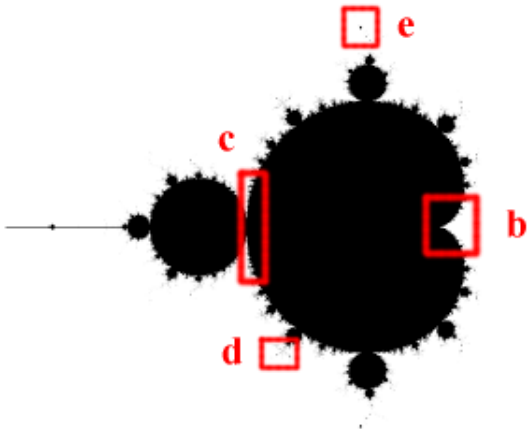
Abbildung 16: Spiralen mit unterschiedlicher Armanzahl

(a) Sterne findet man häufig...

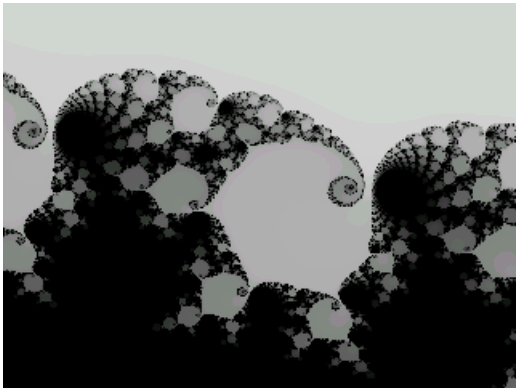
(b) ...in der Nähe von Spiralen.

Abbildung 17: Spiralen im Mandelbrotfraktal

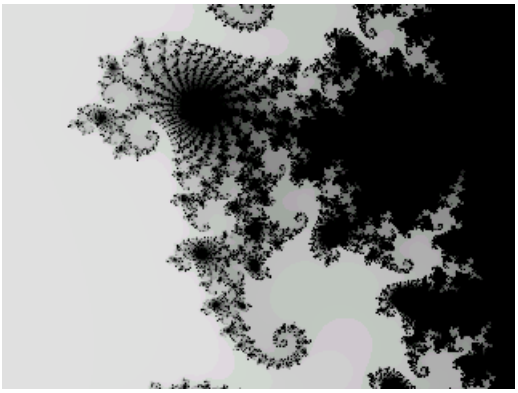
Oft sieht man auch eine Figur, die dem des ganzen Mandelbrotfraktals entspricht, also ein „Mini-Mandelbrotfraktal“. Man entdeckt sie an vielen Orten im Apfelmännchen, z. B. im Außenbereich an den Knospen des Hauptkörpers. Diese Erscheinungen (Abbildung 18) sind ein Indiz für die Selbstähnlichkeit des Mandelbrotfraktals, siehe hierzu auch Abschnitt 2.1.5.1.



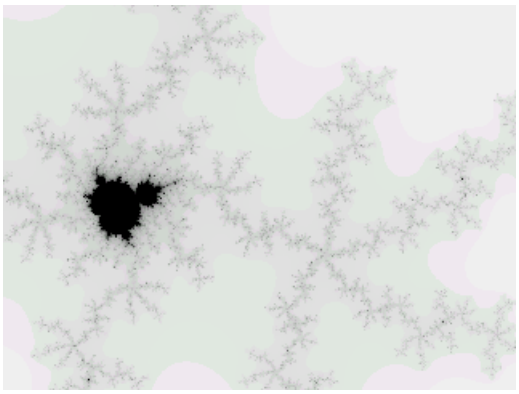
(a) Übersicht



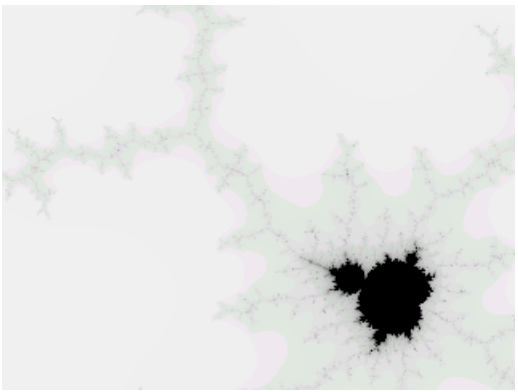
(b) Elefant



(c) Seepferdchen



(d)



(e)

Abbildung 18: Tiere und Mini-Apfelmännchen

4.2. Feigenbaum

4.2.1. Unveränderlichkeit des Feigenbaumdiagramms

Zunächst vermutet man, dass das Aussehen des Feigenbaumdiagramms vom Startwert der Iteration abhängt. Betrachtet man jedoch z. B. Abbildung 19, so stellt man das Gegenteil fest. Das Feigenbaumdiagramm ist vom Startwert unabhängig.

