Herzog-Christian-August-Gymnasium
Sulzbach-Rosenberg

Kollegstufenjahrgang 20... /.....

F A C H A R B E I T
aus dem Fach

.....

Thema: .....
.....

Verfasser: .....
Leistungskurs .....
Kursleiter .....
Abgabetermin: .....

Erzielte Note ..... in Worten: .....
Erzielte Punkte: ..... in Worten: .....
(einfache Wertung)

Erzielte Punkte in der mündlichen Prüfung: .....
(in Worten)

Abgabe beim Kursleiter am: .....

.....
(Unterschrift des Kursleiters)

1 EINLEITUNG

[1]. Das Wort Fraktal kommt vom lat. Adjektiv *fractus* (zerbrochen). Fraktale sind geometrischen Figuren wie Dreiecke oder Vierecke, unterscheiden sich jedoch in vielerlei Hinsicht, wie zum Beispiel der Tatsache, dass sie unendlich viele Details besitzen, weshalb sie auch manchmal als mathematische Monster bezeichnet werden.

2 EIGENSCHAFTEN VON FRAKTALEN

2.1 CANTOR-MENGE

[2]Als Beispiel für ein Fraktal betrachten wir die Cantor-Menge, die folgendermaßen konstruiert wird:

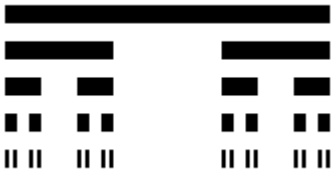


Abbildung 1: Konstruktion der Cantor Menge

Konstruktion:

Man beginnt mit dem abgeschlossenen reellen Intervall I von 0 bis 1.

Nun entfernt man sein mittleres Drittel, also das offene Intervall von 1/3 bis 2/3 und erhält zwei abgeschlossene Teilintervalle I_1, I_2. Auf jedem Teilintervall wiederholen wir denselben Vorgang. Wenn man dies unendlich oft wiederholt, ist der Grenzwert die sogenannte Cantor-Menge (benannt nach dem deutschen Mathematiker Georg Cantor). Dieses wiederholte Ausführen eines Algorithmus nennt man Iteration (vom lat. *iterare* „wiederholen“).Das Ausgangsobjekt, in diesem Fall das Intervall, nennt man Initiator. Dieses wird durch einen sog. Generator ersetzt, hier einem Intervall, dem das mittlere Drittel fehlt, d.h. durch zwei verkürzte Intervalle ersetzt. Dadurch wird ein neuer Initiator erzeugt, auf den wir dieses Verfahren erneut anwenden usw.

Es fällt auf, dass jede Iteration die vorherige im kleineren Maßstab enthält. Eine weitere Auffälligkeit ist, dass bei vielfacher Iteration immer mehr Intervalle entstehen, deren Größe immer mehr abnimmt.

Anzahl der Teilintervalle nach n-ter Iteration: $N_n = 2^n$

$\lim_{n->\infty} \mathrm{N_n} = \infty$

Gesamtlänge der Teilintervalle nach n-ter Iteration: $L_n = \left(\frac{2}{3}\right)^n$

$\lim_{n->\infty} \mathrm{L_n} = 0$

2.2 SELBSTÄHNLICHKEIT

2.2.1 KOCHKURVE

[3]Die Kochkurve ist ein vom schwedischen Mathematiker Helge von Koch 1904 vorgestelltes Beispiel für eine überall stetige aber nirgends differenzierbare Kurve. Sie ist ein sehr beliebtes Beispiel für Fraktale.

2.2.1.1 Konstruktion:

Man nimmt eine bestimmte Strecke s (Initiator). Das mittlere Drittel der Strecke wird durch ein gleichseitiges Dreieck ersetzt und anschließend das mittlere Drittel entfernt (1.Iteration). Diese Ersetzung wird nun für jede der 4 neu entstandenen Strecken wiederholt (2.Iteration). Die n-te Iteration strebt im Grenzwert $n->\infty$ gegen die Kochkurve.

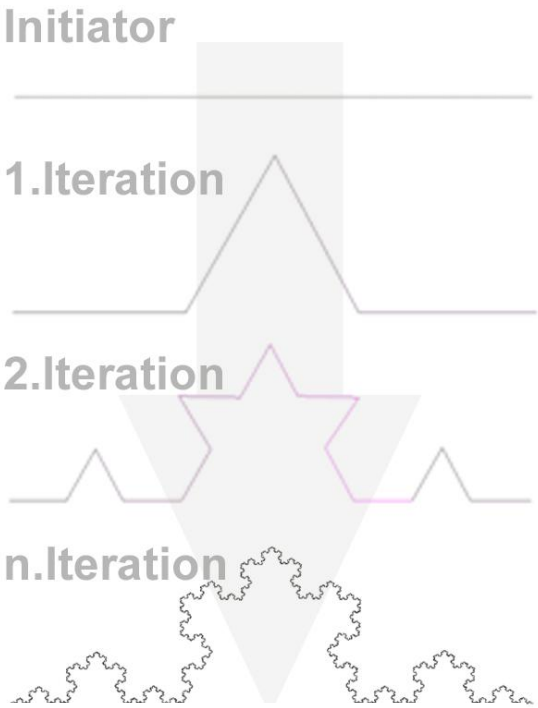


Abbildung 2: Konstruktion der Kochkurve

2.2.1.2 Eigenschaften der Kochkurve

Gesamtanzahl der Linien bei n-ter Iterationsstufe $N_n = 4^n$

$\lim_{n->\infty} \mathrm{N_n} = \infty$

Länge einer Teillinie bei n-ter Iterationsstufe: : $l_n = \left(\frac{1}{3}\right)^n \cdot s$

$\lim_{n->\infty} \mathrm{l_n} = 0$

Gesamtlänge bei der n. Iterationsstufe (Länge einer Linie · Anzahl der Linien):

$$L_n = \left(\frac{1}{3}\right)^{\mathrm{n}} \cdot \mathrm{s} \cdot 4^n = (\frac{1}{3} \cdot 4)^n \cdot \mathrm{s} = (\frac{4}{3})^n \cdot \mathrm{s}$$

$$\lim_{n->\infty} L_n = +\infty$$

Fläche unter der „Zacke“:

Anzahl der Zacken bei k-ter Iterationsstufe:

$$N_{Zacken\ (k)} = \begin{cases} 4^{k-1} & (k > 0) \\ 0 & (k = 0) \end{cases}$$

Fläche einer Zacke bei k-ter Iterationsstufe:

$$A_{Zacke\,(k)} = \begin{cases} \frac{\sqrt{3}}{4} \cdot \frac{s^2}{9^k} & (k > 0) \\ 0 & (k = 0) \end{cases}$$

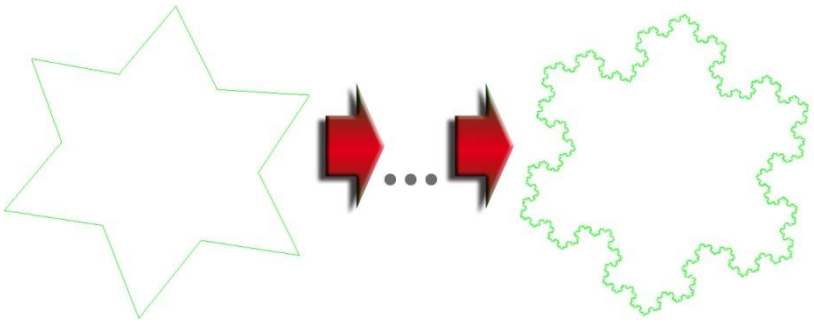
Die Gesamtfläche aller Zacken bei k-ter Iteration:

$$A_{Ges\,(n)} = \sum_{k=0}^{n} N_{Zacken\,(k)} * A_{Zacke\,(k)} = \sum_{k=1}^{n} N_{Zacken\,(k)} * A_{Zacke\,(k)}$$

$$= \sum_{k=0}^{n-1} N_{Zacken\,(k+1)} * A_{Zacke\,(k+1)} = \sum_{k=0}^{n-1} 4^k * \frac{\sqrt{3}}{4} * \frac{s^2}{9^{k+1}} = \frac{\sqrt{3}*s^2}{36} * \sum_{k=0}^{n-1} \left(\frac{4}{9}\right)^k$$

$$= \frac{\sqrt{3}*s^2}{36} * \frac{1-\left(\frac{4}{9}\right)^n}{1-\frac{4}{9}} = \frac{\sqrt{3}*s^2}{20}\left(1 - \left(\frac{4}{9}\right)^n\right)$$

2.2.2 KOCHINSEL



Kochinsel 1. Iteration Kochinsel

Wenn man drei um 120° gegeneinander geneigte Kochkurven zusammensetzt, begrenzen diese ein Gebiet, das man als Kochinsel bezeichnet.

Für die Fläche (gleichseitiges Dreieck + 3* Gesamtfläche der Zacken) gilt dann:

$$A_{I(n)} = \frac{s^2}{4} * \sqrt{3} + 3 * \frac{\sqrt{3}*s^2}{20} * \left(1 - \left(\frac{4}{9}\right)^n\right) = \frac{2}{5}*\sqrt{3} * s^2 - \frac{3\sqrt{3}}{20} * \left(\frac{4}{9}\right)^n * s^2$$

$$\lim_{n->\infty} \mathrm{A}_{\mathrm{l(n)}} = \frac{2}{5} * \sqrt{3} * s^2$$

Für den Umfang gilt:

$$U_{I(n)} = 3 * 4^{n\left(\frac{1}{3}\right)^n} * s = 3 * \left(\frac{4}{3}\right)^n * s$$

$$\lim_{n->\infty} \mathrm{U}_{\mathrm{l(n)}} = \infty$$

Wenn man einen Vermessungsingenieur, mit einem Lineal der Länge $l_n = \left(\frac{1}{3}\right)^n \cdot s$ versehen, beauftragen würde, den Umfang der Kochinsel zu vermessen, bekäme er den endlichen „Umfang“ $U_{I(n)}$ heraus. Kürzere Lineale liefern immer

größere „Umfänge“. „Unendlich kurze Lineale“ würden einen unendlich langen Umfang liefern. Dies ist erstaunlich, da die Fläche der Kochinsel natürlich endlich ist. Die Küstenlinie einer realen Insel wie z. B. England besitzt eine ähnlich fraktale Struktur. Es ist daher prinzipiell unmöglich die wirkliche „Länge“ der Küste von England, das natürlich eine endliche Fläche besitzt, mit Hilfe von Linealen zu ermitteln, da extrem kurze Lineale, die man jeder Unebenheit anlegen kann, eine unendlich lange Küstenlänge ergäben.

2.2.3 Definition von Selbstähnlichkeit

Nun geh ich etwas näher auf das zuvor erwähnte Phänomen ein, dass höhere Iterationen in niedrigeren Iterationen enthalten sind. Generell findet man in einem Fraktal das Fraktal selber in verkleinerter und/oder verdrehter, gespiegelter, verzerrter Form wieder. Diese Tatsache nennt man Selbstähnlichkeit.
Definition [4]: „Selbstähnlichkeit ist die Eigenschaft von Gegenständen, Körpern, Mengen oder geometrischen Objekten, bei Vergrößerung dieselben oder ähnliche Strukturen aufzuweisen wie im Anfangszustand“. Auch in der Natur kann man oft eine gewisse Selbstähnlichkeit erkennen, so ähnelt die Aorta (Hauptschlagader) den Arterien, welche sich verzweigen und diese den Arteriolen, die sich noch viel stärker verzweigen, also liegt eine gewisse Selbstähnlichkeit des Blutkreislaufs vor.
Da konstruierte Fraktale wie die Cantor Menge theoretisch unendlich iteriert werden, erhält man, egal wie oft man das Objekt vergrößert, immer eine exakte Kopie des Originals. Deshalb spricht man von exakter Selbstähnlichkeit. Diese tritt nur bei mathematischen Fraktalen wie der Cantor-Menge auf, und nicht bei natürlichen Fraktalen, dort wäre dies auch wegen physikalischer Beschränkungen gar nicht möglich, spätestens bei der Größe von Atomen wäre es mit Selbstähnlichkeit zu Ende.

2.2.4 *Selbstähnlichkeit bei der Kochkurve*

Wenn man zum Beispiel aus dem Bild einer Kochkurve das linke Drittel abschneidet und einzeln betrachtet, so ergibt dieser Teil, wenn man ihn 3 fach vergrößert, das Original. Ebenso kann man mit den restlichen 3 Teilen verfahren, allerdings sind die mittleren 2 Stücke um $\pm$ 60° gedreht.

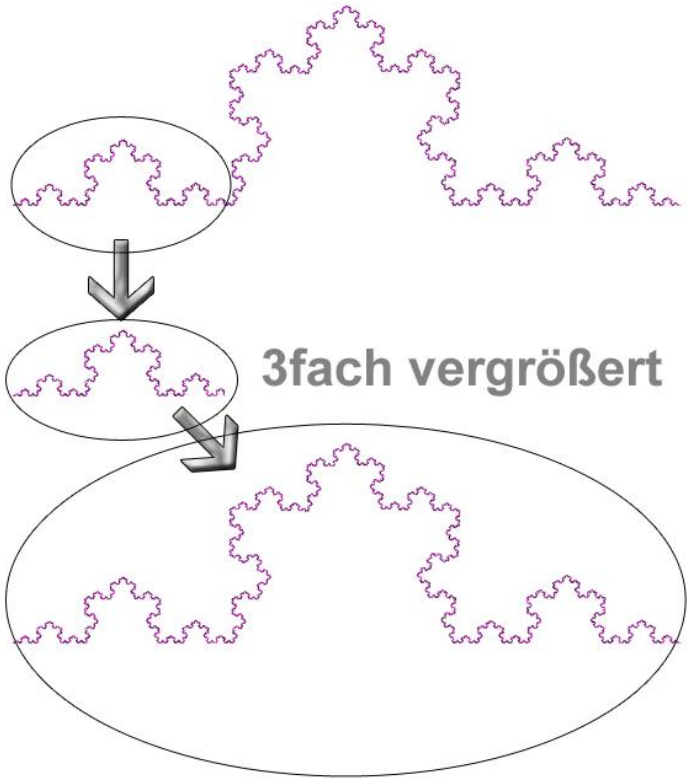


Abbildung 3: Selbstähnlichkeit bei der Kochkurve verdeutlicht durch Vergrößerung

2.3 Dimension

2.3.1 *Sierpinski Dreieck*

Konstruktion:

Man nehme ein Dreieck der Seitenlänge s (Initiator), verbinde die Seitenmittelpunkte und erzeuge damit 4 neue Dreiecke (1.Iteration). Von diesen lässt man das mittlere Dreieck weg, und wiederholt die Prozedur mit den restlichen Dreiecken (Iteration).

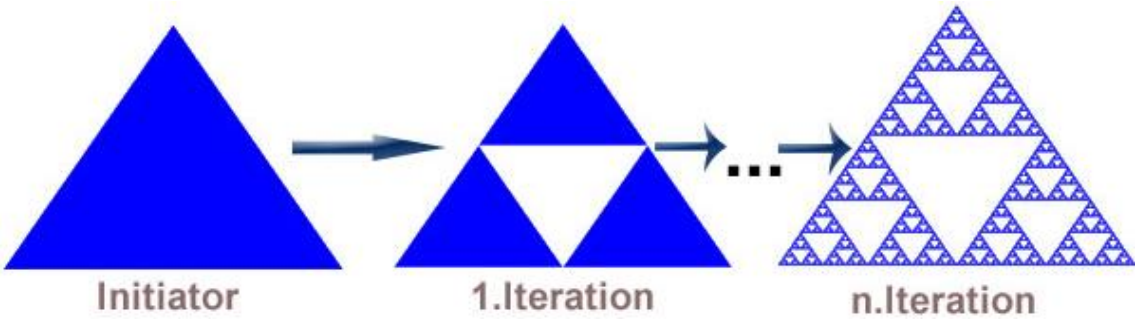


Abbildung 4: Konstruktion des Sierpinski Dreiecks

2.3.2 *Eigenschaften des Sierpinski-Dreiecks*

Anzahl der Dreiecke bei n. Iteration: 3^n

Seitenlänge bei n. Iteration: $L_n = (\frac{1}{2})^n \cdot s$

Gesamtfläche: $A_n = (\frac{3}{4})^n \cdot \frac{\sqrt{3}}{4} s^2$

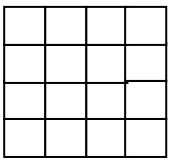
$\lim_{n\to\infty} A_n = 0$

2.3.3 DIE ÄHNLICHKEITSDIMENSION

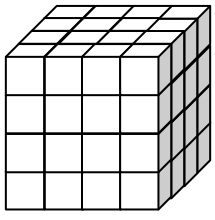
Eine weitere Eigenschaft der Cantor-Menge besteht darin, dass die Intervalle mit höherer Iteration n, immer kürzer werden, und aus einer Vogelperspektive betrachtet punktförmig erscheinen. Aus der Nähe betrachtet bleibt der Liniencharakter, aus der Ferne betrachtet erscheint ein Punktcharakter. Ist die Cantor-Menge punktförmig oder linienförmig? Um diese Frage zu beantworten, bestimmt man die Dimension.
Bevor die Dimension der Cantor-Menge analysiert werden kann, benötigt man erst einmal eine Methode um die Dimension festzustellen. Dazu benutze ich die sogenannte Ähnlichkeits-Dimension, welche den Vorteil hat, dass sie leicht verständlich und ohne Limes berechenbar ist.
Um diese zu verstehen, schaut man sich an, wie sich einfache geometrische Objekte aus ihren eigenen Bildern unter zentrischen Streckungen zusammensetzen lassen.

- Um eine eindimensionale Strecke der Länge *l* aus  Bildern mit Streckungsfaktor $k = \frac{1}{n}$ zusammenzusetzen benötigen wir $N=n^1$ Bilder.

- Wenn dagegen ein Quadrat der Dimension 2 mit Seitenlänge *l* aus ihren Bildern mit Streckungsfaktor $k = \frac{1}{n}$ zusammengesetzt werden soll, benötigen wir $N=n^2$ Bilder.



- Soll schließlich ein dreidimensionaler Würfel mit Kantenlänge *l* aus seinen Bildern mit Streckungsfaktor $k = \frac{1}{n}$ zusammengesetzt werden, benötigen wir $N=n^3$ Bilder.



Zwischen Dimension, Streckungsfaktor und Anzahl selbstähnlicher Teile scheint ein Zusammenhang zu bestehen.

Ein geometrisches Objekt wird aus seinen Bildern mit Streckungsfaktor $k = \frac{1}{n}$ zusammengesetzt, wobei wir dafür $N = n^D$ Bilder benötigen.

Wir verwenden die letzte Beziehung um die Dimension D allgemein zu definieren.

$$N = n^D = \left(\frac{1}{k}\right)^D \Rightarrow \ln(N) = D * ln\left(\frac{1}{k}\right) \Rightarrow \boxed{D = \frac{\ln(N)}{\ln\left(\frac{1}{k}\right)}}$$

Im Falle des Quadrats ergibt sich nach dieser Definition.

$D = \frac{ln(n^2)}{ln(n)} = 2$, wie es sein soll.

2.3.4 Dimension des Sierpinski-Dreiecks

Nun wird diese Regel auf das vorher besprochene Sierpinski-Dreieck angewandt.
Um das Sierpinski-Dreieck zusammenzusetzen, benötigen wir drei Bilder (N=3) mit Streckungsfaktor k $= \frac{1}{2}$.

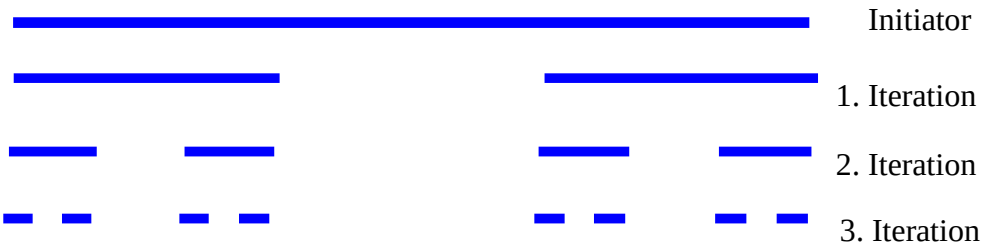
$$\mathrm{D} = = \frac{\ln(3)}{\ln(\frac{1}{0.5})} = \frac{\ln(3)}{\ln(2)} \approx 1.584962500$$

Die Dimension des Sierpinski-Dreiecks ist also nicht geradzahlig!
Dies ist eine Besonderheit der Fraktale. Ihre Dimension ist, im Gegensatz zur normalen Geometrie, in der nur ganze Dimensionen wie 0, 1, 2 oder 3 auftauchen, gebrochen oder sogar irrational. Das Sierpinski-Dreieck mit seiner Dimension $D \approx 1.584$, die zwischen Eins und Zwei liegt, hat demnach sowohl streckenhafte wie flächenhafte Züge.

2.3.5 Dimension der Cantor Menge

Wie bereits erwähnt ist die Cantor-Menge ein „Etwas“ zwischen Strecke und Punkt.
Um diese vage Aussage zu konkretisieren, berechnen wir seine fraktale Dimension.
Wir setzen die Cantor-Menge aus zwei Bildern (N=2) mit Streckungsfaktor $k = \frac{1}{3}$ zusammen.

$$D = \frac{\ln(2)}{\ln(3)} \approx 0.6309297535$$



2.3.6 Dimension der Kochkurve

Die Kochkurve setzt sich aus vier Bildern (N = 4) mit dem Streckungsfaktor $k = \frac{1}{3}$ zusammen.

$\text{Dim} = \frac{\ln(4)}{\ln(3)} \approx 1.261859536$

Damit hat die Kochkurve sowohl linienhafte wie flächenhafte Züge.

2.4 ITERATED FUNCTION SYSTEMS

Jetzt da die Konstruktionen einiger Fraktale bekannt schreibe ich eine Methode, mit der man beliebige Fraktale erzeugen kann, nur indem man Parameter von Gleichungen ändert. Doch dazu benötigt man einige Funktionen, mit denen man Punkte transformieren kann.

2.4.1 DIE VERSCHIEBUNGS-DREHSTRECKUNG

2.4.1.1 Translation

Wenn man einen Punkt P (x, y) um $\binom{e}{f}$ verschieben will, so ist der x Wert des neuen Punktes x+e und der y-Wert y+f. Es gelten also folgende zwei Formeln:

$x' = x + e$
$y' = y + f$

In Matrizenschreibweise:

$$\overrightarrow{x'} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \vec{x} + \binom{e}{f}$$

2.4.1.2 Rotation

Der Punkt (x, y) habe die Polarkoordinaten r und α. Dann gilt:

$x = r * \cos(\propto)$
$y = r * \sin(\propto)$

Nun drehen wir den Punkt P um den Ursprung um den Winkel φ. Es gilt mit den Additionstheoremen:

$$x' = r * \cos(\propto - \varphi) = r * \cos \propto * \cos\varphi + r * \sin \propto * \sin\varphi = \cos\varphi * x + \sin\varphi * y$$
$$y' = r * \sin(\propto - \varphi) = r * \sin \propto * \cos\varphi - r * \cos \propto * \sin\varphi = -\sin\varphi * x + \cos\varphi * y$$

$\mathbf{x' = \cos\varphi \cdot x + \sin\varphi \cdot y}$
$\mathbf{y' = -\sin\varphi \cdot x + \cos\varphi \cdot y}$

Oder in der Matrizenschreibweise:

$$\overrightarrow{x'} = \begin{bmatrix} cos\varphi & sin\varphi \\ -sin\varphi & cos\varphi \end{bmatrix} \cdot \vec{x}$$

2.4.1.3 Bewegungen

Bewegungen oder eigentliche Kongruenzabbildungen entstehen durch Hintereinanderausführungen von Rotationen und Translationen. Daher lassen sie sich folgendermaßen darstellen.

