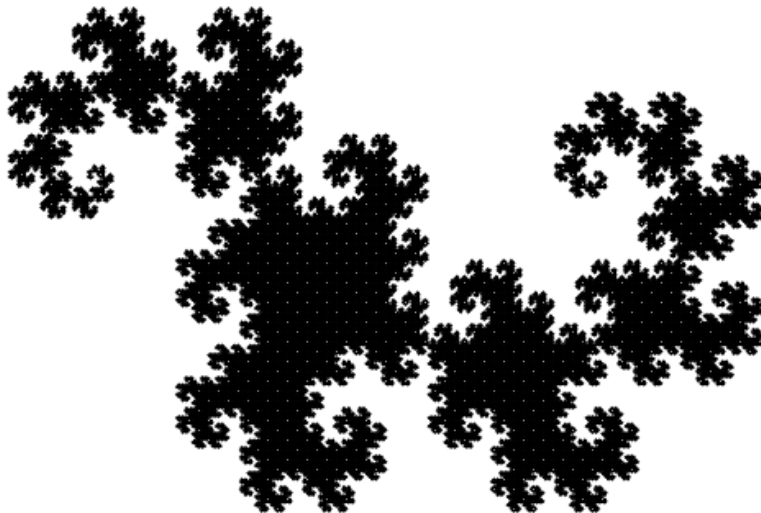


# GRAFISCHE DARSTELLUNG VON FRAKTALEN

Portfolioarbeit

Von Jonah Sebright

Klasse 6A



## Inhaltsverzeichnis

Inhaltsverzeichnis .....	1
1. Einleitung .....	2
1.1 Was sind Fraktale? .....	2
1.2 Ziele des Projekts .....	2
1.3 Vorschau Resultate .....	2
2. Hauptteil (Material und Methoden) .....	3
2.1 Programmiersprache .....	3
2.2 Bibliotheken .....	3
2.2.1 StdDraw .....	3
2.2.2 Flanagan .....	3
2.3 Reduktion des Problems .....	4
2.4 Entwicklung der Module .....	5
2.5 Entwicklung der Algorithmen .....	5
2.5.1 Schneeflocke .....	5
2.5.2 Pfeilspitze .....	6
2.5.3 Drachenkurve .....	7
2.6 Testen .....	7
3. Resultate .....	8
3.1 Grafisch dargestellte Fraktale .....	8
3.2 Effizienz .....	8
3.3 Funktionen des Main-Programms .....	12
4. Diskussion .....	12
Flächendrachenkurve .....	12
4.1 Optimierungsmöglichkeiten .....	13
4.2 Rückblick auf das Problem .....	13
6. Quellenverzeichnis .....	14
7. Abbildungsverzeichnis .....	14
8. Figurenverzeichnis .....	15
9. Tabellenverzeichnis .....	15
10. Quellcodeverzeichnis .....	15
11. Anhang .....	15
11.1 Weitere Diagramme .....	15
11.2 Gesamter Quellcode .....	15

## 1. Einleitung

### 1.1 Was sind Fraktale?

Fraktale sind geometrische Gebilde, die sich in nicht ganzzahligen („fraktalen“) Dimensionen befinden. Benoît Mandelbrot zeigte erstmals, dass Fraktale überall in der Natur existieren. Fraktale werden beispielsweise verwendet, um Längen von Küsten zu berechnen (vgl. Radons, kein Datum).

### 1.2 Ziele des Projekts

Das Hauptziel dieses Projekts ist, mit den gelernten Fähigkeiten und dem erworbenen Wissen ein konkretes Informatik-Problem zu lösen. Das in diesem Projekt untersuchte Problem soll bestimmte Monsterkurven, wie die „Schneeflocke“, die „Pfeilspitze“ und die „Drachenkurve“, visuell darstellen. Das Vorgehen beim Lösen des Problems wird ebenfalls in dieser Portfolioarbeit dargestellt.

Zu diesem Problem soll ein Programm erstellt werden, das dem Benutzer eine Annäherung des ausgewählten Fraktals bis zu einer gewünschten Stufe grafisch wiedergibt. Zur Zeichnung der Kurve soll die Zeichnungs-Bibliothek „StdDraw“ verwendet werden. Sobald die Darstellung vollendet ist, wird der Benutzer gefragt, ob er die Zeichnung als PDF-Datei speichern möchte. Dabei kann der Benutzer den Dateinamen selbst eingeben.

Für die grafische Darstellung der Fraktale müssen die Algorithmen der einzelnen Fraktale geschrieben werden, die alle das Basisprinzip der Rekursion implementieren. Zudem verwenden alle Algorithmen die Hilfs-Klasse *Schildkroete* [INSERT\\_LINK\\_TO](#) chapter.

Ausserdem werden in diesem Projekt die Effizienz der Algorithmen gemessen und in der Diskussion nach Optimierungsmöglichkeiten gesucht. Bei der Drachenkurve wird auch die Fläche der Kurve programmatisch gemessen und mit mathematischen Erkenntnissen. Die unendlich lange Drachenkurve sollte eine endliche Fläche von einem Viertel der Gesamtfläche bedecken.