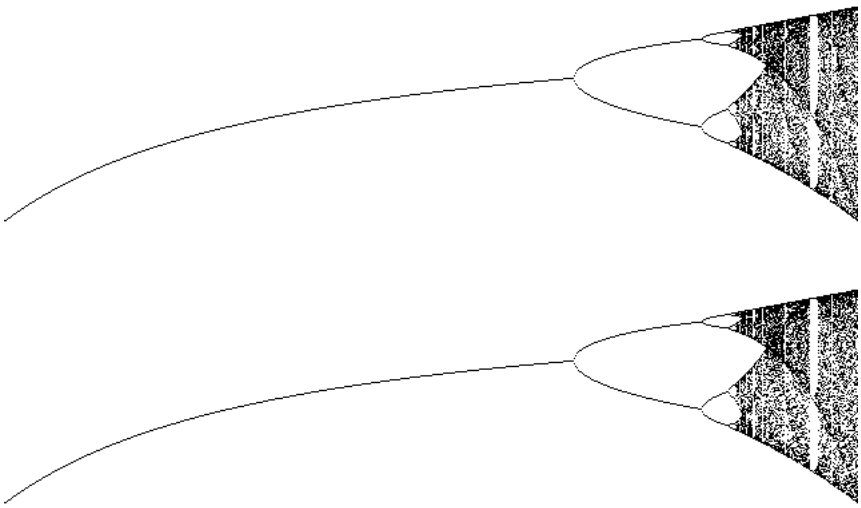


Abbildung 19: Die Unveränderlichkeit des Feigenbaumdiagramms: Oben $m_0 = 0.1$, unten $m_0 = 0.9$

4.2.2. Verschiedene Wachstumsvorgänge

Wir wollen nun abschließend den rechten Rand des Feigenbaumdiagramms untersuchen und herausfinden, warum sich dort der Graf „auffächert“. Dazu stellen wir das Wachstum für einige ausgewählte μ in einem n-m_n-Diagramm (siehe Abschnitt 2.2) dar.

Beispiel $\mu = 0.3$
In Abbildung 20 ist die Iteration für $m_0 = 0.1$ und $\mu = 0.3$ dargestellt. Klar zu erkennen ist die Schwächung des Wachstums, welches schließlich 0 erreicht.

Beispiel $\mu = 2.4$
Genau anders herum verhält es sich mit dem Wachstum bei $\mu = 2.4$ (Abbildung 21). Denn hier *steigt* das Wachstum bis zu einem bestimmten Wert an, um schließlich bei diesem zu verbleiben.

Beispiele $\mu = 3.2$**,** $\mu = 3.5$ **und** $\mu = 3.56$
Mit $\mu = 3.2$ verhält sich m_n periodisch (dargestellt in Abbildung 22). Extremer wird dies noch bei erneuter Steigerung von μ. Beispielhaft dargestellt in den Abbildungen 23 und 24 sind periodische Zyklen mit 4 und 8 Grenzwerten. Diese unterschiedliche Grenzwertanzahl äußert sich im Feigenbaumdiagramm durch die Verzweigungen des Graphen.

Beispiel $\mu = 3.9$
Unübersichtlich und *chaotisch* wird das Verhalten von m_n für $\mu = 3.9$ in Abbildung 25. Hier ist es nicht mehr möglich, ein Muster zu erkennen, um die nächsten Iterationsschritte vorauszusagen. Dies ist der Grund für das Chaos im rechten Bereich des Feigenbaumdiagramms. Da das Wachstum nicht mehr gegen einen bestimmten Wert tendiert oder eine Periodizität zeigt, kann im Diagramm keine zusammenhängende Linie mehr entstehen. Zufällig verstreute Punkte sind die Folge.