$x' = \cos\varphi \cdot x + \sin\varphi \cdot y + e$

$y' = -\sin\varphi \cdot x + \cos\varphi \cdot y + f$

Matrizenschreibweise: $\overrightarrow{x'} = \begin{bmatrix} cos\varphi & \sin\varphi \\ -sin\varphi & cos\varphi \end{bmatrix} \cdot \vec{x} + \begin{pmatrix} e \\ f \end{pmatrix}$

Kongruenzen erhalten bekanntlich die Abstände zweier Punkte.

2.4.1.4 Ähnlichkeitsabbildungen

Wenn man einen Punkt(x, y) am Ursprung mit Streckungsfaktor k zentrisch streckt gilt:

$x' = kx$

$y' = ky$

Ähnlichkeitsabbildungen entstehen durch die Hintereinanderausführungen von Bewegungen und zentrischen Streckungen. Daher lassen sie sich darstellen durch:

$x' = k \cdot \cos\varphi \cdot x + k \cdot \sin\varphi \cdot y + e$

$y' = -k \cdot \sin\varphi \cdot x + k \cdot \cos\varphi \cdot y + f$

Matrizenschreibweise:

$\overrightarrow{x'} = \begin{bmatrix} k \cdot cos\varphi & k \cdot \sin\varphi \\ -k \cdot sin\varphi & k \cdot cos\varphi \end{bmatrix} \cdot \vec{x} + \begin{pmatrix} e \\ f \end{pmatrix}$

Ähnlichkeitsabbildungen mit |k|>1 vergrößern die Abstände zweier Punkte. Ähnlichkeitsabbildungen mit |k|<1 verkleinern dagegen die Abstände zweier Punkte um den Faktor |k|<1 und man bezeichnet sie daher als kontrahierend. Für |k|=1 erhalten wir dagegen Kongruenzabbildungen.

2.4.2 KOCHKURVE ALS IFS FRAKTAL

Wenn alle Funktionen kontrahierend sind, dann gibt es einen eindeutigen Attraktor, das heißt alle Punkte nähern sich je öfters iteriert werden immer näher einem Endzustand an.

Mit den vorgestellten Mitteln kann man nun eine Punktemenge verformen. Um damit ein Fraktal zu erzeugen, benötigt man einen festen Satz von Ähnlichkeitsabbildungen, die iteriert auf eine Ausgangsmenge angewandt werden. Bei der Kochkurve zum Beispiel müssen bei der vorliegenden Punktemenge (einer Strecke) vier verschiedene Transformationen iteriert werden (siehe auch Abbildung 7). Eine solche Vorgehensweise bezeichnet man als IFS (iteriertes Funktionen-System).

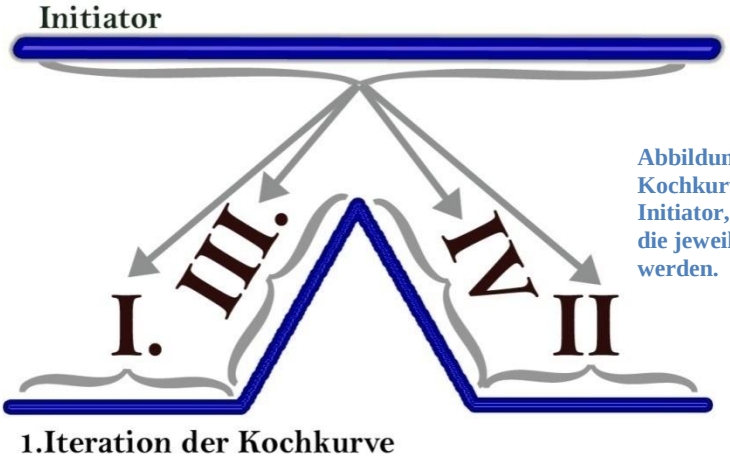


Abbildung 5: Die 4 Transformationen des Kochkurven IFS. Es werden von dem Initiator, der Strecke, vier Kopien angefertigt, die jeweils unterschiedlich transformiert werden.

I.	Kontraktion um 1/3
II.	Kontraktion um 1/3 und Verschiebung um 2/3
III.	Kontraktion um 1/3, 60 Grad Drehung und Verschiebung um 1/3
IV.	Kontraktion um 1/3. -60 Grad Drehung und Verschiebung um $[\frac{1}{2};\frac{\sqrt{3}}{2}]$

Es gibt also 4·2 Formeln:

$x' = \frac{1}{3} * x + 0 * y + 0$ $y' = \frac{1}{3} * x + 0 * y + 0$	I
$x' = \frac{1}{3} * x + 0 * y + \frac{2}{3} * s$ $y' = \frac{1}{3} * x + 0 * y + \frac{2}{3} * s$	II
$x' = \frac{1}{3} * cos(60) * x + sin(60) * y + \frac{1}{3} * s$ $y' = -\frac{1}{3} * sin(60) * x + cos(60) * y + \frac{1}{3} * s$	III
$x' = \frac{1}{3} * cos(-60) * x + sin(-60) * y + \frac{1}{2} * s$ $y' = -\frac{1}{3} * sin(-60) * x + cos(-60) * y + \frac{\sqrt{3}}{2} * s$	IV

$$w_1 = \frac{1}{3} \cdot \begin{bmatrix} cos(0) & \sin(0) \\ -\sin(0) & \cos(0) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{(I)}$$

$$w_2 = \frac{1}{3} \cdot \begin{bmatrix} cos(0) & \sin(0) \\ -\frac{1}{3} * \sin(0) & \cos(0) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \frac{2}{3} \\ 0 \end{bmatrix} \quad \text{(II)}$$

$$w_3 = \frac{1}{3} \cdot \begin{bmatrix} cos(60) & \sin(60) \\ -\sin(60) & \cos(60) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \frac{1}{3} \\ 0 \end{bmatrix} \quad \text{(III)}$$

$$w_4 = \frac{1}{3} \cdot \begin{bmatrix} cos(-60) & \sin(-60) \\ -\sin(-60) & \cos(-60) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix} \quad \text{(IV)}$$

Bzw. 4 Matrizen

Alle Transformationen, die für ein IFS nötig, sind fast man nun zusammen (A ist eine beliebige kompakte Menge, das bedeutet, dass die Menge nicht unendliche Ausdehnung hat, und der Rand Teil der Menge ist. $w_1, w_2, \dots, w_n$ seien Kontraktionen der Ebene): $W(A) = w_1(A) \cup w_2(A) \cup w_3(A) \cup w_4(A) \dots \cup w_n(A)$ ist dann die Vereinigung der Bilder von A unter den verschiedenen Abbildungen $w_1, \dots, w_n$.
Wenn man nun diesen Operator auf die Punktemenge anwendet erhält man eine neue Punktemenge, die .aus n sich möglicherweise überlappenden Gebieten besteht. Je öfter man den Operator auf eine Startmenge A anwendet, desto mehr nähert sich die Bildpunktemenge einem Attraktor, in diesem Fall der Kochkurve an.
Es wird also aus A eine neue Punktmenge erzeugt: Punktemenge $A_1 = W(A_0)$.Aus dieser kann man nun A_2 berechnen, usw. bis irgendwann $A_\infty = W(A_\infty)$. Wegen der Kontrahierenden Wirkung der $w_k$werden beliebige Punkte bei Iteration immer näher an A_∞ herangezogen. Man nennt A_∞daher Attraktor.

2.5 LINDENMAYER-SYSTEM

[5]Das Lindenmayer-System oder auch L-System genannt wurde 1968 von dem ungarischen Biologen Aristid Lindenmayer entwickelt, um Pflanzen realitätsnah modellieren zu können. Es werden Befehle durch Buchstaben ersetzt und diese werden nach vorgegebenen Regeln wiederholt ersetzt. Um diese Zeichenketten sichtbar zu machen, benutze ich Turtle-Grafik, ein Begriff, der von der LOGO-Programmiersprache abstammt. Dazu stellt man sich am besten eine Schildkröte vor, die bei ihren

Bewegungen eine sichtbare Spur im Sand hinterlässt. Die inneren Zustände der Turtle bestehen aus seiner aktuellen Position (x, y) und seiner aktuellen Blickrichtung w. Daneben existiert noch der feste Winkel a, der angibt, um welchen Betrag sich die Turtle bei einer Rotation drehen kann und die Größe d, die angibt um wie viel sich die Turtle bei einem Schritt nach vorne bewegt. Die Turtle verfügt über ein rudimentäres Gedächtnis in Form eines Stacks. (Ein Stack ist ein Stapel von inneren Zuständen, bei dem man nur etwas oben auf legen kann (push) oder das oberste Element entnehmen kann (pop)).
Die Turtle Grafik meines Programms beherrscht folgende Befehle:

„F“: gehe vorwärts (mit Spur im Sand)
„!“: gehe vorwärts (ohne Spur im Sand)
„+“: nach links drehen
„-“: nach rechts drehen
„[“: der innerer Zustand wird auf den Stack• gelegt (push)
„]“: der innerer Zustand wird vom Stack• geholt (pop)

d, a und w_0 müssen als Startbedingungen angegeben werden (d dient zur Skalierung, damit das Fraktal nicht so groß gezeichnet wird, dass es auf dem Bildschirm nicht mehr komplett angezeigt werden kann, a beeinflusst das Aussehen des Fraktals, und w_0, wie es gedreht ist).
Bei einem L-System beginnt man nun mit einer Zeichenkette aus Befehlen, dem Axiom, z. B. „F--F--F“ und bearbeitet nun diese Zeichenkette nach bestimmten Regeln, z. B. wird jedes vorkommende F durch F+F--F+F („Zacke“) ersetzt. Es entsteht also die Zeichenkette F+F--F+F--F+F--F+F--F+F--F+F. Diese Ersetzung wird nun jeweils mit der resultierenden Zeichenkette n fach durchgeführt. Diese Zeichenketten lasse ich dann in meinem Programm mit einer Implementation von Turtle-Grafik zeichnen. Wenn man das vorherige Beispiel, dass ich für eine Zeichenkette angegeben habe, mit dem Winkel a = 60 Grad zeichnet, erhält man die Kochinsel. Man kann auch mehrere Regeln angeben, und beliebige Buchstaben für diese Regeln verwenden.

Definition:
[5]Ein L-System ist ein Quadrupel G = (V, S, ω, P), wobei

- V die Menge der Zeichen ist, die als Variable angesehen werden sollen. (z. B.: F)

- S die Menge der Zeichen ist, die als Konstanten angesehen werden sollen. Die Zeichen aus V und S bilden das Alphabet des L-Systems. (z. B.: +,-,F,!)
- ω ein Wort über dem Alphabet ist, welches als Startwort oder Axiom des L-Systems bezeichnet wird. (z. B.: F--F--F)
- P eine Menge von geordneten Paaren aus Wörtern über dem Alphabet ist, welche die Ersetzungsregeln definieren.

  Ist das erste Wort eines jeden Paares ein einzelner Buchstabe aus V, und zu jeder Variablen eine Ersetzungsregel bekannt, so spricht man von einem kontextfreien L-System. Wenn mehr als ein Buchstabe auf der linken Seite der Ersetzungsregel steht, spricht man von einem kontextsensitiven. In meinem Programm sind nur kontextfreie L-Systeme möglich, da kontextsensitive zu Mehrdeutigkeiten führen können (wird ABC, wenn die es die Ersetzungsregeln AB->D und BC->F existieren, DC oder AF?). Das L-System für die Kochinsel sieht also wie folgt aus:
  G={F;{+,-,F};F--F--F;F->F+F--F+F}

3 ANHANG

Programmcode

Wichtig sind die Klassen in fraktale_sammlung.dll, insbesondere die paint(int n,array<PointF>^ points) Funktionen (dort wird das jeweilige Fraktal gezeichnet). Manchmal zeichnet die paint Funktion nicht direkt das Fraktal, sondern ruft eine andere auf um dies zu tun. Dies hat dann technische Gründe.
Eher unwichtig hingegen sind die Klassen in Fraktale.exe, da diese die Fenster, und andere Grafische Elemente erstellen und steuerern.

3.1 FRAKTALE_SAMMLUNG.DLL

In dieser DLL sind die Klassen, die die Fraktale zeichnen.

3.1.1 FRAKTAL_SAMMLUNG.H

```
#pragma once
// fraktal_sammlung.h
// Enthält die Klasse Fraktal, welche als Basisklasse für alle Fraktale dient
////////////////////////////////////////////////////////////////////////////////
///     Prototyp für eine Fraktal Subklasse
////////////////////////////////////////////////////////////////////////////////
/// die Funktionen paint(int n,array<PointF>^ points)
/// und check() müssen überschrieben werden da sie als abstrakt definiert sind!
////////////////////////////////////////////////////////////////////////////////
//public ref class einFraktal : public Fraktal
//{
//public:
//  einFraktal(){name="einFraktal";neededpoints=0;numberofpoints=2;}
//      virtual void paint(int n,array<PointF>^ points)override{/* TODO: CODE DER FRAKTAL
ERZEUGT*/}
//      virtual int check()override{return 0; /* TODO: CODE DER DIE ANZAHL DER OBJEKTE
VORAUSBERECHNET*/}
//};

#define MAX_MODES 8   /// Anzahl der Sublassen der Klasse Fraktal in dieser dll
#include "Drawing.h"
#include "MorphMatrix.h"
#define PI 3.14159265

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Drawing;
using namespace System::Drawing::Drawing2D;
using namespace System::Threading;
namespace fraktal_sammlung {
public  ref  class Fraktal
        {
        public: array<PointF>^ points;
        public: Color front,back;
        public: bool random,clear,Multithreading,draw_info,abbr_distance;
        public: int linien,neededPoints,n,laenge;
        protected: Random^ rand;
        protected: int minpoints;
        public: double wachstum;
        public: TimeSpan^ time;
        public: String ^status, ^name, ^filetyp;
        protected: Thread^ thread1;
        protected: Draw^ graphics;
        public:  Fraktal()
                        {
                                rand = gcnew Random();
                                linien = 0;
                                random = 0;
                                MinPoints = 0;
                                Multithreading = 1;
                                name="Fraktal";filetyp="*";
```

```
                                points = gcnew array<PointF>(1);
                                clear = 1;
                                abbr_distance = 1;
                        }
        public:  ~Fraktal(){stopthread();}
        public:  void paint(Draw^ g)
                        {
                                if(n){
                                // Vorbereitung um Fraktal zu zeichnen, Bitmap von letztem
Fraktal wird gelöscht,
                                // Infos zum aktuellen Fraktal werden geschrieben, neuer
Thread der das Fraktal zeichnet wird gestartet
                                  graphics = g;
                                  if(clear)graphics->Clear(back);
                                  if(points-
>Length>1&&point_distance(points[0],points[1])<10){throw gcnew Exception("Punktabstand
ist sehr klein,bitte vergrössern");}
                                  if(draw_info){wachstum =
Math::Pow(check(),1.0/n);write3(info());}
                                  starthread();
                                }
                                 // wenn n < 1 sind nocht nicht genügend Punkte angegeben,
also werden die bisherigen mit der aktuellen Mausposition verbunden und gezeichnet
                                else{drawpoints(g);} // if(n)
                        }
        public:  void paint(Graphics^g){paint(gcnew Draw(g));/* Graphics ist Klasse des
net Frameworks für Grafik-Operationen, ich benütze eigene Klasse Draw dafür*/}
        public: void paint(Graphics^g,Bitmap^b){
                                /// Bitmap ist Initiator
                                        paint(gcnew PictureInitiator(g,b));
                        }
        protected: virtual void paint(int n,array<PointF>^ points)=0;
        public:  void  setmousepos(PointF p){if(!n){points[points->Length-1]=p;}
                         // damit wird die Mausposition dem Fraktal mitgeteilt
                         }
        private:  void starthread()
                        {
                                // Startet einen Thread, der die Funktion paint()
ausführt,
                                // oder führt die funktion paint() ohne neuen Thread aus,
                                // falls Multithreading  deaktiviert
                                        try
                                        {
                                                if (Multithreading)
                                                {
                                                        if(thread1!=nullptr)
                                                        {if(thread1->IsAlive){thread1-
>Abort();}}
                                                        delete thread1;
                                                        thread1 = gcnew Thread( gcnew
ThreadStart(this,&Fraktal::paint) ) ;
                                                        thread1->Start();
                                                }
                                                else{paint();}
                                        }
                                        catch (System::Exception^ error)
                                        {
#ifdef _DEBUG
                                                MessageBox::Show("Fehler: "+error-
>Message);
#endif // _DEBUG
                                                status = error->Message;
                                        }

                        }

        public:   void suspendthread(){
                                 // Thread anhalten, falls er existiert und noch nicht
angehalten ist
                                 if(thread1!=nullptr&&thread1->IsAlive&&thread1-
>ThreadState!=ThreadState::Suspended)
                                   {
                                     thread1->Suspend();
                                   }
                                 else
                                   {
                                     if(thread1-
>ThreadState==ThreadState::Suspended){thread1->Resume();}
```

```
                                }
                        };
        public:   void stopthread(){
                                // Thread stoppen
                                if(thread1!=nullptr&&thread1->IsAlive){
                                if(thread1->ThreadState ==
ThreadState::Suspended){thread1->Resume();  }
                                thread1->Abort();
                        }}
        private: void paint(){
                                // Code, den ein neuer Thread startet
                                DateTime begin;
                                begin  = DateTime::Now;
                                linien = 0;
                                status = "Generierung gestartet";
                                paint(n,points);   // spezifischer Fraktal erzeugungs
Code
                                status = "Generierung abgeschlossen, benötigte Zeit: " +
(DateTime::Now - begin).Seconds + " sec, " +  progress()*100+ "% gezeichnet";
                        }
        public:   property bool ThreadIsAlive
                                        {
                                                // Information ob Thread am Leben ist
                                                bool get()
                                                {
                                                        if(thread1!=nullptr){return
thread1->IsAlive;}
                                                        else return 0;
                                                }
                        }
        protected:double point_distance(PointF A,PointF B){
                                // Gibt den Abstand zwischen a und b zurück
                                return Math::Sqrt(Math::Pow(A.X-B.X,2)+Math::Pow(A.Y-
B.Y,2));
                        }
        protected:double point_distance_sqr(PointF A,PointF B){
                                // Gibt das Quadrat des Abstands zwischen a und b zurück
(schneller als Abstand)
                               if(!abbr_distance){return 2;}
                               return (A.X-B.X)*(A.X-B.X)+(A.Y-B.Y)*(A.Y-B.Y);
                        }
        public: static String^ read_word(IO::StreamReader^ strReader)
                        {
                                // liest aus einer Datei ein einziges Wort aus
                                Char buffer;String^ string;
                                do
                                {
                                        buffer=strReader->Read();
                                } while(!(strReader->EndOfStream) && ( (buffer == ' ')
|| (buffer == '\r') || (buffer == '\n')));
                                while (!(strReader->EndOfStream) && !(buffer == ' ') &&
!(buffer == '\r') && !(buffer == '\n'))
                                {
                                        string = String::Concat(string,buffer);
                                        buffer = strReader->Read();
                                }
                                if(strReader->EndOfStream){string =
String::Concat(string,buffer);}
                                return string;
                        }

        public: property int MinPoints
                        {
                          // MinPoints = minimale Anzahl der Punkte ab der ein Fraktal
erstellt werden kann
                          // neededpoints = Anzahl der Punkte, ab der das Fraktal
erstellt wird, d.h. neededpoints >= Minpoints
                                int get(){ return minpoints;}
                        protected:     void set(int x) { minpoints = x; if
(neededPoints<MinPoints) {neededPoints=MinPoints;}}
                        }
        public:  virtual int check()=0; // berechnet wie viele Objekte gezeichnet werden
müssen
        public:  virtual void addpoint(PointF point){
                                // Fügt einen Punkt hinzu
                                if (neededPoints<MinPoints)
                                {
```

```
                                        throw gcnew System::Exception("Achtung, es werden
mindestens "+MinPoints+" Punkte benötigt für diese Fraktal");
                                }
                                if(points->Length>=this->neededPoints)
                                {
                                        n++;
                                }
                                else
                                {
                                        System::Array::Resize(points,points->Length+1);
                                        points[points->Length-2]=point;
                                }
                        }
        public:  virtual void addpoints(array<PointF>^ p_oints){
                                         // Fügt array von Punkten hinzu
                                                if (neededPoints<MinPoints)
                                                {
                                                        throw gcnew
System::Exception("Achtung, es werden mindestens "+MinPoints+" Punkte benötigt für diese
Fraktal");
                                                }
                                                for(int i=0;i<p_oints->Length;i++)
                                                {
                                                if(points->Length>=this->neededPoints)
                                                {
                                                        n++;
                                                        return;
                                                }
                                                else
                                                {

System::Array::Resize(points,points->Length+1);
                                                        points[points->Length-
2]=p_oints[i];
                                                }
                                                }
                                         }
        public:  virtual void subtractpoint()
                        {
                                // entfernt einen Punkt
                                if(points->Length>1)
                                {
                                        System::Array::Resize(points,points->Length-1);
                                }
                        }
        public:  virtual void drawpoints(Draw^g){
                                // bisher angegebene Punkte werden gezeichnet
                                if(points->Length>1&&n==0)
                                {
                                        g->DrawPolygon(gcnew Pen(front),points);
                                }
                        }
        public:  virtual String^ info()
                        {
                                String^ info;
                                info += name + ", " + Convert::ToString(n) + ".Iteration ,
";
                                info += Convert::ToString(check()) + " Objekte " ;
                                if(wachstum&&wachstum==int(wachstum)){info += ", Wachstum
ist "+Convert::ToString(wachstum)+"^n";}
                                return info;
                        }
        public:  double progress()
                        {
                                // Berechnet Fortschritt der Fraktal zeichnung
                                if(thread1->IsAlive){
                                int tmp = check();
                                status = linien + " Linien von " + tmp +" gezeichnet";
                                if(tmp){return double(linien) / tmp;}
                                else return 0;}
                        }
        public:  virtual array<Fraktal^>^ Load(IO::StreamReader^ stream){
                                // Lädt eine Liste von Fraktalen (Standardmäßig alle
Fraktale dieser dll)
                                array<Fraktal^>^ fraktal_list = gcnew array<Fraktal^>(0);
                                for(int i=0;i<=MAX_MODES;i++)
                                {
```

```
                                        Array::Resize(fraktal_list,fraktal_list-
>Length+1);
                                        fraktal_list[fraktal_list->Length] =
change_mode(i);
                                }
                        return fraktal_list;
                }
        protected: void write3(String^ buf)
                        {
                                // Schreibt einen Text
                                SolidBrush^ brush =gcnew SolidBrush(Color::Red);
                                Font^ font=gcnew Font(L"Arial", 10);
                                graphics->DrawString(buf,font,brush,PointF(270,80));
                        }
        public: float object_extend(){
                               // Berechnet den Durchmesser eines, des Polygon umgebendem,
Rechtecks
                               // nötig um abzuschätzen, ab wie vielen Iterationen bei
einem IFS die Objektgrösse < 1 ist
                                PointF min,max;
                                min=max=points[0];
                                for(int i=1;i<points->Length;i++){
                                        if(points[i].X<min.X){min.X=points[i].X;}
                                        if(points[i].Y<min.Y){min.Y=points[i].Y;}
                                        if(points[i].X>max.X){max.X=points[i].X;}
                                        if(points[i].Y>max.Y){max.Y=points[i].Y;}
                                }
                                return Math::Sqrt(Math::Pow(max.X-
min.X,2)+Math::Pow(max.Y-min.Y,2));
                        }
        public:  Fraktal^ change_mode(int setmode);
        property array<String^>^ListFractals{
                 // Listet die namen aller Fraktale dieser dll auf
                                array<String^>^get()
                                {  array<String^>^ s = gcnew array<String^>(MAX_MODES+1);
                                for(int i=0;i<=MAX_MODES;i++)
                                {
                                        Fraktal^f = change_mode(i);
                                        s[i]=f->name;
                                }
                                return s;
                                }
                        }
                        };


#include "CantorMenge.h"
#include "Farn.h"
#include "IFS.h"
#include "IFS_2.h"
#include "Kochkurve.h"
#include "L-System.h"
#include "Pytargorasbaum.h"
#include "Schneeflocke.h"
#include "SierpinskiDreieck.h"

inline Fraktal^ Fraktal::change_mode(int setmode){
        ////////////////////////////////////////////////////////////////////////////
        ///   Fraktal ändern
        ////////////////////////////////////////////////////////////////////////////
        Fraktal^f;
        if(setmode>MAX_MODES){throw gcnew System::Exception("MAX_MODE überschritten");}
        switch(setmode)
        {
        case 0:
                f = gcnew Kochkurve;
                break;
        case 1:
                f = gcnew Schneeflocke;
                break;
        case 2:
                f= gcnew Cantor;
                break;
        case 3:
                f= gcnew Sierpinski;
                break;
        case 4:
                f= gcnew Farn;
```

```
                break;
        case 5:
                f= gcnew Pytargorasbaum;
                break;
        case 6:
                f= gcnew LSystem;
                break;
        case 7:
                f= gcnew IFS;
                break;
        case 8:
                f= gcnew IFS_2;
                dynamic_cast<IFS_2^>(f)->addrule(0.3333,0,0,0);
                dynamic_cast<IFS_2^>(f)->addrule(0.3333,60,1.0/3.0,0);
                dynamic_cast<IFS_2^>(f)->addrule(0.3333,-60,0.5,-0.288);
                dynamic_cast<IFS_2^>(f)->addrule(0.3333,0,2.0/3.0,0);
                break;
        default:
                throw gcnew Exception("Keinem Fraktal wurde die nummer "+setmode+"
zugeordnet");
        }
return f;
}
};
```

3.1.2 CANTORMENGE.H

```
public ref class Cantor : public Fraktal
{
public:
        virtual int check() override {return Math::Pow(2.0,n);}
        Cantor(){MinPoints = 2;name="Cantor Menge";}
        virtual void paint(int n,array<PointF>^ points) override
        {  array<PointF>^ tmppoints = gcnew array<PointF>(2);
        // Generator
        if(n==0)
        {
                Pen^ pen = gcnew Pen(front, 10);
                graphics->DrawPolygon(pen, points);
                this->linien++;
                return;
        }
        //rekursiver aufruf der funktion
        double i,j;
        if(random) {i=rand->NextDouble(); // wenn random aktiviert ist wird die Strecke
nicht in drei gleich grosse Teile geteilt , sondern mit zufälligem teilverhältnis
                   j=rand->NextDouble();  // wenn j= 0.3 , dann geht die 1.Strecke von
Punkt 0 bis zu 0.3 * 01 , und wenn i=0.3 dann geht die zweite Linie von 0.3 *01 los, und
endet bei 1
        if(i>j){double tmp;tmp=i;i=j;j=tmp;}}
        else {i = 0.333;j = 0.666;} // wenn random = 0 wird die Strecke in drei gleich
große Teile geteilt
        //Erzeugung des ersten Teilstücks , von a bis (i=0.33)*ab
        tmppoints[0]   = points[0];                                    // Anfangspunkt
des ersten Teilstücks = a der Originallinie
        tmppoints[1].X = points[0].X + i * (points[1].X-points[0].X);
        tmppoints[1].Y = points[0].Y + i * (points[1].Y-points[0].Y);
        paint(n-1,tmppoints);
        //
        //
        //Erzeugung des zweites Teilstücks , von (j=0.66)*ab bis b
        tmppoints[0].X = points[0].X + j * (points[1].X-points[0].X);
        tmppoints[0].Y = points[0].Y + j * (points[1].Y-points[0].Y);
        tmppoints[1]   = points[1];
        paint(n-1,tmppoints);
        //
        //
        return ;
        }

};
```

3.1.3 FARN.H

```
public ref class Farn : public Fraktal{
        int counter;
public:
        Farn(){MinPoints = 2;name="Farn";}
```

```
        virtual int check() override {return Math::Pow(3.0,n+1)/2-3.0/2.0+1;}
        virtual void paint(int n,array<PointF>^ points) override
                //v3,v4==d1,d2
        {
                if(counter<101&&counter>=0)counter++;else{counter = 0;}
                Pen^  pen = gcnew Pen(System::Drawing::Color::FromArgb(30,counter,30),3);
                graphics->DrawPolygon(pen, points);
                this->linien++;
                if(n>0&&point_distance_sqr(points[0],points[1])>1)
                {
                        array<PointF>^ Tpoints = gcnew array<PointF>(points->Length);
                        Matrix ma,mb,mc;
                        ma.Translate( - points[0].X , - points[0].Y ,
MatrixOrder::Append);   //Punkte auf Ursprung verschieben
                        mb.Translate( - points[0].X , - points[0].Y ,
MatrixOrder::Append);
                        mc.Translate( - points[0].X , - points[0].Y ,
MatrixOrder::Append);
                        if(!random){
                                ma.Rotate( -60 , MatrixOrder::Append);              //die
Punkte werden gedreht
                                mb.Rotate( -10 , MatrixOrder::Append);
                                mc.Rotate(  50 , MatrixOrder::Append);}
                        else{
                                ma.Rotate( -(rand->Next(120)) , MatrixOrder::Append);
//die Punkte werden gedreht
                                mb.Rotate( -(rand->Next(20))  , MatrixOrder::Append);
                                mc.Rotate(   rand->Next(100)  , MatrixOrder::Append);}
                        ma.Scale( 0.5 , 0.5 , MatrixOrder::Append);
                        mb.Scale( 0.8 , 0.8 , MatrixOrder::Append);
                        mc.Scale( 0.5 , 0.5 , MatrixOrder::Append);
                        ma.Translate(points[1].X , points[1].Y,MatrixOrder::Append);
//Punkte zurück verschieben
                        mb.Translate(points[1].X , points[1].Y,MatrixOrder::Append);
                        mc.Translate(points[1].X , points[1].Y,MatrixOrder::Append);

                        for(int i=0;i<points->Length;i++) // Kopie der Punkte wird
angefertigt
                        {
                                Tpoints[i] =  points[i];
                        }
                        ma.TransformPoints(Tpoints);      // Kopie der Punkte wird mit der
Matrix ma transformiert
                        paint(n-1,Tpoints);               // selbstaufruf auf mit den
transformiertem Punkten

                        for(int i=0;i<points->Length;i++) // Kopie der Punkte wird
angefertigt
                        {
                                Tpoints[i] =  points[i];
                        }
                        mb.TransformPoints(Tpoints);      // Kopie der Punkte wird mit der
Matrix mb transformiert
                        paint(n-1,Tpoints);               // selbstaufruf auf mit den
transformiertem Punkten

                        for(int i=0;i<points->Length;i++) // Kopie der Punkte wird
angefertigt
                        {
                                Tpoints[i] =  points[i];
                        }
                        mc.TransformPoints(Tpoints);      // Kopie der Punkte wird mit der
Matrix mc transformiert
                        paint(n-1,Tpoints);               // selbstaufruf auf mit den
transformiertem Punkten
                }
                return ;
        }

};
```

3.1.4 IFS.H

```
public ref class IFS : public Fraktal{
private: Generic::Stack<Matrix^>^ stmat;
public:
        array<Matrix^>^ matrizen;
        array<MorphMatrix^>^ mm;
```

```
        bool morph;
        IFS(){morph = 0;MinPoints = 2;neededPoints=3;name="IFS-Fraktal (Matrix)";matrizen
=  gcnew array<Matrix^>(0);}
        virtual int check() override {return Math::Pow(double(matrizen->Length),n);}
        void add_matrix(Matrix^ m){
                Array::Resize(matrizen,matrizen->Length+1);
                matrizen[matrizen->Length-1] = m;
        }
        void reset(){matrizen = gcnew  array<Matrix^>(0);}
        void set_matrix(Matrix^ m,int n){ // Matrix[n] durch Matrix m ersetzen
                if(matrizen->Length>=n)
                {matrizen[n] = m;}else throw gcnew System::Exception("gewählte Matrix
existiert nicht!");
        }
        void del_matrix(){  // die letzte Matrix löschen
                if(matrizen->Length>0)
                        Array::Resize(matrizen,matrizen->Length-1);
        }
        void del_matrix(int n){ // Matrix[n] löschen
                if(n<matrizen->Length){
                        if(n<matrizen->Length-1){
                                for (int i=n;i<matrizen->Length+1;i++)
                                {
                                        matrizen[n]=matrizen[n+1];
                                }}
                        Array::Resize(matrizen,matrizen->Length-1);
                }}
        array<String^>^ list_matrices(){ // alle Matrizen in einem String array auflisten
(für jede Matrix wird der array um eins erhöht und "Matrix [zahl] hineingeschrieben")
                array<String^>^s = gcnew array<String^>(0);
                for(int i=0;i<matrizen->Length;i++){
                        Array::Resize(s,s->Length+1);
                        s[i]="Matrix "+i;
                }
                return s;
        }
        static String^ read_word(IO::StreamReader^ strReader)
        {
                Char buffer;String^ string;
                do
                {
                        buffer=strReader->Read();
                } while(!(strReader->EndOfStream) && ( (buffer == ' ')  || (buffer ==
'\r') || (buffer == '\n')));
                while (!(strReader->EndOfStream) && !(buffer == ' ') && !(buffer == '\r')
&& !(buffer == '\n'))
                {
                        string = String::Concat(string,buffer);
                        buffer = strReader->Read();
                }
                if(strReader->EndOfStream){string = String::Concat(string,buffer);}
                return string;
        }
        virtual array<Fraktal^>^Load(System::IO::StreamReader^ reader) override{
                array<IFS^>^ ifs_array = gcnew array<IFS^>(0);
                int matrixIndex;
                String^ string;
                double m11,m12,m21,m22,m_x,m_y;
                while(!(reader->EndOfStream)){
                        string = read_word(reader);
                        if(string=="name"){
                                Array::Resize(ifs_array,ifs_array->Length+1);
                                ifs_array[ifs_array->Length-1] = gcnew IFS;
                                ifs_array[ifs_array->Length-1]->name = read_word(reader);
                        }

        if(string=="matrix"){matrixIndex=Convert::ToInt32(read_word(reader));ifs_array[if
s_array->Length-1]->add_matrix(gcnew Matrix);}
                        if(string==">"){ifs_array[ifs_array->Length-1]-
>matrizen[matrixIndex] = gcnew Matrix(m11,m12,m21,m22,m_x,m_y);
                        }
                        if(string=="//"){reader->ReadLine();}
                        if(string=="m11")
                        {
                                m11 = Convert::ToDouble(read_word(reader));
                        }
                        if(string=="m12")
                        {
```

```
                        m12 = Convert::ToDouble(read_word(reader));
                    }
                    if(string=="m21")
                    {
                        m21 = Convert::ToDouble(read_word(reader));
                    }
                    if(string=="m22")
                    {
                        m22 = Convert::ToDouble(read_word(reader));
                    }
                    if(string=="m_x"){
                        m_x = Convert::ToDouble(read_word(reader));
                    }
                    if(string=="m_y"){
                        m_y = Convert::ToDouble(read_word(reader));
                    }
                }
                return ifs_array;
        }
        virtual void paint(int n,array<PointF>^ points) override{

                stmat = gcnew Generic::Stack<Matrix^>; // Der Stack wird erstellt
                stmat->Push(gcnew Matrix);             // eine Identitätsmatrix wird
hinzugefügt
                write3("\n Objekt Durchmesser:" + object_extend());
                delete[] mm;
                mm = gcnew array<MorphMatrix^>(0);
                graphics->graphics->TranslateTransform(points[0].X,points[0].Y);
                // "Papier" auf das geschrieben wird wird verschoben (wichtig, da die
Punkte an den Ursprung versetzt
                //  werden müssen, Drehung um den Ursprung ...)
                paint_ifs(n,points);    // Iterativer Zeichenvorgang
                if(morph){
                    bool allthreadsready=0;graphics->graphics->TranslateTransform(-250,-
250);
                      do{
                            allthreadsready=1;
                               graphics->Clear(Color::Transparent);
                            for(int i=0;i<mm->Length;i++)
                               {
                                      if(mm[i]->IsAlive){  // Wenn Thread von
Morphingobjekt noch läuft....
                                         if(mm[i]->bitmap!=nullptr){graphics-
>DrawImage(mm[i]->bitmap);}  // Bild von Morphingthread wird gezeichnet
                                         allthreadsready=0;
                                   }
                               }
                            thread1->Sleep(50);
                          }while(!allthreadsready);
                    graphics->graphics->TranslateTransform(250,250);
                       morph = 0;stmat = gcnew Generic::Stack<Matrix^>;stmat->Push(gcnew
Matrix);
                       paint_ifs(n,points);
                       graphics->graphics->ResetTransform();
                       morph = 1;
                }

        }
        void paint_ifs(int n,array<PointF>^ points){
                /// Wenn n = 0 wird die Matrix auf die Punkte angewandt und diese werden
gezeichnet, falls nicht
                /// siehe weiter unten . .
                Matrix^ m;
                if(n){
                        ///  Die übergebene Matrix wird mit jeder der Matrizen die das IFS
                        ///  erzeugen Multipliziert, dieser Vorgang wird mit den jeweils
Resultierenden Matrizen
                        ///  wiederholt (Selbstaufruf,die Matrizen werden per Stack
übergeben)             if(matrizen->Length == 0){/*throw gcnew Exception("keine
Matrix");*/MessageBox::Show("keine Matrix
angegeben","Achtung",MessageBoxButtons::OK,MessageBoxIcon::Hand);n=0;}
                        for (int i=0;i<matrizen->Length;i++)
                        {
                                try
                                {
                                    m = stmat->Pop();  // letzte Matrix wird aus dem Stack
geholt
```

```
                                stmat->Push(m->Clone()); // dieselbe Matrix wird wieder
zurück auf den Stack gelegt
                                m->Multiply(matrizen[i],MatrixOrder::Append); // die
geholte Matrix wird mit der i.Matrix aus den Matrizen zur IFS erzeugung Multipliziert
                                stmat->Push(m->Clone()); // die bearbeitete Matrix wird
auf den Stapel gelegt
                            }catch (System::Exception^){}
                            paint_ifs(n-1,points); // Selbstaufruf , n:=n-1
                        }
                        if(stmat->Count>0){stmat->Pop();}
                }
                else{ // n == 0
                        // Ursprung der Punkte wird an points[0] gesetzt , Punkte werden
durch Matrix Tranformiert                   array<PointF>^ copy;copy =
(array<PointF>^)(points->Clone());
                        Matrix matrix;matrix.Translate(-points[0].X,-
points[0].Y);matrix.TransformPoints(copy);
                        m = stmat->Pop();
                        m = gcnew Matrix(m->Elements[0],
                                        m->Elements[1],
                                        m->Elements[2],
                                        m->Elements[3],
                                        point_distance(points[0],points[1])*m->Elements[4],
                                        point_distance(points[0],points[1])*m->Elements[5]);
                        // x,y verschiebung wird auf die Linienlänge angepasst, um die
gewünschte Grösse des Fraktals zu
                        // erzeugen(sonst würde das Fraktal bei einer Einstellung immer
die selbe grösse haben)
                        if(morph){
                                // Wenn morph = 1 werden die Punkte, statt sofort
gezeichnet zu werden, erst von der
                                // Identitätsmatrix in die Transformierten Positionen
umgewandelt, so wird der Entstehungsprozess
                                // besser ersichtlich
                                Array::Resize(mm,mm->Length+1);
                                mm[mm->Length-1] = gcnew MorphMatrix(gcnew
Matrix,m,copy,4);
                        }
                        else{
                                m->TransformPoints(copy);
                                graphics->DrawPolygon(gcnew Pen(front),copy);
                        }
                        linien++;
                }
        }

};
```

3.1.5 IFS2.H

```
public ref class IFS_2 : public Fraktal
{
public:
        array<double>^ streckung;
        array<double>^ drehung;
        array<PointF>^ verschiebung;
        float d;
public:
  IFS_2(){
          name="IFS-Fraktal";MinPoints=2;neededPoints=2;
          streckung    = gcnew array<double>(0);
          drehung      = gcnew array<double>(0);
          verschiebung = gcnew array<PointF>(0);
  }
  float max_scale(){
          float max;
          for (int i=0;i<streckung->Length;i++)
          {if(streckung[i]>max){max=streckung[i];}}
          return max;
  }
  virtual void paint(int n,array<PointF>^ points)override
  {
          try
          {
                  // Abschätzung wie oft iteriert werden muss,
          // indem der Durchmesser eines, des Polygon umgebendem, Rechtecks genommen
wird,
          // und dann in die Formel (länge)*(grösste Streckung)^n = 1
```

```
          // ==> n = - ln(länge)/ln(grösste Streckung) eingesetzt wird
          // (eine beliebige Strecke hat nach einer Iteration im IFS die maximal die
Länge: Urpsrungslänge*die grösste Streckung)
          // Nach 1 wird gesucht, da es sinnlos ist nocheinmal zu iterieren, wenn die
Ausdehnug des Objekts < 1 Pixel ist
          write3("\nGrösstes Objekt < 1 bei n: "+-
Math::Log(object_extend())/Math::Log(max_scale()));
        // Eine Kopie der Punkte wird angefertigt, die Kopie wird an den Ursprung
verschoben  array<PointF>^ cpy = gcnew array<PointF>(points->Length);points-
>CopyTo(cpy,0);
                // Die Punkte werden an den Urpsrung verschoben  Matrix m;m.Translate(-
points[0].X,-points[0].Y);m.TransformPoints(cpy);

          d = point_distance(points[0],points[1]);
          // Iterativer Algorithmus wird gestartet
          punktberechnen(n,cpy);
          }

          catch (Exception^ e)
          {

          }

  }
        array<PointF>^ transformpoints(double k,double winkel,double x_v,double
y_v,array<PointF>^points){
                // Drehstreckung und verschiebung der Punkte
                array<PointF>^ newpoints = gcnew array<PointF>(0);
                for(int i=0;i<points->Length;i++){
                  Array::Resize(newpoints,newpoints->Length+1);

newpoints[i]=PointF(k*Math::Cos(0.0175*winkel)*points[i].X+k*Math::Sin(0.0175*winkel)*po
ints[i].Y+d*x_v,-
k*Math::Sin(0.0175*winkel)*points[i].X+k*Math::Cos(0.0175*winkel)*points[i].Y+d*y_v);
                }
                return newpoints;
        }
        void punktberechnen(int n,array<PointF>^ p){
                if(n){
// alle Punkte werden jeweils für alle Regeln transformiert, diese Transformierten
Punkte werden jeweils zum Selbstaufruf eingesetzt bis n=0
                        for (int i=0;i<streckung->Length;i++)
                        {
                         array<PointF>^c_points;
 /// Transformpoints führt die i-te Transformation für die punkte p durch
                         c_points =
transformpoints(streckung[i],drehung[i],verschiebung[i].X,verschiebung[i].Y,p);
/// Diese sind ein Parameter für den Selbstaufruf
                 punktberechnen(n-1,c_points);
                        }
                }
                else {
// n=0    , Punkte werden um den Vektor, der die Punkte vorher an den Ursprung
verschoben hat  zurückverschoben, danach wird das Polygon gezeichnet
        Matrix^ m2=gcnew Matrix;m2->Translate(points[0].X,points[0].Y);
                        m2->TransformPoints(p);
                        graphics->DrawPolygon(gcnew Pen(front),p);linien++;
                    }
        }
        void addrule(double k,double winkel,double x_v,double y_v){
                // eine Transformationsregel wird hinzugefügt
                Array::Resize(streckung,streckung->Length+1);
                Array::Resize(drehung,drehung->Length+1);
                Array::Resize(verschiebung,verschiebung->Length+1);
         streckung[streckung->Length-1] = k;
                drehung[drehung->Length-1] = winkel;
                verschiebung[verschiebung->Length-1] = PointF(x_v,y_v);
        }
        void delrule(){
                // eine Transformationsregel wird entfernt
                if(streckung->Length>0){
                Array::Resize(streckung,streckung->Length-1);
                Array::Resize(drehung,drehung->Length-1);
                Array::Resize(verschiebung,verschiebung->Length-1);}
        }
        property int Rules{
                int get(){
                        return streckung->Length;
```

```
		}
	}
	virtual int check()override{return Math::Pow(double(streckung->Length),n);}
	array<String^>^ list(){
		// Listet alle Transformationsregeln in einem String array auf
		array<String^>^ s = gcnew array<String^>(0);
		for(int i=0;i<streckung->Length;i++)
		{
			Array::Resize(s,s->Length+1);
			s[i]="Regel " + i;
		}
		return s;
	}
};
```