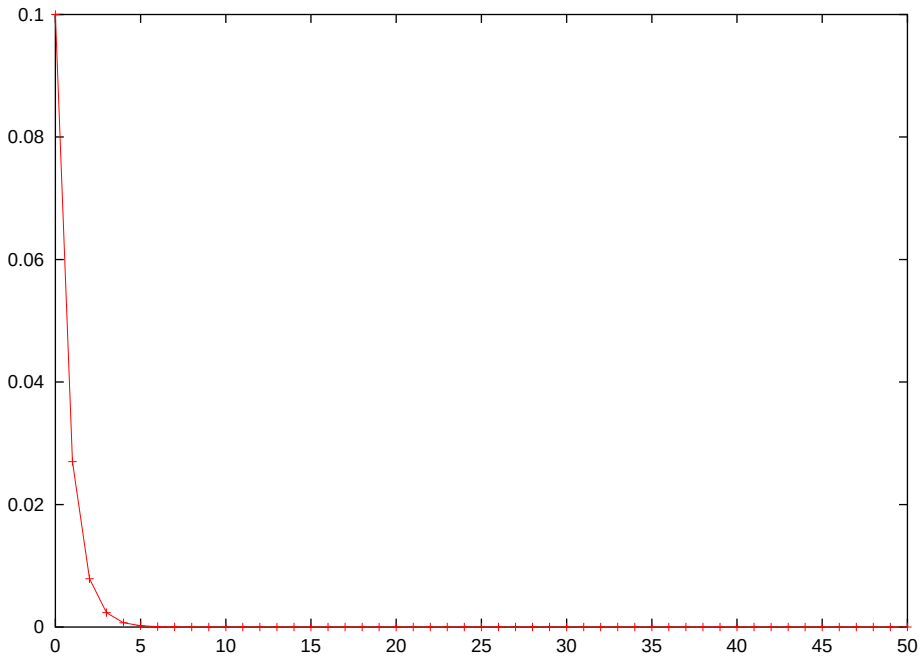


Abbildung 20: Wachstum mit $\mu = 0.3$

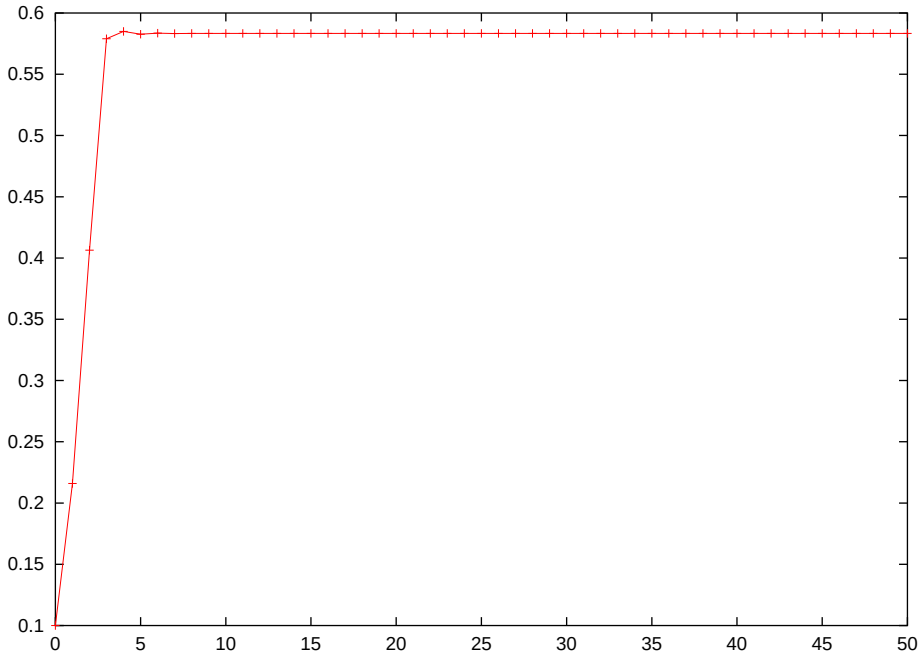


Abbildung 21: $\mu = 2.4$

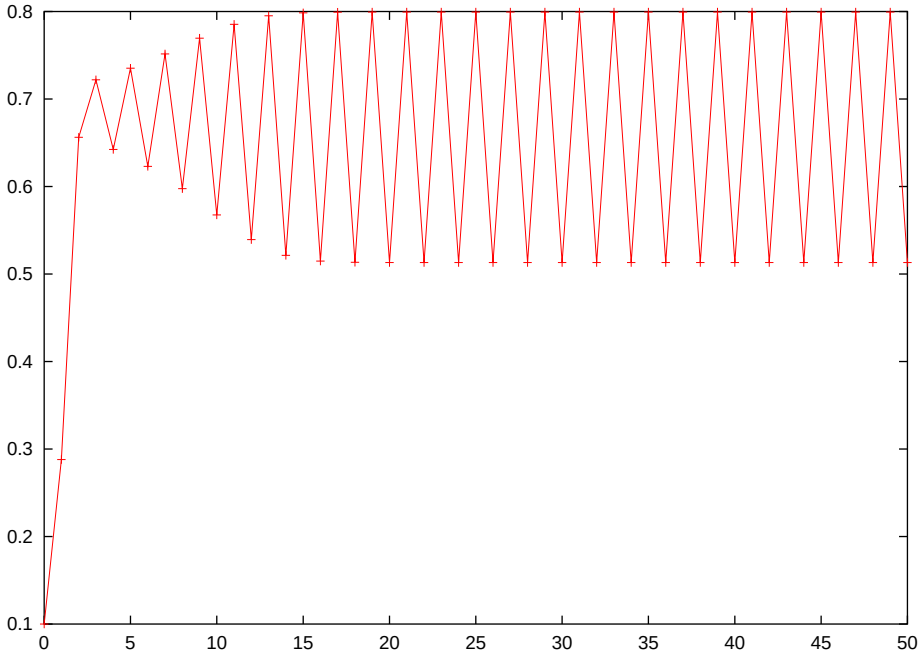


Abbildung 22: $\mu = 3.2$

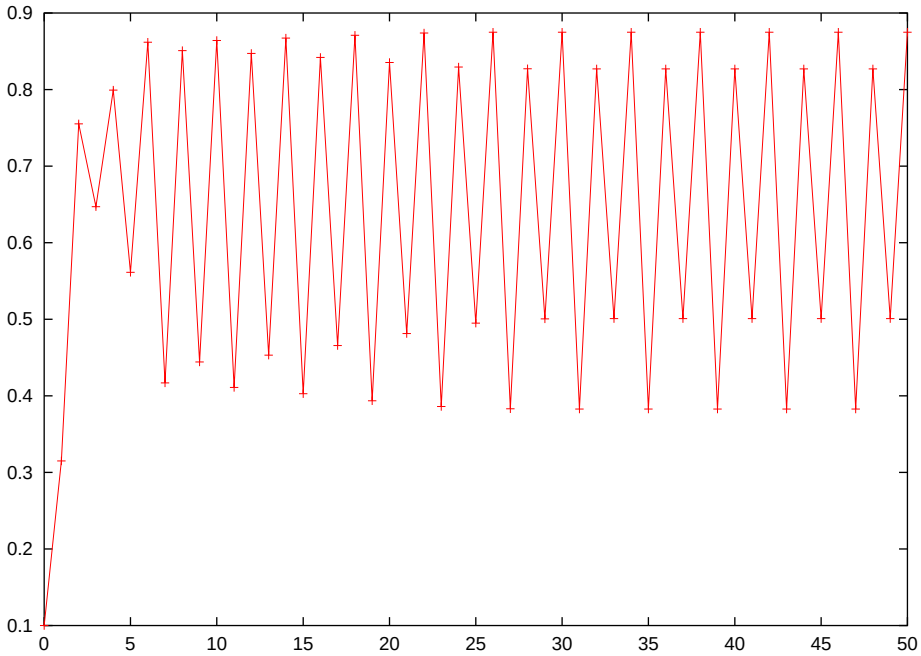


Abbildung 23: $\mu = 3.5$

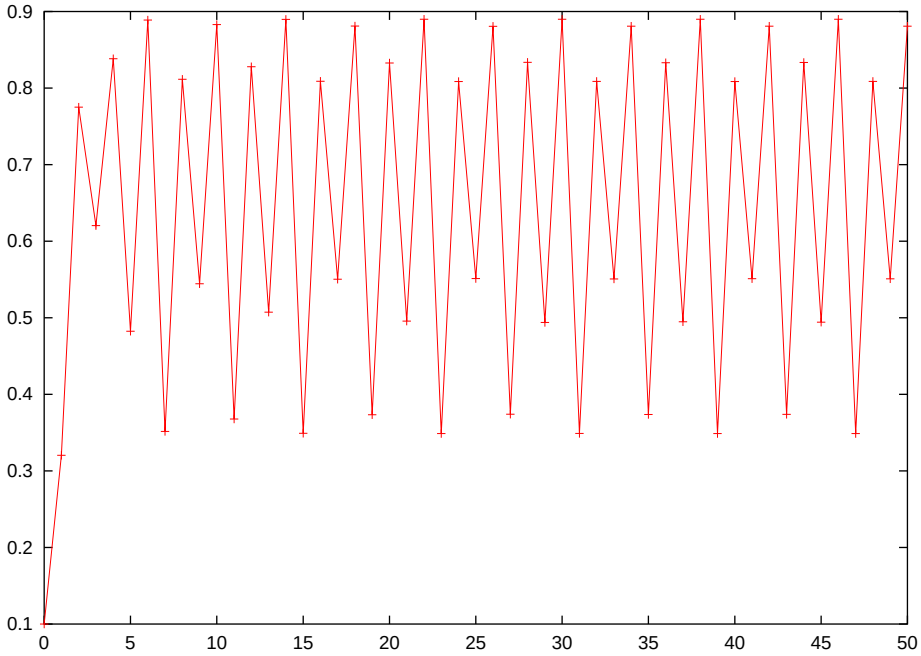


Abbildung 24: $\mu = 3.56$

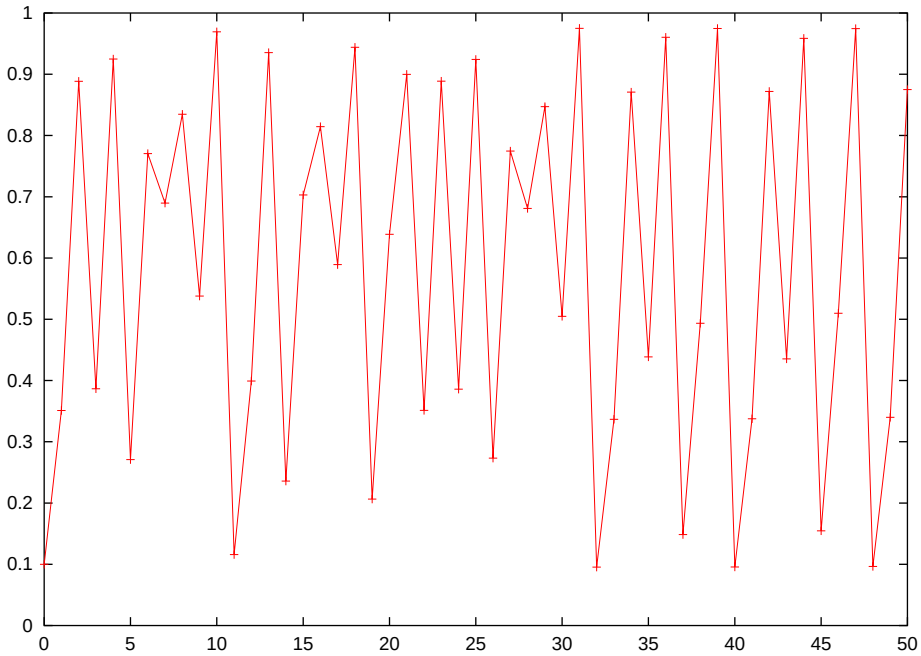


Abbildung 25: $\mu = 3.9$

A. Code-Listings

Die besprochenen Berechnungsfunktionen des Programms ChaosExplorer werden im Folgenden abgedruckt. Zum leichten Auffinden einzelner Quelltextstellen ist der Programmcode mit Zeilennummern versehen.

A.1. Berechnung des Mandelbrotfraktals

Listing 1: Iteration eines Bildpunktes

```
1  int Mandelbrot::Calc::iterateComplex( const compl& pos )
2  {
3    compl cpl( 0, 0 );
4    for( int n = 0; n < m_input->maxIterations(); n++ )
5    {
6      cpl = pow( cpl, m_input->exponent() ) + pos;
7
8      double mag = pow( cpl.real(), 2 ) + pow( cpl.imag(), 2 );
9
10     if( mag >= 4 ) // sqrt( mag ) >= 2
11       break;
12   }
13
14   return n;
15 }
```