3.1.6 KOCHKURVE.H

```
public ref class Kochkurve : public Fraktal
{
public:
	Kochkurve(){MinPoints = 2;name="Kochkurve";laenge=0;}
	virtual void addpoint(PointF point) override{
		if(points->Length>=this->neededPoints){n++;PointF tmp;tmp =
points[0];points[0]=points[1];points[1]=tmp;}
		else{
			System::Array::Resize(points,points->Length+1);
			points[points->Length-2]=point;
		}
	}

	virtual int check() override {return Math::Pow(4.0,n);}
	virtual void paint(int n,array<PointF>^ points) override
	{
		//Generator //
		array<PointF>^ Tpoints = gcnew array<PointF>(points->Length);
		Pen^  pen = gcnew Pen(front);
		if(n==0||point_distance_sqr(points[0],points[1])<=1) // Wenn n=0 oder der
Abstand der beiden Punkte > 1 Pixel ist wird gezeichnet
		{
			graphics->DrawPolygon(pen, points);
			this->linien++;
			return ;}
		if(random&&int(rand->Next(2))>0) // Wenn zufall aktiviert ist entscheidet
dies ob die "Zacke" oben oder unten erscheint (wenn unten wird Punkt 1 und Punkt 2
vertauscht)
	{double tmp;tmp = points[0].X;points[0].X = points[1].X;points[1].X = tmp;
		tmp = points[0].Y;points[0].Y = points[1].Y;points[1].Y = tmp;}
		Matrix ma,mb,mc,md; /// 4 Matrizen für 4 Transformationen ...
		ma.Translate(-points[0].X , -points[0].Y,MatrixOrder::Append);   //Punkte
auf Ursprung projezieren
		mb.Translate(-points[0].X , -points[0].Y,MatrixOrder::Append);
		mc.Translate(-points[0].X , -points[0].Y,MatrixOrder::Append);
		md.Translate(-points[0].X , -points[0].Y,MatrixOrder::Append);
		/*ma.Rotate(0,   MatrixOrder::Append);*/
//die Punkte werden gedreht
		mb.Rotate( 60,MatrixOrder::Append);
		mc.Rotate( 120,MatrixOrder::Append);
		/*md.Rotate(0,   MatrixOrder::Append);*/
		ma.Scale(0.333,0.333,MatrixOrder::Append);                        //die
entstehenden Linien sind 1/3 der Ausgangslinie
		mb.Scale(0.333,0.333,MatrixOrder::Append);
		mc.Scale(0.333,0.333,MatrixOrder::Append);
		md.Scale(0.333,0.333,MatrixOrder::Append);
		ma.Translate(0.00 , 0.00);                                        //die
Punkte werden verschoben
		mb.Translate(0.333*(points[1].X-points[0].X) , 0.333*(points[1].Y-
points[0].Y),MatrixOrder::Append);
		mc.Translate(0.666*(points[1].X-points[0].X) , 0.666*(points[1].Y-
points[0].Y),MatrixOrder::Append);
		md.Translate(0.666*(points[1].X-points[0].X) , 0.666*(points[1].Y-
points[0].Y),MatrixOrder::Append);
		ma.Translate(points[0].X , points[0].Y,MatrixOrder::Append);     //Punkte
werden vom Ursprung an Position zurück verschoben
		mb.Translate(points[0].X , points[0].Y,MatrixOrder::Append);
		mc.Translate(points[0].X , points[0].Y,MatrixOrder::Append);
		md.Translate(points[0].X , points[0].Y,MatrixOrder::Append);

	////////////////////////////////////////////////////////////////////////
```

```
                // 1.Kopie
                points->CopyTo(Tpoints,0);
                ma.TransformPoints(Tpoints);
                /*laenge+=sqrt(pow(Tpoints[0].X-Tpoints[1].X,2)+pow(Tpoints[0].Y-
Tpoints[1].Y,2));*/
                paint(n-1,Tpoints);

        ////////////////////////////////////////////////////////////////////////
                // 2.Kopie
                points->CopyTo(Tpoints,0);
                mb.TransformPoints(Tpoints);
                /*laenge+=sqrt(pow(Tpoints[0].X-Tpoints[1].X,2)+pow(Tpoints[0].Y-
Tpoints[1].Y,2));            */
                paint(n-1,Tpoints);

        ////////////////////////////////////////////////////////////////////////
                // 3.Kopie ("Zacke" erscheint wegen Drehung > 90° auf der falschen Seite,
deshalb werden alle Punkte vertauscht)
        ////////////////////////////////////////////////////////////////////////
                for(int i=0;i<points->Length;i++)
                {
                        Tpoints[i] =  points[points->Length-i-1];
                }
                Tpoints[0] = points[1];
                Tpoints[1] = points[0];
                mc.TransformPoints(Tpoints);
                /*laenge+=sqrt(pow(Tpoints[0].X-Tpoints[1].X,2)+pow(Tpoints[0].Y-
Tpoints[1].Y,2));*/
                paint(n-1,Tpoints);

        ////////////////////////////////////////////////////////////////////////
                // 4.Kopie
                points->CopyTo(Tpoints,0);
                md.TransformPoints(Tpoints);
                /*laenge+=pow(double(pow(double(Tpoints[0].X-
Tpoints[1].X),double(2.0))+pow(double(Tpoints[0].Y-
Tpoints[1].Y),double(2.0))),double(0.5));*/
                paint(n-1,Tpoints);
                return ;

        }

};
```

3.1.7 L-SYSTEM.H

```
public ref class stack
{  /// Ein Stack ist ein Stapelspeicher, das heißt , wenn man Informationen speichert
werden diese auf einen Stapel gelegt , und wenn man neue Informationen abruft werden die
obersten Informationen gegeben und der Stapel wird um eins verkleinert
        /// bei manchen L-Systemen zum Speichern von Position und Winkel
public:
        int counter;
        array<PointF>^v;  // Speichert die aktuelle Position des L-Systems
        array<double>^w;   // Speichert den aktuell eingestellten Winkel
        stack(){w = gcnew array<double>(100);v = gcnew array<PointF>(100);counter=0;}
        void push(PointF V,double
W){w[counter]=W;v[counter]=V;if(counter<100)counter++;else
System::Windows::Forms::MessageBox::Show("Stack ist voll","Fehler");} /// Daten werden
auf den Stapel gelegt  (für das Programm werden die aktuelle Position der Schildkröte
und der Winkel gespeichert)
        PointF popv(){if(counter>0){counter--;return v[counter];}else
System::Windows::Forms::MessageBox::Show("Stack Fehler (Stack leer)","Fehler");return
PointF(0,0);}  /// Position wird zurückgegeben und der Stapel wird um 1 verkleinert...
        double popw(){return w[counter];} /// Winkel wird zurückgegeben
        void reset(){counter = 0;}
};
public ref class LSystem : public Fraktal
{
        // Turtle Graphik
        //F : Zeichne eine Linie mit einer vorgegebenen Länge in Richtung d
        //! : Bewege dich um eine vorgegebene Länge in Richtung d ohne zu zeichnen
        //+ : Drehe dich um den Winkel a gegen den UhrzeigerMath::Sinn.
        //- : Drehe dich um den Winkel a im UhrzeigerMath::Sinn
        //[ : Merke dir die aktuelle Position und Richtung
        //] : Gehe zurück zur zuletzt gemerkten Position und Richtung
public:
```

```
        /// Regeln die für ein L-System benötigt werden
        /// Axiom  : Startzeichenkette
        /// Symbol : Zeichen das ersetzt wird
        /// substitution : Zeichenkette durch die das Symbol ersetzt wird
        /// Winkel : im moment eingestellter Winkel
        array<String^>^substitution;
        String^axiom,^gesamtkette;
        array<Char>^ symbol;
        double add_winkel,winkel,startwinkel,d,x,y;
        PointF pos,lastpos,startpos; // aktuelle Position + alte Position + Startposition
        stack Stack;        // Stack für Positionen + Winkel
        bool write_string;
        LSystem()
        {
                name="L-System";filetyp="lsystem";
                MinPoints = 0;
                neededPoints = 2;
                write_string = 0;
                startpos = PointF(200,200);
                x=500;y=3;
                add_winkel = 90;
                reset(0);
        }
        void reset(int n)
        {
                pos = lastpos = startpos;
                winkel = startwinkel;
                /// Abstand korrigieren (bei jedem n anders)
                d =  x/(Math::Pow(y,n));
                gesamtkette="";
        }
        void reset_rules(){
                substitution = gcnew  array<String^>(0);
                symbol       = gcnew  array<Char>(0);
        }
        void addrule(String^ rule,Char Symbol){
                /// Regel hinzufügen, Parameter sind das Symbol dass ersetzt werden soll
                /// und die Zeichenkette durch die das Symbol ersetzt werden soll
                Array::Resize(substitution,substitution->Length+1);
                substitution[substitution->Length-1] = rule;
                Array::Resize(symbol,symbol->Length+1);
                symbol[symbol->Length-1] = Symbol;
        }
        int checksymbol(char s)
        {
                /// Kontrolle ob der übergebene Buchstabe durch eine Zeichenkette ersetzt
werden muss
                for(int i=0;i<symbol->Length;i++)
                {if(s==symbol[i])return i;}
                return -1;
                /// wenn nein wird -1 zurückgegeben , ansonsten die Position der Regel
angegeben (auch 0 möglich,deshalb -1 falls keine Regel)
        }
        virtual int check() override {
                return 0;
        }
        // Generator des L-Systems
        void paint_lsystem(int n,array<PointF>^ points,Draw^ graphics,String^ axiom)
        {

                String^ lsystem3; /// hier wird die ersetzte Version des L-Systems
gespeichert
                if(!n) /// wenn n = 0 axiom an Interpreter schicken (Graphische Ausgabe)
                {
                        for(int i = 0;i<axiom->Length;i++)
                        {
                                Interpreter(axiom[i],graphics); /// alle Buchstaben müssen
einzeln an den Interpreter geschickt werden um Fehler zu vermeiden
                        }
            if(write_string)gesamtkette+=axiom;   /// Damit lässt sich das gesamt
ersetzte L-System speichern , ist aber Standardmässig ausgeschaltet , da es sehr viel
Speicher verbrauchen würde bei vielfacher Itertion
                        return ;
                }
                for(int i = 0; i<axiom->Length;i++) /// alle Symbole der Variablen axiom
werden untersucht ob sie durch eine Zeichenkette ersetzt werden müssen
                {
```

```
                        int check = checksymbol(axiom[i]); /// In der Variable check wird
gespeichert ob das Symbol ersetzt werden muss , und wenn ja an welcher Position es sich
befindet
                        if(!(check==-1)) // wenn check ungleich -1 dann ist axiom[i] es
ein Symbol das ersetzt werden muss
                        {
                                /// ersetze Zeichen nach der Regel

                                lsystem3 = String::Concat(lsystem3,substitution[check]);
/// hänge die ersetzung an die Zeichenkette lsystem3 an.
                        }
                        else
                        {
                                /// wenn das Zeichen aus axiom nicht ersetzt werden muss
wird es einfach an Lsystem3 angefügt
                                lsystem3=String::Concat(lsystem3,axiom[i]);
                        }
                }

                /// lsystem3 verarbeiten
                /// lsystem3 wird durchsucht nach einem Symbol das ersetzt werden muss,
wenn dies der Fall ist ruft die Funktion sich selbst auf mit dem Symbol das ersetzt wird
als axiom
                for(int i = 0; i<lsystem3->Length;i++)
                {
                        int check = checksymbol(lsystem3[i]);
                        if(!(check==-1))
                        {       // ein zu ersetzendes Symbol gefunden
                                axiom = Convert::ToString(symbol[check]);
                         paint_lsystem(n-1,points,graphics,axiom);  /// selbstaufruf mit Symbol
als axiom
                        }
                        else
                        {
                                Interpreter(lsystem3[i],graphics);  /// Zeichen wird an den
Interpreter geschickt
                         if(write_string)gesamtkette+=lsystem3[i];
                        }
                }
                return;

        }
        virtual void paint(int n,array<PointF>^ points) override
        {

                if(points->Length>1){
                        startwinkel = Math::Atan2((points[1].Y-
points[0].Y)*PI/180,(points[1].X-points[0].X)*PI/180)*180/PI; // Punkte werden in
Polarkoordinaten umgerechnet
                        x = Math::Sqrt(Math::Pow(points[1].X-
points[0].X,2)+Math::Pow(points[1].Y-points[0].Y,2));
                        startpos = points[0]; // Schrittlänge ist länge der Linie zwischen
den ersten beiden angegebenen Punkten
                }
                reset(n); // alle Werte werden auf Startposition gesetzt (womöglich vom
letzten aufruf noch verändert)
                paint_lsystem(n,points,graphics,axiom);
        }
        void Interpreter(char c,Draw^ graphics)
        {  /// Hier werden die Symbole in Befehle umgewandelt und ausgeführt
                /// Funktion der Symbole -> siehe "Turtle Grafik"
                switch(c)
                {
                case '!':
                        nopaint();
                        break;
                case '+':
                        turn1();
                        break;
                case '-':
                        turn2();
                        break;
                case '[':
                        stackpush();
                        break;
                case ']':
                        stackpop();
                        break;
```

```
                case '|':
                        winkel+=180;
                default:
                        /// Check ob Symbol einem Symbol aus einer Regel entspricht
                        if(!(checksymbol(c)==-1))
                        {forward(graphics);}
                        break;
                }
        }
        /// Turtle Grafik
        void forward(Draw^ graphics)
        {
                /// Der Punkt bis zu dem die Linie gezeichnet wird wird anhand des
eingestellten Winkels und der länge ausgerechnet
                Pen^ pen = gcnew Pen(front);
                nopaint();
                graphics->DrawLine(pen,lastpos,pos);
                this->linien++;/**/
        }
        void nopaint()
        {
                lastpos.X=pos.X;
                lastpos.Y=pos.Y;
                pos.Y+=Math::Sin(winkel*PI/180)*d;
                pos.X+=Math::Cos(winkel*PI/180)*d;
        }
        void turn1() /// +
        {
                winkel-=add_winkel;  //Achtung + und - müssen vertauscht werden, da auf
der Abbildungsfläche x=y=0 am oberen linken Rand ist
                if(random){winkel+= rand->Next(-10,10);}
        }
        void turn2() /// -
        {
                winkel+=add_winkel;  //Achtung + und - müssen vertauscht werden, da auf
der Abbildungsfläche x=y=0 am oberen linken Rand ist
                if(random){winkel+= rand->Next(-10,10);}
        }
        void stackpush() /// [ (Winkel und Position auf Stack legen)
        {Stack.push(pos,winkel);}
        void stackpop()  /// ] (Winkel und Position von Stack holen)
        {pos=Stack.popv();winkel=Stack.popw();}
        virtual array<Fraktal^>^ Load(IO::StreamReader^ reader) override{
                array<LSystem^>^ l = gcnew array<LSystem^>(0);
                String^ string;int i=-1;
                while(!(reader->EndOfStream)){
                        string = read_word(reader);
                        if(string=="angle")
                        {
                                l[i]->add_winkel = Convert::ToDouble(read_word(reader));
                        }
                        if(string=="axiom")
                        {
                                l[i]->axiom = read_word(reader);
                        }
                        if(string=="name")
                        {
                                //throw gcnew System::Exception("Name des L-Systems ändern
derzeit nicht unterstützt - Dateifehler");
                                i++;
                                Array::Resize(l,i+1);l[i]=gcnew LSystem;
                                l[i]->name = read_word(reader);
                        }
                        if(string=="linienlaenge_x"){l[i]->x =
Convert::ToDouble(read_word(reader));}
                        if(string=="linienlaenge_y"){l[i]->y =
Convert::ToDouble(read_word(reader));}
                        if(string=="startpos"){l[i]->startpos =
PointF(Convert::ToDouble(read_word(reader)),Convert::ToDouble(read_word(reader)));}
                        if(string=="drehung"){l[i]->startwinkel =
Convert::ToDouble(read_word(reader));}
                        if(string=="rules")
                        {
                                for(int j=0;!(reader->EndOfStream)&&j>=0;j++)
                                {
                                        String^ str = read_word(reader);
                                        if(str=="!rules"){j=-2;}else{
                                        System::Array::Resize(l[i]->substitution,j+1);
```

```
                                        System::Array::Resize(l[i]->symbol,j+1);
                                        l[i]->symbol[j]= str[0];
                                        l[i]->substitution[j]= str->Substring(2);}
                                }
                        }
                }
                 return l;
        }
};
```

3.1.8 *PYTHAGORASBAUM.H*

```
public ref class Pythagorasbaum : public Fraktal{
        double winkel;
public:
        Pythagorasbaum(){MinPoints = 4;name="Pythagorasbaum";}
        virtual int check() override {return points->Length*(Math::Pow(2.0,n+1)-1);}
        virtual void paint(int n,array<PointF>^ points) override
                //v3,v4==d1,d2
        {
                Pen^  pen = gcnew Pen(front);
                graphics->DrawPolygon(pen, points);
                linien+=points->Length;
                if(random) winkel = rand->Next(90);else winkel= 45;
                if(n>0&&point_distance_sqr(points[0],points[1])>1)
                {
                        array<PointF>^ Tpoints = gcnew array<PointF>(points->Length);
                        Matrix ma,mb,mc,md;
                        ma.Translate( - points[0].X , - points[0].Y ,
MatrixOrder::Append);   //Punkte auf Ursprung projezieren
                        mb.Translate( - points[1].X , - points[1].Y ,
MatrixOrder::Append);
                        ma.Rotate(     -winkel , MatrixOrder::Append);              //die
Punkte werden gedreht
                        mb.Rotate(   90-winkel , MatrixOrder::Append);              // da
ein Winkel = winkel, der andere 90° muss der letzte 90-winkel sein
                        double verhaeltnis = Math::Sin((90.0-winkel)*PI/180);  // Je
grösser das eine Viereck, desto kleiner muss das andere sein
                        ma.Scale(verhaeltnis, verhaeltnis, MatrixOrder::Append);
                        verhaeltnis = Math::Sin(winkel*PI/180);
                        mb.Scale(verhaeltnis, verhaeltnis, MatrixOrder::Append);
                        ma.Translate(points[3].X-points[0].X,points[3].Y-
points[0].Y,MatrixOrder::Append);
                        mb.Translate(points[2].X-points[1].X,points[2].Y-
points[1].Y,MatrixOrder::Append);                       //die Punkte werden verschoben

                        ma.Translate(points[0].X , points[0].Y,MatrixOrder::Append);
//Punkte zurück projezieren
                        mb.Translate(points[1].X , points[1].Y,MatrixOrder::Append);

                        points->CopyTo(Tpoints,0);  // Kopie der Punkte wird mit Matrix ma
Transformiert und für Selbstaufruf verwendet
                        ma.TransformPoints(Tpoints);
                        paint(n-1,Tpoints);

            points->CopyTo(Tpoints,0);  // Kopie der Punkte wird mit Matrix mb
Transformiert und für Selbstaufruf verwendet
                        mb.TransformPoints(Tpoints);
                        paint(n-1,Tpoints);          }
                return ;
        }
public:  virtual void drawpoints(Draw^ g)override{
// Wenn 3 Punkte angegeben sind wird ein Parallelogramm ergänzt
                         if(points->Length>1&&n==0)
                         {
                                 if(points->Length==3){
                                         System::Array::Resize(points,4);
                                         points[3]=PointF(points[2].X-
points[1].X+points[0].X,points[2].Y-points[1].Y+points[0].Y);
                                         g->DrawPolygon(gcnew Pen(front),points);
                                         System::Array::Resize(points,3);

                                 }
                                 g->DrawPolygon(gcnew Pen(front),points);
                         }
                 }
public:  virtual void addpoint(PointF point)override{
```

```
                        if (neededPoints<MinPoints)
                        {
                                throw gcnew System::Exception("Achtung, es werden
mindestens "+MinPoints+" Punkte benötigt für diese Fraktal");
                        }
                        if(points->Length>=this->neededPoints)
                        {
                                n++;
                        }
                        else
                        {
                                if(points->Length==3){
                                        System::Array::Resize(points,4);
                                points[3]=PointF(points[2].X-
points[1].X+points[0].X,points[2].Y-points[1].Y+points[0].Y);n++;
}else
                                System::Array::Resize(points,points->Length+1);
                                points[points->Length-2]=point;
                        }
                }

};
```

3.1.9 SCHNEEFLOCKE.H

```
public ref class Schneeflocke : public Fraktal
{
public:
        Schneeflocke(){MinPoints = 3;name="Schneeflocke";}
        virtual int check() override {return points->Length*Math::Pow(4.0,n);}
        virtual void paint(int n,array<PointF>^ points) override
        {

                {
                        Fraktal^ k = gcnew Kochkurve; // Kochkurve wird erstellt
                        if(random)k->random = 1; // Kochkurve wird konfiguriert ... ,
                        k->n = n;
                        k->front = front;
                        k->Multithreading=0;     // kein Multithreading, da sich die
Threads gegenseitig stören
                        k->clear=0;              // Oberfläche darf vor dem zeichnen nicht
gelöscht werden, da sonst jede Kochkurve die vorherige löschen würde
                        k->draw_info = 0;        // Info würde Infos der einzelnen
Kochkurven anzeigen
                        for(int i=0;i<int(points->Length-1);i++) // es werden vom 1.Punkt
zum 2.Punkt, vom 2. zum 3. usw, bis zum letzten Kochkuven gezeichnet und vom letzen zum
ersten damit die Schneeflocke geschlossen ist                        {
                                k->points[0] = points[i]; // 1. Punkt der Kochkurve ist i.
Punkt der Schneeflocke
                                k->points[1] = points[i+1]; // 2. Punkt i+1 Punkt
                k->points[1] = points[i+1];
                                k->paint(graphics);
                                linien = k->linien;
                        }
                        k->points[0]=points[points->Length-1]; // vom letzten Punkt zum
ersten wird auch eine Kochkurve gezeichnet, um das Polygon abzuschliesen
        k->points[1]=points[0];
                        k->paint(graphics);
                        linien = k->linien;
                }
        }
};
```

3.1.10 SIERPINSKIDREIECK.H

```
public ref class Sierpinski : public Fraktal
{
public:
        Sierpinski(){MinPoints = 3;name="Sierpinski";};
        virtual int check() override {return Math::Pow(3.0,n);}
        virtual void paint(int n,array<PointF>^ points) override
        {
                SolidBrush^ brush =gcnew SolidBrush(front);
                array<PointF>^ tmppoints = gcnew array<PointF>(points->Length);
                //wenn die Funktion bei n=0 angelangt ist werden die Werte gespeichert
                if(n==0||point_distance_sqr(points[0],points[1])<=1)
                {
                        graphics->FillPolygon(brush, points);
```

```
                        this->linien++;
                        return;
                }
                //rekursiver aufruf der funktion
                // die Punkte des Dreiecks das entfernt wird
                array<PointF>^ p = gcnew array<PointF>(3);
                p[0].X = 0.5 * (points[0].X + points[1].X); // p1:=(1/2)(a+b)
                p[0].Y = 0.5 * (points[0].Y + points[1].Y);
                p[1].X = 0.5 * (points[0].X + points[2].X); // p2:=(1/2)(a+c)
                p[1].Y = 0.5 * (points[0].Y + points[2].Y);
                p[2].X = 0.5 * (points[1].X + points[2].X); // p3:=(1/2)(b+c)
                p[2].Y = 0.5 * (points[1].Y + points[2].Y);
                if(random){Matrix m;
                m.Translate(-0.333*(points[0].X+points[1].X+points[2].X),-
0.333*(points[0].Y+points[1].Y+points[2].Y),MatrixOrder::Append);
                m.Rotate(rand->Next(60),MatrixOrder::Append);
                m.Translate( 0.333*(points[0].X+points[1].X+points[2].X),
0.333*(points[0].Y+points[1].Y+points[2].Y),MatrixOrder::Append);
                m.TransformPoints(p);}
                //erstes Dreieck
                tmppoints[0] = points[0];            // a des Original Dreiecks
                tmppoints[1] = p[0];                   // p1
                tmppoints[2] = p[1];                   // p2
                paint(n-1,tmppoints);
                //
                //
                //zweites Dreieck
                tmppoints[0] = p[0];                                  // p1
                tmppoints[1] = points[1];                          // b des Original
Dreiecks
                tmppoints[2] = p[2];                                  // p3
                paint(n-1,tmppoints);
                //
                //
                //drittes Dreieck
                tmppoints[0] = p[2];
                tmppoints[1] = p[1];
                tmppoints[2] = points[2];                          // c:= c
                paint(n-1,tmppoints);
                //
                return ;
        }

};
```

3.1.11 DRAWING.H

```
using namespace System;
using namespace System::Threading;
using namespace System::Drawing;
using namespace System::Drawing::Drawing2D;
using namespace System::Windows::Forms;
public ref class Draw
{
        // Klasse, die grundlegende Grafikfunktionen kapselt
public:
        Graphics^ graphics;
        Draw(Graphics^ g){graphics = g;}
        Draw(){}
        virtual void DrawLine(Pen^ pen,PointF point1,PointF point2){
                try{graphics->DrawLine(pen,point1,point2);}
                catch (Exception^){}
        }
        virtual void DrawPolygon(Pen^ pen,array<PointF>^ points){
                try{graphics->DrawPolygon(pen,points);}
                catch (Exception^){}
        }
        void DrawPolygon(Pen^ pen,array<PointF>^ points,Thread^ thread){
                try{graphics->DrawPolygon(pen,points);}
                catch (Exception^){}
        }
        void DrawString(String^ text,Font^ font,Brush^ brush,PointF position){
                try{graphics->DrawString(text,font,brush,position);}
                catch (Exception^){}
        }
        virtual void FillPolygon(Brush^ brush,array<PointF>^ points){
                try{graphics->FillPolygon(brush,points);}
                catch (Exception^){}
```

```
        }
        virtual void DrawImage(Bitmap^b){graphics->DrawImage(b,PointF(0,0));}
        void Clear(Color color){
                try{graphics->Clear(color);}
                catch (Exception^){}
        }
};
public  ref class PictureInitiator : public Draw{
        // Subfunktion der Klasse Draw, zeichnet statt Linien, oder Flächen ein Bild
        Bitmap^ pic;
public:
        PictureInitiator(Graphics^g,Bitmap^initiator){
                graphics = g;pic = initiator;
        }
        void DrawPicture(array<PointF>^ points){
                try{
                        array<PointF>^ Tpoints = gcnew array<PointF>(points->Length);
                        points->CopyTo(Tpoints,0);
                        if(points->Length==2){
                                Array::Resize(Tpoints,3);
                                Tpoints[2].X=2*Tpoints[1].X-
Tpoints[0].X;Tpoints[2].Y=2*Tpoints[0].Y-Tpoints[1].Y;}
                        if(points->Length!=3){Array::Resize(Tpoints,3);}
                        PointF tmp;tmp=Tpoints[0];Tpoints[0]=Tpoints[1];Tpoints[1]=tmp;
                        graphics->DrawImage(pic,Tpoints);
                }
                catch (Exception^){}}
        virtual void DrawPolygon(Pen^ pen,array<PointF>^ points)override{
                DrawPicture(points);
        }

        virtual void FillPolygon(Brush^ brush,array<PointF>^ points)override{
                DrawPicture(points);
        }
        virtual void DrawLine(Pen^ pen,PointF point1,PointF point2)override{
                array<PointF>^ points = gcnew array<PointF>(2);
                points[0]=point1;points[1]=point2;
                DrawPicture(points);
        }

};
```

3.1.12 MORPGMATRIX.H

```
using namespace System;
using namespace System::Threading;
using namespace System::Drawing;
using namespace System::Drawing::Drawing2D;
using namespace System::Windows::Forms;
// Feature noch unter Entwicklung, Morpht eine Matrix in eine andere
public ref class MorphMatrix{
        DateTime end;
        Matrix ^m_1, ^m_2;
        array<PointF>^ points;
        float time;
        Thread^ thread;
public:Bitmap^ bitmap;
public:Color front,back;
        Matrix^ MMatrix(Matrix^ m1,Matrix^ m2,double progress){
                // Matrix wird in einer andere umgewandelt
                if(progress>1||progress<0){MessageBox::Show("Fortschritt bitte zwischen 0
und 1 angeben");}
                array<float>^ elements1 = gcnew array<float>(6);m1-
>Elements.CopyTo(elements1,0);
                array<float>^ elements2 = gcnew array<float>(6);m2-
>Elements.CopyTo(elements2,0);
                return gcnew Matrix((1-progress)*elements1[0]+progress*elements2[0],
                        (1-progress)*elements1[1]+progress*elements2[1],
                        (1-progress)*elements1[2]+progress*elements2[2],
                        (1-progress)*elements1[3]+progress*elements2[3],
                        (1-progress)*elements1[4]+progress*elements2[4],
                        (1-progress)*elements1[5]+progress*elements2[5]);

        }

public:MorphMatrix(Matrix ^m1,Matrix ^m2,array<PointF>^ Points,float time_to_morph)
                {points = Points;time = time_to_morph;m_1 = m1;m_2 =
m2;starthread();front = Color::Black;}
```

```
		property bool IsAlive{
			bool get(){return thread->IsAlive;}
		}
private:  void doIt(){
				end = DateTime::Now.AddSeconds(time);
				bitmap = gcnew Bitmap(500,500);
				Graphics^g = Graphics::FromImage(bitmap);
				g->TranslateTransform(250,250);
				for(;DateTime::Now<end;)
				{
					array<PointF>^ p_cpy = gcnew array<PointF>(points-
>Length);points->CopyTo(p_cpy,0);
					TimeSpan^ delay = end-DateTime::Now;
					MMatrix(m_1,m_2,1-(delay->Seconds+float(delay-
>Milliseconds)/1000)/time)->TransformPoints(p_cpy);
					g->Clear(Color::Transparent);
					g->DrawPolygon(gcnew Pen(front),p_cpy);
					thread->Sleep(100);
				}
		 }
		 void starthread()
		 {
			try
			{
				thread = gcnew Thread( gcnew
ThreadStart(this,&MorphMatrix::doIt) ) ;
				thread->Start() ;
			}
			catch (System::Exception^ error)
			{
#ifdef _DEBUG
				MessageBox::Show("Fehler: "+error->Message);
#endif // _DEBUG
			}

		 }
		void stopthread(){
			if(thread!=nullptr&&thread->IsAlive){thread->Abort();}
		 }
		~MorphMatrix(){stopthread();}
};
```

3.2 FRAKTALE.EXE

Fenster und Input

3.2.1 FRAKTALE.CPP

```
// Fraktale.cpp: Hauptprojektdatei.

#include "stdafx.h"
#include "mainWindow.h"
using namespace Fraktale;
[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
	// Aktivieren visueller Effekte von Windows XP, bevor Steuerelemente erstellt
werden
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);
	// Hauptfenster erstellen und ausführen
	if(!System::IO::File::Exists("fraktal_sammlung.dll")){System::Windows::Forms::Mes
sageBox::Show("fraktal_sammlung.dll nicht gefunden, bitte überprüfen sie ob alle nötigen
Dateien im selben Verzeichnis sind wie die Exe");}
	Application::Run(gcnew mainWindow());
	Application::Exit();
	return 0;
}
```

3.2.2 FILEMANAGMENT.H

```
using namespace System::IO;
public ref class FileManagment{
	/// Verwaltung von Dateien (Laden von Bilder,Fraktale,Fehlerbehandlung)
private: String ^filename, ^name, ^filetyp;
		bool warnings;
		StreamReader^ loader;
public:  FileManagment(String^Filename){warnings = 1;filename = Filename;filetyp = "*";}
```

```
                FileManagment(String^Filename,String^ Filetyp){warnings = 1;filename =
Filename;FileTyp = Filetyp;}
                property bool Warnings{
                        bool get() { return warnings; }
                        void set(bool val) { warnings = val; }
                }
                void GetPath(){
                        if(!(filename=="none")){
                        if(!IO::File::Exists(filename)){
                                if(!warnings||MessageBox::Show(filename+" wurde nicht
gefunden, bitte Datei angeben",
                                                "Datei
fehlt",MessageBoxButtons::OKCancel,MessageBoxIcon::Information) ==
System::Windows::Forms::DialogResult::OK){
                                                OpenFileDialog^ dialog = gcnew
OpenFileDialog();
                                                dialog->Filter = name+L" Dateien
(*."+filetyp+")|*."+filetyp+"|alle Dateien|*.*";
                                                dialog->Title = L"Lade "+ name;
                                                dialog->RestoreDirectory = true;
                                                if(dialog->ShowDialog() ==
Windows::Forms::DialogResult::OK)
                                                {if(IO::File::Exists(dialog-
>FileName)){filename = dialog->FileName;return;}}}
                                filename = "none";
                        }else{loader = gcnew System::IO::StreamReader(filename);}}
                }
                Image^ LoadPicture(){
                        GetPath();
                        if(filename=="none"){return gcnew Bitmap(1,1);}
                        {Image^ bmp =  Bitmap::FromFile(filename);return bmp;}
                }
                array<Fraktal^>^ LoadFractals(Fraktal^ f){
                        {
                                GetPath();
                                if(filename=="none"){array<Fraktal^>^f = gcnew
array<Fraktal^>(1);f[0]=gcnew Kochkurve;return f;}
                                array<Fraktal^>^ LoadedFractals;
                                LoadedFractals  = f->Load(loader);
                                return LoadedFractals;
                        }
                }
public:  property String^ FileName{
                        String^ get()
                        {
                                return filename;
                        }
                        void set(String^ value)
                        {
                                filename = value;
                    }
                }
                property String^ Name{
                                String^ get()
                                {
                                        return name;
                                }
                                void set(String^ value)
                                {
                                        name = value;
                                }
                        }
                property String^ FileTyp{
                                String^ get()
                                {
                                        return filetyp;
                                }
                                void set(String^ value)
                                {
                                        filetyp = value;
                                }
                        }
};
```