Listing 2: Unoptimierte Berechnung des Bildes

```
1  void Mandelbrot::Calc::calcUnOptimized()
2  {
3    IntMatrix& matrix = m_output->resultMatrix();
4
5    const double step_x = m_input->coords().width()
6                          / m_input->picSize().width;
7    const double step_y = m_input->coords().height()
8                          / m_input->picSize().height;
9
10   for( unsigned int y = 0;
11        y < matrix.height() && m_calcing; y++ )
12   {
13     for( unsigned int x = 0; x < matrix.width(); x++ )
14       matrix( x, y ) = iterateComplex(
15                   compl( m_input->coords().tl.x + x * step_x,
16                          m_input->coords().tl.y + y * step_y ) );
17
18     emit rectPainted( UIntRect( 0, y, matrix.width() - 1, y ) );
19   }
```

```
20 }
```

Listing 3: Berechnung der Farbpalette

```
1 void Mandelbrot::Calc::calcPalette()
2 {
3   vector<QColor>& palette = m_output->palette();
4
5   palette.resize( m_input->maxIterations() + 1 );
6
7   int red, green, blue;
8   double pos;
9   for( unsigned int i = 0; i < palette.size(); i++ )
10  {
11    pos = (double) i / ( palette.size() - 1 );
12    m_input->colorFade().colorbypos( pos, red, green, blue );
13    palette[i].setRgb( red, green, blue );
14  }
15 }
```

Listing 4: Ausgabe des Bildes auf dem Bildschirm

```
1 void Mandelbrot::CalcResult::rectPainted( const UIntRect& rect )
2 {
3   if( m_resultMatrix.width() != m_pixmap.width()
4       || m_resultMatrix.height() != m_pixmap.height() )
5   {
6     m_pixmap.resize( m_resultMatrix.width(),
7                      m_resultMatrix.height() );
8     m_pixmap.fill( Qt::black );
9   }
10
11  if( rect.br.x < m_resultMatrix.width()
12      && rect.br.y < m_resultMatrix.height() )
13  {
14    QPainter painter( &m_pixmap );
15    for( int y = rect.tl.y; y <= rect.br.y; y++ )
16    {
17      for( int x = rect.tl.x; x <= rect.br.x; x++ )
18      {
19        int& n = m_resultMatrix( x, y );
20        if( n != -1 )
21        {
22          painter.setPen( m_palette[ n % m_palette.size() ] );
23          painter.drawPoint( x, y );
24        }
25      }
26    }
```

```
27   }
28
29   emit pixmapChanged( QRect( rect.tl.x, rect.tl.y,
30                       rect.width() + 1, rect.height() + 1 ) );
31 }
```

Listing 5: Die Implementierung des Interleave-Berechnungsmodus

```
1  void Mandelbrot::Calc::calcInterleaved()
2  {
3    IntMatrix& matrix = m_output->resultMatrix();
4
5    const double step_x = m_input->coords().width()
6                          / m_input->picSize().width;
7    const double step_y = m_input->coords().height()
8                          / m_input->picSize().height;
9    const unsigned int interleave_step = 5;
10
11   for( unsigned int y = 0;
12        y < matrix.height() && m_calcing; y++ )
13   {
14     const double coord_y = m_input->coords().tl.y
15                            + y * step_y;
16
17     int mirror_y = ( -coord_y - m_input->coords().tl.y )
18                    / step_y;
19
20     if( m_input->coords().tl.y > 0
21         && coord_y < 0 && mirror_y >= 0 )
22     {
23       for( unsigned int x = 0; x < matrix.width(); x++ )
24         matrix( x, y ) = matrix( x, mirror_y );
25     }
26     else
27     {
28       unsigned int saved_x = 0;
29       matrix( saved_x, y ) = iterateComplex(
30                                compl( m_input->coords().tl.x,
31                                       coord_y ) );
32
33       while( saved_x < matrix.width() - 1 )
34       {
35         int& iter_saved_x = matrix( saved_x, y );
36
37         unsigned int x = saved_x + interleave_step;
38         if( x > matrix.width() - 1 )
39           x = matrix.width() - 1;
```

```
40
41          for( ; x > saved_x; x-- )
42          {
43            int& iter_x = matrix( x, y );
44            iter_x = iterateComplex(
45                       compl( m_input->coords().tl.x + x * step_x,
46                              coord_y ) );
47
48            if( iter_x == iter_saved_x )
49              break;
50          }
51
52          for( unsigned int curr_x = saved_x + 1;
53               curr_x < x; curr_x++ )
54            matrix( curr_x, y ) = iter_saved_x;
55
56          saved_x += interleave_step;
57
58        }
59      }
60
61      emit rectPainted( UIntRect( 0, y, matrix.width() - 1, y ) );
62    }
63 }
```

Listing 6: Die Bildberechnung nach der Kästchenmethode

```
1 void Mandelbrot::Calc::calcSquares( const UIntRect& rect )
2 {
3
4   if( ! m_calcing )
5     return;
6
7   IntMatrix& matrix = m_output->resultMatrix();
8
9   const double step_x = m_input->coords().width()
10                        / m_input->picSize().width;
11  const double step_y = m_input->coords().height()
12                        / m_input->picSize().height;
13
14  if( rect.width() <= 2 || rect.height() <= 2 )
15  {
16    // rect is quite small, simply calc all pixels
17    for( int y = rect.tl.y; y <= rect.br.y; y++ )
18      for( int x = rect.tl.x; x <= rect.br.x; x++ )
19        matrix( x, y ) = iterateComplex(
20                           compl( m_input->coords().tl.x +
```

```
21                                      x * step_x,
22                                      m_input->coords().tl.y +
23                                      y * step_y ) );
24
25      emit rectPainted( rect );
26    }
27    else
28    {
29      const double coord_tl_x = m_input->coords().tl.x
30                                + rect.tl.x * step_x;
31      const double coord_tl_y = m_input->coords().tl.y
32                                + rect.tl.y * step_y;
33      const double coord_br_x = m_input->coords().tl.x
34                                + rect.br.x * step_x;
35      const double coord_br_y = m_input->coords().tl.y
36                                + rect.br.y * step_y;
37
38      int& tl = matrix( rect.tl.x, rect.tl.y );
39      if( tl == -1 )
40        tl = iterateComplex( compl( coord_tl_x, coord_tl_y ) );
41
42      int& tr = matrix( rect.br.x, rect.tl.y );
43      if( tr == -1 )
44        tr = iterateComplex( compl( coord_br_x, coord_tl_y ) );
45
46      int& bl = matrix( rect.tl.x, rect.br.y );
47      if( bl == -1 )
48        bl = iterateComplex( compl( coord_tl_x, coord_br_y ) );
49
50      int& br = matrix( rect.br.x, rect.br.y );
51      if( br == -1 )
52        br = iterateComplex( compl( coord_br_x, coord_br_y ) );
53
54      if( tl != tr || tl != bl || tl != br ) // quick check
55                                             // with the
56                                             // four corners
57      {
58        goto divide;
59      }
60      else
61      {
62        // check top and bottom corner
63        for( int x = rect.tl.x + 1; x < rect.br.x; x++ )
64        {
65          int& t = matrix( x, rect.tl.y );
66          int& b = matrix( x, rect.br.y );
```

```
67
68          const double coord_x = m_input->coords().tl.x
69                                 + x * step_x;
70
71          if( t == -1 )
72            t = iterateComplex( compl( coord_x, coord_tl_y ) );
73          if( b == -1 )
74            b = iterateComplex( compl( coord_x, coord_br_y ) );
75
76          if( tl != t || tl != b )
77            goto divide;
78
79        }
80
81        // check left and right corner
82        for( int y = rect.tl.y + 1; y < rect.br.y; y++ )
83        {
84          int& l = matrix( rect.tl.x, y );
85          int& r = matrix( rect.br.x, y );
86
87          const double coord_y = m_input->coords().tl.y
88                                 + y * step_y;
89
90          if( l == -1 )
91            l = iterateComplex( compl( coord_tl_x, coord_y ) );
92          if( r == -1 )
93            r = iterateComplex( compl( coord_br_x, coord_y ) );
94
95          if( tl != l || tl != r )
96            goto divide;
97        }
98
99        // all corner pixels have the same value => fill the rect
100       for( y = rect.tl.y + 1; y < rect.br.y; y++ )
101         for( x = rect.tl.x + 1; x < rect.br.x; x++ )
102           matrix( x, y ) = tl;
103
104       emit rectPainted( rect );
105     }
106   }
107
108 return;
109
110 divide:
111   //divide rect
112   if( rect.width() > rect.height() )
```

```
113   {
114     calcSquares( IntRect( rect.tl.x, rect.tl.y,
115                           rect.tl.x + rect.width() / 2,
116                           rect.br.y ) );
117     calcSquares( IntRect( rect.tl.x + rect.width() / 2,
118                           rect.tl.y, rect.br.x, rect.br.y ) );
119   }
120   else
121   {
122     calcSquares( IntRect( rect.tl.x, rect.tl.y, rect.br.x,
123                            rect.tl.y + rect.height() / 2 ) );
124     calcSquares( IntRect( rect.tl.x,
125                           rect.tl.y + rect.height() / 2,
126                           rect.br.x, rect.br.y ) );
127   }
128
129 }
```