3.2.3 MAINWINDOW.H

```
#pragma once
// Hauptfenster, Steuerung der Fraktale, Eingabe, Oberfläche
```

```
namespace Fraktale {
	using namespace fraktal_sammlung;
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Drawing;
	using namespace System::Threading;
	using namespace System::IO;
//#define mouspointer
#ifdef mouspointer
	[DllImport("user32.dll", EntryPoint = "LoadCursorFromFileW",CharSet =
CharSet::Unicode)]
	extern IntPtr LoadC(String^ fileName);
#endif
	 /*<summary>
	 Zusammenfassung für mainWindow
	 Warnung: Wenn Sie den Namen dieser Klasse ändern, müssen Sie auch
		die Ressourcendateiname-Eigenschaft für das Tool zur Kompilierung
verwalteter Ressourcen ändern,
		das allen RESX-Dateien zugewiesen ist, von denen diese Klasse abhängt.
		Anderenfalls können die Designer nicht korrekt mit den lokalisierten
Ressourcen
		arbeiten, die diesem Formular zugewiesen sind.
	 </summary>*/
	public ref class mainWindow : public System::Windows::Forms::Form
	{
	public:
		mainWindow(void)
		{
			InitializeComponent();
		    set_mode(0);
			this->DoubleBuffered = 1;
			Screen^ s = Screen::PrimaryScreen;
			bmp_disp = gcnew Bitmap(s->WorkingArea.Width,s-
>WorkingArea.Height);
			this->button1->BackColor = FrontColorDialog->Color;
			BackColorDialog->Color = this->BackColor;
			this->button8->BackColor = BackColorDialog->Color;
			this->comboBox1->Items->Clear();
			this->comboBox1->Items->AddRange(fraktal->ListFractals);
			this->comboBox1->Text = fraktal->name;
			resourcen = gcnew array<FileManagment^>(4);
			resourcen[0] = gcnew FileManagment("nothing.bmp");
			resourcen[1] = gcnew FileManagment("working.bmp");
			resourcen[2] = gcnew FileManagment("lsysteme.lsystem");
			resourcen[3] = gcnew FileManagment("fraktale.ifs");
			this->Invalidate();
			this->toolStripDropDownButton2->Image = resourcen[0]-
>LoadPicture();
#ifdef mouspointer
			if(System::IO::File::Exists("sierpinsk.ani"))
			{
				try
				{
					this->Cursor = gcnew
System::Windows::Forms::Cursor(LoadC("sierpinsk.ani"));
				}
				catch (System::DllNotFoundException^e)
				{
					MessageBox::Show(e->Message);
				}

			}
#endif
		}

	protected:
		/// <summary>
		/// Verwendete Ressourcen bereinigen.
		/// </summary>
		~mainWindow()
		{
			if (components)
			{
				delete components;
				fraktal->stopthread();
```

```
                }
            }

        private: Fraktal^ fraktal;
        private: Bitmap^ bmp_disp;
        array<Fraktal^>^ LoadedFractal;
        private: array<FileManagment^>^ resourcen;
        private: System::Windows::Forms::Button^  button2;
        private: System::Windows::Forms::TextBox^  lsys_rule;
        private: System::Windows::Forms::Button^  lsys_ok;
        private: System::Windows::Forms::TextBox^  textBox2;
        private: System::Windows::Forms::ListBox^  listBox1;
        private: System::Windows::Forms::Button^  lsys_abort;

        private: System::Windows::Forms::Label^  label2;
        private: System::Windows::Forms::Label^  label3;
        private: System::Windows::Forms::Label^  label4;
        private: System::Windows::Forms::Label^  label1;
        private: System::Windows::Forms::Button^  load;


        private: System::Windows::Forms::GroupBox^  groupBox1;
        private: System::Windows::Forms::GroupBox^  groupBox2;
        private: System::Windows::Forms::ContextMenuStrip^  contextMenuStrip1;

        private: System::Windows::Forms::ComboBox^  comboBox1;
        private: System::Windows::Forms::Timer^  timer1;
        private: System::Windows::Forms::NumericUpDown^  numericUpDown1;
        private: System::Windows::Forms::Label^  label6;

        private: System::Windows::Forms::GroupBox^  groupBox5;
        private: System::Windows::Forms::Button^  button1;
        private: System::Windows::Forms::ColorDialog^  FrontColorDialog;
        private: System::Windows::Forms::ColorDialog^  BackColorDialog;
        private: System::Windows::Forms::Button^  button8;
        private: System::Windows::Forms::GroupBox^  groupBox6;
        private: System::Windows::Forms::Label^  label5;
        private: System::Windows::Forms::CheckBox^  checkBox1;
        private: System::Windows::Forms::TextBox^  textBox3;

        private: System::Windows::Forms::SaveFileDialog^  saveFileDialog1;
        private: System::Windows::Forms::Button^  Save;
        private: System::Windows::Forms::StatusStrip^  statusStrip1;
        private: System::Windows::Forms::ToolStripProgressBar^  toolStripProgressBar1;
        private: System::Windows::Forms::ToolStripStatusLabel^  toolStripStatusLabel1;
        private: System::Windows::Forms::Label^  label7;
        private: System::Windows::Forms::ToolStripDropDownButton^
toolStripDropDownButton1;
        private: System::Windows::Forms::ToolStripMenuItem^  ausToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^  anToolStripMenuItem;
        private: System::Windows::Forms::Button^  lsys_settings;

        private: System::Windows::Forms::GroupBox^  groupBox3;
        private: System::Windows::Forms::GroupBox^  groupBox4;
        private: System::Windows::Forms::Button^  ifs_settings;

        private: System::Windows::Forms::Button^  ifs_load;


        private: System::Windows::Forms::Button^  ifs_ok;
        private: System::Windows::Forms::Button^  ifs_abort;


        private: System::Windows::Forms::ListBox^  listBox2;
        private: System::Windows::Forms::Label^  label11;




private: System::Windows::Forms::PictureBox^  pictureBox1;
private: System::Windows::Forms::ComboBox^  comboBox2;
```

```
private: System::Windows::Forms::NumericUpDown^  numericUpDown2;
private: System::Windows::Forms::Label^  label12;
private: System::Windows::Forms::ToolStripStatusLabel^  toolStripDropDownButton2;
private: System::Windows::Forms::CheckBox^  checkBox2;
private: System::Windows::Forms::ToolTip^  toolTip1;
private: System::Windows::Forms::Button^  button3;
private: System::Windows::Forms::OpenFileDialog^  OpenInitiator;
private: System::Windows::Forms::CheckBox^  checkBox3;
private: System::Windows::Forms::TextBox^  textBox1;
private: System::Windows::Forms::CheckBox^  checkBox4;
private: System::Windows::Forms::GroupBox^  groupBox7;
private: System::Windows::Forms::Label^  label13;
private: System::Windows::Forms::Label^  label14;
private: System::Windows::Forms::Label^  label15;
private: System::Windows::Forms::TextBox^  ifs_vy;
private: System::Windows::Forms::TextBox^  ifs_scale;


private: System::Windows::Forms::TextBox^  ifs_rotate;

private: System::Windows::Forms::TextBox^  ifs_vx;

private: System::Windows::Forms::GroupBox^  groupBox8;
private: System::Windows::Forms::Button^  ifs2_load;
private: System::Windows::Forms::Button^  ifs2_ok;
private: System::Windows::Forms::Button^  ifs2_cancel;
private: System::Windows::Forms::ListBox^  listBox3;
private: System::Windows::Forms::PictureBox^  pictureBox2;
private: System::Windows::Forms::CheckBox^  checkBox5;




private: System::ComponentModel::IContainer^  components;
protected:
private:
		/// <summary>
		/// Erforderliche Designervariable.
		/// </summary>


#pragma region Windows Form Designer generated code
		/// <summary>
		/// Erforderliche Methode für die Designerunterstützung.
		/// Der Inhalt der Methode darf nicht mit dem Code-Editor geändert
werden.
		/// </summary>
		void InitializeComponent(void)
		{
			this->components = (gcnew System::ComponentModel::Container());
			System::ComponentModel::ComponentResourceManager^  resources =
(gcnew System::ComponentModel::ComponentResourceManager(mainWindow::typeid));
			this->button2 = (gcnew System::Windows::Forms::Button());
			this->lsys_rule = (gcnew System::Windows::Forms::TextBox());
			this->lsys_ok = (gcnew System::Windows::Forms::Button());
			this->textBox2 = (gcnew System::Windows::Forms::TextBox());
			this->listBox1 = (gcnew System::Windows::Forms::ListBox());
			this->lsys_abort = (gcnew System::Windows::Forms::Button());
			this->label2 = (gcnew System::Windows::Forms::Label());
			this->label3 = (gcnew System::Windows::Forms::Label());
			this->label4 = (gcnew System::Windows::Forms::Label());
			this->label1 = (gcnew System::Windows::Forms::Label());
			this->load = (gcnew System::Windows::Forms::Button());
			this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
			this->checkBox4 = (gcnew System::Windows::Forms::CheckBox());
			this->textBox1 = (gcnew System::Windows::Forms::TextBox());
			this->groupBox5 = (gcnew System::Windows::Forms::GroupBox());
			this->lsys_settings = (gcnew System::Windows::Forms::Button());
			this->textBox3 = (gcnew System::Windows::Forms::TextBox());
			this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());
			this->button3 = (gcnew System::Windows::Forms::Button());
			this->checkBox3 = (gcnew System::Windows::Forms::CheckBox());
			this->checkBox2 = (gcnew System::Windows::Forms::CheckBox());
			this->label12 = (gcnew System::Windows::Forms::Label());
			this->numericUpDown2 = (gcnew
System::Windows::Forms::NumericUpDown());
```

```
                    this->numericUpDown1 = (gcnew
System::Windows::Forms::NumericUpDown());
                    this->checkBox1 = (gcnew System::Windows::Forms::CheckBox());
                    this->groupBox6 = (gcnew System::Windows::Forms::GroupBox());
                    this->button1 = (gcnew System::Windows::Forms::Button());
                    this->button8 = (gcnew System::Windows::Forms::Button());
                    this->label6 = (gcnew System::Windows::Forms::Label());
                    this->Save = (gcnew System::Windows::Forms::Button());
                    this->comboBox1 = (gcnew System::Windows::Forms::ComboBox());
                    this->contextMenuStrip1 = (gcnew
System::Windows::Forms::ContextMenuStrip(this->components));
                    this->timer1 = (gcnew System::Windows::Forms::Timer(this-
>components));
                    this->label5 = (gcnew System::Windows::Forms::Label());
                    this->statusStrip1 = (gcnew
System::Windows::Forms::StatusStrip());
                    this->toolStripProgressBar1 = (gcnew
System::Windows::Forms::ToolStripProgressBar());
                    this->toolStripDropDownButton1 = (gcnew
System::Windows::Forms::ToolStripDropDownButton());
                    this->ausToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                    this->anToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                    this->toolStripDropDownButton2 = (gcnew
System::Windows::Forms::ToolStripStatusLabel());
                    this->toolStripStatusLabel1 = (gcnew
System::Windows::Forms::ToolStripStatusLabel());
                    this->label7 = (gcnew System::Windows::Forms::Label());
                    this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());
                    this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());
                    this->groupBox4 = (gcnew System::Windows::Forms::GroupBox());
                    this->ifs_settings = (gcnew System::Windows::Forms::Button());
                    this->ifs_load = (gcnew System::Windows::Forms::Button());
                    this->ifs_ok = (gcnew System::Windows::Forms::Button());
                    this->ifs_abort = (gcnew System::Windows::Forms::Button());
                    this->listBox2 = (gcnew System::Windows::Forms::ListBox());
                    this->label11 = (gcnew System::Windows::Forms::Label());
                    this->comboBox2 = (gcnew System::Windows::Forms::ComboBox());
                    this->FrontColorDialog = (gcnew
System::Windows::Forms::ColorDialog());
                    this->BackColorDialog = (gcnew
System::Windows::Forms::ColorDialog());
                    this->saveFileDialog1 = (gcnew
System::Windows::Forms::SaveFileDialog());
                    this->toolTip1 = (gcnew System::Windows::Forms::ToolTip(this-
>components));
                    this->OpenInitiator = (gcnew
System::Windows::Forms::OpenFileDialog());
                    this->groupBox7 = (gcnew System::Windows::Forms::GroupBox());
                    this->pictureBox2 = (gcnew System::Windows::Forms::PictureBox());
                    this->listBox3 = (gcnew System::Windows::Forms::ListBox());
                    this->label13 = (gcnew System::Windows::Forms::Label());
                    this->label14 = (gcnew System::Windows::Forms::Label());
                    this->label15 = (gcnew System::Windows::Forms::Label());
                    this->ifs_vy = (gcnew System::Windows::Forms::TextBox());
                    this->ifs_scale = (gcnew System::Windows::Forms::TextBox());
                    this->ifs_rotate = (gcnew System::Windows::Forms::TextBox());
                    this->ifs_vx = (gcnew System::Windows::Forms::TextBox());
                    this->groupBox8 = (gcnew System::Windows::Forms::GroupBox());
                    this->ifs2_load = (gcnew System::Windows::Forms::Button());
                    this->ifs2_ok = (gcnew System::Windows::Forms::Button());
                    this->ifs2_cancel = (gcnew System::Windows::Forms::Button());
                    this->checkBox5 = (gcnew System::Windows::Forms::CheckBox());
                    this->groupBox1->SuspendLayout();
                    this->groupBox5->SuspendLayout();
                    this->groupBox2->SuspendLayout();
                    (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->numericUpDown2))->BeginInit();
                    (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->numericUpDown1))->BeginInit();
                    this->groupBox6->SuspendLayout();
                    this->statusStrip1->SuspendLayout();
                    this->groupBox3->SuspendLayout();
                    (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox1))->BeginInit();
                    this->groupBox4->SuspendLayout();
                    this->groupBox7->SuspendLayout();
```

```
                        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox2))->BeginInit();
                        this->groupBox8->SuspendLayout();
                        this->SuspendLayout();
                        //
                        // button2
                        //
                        this->button2->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->button2->Location = System::Drawing::Point(91, 171);
                        this->button2->Margin = System::Windows::Forms::Padding(4);
                        this->button2->Name = L"button2";
                        this->button2->Size = System::Drawing::Size(92, 30);
                        this->button2->TabIndex = 1;
                        this->button2->Text = L"Pause";
                        this->button2->UseVisualStyleBackColor = true;
                        this->button2->Click += gcnew System::EventHandler(this,
&mainWindow::button2_Click);
                        //
                        // lsys_rule
                        //
                        this->lsys_rule->Location = System::Drawing::Point(91, 32);
                        this->lsys_rule->Margin = System::Windows::Forms::Padding(4);
                        this->lsys_rule->Name = L"lsys_rule";
                        this->lsys_rule->Size = System::Drawing::Size(123, 22);
                        this->lsys_rule->TabIndex = 3;
                        this->lsys_rule->TextChanged += gcnew System::EventHandler(this,
&mainWindow::textBox1_TextChanged);
                        //
                        // lsys_ok
                        //
                        this->lsys_ok->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"lsys_ok.BackgroundImage")));
                        this->lsys_ok->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->lsys_ok->Location = System::Drawing::Point(123, 58);
                        this->lsys_ok->Margin = System::Windows::Forms::Padding(4);
                        this->lsys_ok->Name = L"lsys_ok";
                        this->lsys_ok->Size = System::Drawing::Size(28, 22);
                        this->lsys_ok->TabIndex = 4;
                        this->lsys_ok->UseVisualStyleBackColor = true;
                        this->lsys_ok->Click += gcnew System::EventHandler(this,
&mainWindow::lsys_ok_Click);
                        //
                        // textBox2
                        //
                        this->textBox2->Location = System::Drawing::Point(12, 43);
                        this->textBox2->Margin = System::Windows::Forms::Padding(4);
                        this->textBox2->Name = L"textBox2";
                        this->textBox2->Size = System::Drawing::Size(85, 22);
                        this->textBox2->TabIndex = 6;
                        this->textBox2->Text = L"F--F--F";
                        this->textBox2->TextChanged += gcnew System::EventHandler(this,
&mainWindow::textBox2_TextChanged);
                        //
                        // listBox1
                        //
                        this->listBox1->ItemHeight = 16;
                        this->listBox1->Items->AddRange(gcnew cli::array< System::Object^
>(1) {L"F:F+F--F+F"});
                        this->listBox1->Location = System::Drawing::Point(105, 43);
                        this->listBox1->Margin = System::Windows::Forms::Padding(4);
                        this->listBox1->Name = L"listBox1";
                        this->listBox1->Size = System::Drawing::Size(111, 52);
                        this->listBox1->TabIndex = 7;
                        this->listBox1->SelectedIndexChanged += gcnew
System::EventHandler(this, &mainWindow::listBox1_SelectedIndexChanged);
                        //
                        // lsys_abort
                        //
                        this->lsys_abort->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"lsys_abort.BackgroundImage")));
                        this->lsys_abort->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->lsys_abort->Location = System::Drawing::Point(153, 58);
                        this->lsys_abort->Margin = System::Windows::Forms::Padding(4);
```

```
                    this->lsys_abort->Name = L"lsys_abort";
                    this->lsys_abort->Size = System::Drawing::Size(25, 22);
                    this->lsys_abort->TabIndex = 8;
                    this->lsys_abort->UseVisualStyleBackColor = true;
                    this->lsys_abort->Click += gcnew System::EventHandler(this,
&mainWindow::button5_Click);
                    //
                    // label2
                    //
                    this->label2->AutoSize = true;
                    this->label2->Location = System::Drawing::Point(8, 75);
                    this->label2->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                    this->label2->Name = L"label2";
                    this->label2->Size = System::Drawing::Size(100, 17);
                    this->label2->TabIndex = 10;
                    this->label2->Text = L"Winkel =        °";
                    //
                    // label3
                    //
                    this->label3->AutoSize = true;
                    this->label3->Location = System::Drawing::Point(87, 12);
                    this->label3->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                    this->label3->Name = L"label3";
                    this->label3->Size = System::Drawing::Size(123, 17);
                    this->label3->TabIndex = 11;
                    this->label3->Text = L"Regel hinzufügen:";
                    //
                    // label4
                    //
                    this->label4->AutoSize = true;
                    this->label4->Location = System::Drawing::Point(16, 23);
                    this->label4->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                    this->label4->Name = L"label4";
                    this->label4->Size = System::Drawing::Size(49, 17);
                    this->label4->TabIndex = 12;
                    this->label4->Text = L"Axiom:";
                    //
                    // label1
                    //
                    this->label1->AutoSize = true;
                    this->label1->Location = System::Drawing::Point(101, 23);
                    this->label1->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                    this->label1->Name = L"label1";
                    this->label1->Size = System::Drawing::Size(127, 17);
                    this->label1->TabIndex = 13;
                    this->label1->Text = L"Ersetzunggsregeln";
                    //
                    // load
                    //
                    this->load->Location = System::Drawing::Point(24, 32);
                    this->load->Margin = System::Windows::Forms::Padding(4);
                    this->load->Name = L"load";
                    this->load->Size = System::Drawing::Size(67, 26);
                    this->load->TabIndex = 16;
                    this->load->Text = L"Laden";
                    this->load->UseVisualStyleBackColor = true;
                    this->load->Click += gcnew System::EventHandler(this,
&mainWindow::button7_Click);
                    //
                    // groupBox1
                    //
                    this->groupBox1->Controls->Add(this->checkBox4);
                    this->groupBox1->Controls->Add(this->textBox1);
                    this->groupBox1->Controls->Add(this->groupBox5);
                    this->groupBox1->Controls->Add(this->textBox3);
                    this->groupBox1->Controls->Add(this->listBox1);
                    this->groupBox1->Controls->Add(this->textBox2);
                    this->groupBox1->Controls->Add(this->label4);
                    this->groupBox1->Controls->Add(this->label2);
                    this->groupBox1->Controls->Add(this->label1);
                    this->groupBox1->FlatStyle =
System::Windows::Forms::FlatStyle::Popup;
                    this->groupBox1->Location = System::Drawing::Point(11, 321);
                    this->groupBox1->Margin = System::Windows::Forms::Padding(4);
```

```
                        this->groupBox1->Name = L"groupBox1";
                        this->groupBox1->Padding = System::Windows::Forms::Padding(4);
                        this->groupBox1->Size = System::Drawing::Size(239, 273);
                        this->groupBox1->TabIndex = 17;
                        this->groupBox1->TabStop = false;
                        this->groupBox1->Text = L"L-System";
                        this->groupBox1->Visible = false;
                        this->groupBox1->Enter += gcnew System::EventHandler(this,
&mainWindow::groupBox1_Enter);
                        //
                        // checkBox4
                        //
                        this->checkBox4->AutoSize = true;
                        this->checkBox4->Location = System::Drawing::Point(8, 245);
                        this->checkBox4->Name = L"checkBox4";
                        this->checkBox4->Size = System::Drawing::Size(174, 21);
                        this->checkBox4->TabIndex = 27;
                        this->checkBox4->Text = L"Zeichenkette anzeigen";
                        this->checkBox4->UseVisualStyleBackColor = true;
                        this->checkBox4->CheckedChanged += gcnew
System::EventHandler(this, &mainWindow::checkBox4_CheckedChanged);
                        //
                        // textBox1
                        //
                        this->textBox1->AcceptsReturn = true;
                        this->textBox1->Location = System::Drawing::Point(8, 190);
                        this->textBox1->Multiline = true;
                        this->textBox1->Name = L"textBox1";
                        this->textBox1->ReadOnly = true;
                        this->textBox1->ScrollBars =
System::Windows::Forms::ScrollBars::Vertical;
                        this->textBox1->Size = System::Drawing::Size(223, 52);
                        this->textBox1->TabIndex = 26;
                        this->textBox1->Text = L"....";
                        //
                        // groupBox5
                        //
                        this->groupBox5->Controls->Add(this->lsys_settings);
                        this->groupBox5->Controls->Add(this->label3);
                        this->groupBox5->Controls->Add(this->lsys_rule);
                        this->groupBox5->Controls->Add(this->lsys_ok);
                        this->groupBox5->Controls->Add(this->load);
                        this->groupBox5->Controls->Add(this->lsys_abort);
                        this->groupBox5->Location = System::Drawing::Point(8, 103);
                        this->groupBox5->Margin = System::Windows::Forms::Padding(4);
                        this->groupBox5->Name = L"groupBox5";
                        this->groupBox5->Padding = System::Windows::Forms::Padding(4);
                        this->groupBox5->Size = System::Drawing::Size(223, 86);
                        this->groupBox5->TabIndex = 25;
                        this->groupBox5->TabStop = false;
                        //
                        // lsys_settings
                        //
                        this->lsys_settings->Location = System::Drawing::Point(3, 32);
                        this->lsys_settings->Margin = System::Windows::Forms::Padding(4);
                        this->lsys_settings->Name = L"lsys_settings";
                        this->lsys_settings->Size = System::Drawing::Size(24, 26);
                        this->lsys_settings->TabIndex = 26;
                        this->lsys_settings->Text = L"#";
                        this->lsys_settings->UseVisualStyleBackColor = true;
                        this->lsys_settings->Click += gcnew System::EventHandler(this,
&mainWindow::button3_Click);
                        //
                        // textBox3
                        //
                        this->textBox3->Location = System::Drawing::Point(65, 71);
                        this->textBox3->Margin = System::Windows::Forms::Padding(4);
                        this->textBox3->Name = L"textBox3";
                        this->textBox3->Size = System::Drawing::Size(29, 22);
                        this->textBox3->TabIndex = 9;
                        this->textBox3->Text = L"60";
                        this->textBox3->TextChanged += gcnew System::EventHandler(this,
&mainWindow::textBox3_TextChanged);
                        //
                        // groupBox2
                        //
                        this->groupBox2->Controls->Add(this->button3);
                        this->groupBox2->Controls->Add(this->checkBox3);
```

```
                        this->groupBox2->Controls->Add(this->checkBox2);
                        this->groupBox2->Controls->Add(this->label12);
                        this->groupBox2->Controls->Add(this->numericUpDown2);
                        this->groupBox2->Controls->Add(this->button2);
                        this->groupBox2->Controls->Add(this->numericUpDown1);
                        this->groupBox2->Controls->Add(this->checkBox1);
                        this->groupBox2->Controls->Add(this->groupBox6);
                        this->groupBox2->Controls->Add(this->label6);
                        this->groupBox2->Controls->Add(this->Save);
                        this->groupBox2->Controls->Add(this->comboBox1);
                        this->groupBox2->Location = System::Drawing::Point(11, 80);
                        this->groupBox2->Margin = System::Windows::Forms::Padding(4);
                        this->groupBox2->Name = L"groupBox2";
                        this->groupBox2->Padding = System::Windows::Forms::Padding(4);
                        this->groupBox2->Size = System::Drawing::Size(239, 204);
                        this->groupBox2->TabIndex = 18;
                        this->groupBox2->TabStop = false;
                        this->groupBox2->Text = L"Fraktale";
                        this->groupBox2->Enter += gcnew System::EventHandler(this,
&mainWindow::groupBox2_Enter);
                        //
                        // button3
                        //
                        this->button3->Location = System::Drawing::Point(204, 140);
                        this->button3->Margin = System::Windows::Forms::Padding(4);
                        this->button3->Name = L"button3";
                        this->button3->Size = System::Drawing::Size(31, 23);
                        this->button3->TabIndex = 26;
                        this->button3->Text = L"...";
                        this->button3->UseVisualStyleBackColor = true;
                        this->button3->Click += gcnew System::EventHandler(this,
&mainWindow::button3_Click_1);
                        //
                        // checkBox3
                        //
                        this->checkBox3->AutoSize = true;
                        this->checkBox3->Location = System::Drawing::Point(8, 143);
                        this->checkBox3->Margin = System::Windows::Forms::Padding(4);
                        this->checkBox3->Name = L"checkBox3";
                        this->checkBox3->Size = System::Drawing::Size(201, 21);
                        this->checkBox3->TabIndex = 36;
                        this->checkBox3->Text = L"Benutzerdefinierter Initiator";
                        this->checkBox3->UseVisualStyleBackColor = true;
                        this->checkBox3->CheckedChanged += gcnew
System::EventHandler(this, &mainWindow::checkBox3_CheckedChanged);
                        //
                        // checkBox2
                        //
                        this->checkBox2->AutoSize = true;
                        this->checkBox2->Checked = true;
                        this->checkBox2->CheckState =
System::Windows::Forms::CheckState::Checked;
                        this->checkBox2->Location = System::Drawing::Point(8, 116);
                        this->checkBox2->Margin = System::Windows::Forms::Padding(4);
                        this->checkBox2->Name = L"checkBox2";
                        this->checkBox2->Size = System::Drawing::Size(152, 21);
                        this->checkBox2->TabIndex = 34;
                        this->checkBox2->Text = L"Abbr. bei d < 1Pixel";
                        this->toolTip1->SetToolTip(this->checkBox2, L"Die Iteration des
Fraktals wird abgebrochen wenn der Abstand der Punkte auf 1 Pix"
                                L"el geschrumpft ist");
                        this->checkBox2->UseVisualStyleBackColor = true;
                        this->checkBox2->CheckedChanged += gcnew
System::EventHandler(this, &mainWindow::checkBox2_CheckedChanged_1);
                        //
                        // label12
                        //
                        this->label12->AutoSize = true;
                        this->label12->Location = System::Drawing::Point(137, 87);
                        this->label12->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                        this->label12->Name = L"label12";
                        this->label12->Size = System::Drawing::Size(56, 17);
                        this->label12->TabIndex = 33;
                        this->label12->Text = L"Punkte:";
                        //
                        // numericUpDown2
                        //
```

```
                    this->numericUpDown2->Location = System::Drawing::Point(203, 84);
                    this->numericUpDown2->Margin = System::Windows::Forms::Padding(4);
                    this->numericUpDown2->Name = L"numericUpDown2";
                    this->numericUpDown2->Size = System::Drawing::Size(35, 22);
                    this->numericUpDown2->TabIndex = 35;
                    this->toolTip1->SetToolTip(this->numericUpDown2, L"Einstellung wie
viele Punkte sie angeben möchten um das Fraktal zu erzeugen");
                    this->numericUpDown2->ValueChanged += gcnew
System::EventHandler(this, &mainWindow::numericUpDown2_ValueChanged);
                    //
                    // numericUpDown1
                    //
                    this->numericUpDown1->Location = System::Drawing::Point(141, 55);
                    this->numericUpDown1->Margin = System::Windows::Forms::Padding(4);
                    this->numericUpDown1->Name = L"numericUpDown1";
                    this->numericUpDown1->Size = System::Drawing::Size(49, 22);
                    this->numericUpDown1->TabIndex = 22;
                    this->toolTip1->SetToolTip(this->numericUpDown1, L"wie oft der
Algorythmus Iteriert werden soll");
                    this->numericUpDown1->ValueChanged += gcnew
System::EventHandler(this, &mainWindow::numericUpDown1_ValueChanged);
                    //
                    // checkBox1
                    //
                    this->checkBox1->AutoSize = true;
                    this->checkBox1->Location = System::Drawing::Point(8, 87);
                    this->checkBox1->Margin = System::Windows::Forms::Padding(4);
                    this->checkBox1->Name = L"checkBox1";
                    this->checkBox1->Size = System::Drawing::Size(63, 21);
                    this->checkBox1->TabIndex = 25;
                    this->checkBox1->Text = L"zufall";
                    this->toolTip1->SetToolTip(this->checkBox1, L"hiermit wird zufall
in die entstehung des Fraktals gebracht");
                    this->checkBox1->UseVisualStyleBackColor = true;
                    this->checkBox1->CheckedChanged += gcnew
System::EventHandler(this, &mainWindow::checkBox1_CheckedChanged);
                    //
                    // groupBox6
                    //
                    this->groupBox6->Controls->Add(this->button1);
                    this->groupBox6->Controls->Add(this->button8);
                    this->groupBox6->Location = System::Drawing::Point(195, 0);
                    this->groupBox6->Margin = System::Windows::Forms::Padding(4);
                    this->groupBox6->Name = L"groupBox6";
                    this->groupBox6->Padding = System::Windows::Forms::Padding(4);
                    this->groupBox6->Size = System::Drawing::Size(43, 75);
                    this->groupBox6->TabIndex = 26;
                    this->groupBox6->TabStop = false;
                    //
                    // button1
                    //
                    this->button1->FlatStyle =
System::Windows::Forms::FlatStyle::Popup;
                    this->button1->ForeColor =
System::Drawing::SystemColors::ActiveCaption;
                    this->button1->Location = System::Drawing::Point(12, 20);
                    this->button1->Margin = System::Windows::Forms::Padding(4);
                    this->button1->Name = L"button1";
                    this->button1->Size = System::Drawing::Size(23, 18);
                    this->button1->TabIndex = 25;
                    this->button1->UseVisualStyleBackColor = true;
                    this->button1->Click += gcnew System::EventHandler(this,
&mainWindow::button1_Click_1);
                    //
                    // button8
                    //
                    this->button8->FlatStyle =
System::Windows::Forms::FlatStyle::Popup;
                    this->button8->ForeColor =
System::Drawing::SystemColors::ActiveCaption;
                    this->button8->Location = System::Drawing::Point(12, 46);
                    this->button8->Margin = System::Windows::Forms::Padding(4);
                    this->button8->Name = L"button8";
                    this->button8->Size = System::Drawing::Size(23, 18);
                    this->button8->TabIndex = 21;
                    this->button8->UseVisualStyleBackColor = true;
                    this->button8->Click += gcnew System::EventHandler(this,
&mainWindow::button8_Click);
```

```
                        //
                        // label6
                        //
                        this->label6->AutoSize = true;
                        this->label6->Location = System::Drawing::Point(4, 59);
                        this->label6->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                        this->label6->Name = L"label6";
                        this->label6->Size = System::Drawing::Size(105, 17);
                        this->label6->TabIndex = 23;
                        this->label6->Text = L"Iterationsstufe: ";
                        this->label6->Click += gcnew System::EventHandler(this,
&mainWindow::label6_Click);
                        //
                        // Save
                        //
                        this->Save->Location = System::Drawing::Point(7, 171);
                        this->Save->Margin = System::Windows::Forms::Padding(4);
                        this->Save->Name = L"Save";
                        this->Save->Size = System::Drawing::Size(84, 30);
                        this->Save->TabIndex = 29;
                        this->Save->Text = L"Speichern";
                        this->Save->UseVisualStyleBackColor = true;
                        this->Save->Click += gcnew System::EventHandler(this,
&mainWindow::Save_Click);
                        //
                        // comboBox1
                        //
                        this->comboBox1->DropDownStyle =
System::Windows::Forms::ComboBoxStyle::DropDownList;
                        this->comboBox1->Location = System::Drawing::Point(8, 23);
                        this->comboBox1->Margin = System::Windows::Forms::Padding(4);
                        this->comboBox1->Name = L"comboBox1";
                        this->comboBox1->Size = System::Drawing::Size(181, 24);
                        this->comboBox1->TabIndex = 21;
                        this->toolTip1->SetToolTip(this->comboBox1, L"wählen sie hier ein
Fraktal");
                        this->comboBox1->SelectedIndexChanged += gcnew
System::EventHandler(this, &mainWindow::comboBox1_SelectedIndexChanged);
                        //
                        // contextMenuStrip1
                        //
                        this->contextMenuStrip1->Name = L"contextMenuStrip1";
                        this->contextMenuStrip1->Size = System::Drawing::Size(61, 4);
                        //
                        // timer1
                        //
                        this->timer1->Interval = 200;
                        this->timer1->Tick += gcnew System::EventHandler(this,
&mainWindow::timer1_Tick_1);
                        //
                        // label5
                        //
                        this->label5->AutoSize = true;
                        this->label5->Enabled = false;
                        this->label5->Font = (gcnew System::Drawing::Font(L"Arial",
21.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->label5->ForeColor = System::Drawing::Color::CadetBlue;
                        this->label5->Location = System::Drawing::Point(19, 9);
                        this->label5->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                        this->label5->Name = L"label5";
                        this->label5->Size = System::Drawing::Size(129, 42);
                        this->label5->TabIndex = 27;
                        this->label5->Text = L"Fraktal";
                        //
                        // statusStrip1
                        //
                        this->statusStrip1->BackColor =
System::Drawing::SystemColors::Control;
                        this->statusStrip1->Items->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(4) {this->toolStripProgressBar1,
                                this->toolStripDropDownButton1, this-
>toolStripDropDownButton2, this->toolStripStatusLabel1});
                        this->statusStrip1->Location = System::Drawing::Point(0, 693);
                        this->statusStrip1->Name = L"statusStrip1";
```

```
                    this->statusStrip1->Padding = System::Windows::Forms::Padding(1,
0, 19, 0);
                    this->statusStrip1->Size = System::Drawing::Size(1053, 26);
                    this->statusStrip1->TabIndex = 30;
                    this->statusStrip1->Text = L"statusStrip1";
                    //
                    // toolStripProgressBar1
                    //
                    this->toolStripProgressBar1->Name = L"toolStripProgressBar1";
                    this->toolStripProgressBar1->Size = System::Drawing::Size(133,
20);
                    //
                    // toolStripDropDownButton1
                    //
                    this->toolStripDropDownButton1->DisplayStyle =
System::Windows::Forms::ToolStripItemDisplayStyle::Text;
                    this->toolStripDropDownButton1->DropDownItems->AddRange(gcnew
cli::array< System::Windows::Forms::ToolStripItem^  >(2) {this->ausToolStripMenuItem,
                            this->anToolStripMenuItem});
                    this->toolStripDropDownButton1->ImageTransparentColor =
System::Drawing::Color::Magenta;
                    this->toolStripDropDownButton1->Name =
L"toolStripDropDownButton1";
                    this->toolStripDropDownButton1->Size = System::Drawing::Size(88,
24);
                    this->toolStripDropDownButton1->Text = L"Fortschritt";
                    //
                    // ausToolStripMenuItem
                    //
                    this->ausToolStripMenuItem->DisplayStyle =
System::Windows::Forms::ToolStripItemDisplayStyle::Text;
                    this->ausToolStripMenuItem->Name = L"ausToolStripMenuItem";
                    this->ausToolStripMenuItem->Size = System::Drawing::Size(100, 24);
                    this->ausToolStripMenuItem->Text = L"aus";
                    this->ausToolStripMenuItem->ToolTipText = L"Ausschalten erhöht die
Performance , allerdings ist der Fortschritt nicht mehr si"
                            L"chtbar";
                    this->ausToolStripMenuItem->Click += gcnew
System::EventHandler(this, &mainWindow::ausToolStripMenuItem_Click);
                    //
                    // anToolStripMenuItem
                    //
                    this->anToolStripMenuItem->Checked = true;
                    this->anToolStripMenuItem->CheckState =
System::Windows::Forms::CheckState::Checked;
                    this->anToolStripMenuItem->DisplayStyle =
System::Windows::Forms::ToolStripItemDisplayStyle::Text;
                    this->anToolStripMenuItem->Name = L"anToolStripMenuItem";
                    this->anToolStripMenuItem->Size = System::Drawing::Size(100, 24);
                    this->anToolStripMenuItem->Text = L"ein";
                    this->anToolStripMenuItem->Click += gcnew
System::EventHandler(this, &mainWindow::anToolStripMenuItem_Click);
                    //
                    // toolStripDropDownButton2
                    //
                    this->toolStripDropDownButton2->Alignment =
System::Windows::Forms::ToolStripItemAlignment::Right;
                    this->toolStripDropDownButton2->BorderSides =
System::Windows::Forms::ToolStripStatusLabelBorderSides::Right;
                    this->toolStripDropDownButton2->DisplayStyle =
System::Windows::Forms::ToolStripItemDisplayStyle::Image;
                    this->toolStripDropDownButton2->ImageTransparentColor =
System::Drawing::Color::Magenta;
                    this->toolStripDropDownButton2->Margin =
System::Windows::Forms::Padding(0);
                    this->toolStripDropDownButton2->Name =
L"toolStripDropDownButton2";
                    this->toolStripDropDownButton2->Size = System::Drawing::Size(4,
26);
                    this->toolStripDropDownButton2->Text = L"Button2";
                    //
                    // toolStripStatusLabel1
                    //
                    this->toolStripStatusLabel1->Name = L"toolStripStatusLabel1";
                    this->toolStripStatusLabel1->Size = System::Drawing::Size(107,
21);
                    this->toolStripStatusLabel1->Text = L"Status Anzeige";
                    //
```

```
                        // label7
                        //
                        this->label7->AutoSize = true;
                        this->label7->Location = System::Drawing::Point(641, 14);
                        this->label7->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                        this->label7->Name = L"label7";
                        this->label7->Size = System::Drawing::Size(149, 17);
                        this->label7->TabIndex = 31;
                        this->label7->Text = L"LABEL , SPEED TEST";
                        this->label7->Visible = false;
                        this->label7->Click += gcnew System::EventHandler(this,
&mainWindow::label7_Click);
                        //
                        // groupBox3
                        //
                        this->groupBox3->Controls->Add(this->checkBox5);
                        this->groupBox3->Controls->Add(this->pictureBox1);
                        this->groupBox3->Controls->Add(this->groupBox4);
                        this->groupBox3->Controls->Add(this->listBox2);
                        this->groupBox3->Controls->Add(this->label11);
                        this->groupBox3->FlatStyle =
System::Windows::Forms::FlatStyle::Popup;
                        this->groupBox3->Location = System::Drawing::Point(256, 80);
                        this->groupBox3->Margin = System::Windows::Forms::Padding(4);
                        this->groupBox3->Name = L"groupBox3";
                        this->groupBox3->Padding = System::Windows::Forms::Padding(4);
                        this->groupBox3->Size = System::Drawing::Size(239, 184);
                        this->groupBox3->TabIndex = 32;
                        this->groupBox3->TabStop = false;
                        this->groupBox3->Text = L"Iterated Function Systems";
                        this->groupBox3->Enter += gcnew System::EventHandler(this,
&mainWindow::groupBox3_Enter);
                        //
                        // pictureBox1
                        //
                        this->pictureBox1->Location = System::Drawing::Point(132, 41);
                        this->pictureBox1->Margin = System::Windows::Forms::Padding(4);
                        this->pictureBox1->Name = L"pictureBox1";
                        this->pictureBox1->Size = System::Drawing::Size(99, 66);
                        this->pictureBox1->TabIndex = 34;
                        this->pictureBox1->TabStop = false;
                        //
                        // groupBox4
                        //
                        this->groupBox4->Controls->Add(this->ifs_settings);
                        this->groupBox4->Controls->Add(this->ifs_load);
                        this->groupBox4->Controls->Add(this->ifs_ok);
                        this->groupBox4->Controls->Add(this->ifs_abort);
                        this->groupBox4->Location = System::Drawing::Point(10, 133);
                        this->groupBox4->Margin = System::Windows::Forms::Padding(4);
                        this->groupBox4->Name = L"groupBox4";
                        this->groupBox4->Padding = System::Windows::Forms::Padding(4);
                        this->groupBox4->Size = System::Drawing::Size(223, 43);
                        this->groupBox4->TabIndex = 25;
                        this->groupBox4->TabStop = false;
                        //
                        // ifs_settings
                        //
                        this->ifs_settings->Enabled = false;
                        this->ifs_settings->Location = System::Drawing::Point(4, 12);
                        this->ifs_settings->Margin = System::Windows::Forms::Padding(4);
                        this->ifs_settings->Name = L"ifs_settings";
                        this->ifs_settings->Size = System::Drawing::Size(24, 26);
                        this->ifs_settings->TabIndex = 26;
                        this->ifs_settings->Text = L"#";
                        this->ifs_settings->UseVisualStyleBackColor = true;
                        //
                        // ifs_load
                        //
                        this->ifs_load->Location = System::Drawing::Point(27, 12);
                        this->ifs_load->Margin = System::Windows::Forms::Padding(4);
                        this->ifs_load->Name = L"ifs_load";
                        this->ifs_load->Size = System::Drawing::Size(67, 26);
                        this->ifs_load->TabIndex = 16;
                        this->ifs_load->Text = L"Laden";
                        this->ifs_load->UseVisualStyleBackColor = true;
```

```
                        this->ifs_load->Click += gcnew System::EventHandler(this,
&mainWindow::ifs_load_Click);
                        //
                        // ifs_ok
                        //
                        this->ifs_ok->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"ifs_ok.BackgroundImage")));
                        this->ifs_ok->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->ifs_ok->Location = System::Drawing::Point(124, 14);
                        this->ifs_ok->Margin = System::Windows::Forms::Padding(4);
                        this->ifs_ok->Name = L"ifs_ok";
                        this->ifs_ok->Size = System::Drawing::Size(28, 22);
                        this->ifs_ok->TabIndex = 4;
                        this->ifs_ok->UseVisualStyleBackColor = true;
                        this->ifs_ok->Click += gcnew System::EventHandler(this,
&mainWindow::ifs_ok_Click);
                        //
                        // ifs_abort
                        //
                        this->ifs_abort->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"ifs_abort.BackgroundImage")));
                        this->ifs_abort->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->ifs_abort->Location = System::Drawing::Point(155, 14);
                        this->ifs_abort->Margin = System::Windows::Forms::Padding(4);
                        this->ifs_abort->Name = L"ifs_abort";