Listing 7: Der Code für das Boundary Tracing

```
1  void Mandelbrot::Calc::calcBoundaryTraced()
2  {
3    IntMatrix& matrix = m_output->resultMatrix();
4    const DoubleRect& coords = m_input->coords();
5
6    const double step_x = m_input->coords().width()
7                          / m_input->picSize().width;
8    const double step_y = m_input->coords().height()
9                          / m_input->picSize().height;
10
11   const int y_xaxis = m_input->coords().tl.y / -step_y;
12
13   for( unsigned int y = 0;
14        y < matrix.height() && m_calcing; y++ )
15   {
16     const double coord_y = m_input->coords().tl.y + y * step_y;
17     const int mirror_y = ( -coord_y - m_input->coords().tl.y )
18                          / step_y;
19
20     if( m_input->coords().tl.y > 0
21         && coord_y < 0 && mirror_y >= 0 )
22     {
23       for( unsigned int x = 0; x < matrix.width(); x++ )
24         matrix( x, y ) = matrix( x, mirror_y );
25     }
26     else
27     {
```

```
28        for( unsigned int x = 0; x < matrix.width(); x++ )
29        {
30          int& n = matrix( x, y );
31
32          if( n == -1 )
33            n = iterateComplex(
34                  compl( m_input->coords().tl.x + x * step_x,
35                         m_input->coords().tl.y + y * step_y ) );
36          else
37            continue;
38
39          const unsigned int right = 0;
40          const unsigned int down = 1;
41          const unsigned int left = 2;
42          const unsigned int up = 3;
43
44          unsigned int dir = right;
45
46          int curr_x = x;
47          int curr_y = y;
48
49          bool fill = false;
50
51          while(1)
52          {
53            // step ahead
54            switch( dir )
55            {
56              case right:
57                curr_x++;
58                break;
59
60              case down:
61                curr_y++;
62                break;
63
64              case left:
65                curr_x--;
66                break;
67
68              case up:
69                curr_y--;
70                break;
71            }
72
73            if( (int) y < y_xaxis && y_xaxis < curr_y )
```

```
74            {
75              // other side of the x-axis, turn right
76              dir = ( dir + 1 ) % 4;
77              continue;
78            }
79
80            if( curr_x < 0 || curr_x > matrix.width() - 1
81                || curr_y < 0 || curr_y > matrix.height() - 1 )
82            {
83              // offscreen, turn right
84              dir = ( dir + 1 ) % 4;
85              continue;
86            }
87
88            if( fill && dir == down )
89            {
90              int paint_x = curr_x;
91              while( paint_x > 0 )
92              {
93                paint_x--;
94                int& paint_n = matrix( paint_x, curr_y );
95                if( paint_n == -1 )
96                  paint_n = n;
97                else if( paint_n != n )
98                  paint_x = 0;
99              }
100           }
101
102           int& n_check = matrix( curr_x, curr_y );
103           if( n_check == -1 )
104             n_check = iterateComplex(
105                         compl( m_input->coords().tl.x +
106                                curr_x * step_x,
107                                m_input->coords().tl.y +
108                                curr_y * step_y ) );
109
110           if( n == n_check )
111             dir = ( dir + 3 ) % 4; // turn left
112           else
113             dir = ( dir + 1 ) % 4; // turn right
114
115           if( curr_x == x && curr_y == y )
116           {
117             if( ! fill )
118             {
119               fill = true;
```

```
120                 dir = right;
121               }
122               else
123               {
124                 break;
125               }
126             }
127           }
128         }
129       }
130
131       emit rectPainted( UIntRect( 0, y, matrix.width() - 1, y ) );
132     }
133
134 }
```