                        this->ifs_abort->Size = System::Drawing::Size(25, 22);
                        this->ifs_abort->TabIndex = 8;
                        this->ifs_abort->UseVisualStyleBackColor = true;
                        this->ifs_abort->Click += gcnew System::EventHandler(this,
&mainWindow::ifs_abort_Click);
                        //
                        // listBox2
                        //
                        this->listBox2->ItemHeight = 16;
                        this->listBox2->Location = System::Drawing::Point(12, 41);
                        this->listBox2->Margin = System::Windows::Forms::Padding(4);
                        this->listBox2->Name = L"listBox2";
                        this->listBox2->Size = System::Drawing::Size(111, 68);
                        this->listBox2->TabIndex = 7;
                        this->listBox2->SelectedIndexChanged += gcnew
System::EventHandler(this, &mainWindow::listBox2_SelectedIndexChanged);
                        //
                        // label11
                        //
                        this->label11->AutoSize = true;
                        this->label11->Location = System::Drawing::Point(8, 21);
                        this->label11->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                        this->label11->Name = L"label11";
                        this->label11->Size = System::Drawing::Size(62, 17);
                        this->label11->TabIndex = 13;
                        this->label11->Text = L"Matrizen";
                        //
                        // comboBox2
                        //
                        this->comboBox2->DropDownStyle =
System::Windows::Forms::ComboBoxStyle::DropDownList;
                        this->comboBox2->Location = System::Drawing::Point(11, 292);
                        this->comboBox2->Margin = System::Windows::Forms::Padding(4);
                        this->comboBox2->Name = L"comboBox2";
                        this->comboBox2->Size = System::Drawing::Size(181, 24);
                        this->comboBox2->TabIndex = 33;
                        this->comboBox2->SelectedIndexChanged += gcnew
System::EventHandler(this, &mainWindow::comboBox2_SelectedIndexChanged);
                        //
                        // saveFileDialog1
                        //
                        this->saveFileDialog1->Filter = L"Bitmap-Dateien|*.bmp|Alle
Dateien|*.*";
                        this->saveFileDialog1->Title = L"Bild Speichern unter . . ";
                        //
                        // OpenInitiator
                        //
                        this->OpenInitiator->Filter = L"Bilder|*.bmp;*.jpg;*.jpeg;*.gif";
```

```
                        //
                        // groupBox7
                        //
                        this->groupBox7->Controls->Add(this->pictureBox2);
                        this->groupBox7->Controls->Add(this->listBox3);
                        this->groupBox7->Controls->Add(this->label13);
                        this->groupBox7->Controls->Add(this->label14);
                        this->groupBox7->Controls->Add(this->label15);
                        this->groupBox7->Controls->Add(this->ifs_vy);
                        this->groupBox7->Controls->Add(this->ifs_scale);
                        this->groupBox7->Controls->Add(this->ifs_rotate);
                        this->groupBox7->Controls->Add(this->ifs_vx);
                        this->groupBox7->Controls->Add(this->groupBox8);
                        this->groupBox7->FlatStyle =
System::Windows::Forms::FlatStyle::Popup;
                        this->groupBox7->Location = System::Drawing::Point(256, 292);
                        this->groupBox7->Margin = System::Windows::Forms::Padding(4);
                        this->groupBox7->Name = L"groupBox7";
                        this->groupBox7->Padding = System::Windows::Forms::Padding(4);
                        this->groupBox7->Size = System::Drawing::Size(239, 240);
                        this->groupBox7->TabIndex = 34;
                        this->groupBox7->TabStop = false;
                        this->groupBox7->Text = L"Iterated Function Systems";
                        this->groupBox7->Visible = false;
                        //
                        // pictureBox2
                        //
                        this->pictureBox2->Location = System::Drawing::Point(132, 22);
                        this->pictureBox2->Margin = System::Windows::Forms::Padding(4);
                        this->pictureBox2->Name = L"pictureBox2";
                        this->pictureBox2->Size = System::Drawing::Size(99, 66);
                        this->pictureBox2->TabIndex = 35;
                        this->pictureBox2->TabStop = false;
                        //
                        // listBox3
                        //
                        this->listBox3->ItemHeight = 16;
                        this->listBox3->Location = System::Drawing::Point(11, 22);
                        this->listBox3->Margin = System::Windows::Forms::Padding(4);
                        this->listBox3->Name = L"listBox3";
                        this->listBox3->Size = System::Drawing::Size(112, 68);
                        this->listBox3->TabIndex = 33;
                        this->listBox3->SelectedIndexChanged += gcnew
System::EventHandler(this, &mainWindow::listBox3_SelectedIndexChanged);
                        //
                        // label13
                        //
                        this->label13->AutoSize = true;
                        this->label13->Location = System::Drawing::Point(135, 165);
                        this->label13->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                        this->label13->Name = L"label13";
                        this->label13->Size = System::Drawing::Size(73, 17);
                        this->label13->TabIndex = 32;
                        this->label13->Text = L"skalierung";
                        //
                        // label14
                        //
                        this->label14->AutoSize = true;
                        this->label14->Location = System::Drawing::Point(135, 131);
                        this->label14->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                        this->label14->Name = L"label14";
                        this->label14->Size = System::Drawing::Size(61, 17);
                        this->label14->TabIndex = 31;
                        this->label14->Text = L"drehung";
                        //
                        // label15
                        //
                        this->label15->AutoSize = true;
                        this->label15->Location = System::Drawing::Point(135, 93);
                        this->label15->Margin = System::Windows::Forms::Padding(4, 0, 4,
0);
                        this->label15->Name = L"label15";
                        this->label15->Size = System::Drawing::Size(93, 17);
                        this->label15->TabIndex = 30;
                        this->label15->Text = L"verschiebung";
                        //
```

```
                    // ifs_vy
                    //
                    this->ifs_vy->Location = System::Drawing::Point(58, 98);
                    this->ifs_vy->Margin = System::Windows::Forms::Padding(4);
                    this->ifs_vy->Name = L"ifs_vy";
                    this->ifs_vy->Size = System::Drawing::Size(40, 22);
                    this->ifs_vy->TabIndex = 29;
                    this->ifs_vy->Text = L"1";
                    this->ifs_vy->TextChanged += gcnew System::EventHandler(this,
&mainWindow::ifs_vy_TextChanged);
                    //
                    // ifs_scale
                    //
                    this->ifs_scale->Location = System::Drawing::Point(11, 160);
                    this->ifs_scale->Margin = System::Windows::Forms::Padding(4);
                    this->ifs_scale->Name = L"ifs_scale";
                    this->ifs_scale->Size = System::Drawing::Size(85, 22);
                    this->ifs_scale->TabIndex = 28;
                    this->ifs_scale->Text = L"0,5";
                    this->ifs_scale->TextChanged += gcnew System::EventHandler(this,
&mainWindow::ifs_scale_TextChanged);
                    //
                    // ifs_rotate
                    //
                    this->ifs_rotate->Location = System::Drawing::Point(11, 128);
                    this->ifs_rotate->Margin = System::Windows::Forms::Padding(4);
                    this->ifs_rotate->Name = L"ifs_rotate";
                    this->ifs_rotate->Size = System::Drawing::Size(85, 22);
                    this->ifs_rotate->TabIndex = 27;
                    this->ifs_rotate->Text = L"0";
                    this->ifs_rotate->TextChanged += gcnew System::EventHandler(this,
&mainWindow::ifs_rotate_TextChanged);
                    //
                    // ifs_vx
                    //
                    this->ifs_vx->Location = System::Drawing::Point(10, 98);
                    this->ifs_vx->Margin = System::Windows::Forms::Padding(4);
                    this->ifs_vx->Name = L"ifs_vx";
                    this->ifs_vx->Size = System::Drawing::Size(40, 22);
                    this->ifs_vx->TabIndex = 26;
                    this->ifs_vx->Text = L"1";
                    this->ifs_vx->TextChanged += gcnew System::EventHandler(this,
&mainWindow::ifs_vx_TextChanged);
                    //
                    // groupBox8
                    //
                    this->groupBox8->Controls->Add(this->ifs2_load);
                    this->groupBox8->Controls->Add(this->ifs2_ok);
                    this->groupBox8->Controls->Add(this->ifs2_cancel);
                    this->groupBox8->Location = System::Drawing::Point(11, 186);
                    this->groupBox8->Margin = System::Windows::Forms::Padding(4);
                    this->groupBox8->Name = L"groupBox8";
                    this->groupBox8->Padding = System::Windows::Forms::Padding(4);
                    this->groupBox8->Size = System::Drawing::Size(223, 43);
                    this->groupBox8->TabIndex = 25;
                    this->groupBox8->TabStop = false;
                    //
                    // ifs2_load
                    //
                    this->ifs2_load->Enabled = false;
                    this->ifs2_load->Location = System::Drawing::Point(8, 12);
                    this->ifs2_load->Margin = System::Windows::Forms::Padding(4);
                    this->ifs2_load->Name = L"ifs2_load";
                    this->ifs2_load->Size = System::Drawing::Size(67, 26);
                    this->ifs2_load->TabIndex = 16;
                    this->ifs2_load->Text = L"Laden";
                    this->ifs2_load->UseVisualStyleBackColor = true;
                    this->ifs2_load->Click += gcnew System::EventHandler(this,
&mainWindow::ifs2_load_Click);
                    //
                    // ifs2_ok
                    //
                    this->ifs2_ok->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"ifs2_ok.BackgroundImage")));
                    this->ifs2_ok->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                    this->ifs2_ok->Location = System::Drawing::Point(159, 13);
```

```
                    this->ifs2_ok->Margin = System::Windows::Forms::Padding(4);
                    this->ifs2_ok->Name = L"ifs2_ok";
                    this->ifs2_ok->Size = System::Drawing::Size(28, 22);
                    this->ifs2_ok->TabIndex = 4;
                    this->ifs2_ok->UseVisualStyleBackColor = true;
                    this->ifs2_ok->Click += gcnew System::EventHandler(this,
&mainWindow::button6_Click);
                    //
                    // ifs2_cancel
                    //
                    this->ifs2_cancel->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"ifs2_cancel.BackgroundImage")));
                    this->ifs2_cancel->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                    this->ifs2_cancel->Location = System::Drawing::Point(190, 13);
                    this->ifs2_cancel->Margin = System::Windows::Forms::Padding(4);
                    this->ifs2_cancel->Name = L"ifs2_cancel";
                    this->ifs2_cancel->Size = System::Drawing::Size(25, 22);
                    this->ifs2_cancel->TabIndex = 8;
                    this->ifs2_cancel->UseVisualStyleBackColor = true;
                    this->ifs2_cancel->Click += gcnew System::EventHandler(this,
&mainWindow::ifs2_cancel_Click);
                    //
                    // checkBox5
                    //
                    this->checkBox5->AutoSize = true;
                    this->checkBox5->Location = System::Drawing::Point(14, 117);
                    this->checkBox5->Name = L"checkBox5";
                    this->checkBox5->Size = System::Drawing::Size(86, 21);
                    this->checkBox5->TabIndex = 35;
                    this->checkBox5->Text = L"Morphen";
                    this->checkBox5->UseVisualStyleBackColor = true;
                    this->checkBox5->CheckedChanged += gcnew
System::EventHandler(this, &mainWindow::checkBox5_CheckedChanged);
                    //
                    // mainWindow
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
                    this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                    this->BackColor = System::Drawing::Color::Silver;
                    this->ClientSize = System::Drawing::Size(1053, 719);
                    this->Controls->Add(this->label7);
                    this->Controls->Add(this->statusStrip1);
                    this->Controls->Add(this->groupBox7);
                    this->Controls->Add(this->label5);
                    this->Controls->Add(this->groupBox2);
                    this->Controls->Add(this->comboBox2);
                    this->Controls->Add(this->groupBox3);
                    this->Controls->Add(this->groupBox1);
                    this->Icon = (cli::safe_cast<System::Drawing::Icon^  >(resources-
>GetObject(L"$this.Icon")));
                    this->Margin = System::Windows::Forms::Padding(4);
                    this->Name = L"mainWindow";
                    this->Text = L"Fraktal-Programm";
                    this->Paint += gcnew
System::Windows::Forms::PaintEventHandler(this, &mainWindow::mainWindow_Paint);
                    this->Click += gcnew System::EventHandler(this,
&mainWindow::mainWindow_Click);
                    this->MouseMove += gcnew
System::Windows::Forms::MouseEventHandler(this, &mainWindow::mainWindow_MouseMove);
                    this->KeyDown += gcnew
System::Windows::Forms::KeyEventHandler(this, &mainWindow::mainWindow_KeyDown);
                    this->MouseDown += gcnew
System::Windows::Forms::MouseEventHandler(this, &mainWindow::mainWindow_MouseDown);
                    this->Load += gcnew System::EventHandler(this,
&mainWindow::mainWindow_Load);
                    this->groupBox1->ResumeLayout(false);
                    this->groupBox1->PerformLayout();
                    this->groupBox5->ResumeLayout(false);
                    this->groupBox5->PerformLayout();
                    this->groupBox2->ResumeLayout(false);
                    this->groupBox2->PerformLayout();
                    (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->numericUpDown2))->EndInit();
                    (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->numericUpDown1))->EndInit();
                    this->groupBox6->ResumeLayout(false);
```

```
                        this->statusStrip1->ResumeLayout(false);
                        this->statusStrip1->PerformLayout();
                        this->groupBox3->ResumeLayout(false);
                        this->groupBox3->PerformLayout();
                        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox1))->EndInit();
                        this->groupBox4->ResumeLayout(false);
                        this->groupBox7->ResumeLayout(false);
                        this->groupBox7->PerformLayout();
                        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox2))->EndInit();
                        this->groupBox8->ResumeLayout(false);
                        this->ResumeLayout(false);
                        this->PerformLayout();

                }
                property String^ Status{
                        void set(String^ status ){toolStripStatusLabel1->Text = status;}
                        String^ get(){return toolStripStatusLabel1->Text;}
                }
#pragma endregion
        private: bool checkrule(String^ str){
                                if(str==""){MessageBox::Show("Geben sie eine Regel
ein...");return 0;}
                                if(str->IndexOf(':')!=1){MessageBox::Show("Falscher Syntax
, geben sie bitte die Regeln nach dem Muster\n \"X:YZ\" ein um X durch YZ zu ersetzen\n
");return 0;}
                                for (int i = 0;i<this->listBox1->Items->Count;i++)
                                {
                                        if(str[0]==this->listBox1->Items[i]-
>ToString()[0]){MessageBox::Show("Dieses Symbol wurde schon ersetzt (D0L-
System)");return 0;}

                                }
                  return 1;

                         }
        private: void starthread(Graphics^g)
                {
                        if(fraktal->points->Length-fraktal->neededPoints>=0)
                        {
                                this->timer1->Enabled = 1;this->toolStripDropDownButton2-
>Image = resourcen[0]->LoadPicture();
                                try
                                {
                                        if(!(OpenInitiator->FileName==""))
                                        {fraktal-
>paint(g,(Bitmap^)Bitmap::FromFile(OpenInitiator->FileName));}
                                        else fraktal->paint(g);
                                }
                                catch (Exception^e)
                                {
                                        Status = e->Message;
                                }

                    }
                }
        private: void set_label(){

////////////////////////////////////////////////////////////////////////
                                /// Beschriftungen werden aktualisiert (z.B.wenn Fraktal
geändert)

////////////////////////////////////////////////////////////////////////
                                this->label5->Text = fraktal->name;
                                if(fraktal->neededPoints-fraktal->points->Length>0)this-
>label5->Text+=" : Bitte noch " +System::Convert::ToString(fraktal->neededPoints-
fraktal->points->Length)+" Punkt(e) angeben";
                                this->comboBox1->Text = fraktal->name;
                                this->Text = "Fraktale - "+this->label5->Text;
                        }
        private: System::Void button1_Click(System::Object^  sender, System::EventArgs^
e) {
                        }
        private: System::Void mainWindow_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e) {
                                /// Invalidate ...
                                Graphics^ p = e->Graphics;
```

```
                                if(fraktal->n>0)
                                {
                                        p->DrawImage(bmp_disp,0,0);
                                 try
                                   {
                                         Status = fraktal->status;
                                         double tmp = 100 * fraktal->progress();
                                         this->toolStripProgressBar1->Value = int(tmp);
                                   }
                                 catch (System::Exception^ error)
                                   {
                                         Status = error->Message;
#ifdef _DEBUG
                                         MessageBox::Show("Fehler: "+error->Message);
#endif // _DEBUG
                                   }
                                }
                                else
                                {
                                        fraktal-
>setmousepos(PointToClient(Point(MousePosition.X,MousePosition.Y)));
                                  fraktal->paint(p);
                                }
                        }
        private: System::Void button2_Click(System::Object^  sender, System::EventArgs^
e) {
                                        fraktal->suspendthread();
                        }
private: System::Void mainWindow_Click(System::Object^  sender, System::EventArgs^  e)
                {

                }
private: System::Void mainWindow_MouseDown(System::Object^  sender,
System::Windows::Forms::MouseEventArgs^  e)
                {
                        /// Mausklick
                        set_label();
                        if(e->Button==System::Windows::Forms::MouseButtons::Left)
                        {
                                if(fraktal->n>0){
                                numericUpDown1->Value++;
                                }
                                else fraktal-
>addpoint(PointToClient(Point(MousePosition.X,MousePosition.Y)));
                        }
                        if(e->Button==System::Windows::Forms::MouseButtons::Right)
                        {
                                if(fraktal->n>1)
                                { numericUpDown1->Value--; }
                                else{{if(fraktal->n==1){fraktal->n--;}else fraktal-
>subtractpoint();}}
                        }
                        this->numericUpDown1->Value = fraktal->n;
         };
private: System::Void lsys_ok_Click(System::Object^  sender, System::EventArgs^  e) {
             if(checkrule(lsys_rule->Text))
                        {
                                String^ buf = lsys_rule->Text;
                 dynamic_cast<LSystem^>(fraktal)->addrule(buf->Substring(2),buf[0]);
                        }
                        LSystemBoxes_refresh();
                //Item hinzufügen zur Liste
                        //this->listBox1->Items->AddRange(gcnew cli::array<
System::Object^  >(1) {L""});
                        //this->listBox1->Items[this->listBox1->Items->Count-1] =
mainWindow::textBox1->Text;
                }
String^ read_word(IO::StreamReader^ strReader)
                {
                        Char buffer;String^ string;
                        do
                        {
                                buffer=strReader->Read();
                        } while(!(strReader->EndOfStream) && ( (buffer == ' ')  ||
(buffer == '\r') || (buffer == '\n')));
                        while (!(strReader->EndOfStream) && !(buffer == ' ') && !(buffer
== '\r') && !(buffer == '\n'))
                        {
```

```
                                string = String::Concat(string,buffer);
                                buffer = strReader->Read();
                        }
                        if(strReader->EndOfStream){string =
String::Concat(string,buffer);}
                        return string;
                }
private: System::Void SurfaceToData(){
                        /// Die Einstellungen die an der Benutzeroberfläche gemacht
wurden werden
                        /// in eine Instanz der Klasse Rule (Daten zum erzeugen
                        /// des LSystems, ersetzungsregeln, etc.) eingetragen
                                try
                                {
                                        dynamic_cast<LSystem^>(fraktal)->reset_rules();
                                        for(int j = 0;j<this->listBox1->Items->Count;j++)
                                        {
                          String^ str = this->listBox1->Items[j]->ToString();
                                                dynamic_cast<LSystem^>(fraktal)-
>addrule(str->Substring(2),str[0]);
                                        }
                                        /// Axiom ist in TextBox2
                                        dynamic_cast<LSystem^>(fraktal)->axiom=this-
>textBox2->Text->ToString();
                                        /// winkel + in double umwandeln
                                        dynamic_cast<LSystem^>(fraktal)->add_winkel =
Convert::ToDouble(this->textBox3->Text->ToString());
                                }
                                catch (System::Exception^ error)
                                {
                                        Status = error->Message;
                                }
                }
private: System::Void LSystemBoxes_refresh(){
                        /// Oberfläche wird an die aktualisierten LSystem Daten angepasst
                        /// (z.B.beim laden von L-System Dateien)

if(LoadedFractal!=nullptr&&dynamic_cast<array<LSystem^>^>(LoadedFractal)!=nullptr){
                        this->textBox2->Text = dynamic_cast<LSystem^>(fraktal)->axiom;
                        this->textBox3->Text =
Convert::ToString(dynamic_cast<LSystem^>(fraktal)->add_winkel);
                        this->listBox1->Items->Clear();
                        for(int j = 0;j<dynamic_cast<LSystem^>(LoadedFractal[comboBox2-
>SelectedIndex])->symbol->Length;j++)
                        {
                                this->listBox1->Items-
>Add(dynamic_cast<LSystem^>(LoadedFractal[comboBox2->SelectedIndex])->symbol[j] + ":" +
dynamic_cast<LSystem^>(LoadedFractal[comboBox2->SelectedIndex])->substitution[j]);
                        }
                        }
                }
private: System::Void textBox1_TextChanged(System::Object^  sender, System::EventArgs^
e) {
                }
private: System::Void mainWindow_KeyDown(System::Object^  sender,
System::Windows::Forms::KeyEventArgs^  e) {

                }
private: System::Void listBox1_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                }
private: System::Void button5_Click(System::Object^  sender, System::EventArgs^  e) {
                        /// delete rule ...
                        if(this->listBox1->Items->Count>0){
                        this->listBox1->Items->RemoveAt(this->listBox1->Items->Count-1);
                        SurfaceToData();
                        }
                }

private: System::Void button7_Click(System::Object^  sender, System::EventArgs^  e) {
                        /// Load Funktion
                        try
                        {
                                FileManagment^ f = gcnew FileManagment("");
                                    f->Warnings = 0;
                                    f->FileTyp = "lsystem";
                                    f->Name = "L-System";
                                    LoadedFractal = f->LoadFractals(gcnew LSystem);
```

```
                                    if(LoadedFractal==nullptr||LoadedFractal-
>Length==0){MessageBox::Show("Diese Datei enthält keine L-Systeme");return;}
                                    ComboBox2_fill(LoadedFractal);
                                    LSystemBoxes_refresh();
                                    change_frac(LoadedFractal[0]);
                                }
                        catch (System::Exception^ error)
                        {
                                MessageBox::Show( error->Message, "Fehler",
                                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation
);
                        }

                }
private: System::Void groupBox2_Enter(System::Object^  sender, System::EventArgs^  e) {
                }
private: System::Void groupBox1_Enter(System::Object^  sender, System::EventArgs^  e) {
                }
private: System::Void comboBox1_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                        /// Combo Box Fraktal Auswahl
             set_mode(this->comboBox1->SelectedIndex);
                }
private: System::Void set_mode(int n){
                        if(fraktal==nullptr){fraktal=gcnew Kochkurve;}
                        groupBox1->Visible = 0;
                        groupBox3->Visible = 0;
                        groupBox7->Visible = 0;
                        comboBox2->Visible = 0;
                        change_frac(fraktal->change_mode(n));

                       if(dynamic_cast<LSystem^>(fraktal)!=nullptr)
                       {  // wenn fraktal vom typ Lsystem ist dann wird lsystem.lsystem
geladen
                           LoadedFractal = resourcen[2]->LoadFractals(gcnew LSystem);
                           ComboBox2_fill(LoadedFractal);
                           LSystemBoxes_refresh();
                           groupBox1->Visible = 1;
                           comboBox2->Visible = 1;
                        }
                       if(dynamic_cast<IFS^>(fraktal)!=nullptr)
                       {  // wenn fraktal vom typ IFS ist dann wird fraktale.ifs geladen
                              LoadedFractal = resourcen[3]->LoadFractals(gcnew IFS);
                              ComboBox2_fill(LoadedFractal);
                              Matrices_Box_refresh();
                              groupBox3->Location= groupBox1->Location;
                              groupBox3->Visible = 1;
                              comboBox2->Visible = 1;
                       }
                       if(dynamic_cast<IFS_2^>(fraktal)!=nullptr)
                       {  // wenn fraktal vom typ IFS_2 ist dann wird fraktale.ifs
geladen
                              listBox3->Items->Clear();
                              listBox3->Items->AddRange(dynamic_cast<IFS_2^>(fraktal)-
>list());
                              groupBox7->Location= groupBox1->Location;
                              groupBox7->Visible = 1;
                              ifs2_preview();
                       }
                       fraktal->draw_info = 1;
                }
private: System::Void change_frac(Fraktal^f){
                fraktal->stopthread();
                fraktal = f;
                numericUpDown2->Value = fraktal->neededPoints;
                fraktal->random = checkBox1->Checked;
                fraktal->front = FrontColorDialog->Color;}
private: System::Void timer1_Tick_1(System::Object^  sender, System::EventArgs^  e) {
                        // L_System-Zeichenkette anzeigen
         if(dynamic_cast<LSystem^>(fraktal)!=nullptr&&checkBox4->Checked){textBox1-
>Text=dynamic_cast<LSystem^>(fraktal)->gesamtkette;}
             //
                        if(!fraktal->ThreadIsAlive)
                        {this->Invalidate();this->timer1->Enabled = 0;this-
>toolStripDropDownButton2->Image = resourcen[0]->LoadPicture();return;}
                        if(this->anToolStripMenuItem->Checked)
                        {this->Invalidate();}
                }
```

```
private: System::Void textBox2_TextChanged(System::Object^  sender, System::EventArgs^
e) {
                        dynamic_cast<LSystem^>(fraktal)->axiom = textBox2->Text;
                }
private: System::Void numericUpDown1_ValueChanged(System::Object^  sender,
System::EventArgs^  e) {
                        // n wird über die Oberfläche geändert ...
                        fraktal->n =  System::Convert::ToInt32(this->numericUpDown1-
>Value) ;

starthread(Graphics::FromImage(bmp_disp)/*Graphics::FromHwnd(this->Handle)*/);
                }
private: System::Void label6_Click(System::Object^  sender, System::EventArgs^  e) {
                }
private: System::Void mainWindow_MouseMove(System::Object^  sender,
System::Windows::Forms::MouseEventArgs^  e) {
         if (fraktal->n==0&&fraktal->points->Length>1)
         {
                        this->Invalidate();
         }

                }
private: System::Void button1_Click_1(System::Object^  sender, System::EventArgs^  e) {
                        this->FrontColorDialog->ShowDialog();
                        this->button1->BackColor = FrontColorDialog->Color;
                        fraktal->front = FrontColorDialog->Color;
                }
private: System::Void button8_Click(System::Object^  sender, System::EventArgs^  e) {
                        this->BackColorDialog->ShowDialog();
                        this->BackColor = BackColorDialog->Color;
                        this->button8->BackColor = BackColorDialog->Color;
                }
private: System::Void mainWindow_Load(System::Object^  sender, System::EventArgs^  e) {
                }
private: System::Void checkBox1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        fraktal->random = this->checkBox1->Checked;
                }
private: System::Void checkBox2_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                }
private: System::Void Save_Click(System::Object^  sender, System::EventArgs^  e) {
                        if(MessageBox::Show("Achtung, Fraktal muss in dieser Version
andere Farbe als schwarz haben, sonst ist es nicht sichtbar (Hintergrund ist momentan
immer schwarz)","Hinweis"

,MessageBoxButtons::OKCancel,MessageBoxIcon::Information)==::DialogResult::OK){
                        if(this->saveFileDialog1-
>ShowDialog()==Windows::Forms::DialogResult::OK){
                                bmp_disp->Save(this->saveFileDialog1-
>FileName,System::Drawing::Imaging::ImageFormat::Bmp);
                        } }
                }
private: System::Void pictureBox1_Click(System::Object^  sender, System::EventArgs^  e)
{
                }
private: System::Void label7_Click(System::Object^  sender, System::EventArgs^  e) {
                        MessageBox::Show("Deaktiviert...");
                }
private: System::Void anToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {

////////////////////////////////////////////////////////////////////////
                        ///   Fortschrittsanzeige einschalten

////////////////////////////////////////////////////////////////////////
                        anToolStripMenuItem->Checked=1;
                        ausToolStripMenuItem->Checked=0;
                }
private: System::Void ausToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {

////////////////////////////////////////////////////////////////////////
                        ///   Fortschrittsanzeige ausschalten

////////////////////////////////////////////////////////////////////////
                        anToolStripMenuItem->Checked=0;
                        ausToolStripMenuItem->Checked=1;
```

```
                }
private: System::Void textBox3_TextChanged(System::Object^  sender, System::EventArgs^
e) {
                }
private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e) {

/////////////////////////////////////////////////////////////////////////
                 ///   Button für erweiterte Einstellungen des L-Systems
(lsystem_settings)

/////////////////////////////////////////////////////////////////////////
                  lsystem_settings^ settings = gcnew lsystem_settings();
                  settings->lsystem =
dynamic_cast<LSystem^>(LoadedFractal[comboBox2->SelectedIndex]);
                  settings->Show();
                }
private: System::Void groupBox3_Enter(System::Object^  sender, System::EventArgs^  e) {
                }
private: System::Void textBox5_TextChanged(System::Object^  sender, System::EventArgs^
e) {
                }
private: System::Void Matrices_Box_refresh()
                {
            ////////////////////////////////////////////////////////////////////////
                 ///   IFS_ListBox (listet die Matrizen auf) wird aktualisiert

/////////////////////////////////////////////////////////////////////////
                 /// prüfe ob array LoadedFractal dem Typ IFS entspricht (es
befinden sich nur ifs Fraktale im array)
                 /// wenn dies wahr ist , dann werden die Matrizen des ausgewählten
IFS aufgelistet

if(LoadedFractal!=nullptr&&dynamic_cast<array<IFS^>^>(LoadedFractal)!=nullptr){
                  listBox2->Items->Clear();
                 /// downcast, nötig, da geladene Fraktale upgecastet werden damit
der Programmcode allgemein bleibt...
                  listBox2->Items-
>AddRange(dynamic_cast<IFS^>(LoadedFractal[comboBox2->SelectedIndex])->list_matrices());
                 ifs_preview();}
                }
private: System::Void ifs_preview(){
                  Bitmap^b = gcnew Bitmap(pictureBox1->Width,pictureBox1->Height);
                  IFS^f=gcnew IFS;f->matrizen = dynamic_cast<IFS^>(fraktal)-
>matrizen;
                  f->front = fraktal->front;f->clear = 0;
                  f->back = fraktal->back;f->Multithreading = 0;
                  f->neededPoints=4;f->morph = 0;
                  f->addpoint(Point(pictureBox1->Width/3,2*pictureBox1->Height/3));
                  f->addpoint(Point(2*pictureBox1->Width/3,2*pictureBox1-
>Height/3));
                  f->addpoint(Point(2*pictureBox1->Width/3,pictureBox1->Height/3));
                  f->points[3]=Point(pictureBox1->Width/3,pictureBox1->Height/3);
                  f->n = 1;
          Graphics^ g = Graphics::FromImage(b);
                  g->DrawPolygon(gcnew Pen(Color::Red),f->points);
                  f->paint(g);
                  pictureBox1->Image = b;
                }
private: System::Void ifs2_preview(){
                                    Bitmap^b = gcnew Bitmap(pictureBox2-
>Width,pictureBox2->Height);
                                    IFS_2^f=gcnew IFS_2;
                                    f->verschiebung = dynamic_cast<IFS_2^>(fraktal)-
>verschiebung;
                                    f->drehung = dynamic_cast<IFS_2^>(fraktal)-
>drehung;
                                    f->streckung = dynamic_cast<IFS_2^>(fraktal)-
>streckung;
                                    f->front = fraktal->front;f->clear = 0;
                                    f->back = fraktal->back;f->Multithreading = 0;
                                    f->neededPoints=4;
                                    f->addpoint(Point(pictureBox1-
>Width/3,2*pictureBox1->Height/3));
                                    f->addpoint(Point(2*pictureBox1-
>Width/3,2*pictureBox1->Height/3));
                                    f->addpoint(Point(2*pictureBox1-
>Width/3,pictureBox1->Height/3));
```

```
                                        f->setmousepos(Point(pictureBox1-
>Width/3,pictureBox1->Height/3));
                                        f->n = 1;
                                        Graphics^ g = Graphics::FromImage(b);
                                        g->DrawPolygon(gcnew Pen(Color::Red),f->points);
                                        f->paint(g);
                                        pictureBox2->Image = b;
                                }
private: System::Void ifs_ok_Click(System::Object^  sender, System::EventArgs^  e) {
                        dynamic_cast<IFS^>(fraktal)->add_matrix(gcnew Matrix);
                        Matrices_Box_refresh();
                }
private: System::Void listBox2_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {

//////////////////////////////////////////////////////////////////////////
                        ///   Auswahl der ListBox (listet Matrizen für IFS auf) wurde
geändert
                        ///   -> Es wird ein Fenster geöffnet mit dem man die Werte der
                        ///              ausgewählten Matrix bearbeiten kann

//////////////////////////////////////////////////////////////////////////
                        ifs_matrix^ edit_matrix = gcnew ifs_matrix();
                        edit_matrix->ifs = dynamic_cast<IFS^>(LoadedFractal[comboBox2-
>SelectedIndex]);
                        edit_matrix->SelectedIndex = listBox2->SelectedIndex;
                        edit_matrix->Show();
                }
private: System::Void ifs_abort_Click(System::Object^  sender, System::EventArgs^  e) {

//////////////////////////////////////////////////////////////////////////
                        ///   eine Matrix für IFS löschen

//////////////////////////////////////////////////////////////////////////
                        if(listBox2->SelectedIndex==-
1){dynamic_cast<IFS^>(LoadedFractal[comboBox2->SelectedIndex])->del_matrix();}
                        else dynamic_cast<IFS^>(LoadedFractal[comboBox2->SelectedIndex])-
>del_matrix(listBox2->SelectedIndex);
                        Matrices_Box_refresh();
                }

 private: System::Void ComboBox2_fill(array<Fraktal^>^ frac){

//////////////////////////////////////////////////////////////////////////
                         ///   mehrere ifs oder lsysteme aus einer Datei laden und in der
Combo Box anzeigen

//////////////////////////////////////////////////////////////////////////
                        try
                        {
                                comboBox2->Items->Clear();
                                for (int i=0;i<frac->Length;i++)
                                {
                                        comboBox2->Items->Add(frac[i]->name);
                                }
                                // erstes Fraktal wird ausgewählt ....
                                if(LoadedFractal->Length>0&& comboBox2-
>SelectedIndex<0){comboBox2->SelectedIndex = 0; change_frac(LoadedFractal[0]);}
                                else comboBox2->SelectedIndex = comboBox2->SelectedIndex;
                        }
                        catch (System::IO::FileNotFoundException^ error)
                        {
                                MessageBox::Show(error->Message);
                        }
                 }
private: System::Void comboBox2_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                        // von der ComboBox die mehrere ifs und Lsysteme enthält wurde
die Auswahl geändert
                        //ifs = LoadedIfs[comboBox2->SelectedIndex];
                        change_frac(LoadedFractal[comboBox2->SelectedIndex]);
                        Matrices_Box_refresh();
                        LSystemBoxes_refresh();

                }
private: System::Void ifs_load_Click(System::Object^  sender, System::EventArgs^  e) {
                                        try
                                        {
```

```
                                                FileManagment^ f = gcnew FileManagment("");
                                                f->Warnings = 0;
                                                f->FileTyp = "ifs";
                                                f->Name = "IFS-Fraktal";
                                                LoadedFractal = f->LoadFractals(gcnew IFS);
                                                if(LoadedFractal==nullptr||LoadedFractal-
>Length==0){MessageBox::Show("Diese Datei enthält keine IFS-Fraktale");return;}
                                                change_frac(LoadedFractal[0]);
                                                ComboBox2_fill(LoadedFractal);
                                        }
                                        catch (System::Exception^ error)
                                        {
                                                MessageBox::Show( error->Message, "Fehler",
                                                        MessageBoxButtons::OK,
MessageBoxIcon::Exclamation );
                                        }

                }
private: System::Void numericUpDown2_ValueChanged(System::Object^  sender,
System::EventArgs^  e) {
             int i = Convert::ToInt32(numericUpDown2->Value);
                        if(i<=fraktal->MinPoints)
                        {fraktal->neededPoints = fraktal->MinPoints;numericUpDown2->Value
= fraktal->neededPoints;}
                        else fraktal->neededPoints = i;
                }
private: System::Void openFileDialog2_FileOk(System::Object^  sender,
System::ComponentModel::CancelEventArgs^  e) {
                }
private: System::Void checkBox2_CheckedChanged_1(System::Object^  sender,
System::EventArgs^  e) {
                        if(this->checkBox2->Checked)fraktal->abbr_distance = 1;
                        else fraktal->abbr_distance = 0;
                }
private: System::Void button3_Click_1(System::Object^  sender, System::EventArgs^  e) {
                        if(OpenInitiator->ShowDialog() ==
Windows::Forms::DialogResult::OK){checkBox3->Checked = 1;}
                }
private: System::Void checkBox3_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        if(checkBox3->Checked){
                            if(OpenInitiator->FileName=="")OpenInitiator->ShowDialog();
                        }
                        else OpenInitiator = gcnew
System::Windows::Forms::OpenFileDialog;
                }
private: System::Void checkBox4_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        dynamic_cast<LSystem^>(fraktal)->write_string = checkBox4-
>Checked;
                }
private: System::Void button6_Click(System::Object^  sender, System::EventArgs^  e) {

////////////////////////////////////////////////////////////////////////
                        ///     Regel für IFS Fraktal hinzufügen
////////////////////////////////////////////////////////////////////////
                        try
                        {
                                if(Convert::ToDouble(ifs_scale->Text)>1){throw gcnew
System::FormatException("Skalierung muss kleiner 1 sein!! (kontrahierende Funktion)");}
                                dynamic_cast<IFS_2^>(fraktal)-
>addrule(Convert::ToDouble(ifs_scale->Text),Convert::ToDouble(ifs_rotate-
>Text),Convert::ToDouble(ifs_vx->Text),Convert::ToDouble(ifs_vy->Text));
                            listBox3->Items->Clear();
                                listBox3->Items->AddRange(dynamic_cast<IFS_2^>(fraktal)-
>list());
                        }
                        catch (System::FormatException^ exception)
                        {
                                MessageBox::Show(Convert::ToString(exception->Message));
                        }
                }
private: System::Void ifs2_cancel_Click(System::Object^  sender, System::EventArgs^  e)
{
                        dynamic_cast<IFS_2^>(fraktal)->delrule();
                        listBox3->Items->Clear();
                        listBox3->Items->AddRange(dynamic_cast<IFS_2^>(fraktal)->list());
                }
```

```
private: System::Void listBox3_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                        if(listBox3->SelectedIndex>=0)
                        {
                                ifs_rotate->Text =
Convert::ToString(dynamic_cast<IFS_2^>(fraktal)->drehung[listBox3->SelectedIndex]);
                                ifs_vx->Text     =
Convert::ToString(dynamic_cast<IFS_2^>(fraktal)->verschiebung[listBox3-
>SelectedIndex].X);
                                ifs_vy->Text     =
Convert::ToString(dynamic_cast<IFS_2^>(fraktal)->verschiebung[listBox3-
>SelectedIndex].Y);
                                ifs_scale->Text  =
Convert::ToString(dynamic_cast<IFS_2^>(fraktal)->streckung[listBox3->SelectedIndex]);
                        }
                }
private: System::Void ifs_vx_TextChanged(System::Object^  sender, System::EventArgs^  e)
{
                        try
                        {
                                if(listBox3-
>SelectedIndex>=0&&dynamic_cast<IFS_2^>(fraktal)->Rules>0){
                                        dynamic_cast<IFS_2^>(fraktal)-
>verschiebung[listBox3->SelectedIndex].X = Convert::ToDouble(ifs_vx->Text);
                                        ifs2_preview();
                                }
                        }
                        catch (FormatException^ e){Status = e->Message;}
        }
private: System::Void ifs_vy_TextChanged(System::Object^  sender, System::EventArgs^  e)
{
                        try
                        {
                                if(listBox3-
>SelectedIndex>=0&&dynamic_cast<IFS_2^>(fraktal)->Rules>0){
                                        dynamic_cast<IFS_2^>(fraktal)-
>verschiebung[listBox3->SelectedIndex].Y =  Convert::ToDouble(ifs_vy->Text);
                                        ifs2_preview();
                                }
                        } catch (FormatException^ e){Status = e->Message;}
                }
private: System::Void ifs_rotate_TextChanged(System::Object^  sender, System::EventArgs^
e) {
                        try
                        {
                                if(listBox3-
>SelectedIndex>=0&&dynamic_cast<IFS_2^>(fraktal)->Rules>0){
                                        dynamic_cast<IFS_2^>(fraktal)->drehung[listBox3-
>SelectedIndex] =  Convert::ToDouble(ifs_rotate->Text);
                                        ifs2_preview();
                                }

                        }      catch (FormatException^ e){Status = e->Message;}

}
private: System::Void ifs_scale_TextChanged(System::Object^  sender, System::EventArgs^
e) {
                        try
                        {
                                if(listBox3-
>SelectedIndex>=0&&dynamic_cast<IFS_2^>(fraktal)->Rules>0){
                                        dynamic_cast<IFS_2^>(fraktal)->streckung[listBox3-
>SelectedIndex] =  Convert::ToDouble(ifs_scale->Text);
                                        ifs2_preview();
                                }
                    }   catch (FormatException^ e){Status = e->Message;}

        }
private: System::Void ifs2_load_Click(System::Object^  sender, System::EventArgs^  e) {
                        FileManagment^ f = gcnew FileManagment("");
                        f->Warnings = 0;
                        f->FileTyp = "ifs2";
                        f->Name = "IFS-Fraktal";
                        LoadedFractal = f->LoadFractals(gcnew IFS_2);
                }
private: System::Void checkBox5_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        dynamic_cast<IFS^>(fraktal)->morph = checkBox5->Checked;
```

```
                    }
};

}
```