A.2. Darstellung des Feigenbaumdiagramms

Listing 8: Der Code zum Berechnen des Feigenbaumdiagramms

```
1  void Feigenbaum::Calc::run()
2  {
3    m_calcing = true;
4
5    emit started();
6
7    m_output->pixmap().resize( m_input->picSize().width,
8                               m_input->picSize().height );
9    m_output->deleteData();
10
11   for( int x = 0;
12        x < m_input->picSize().width && m_calcing; x++ )
13   {
14     const double u = m_input->coords().tl.x +
15                      (double) m_input->coords().width() /
16                      m_input->picSize().width * x;
17     double result = m_input->startValue();
18     for( int iter = 0;
19          iter < m_input->maxIterations(); iter++ )
20     {
21       result = u * result * ( 1 - result );
22       if( iter >= m_input->maxIterations() -
23           m_input->paintIterations() )
24       {
25         const double percent =
26                ( result - m_input->coords().br.y )
27                / -m_input->coords().height();
```

```
28          const int y = ( 1 - percent )
29                        * m_input->picSize().height;
30
31          m_output->setPixel( UIntPoint( x, y ) );
32        }
33      }
34
35      emit updated( QRect( x, 0, 1, m_input->picSize().height ) );
36    }
37
38    if( m_calcing )
39    {
40      m_calcing = false;
41      emit finished();
42    }
43    else
44    {
45      emit stopped();
46    }
47  }
```

B. Quellennachweise

Bücher

[1] Reinhart Behr, *Ein Weg zur fraktalen Geometrie*, Ernst Klett Schulbuchverlag, 1. Auflage, 1989.

[2] Bjarne Stroustrup, *Die C++-Programmiersprache*, Addison-Wesley, 4. Auflage, 2000.

Internet-Seiten

[3] Paul Derbyshire, *Quick Guide to the Mandelbrot Set*, http://www.globalserve.net/~derbyshire/manguide.html, Version vom 7.12.2002, 23:38 Uhr.

[4] Maarten Egmond, *Speed in Mandlebrot calculations*, Posting in die Newsgroup sci.fractals vom 02.10.1996, gefunden mit Google Groups (http://groups.google.com) am 24.8.2002, 13:07 Uhr.

[5] Michael R. Ganss, *AlmondBread*, http://user.cs.tu-berlin.de/~rms/AlmondBread/, Version vom 29.8.2002, 17:11 Uhr.

[6] Microsoft Encarta Online-Enzyklopädie, *Artikel „Fraktal“*, http://encarta.msn.de, Version vom 06.11.2002, 13:11 Uhr.

[7] Michael Charles Taylor, *FAQ (Frequently Asked Questions) der Newsgroup sci.fractals*, http://spanky.triumf.ca/pub/fractals/docs/SCI_FRACTALS.FAQ, Version vom 21.8.2002, 11:08 Uhr.

[8] J. Bentele und J. Theofel, *Feigenbaumdiagramm - Der geordnete Weg ins Chaos*, http://www.gus.bb.bw.schule.de/fbaum/fbaum.htm, Version vom 9.7.2002, 13:38 Uhr.

[9] Florian Wolferseder, *Mandelbrot - Symmetrie zur reellen Achse*, http://www.wolferseder.de/Mandelbrot-Menge.html, Version vom 10.12.2002, 17:23 Uhr.

C. GNU Free Documentation License

Version 1.2, November 2002

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

C.1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship

could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect

on the meaning of this License.

C.2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section C.3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

C.3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

C.4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections C.2 and C.3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

C.5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section C.4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant

Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

C.6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

C.7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section C.3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

C.8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section C.4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers.

In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section C.4) to Preserve its Title (section C.1) will typically require changing the actual title.

C.9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

C.10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.