LITERATURVERZEICHNIS

1. wikipedia - Fraktal. [Online] http://de.wikipedia.org/wiki/Fraktal.
2. **Peitgen, Heinz-Otto.** Bausteine des Chaos Fraktale. 1992.
3. wikipedia - Kochkurve. [Online] http://de.wikipedia.org/wiki/Koch-Kurve.
4. wikipedia - Selbstähnlichkeit. [Online]
http://de.wikipedia.org/wiki/Selbst%C3%A4hnlichkeit.
5. wikipedia - Lindenmayer-System. [Online]
http://de.wikipedia.org/wiki/Lindenmayer-System.
6. wikipedia - Affine Abbildung. [Online]
http://de.wikipedia.org/wiki/Affine_Abbildung.
7. MSND Bibliothek. [Online] http://msdn2.microsoft.com/de-de/default.aspx.
8. **Stroustrup, Bjarne.** *Die C++ Programmiersprache.*
9. **Barth Mühlbauer, Nikol Wörle.** *Mathematische Formeln und Definitionen.*
10. Webcast: GDI+ mit dem .NET-Framework. [Online]
http://www.microsoft.com/germany/msdn/webcasts/library.aspx?id=118753813.

Ich erkläre hiermit, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benützt habe.

....., den ....

Ort Datum Unterschrift