### 1.3 Vorschau Resultate

Die Zeichnungen des Programms sind zum Staunen schön! Das Haupt-Programm läuft ohne Fehler und beinhaltet alle oben beschriebenen Funktionen und erfüllt somit dessen Ziele. Die einzelnen Algorithmen der Fraktale funktionieren ebenfalls einwandfrei bei beliebiger Stufe. Hier sind ein paar grafische Resultate der verschiedenen Fraktale:

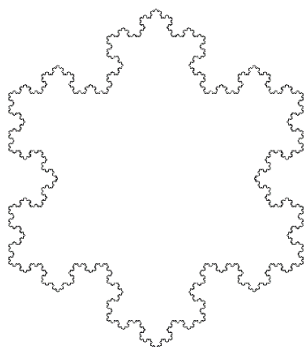


Abb. 3 Schneeflocke bei Stufe 7

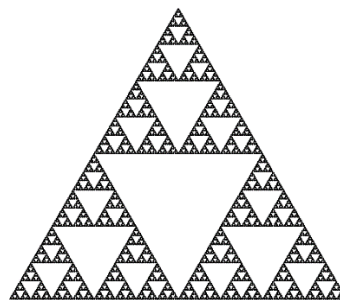


Abb. 4 Pfeilspitze bei Stufe 9

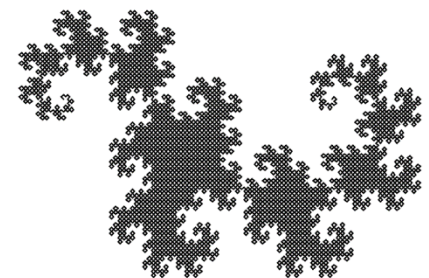


Abb. 2 Drachenkurve bei Stufe 13

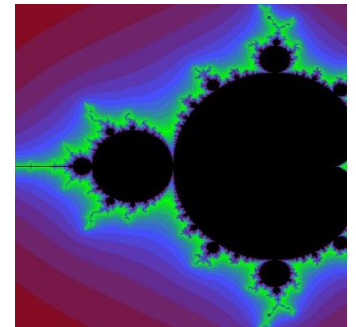


Abb. 1 Mandelbrot Fraktal (WELT, 2010)

Was und wie gemacht und was wird vorgesehlt, Darstellung

Das Programm besteht aus den Modulen *Main*, *Schildkroete*, *Fraktal*, *Input* und den Bibliotheken *StdDraw* und *Flanagan* (siehe Abb. 5 Modularisierung des Programms).

Insgesamt wurden 13 Klassen erstellt mit ca. 500 Linien Quellcode. Eine Gesamtübersicht der Klassen und ihren Beziehungen ist im Anhang in **INSERT\_REFERENCE** ersichtlich.

## 2. Hauptteil (Material und Methoden)

Eine Java-Applikation ist sehr komplex und kann durch falsche Entscheidungen schnell zu Problemen führen. Geschicktes Vorgehen ist daher sehr wichtig und die Grundstrukturen müssen stimmen, bevor Details implementiert werden können. Besonders mit zunehmendem Umfang des Programms ist die Architektur umso wichtiger.

Begonnen wird mit der konkreten Definition des Programms, was schon in der Einleitung gemacht wurde. Danach findet die Reduktion des Problems statt, wobei die Module und deren Beziehungen und der Globalalgorithmus skizziert werden. Anschliessend können die einzelnen Module und Algorithmen konkret implementiert werden. Möglicherweise werden dafür Bibliotheken verwendet (siehe 2.2.1 StdDraw). Erst in diesem Schritt wird der Quellcode verfasst.

Damit das Programm stabil und sicher ist, wurde es mit den erwarteten sowie auch unerwarteten Eingabewerten des Benutzers getestet. Dies ist wichtig, da Benutzer nicht immer die erwarteten Werte eingeben und Fehler des Programms zu vermeiden sind. Zudem muss auch die Effizienz getestet werden, sodass ein langsames Programm bemerkt und möglicherweise optimiert werden kann.

Für die Verwendung des Programms muss zunächst eine Verbindung zwischen dem Benutzer und dem Programm hergestellt werden. Dies wird durch ein GUI (graphical user interface) gemacht, was die Interaktion erlaubt. In diesem Programm müssen lediglich ein paar Fenster zum Einlesen und zur Resultat-Ausgabe geöffnet werden. Die Bibliothek „Flanagan“ (siehe 2.2.2 Flanagan) erfüllt diese Anforderungen.

### 2.1 Programmiersprache

Wie für die Arbeit vorgegeben wurde das Programm ausschliesslich mit Java programmiert. Java ist eine objekt-orientierte Programmiersprache von Oracle und bietet so viele Vorteile, wie unter anderem Polymorphie (vgl. Ullenboom 2012, p .50).

### 2.2 Bibliotheken

Bibliotheken sind eine Sammlung von Unterprogrammen, die von anderen Programmen genutzt werden können. Oft werden die Bibliotheken zur Vereinfachung von Problemen geschrieben.

#### 2.2.1 StdDraw

Für das Programm wurde eine Zeichnungsbibliothek namens „StdDraw“ vom *Computer Science Department* der Universität Princeton verwendet (vgl. Princeton University, 2021). Die Bibliothek bietet sowohl eine Zeichenfläche als auch Methoden an, mit denen die Zeichenfläche manipuliert werden kann. Die wichtigsten Methoden für dieses Projekt sind *line*, *setPenColor*, *setPenRadius*, *clear*. Eine weitere nützliche Funktion ist *save*, mit der die Zeichenfläche als Bild-Datei gespeichert werden kann.

#### 2.2.2 Flanagan

Für das Einlesen der gewünschten Kurve, der Stufe und des Dateinamens des zu speichernden Bildes wurde die Flanagan-Bibliothek (Flanagan, 2010) implementiert. Mit dieser können einfache Eingabefenster geöffnet werden. Die wichtigsten Methoden für dieses Projekt sind *optionBox*, *readInt*, *yesNo* und *readLine*, um die notwendige Information von dem Benutzer zu erhalten.

## 2.3 Reduktion des Problems

Komplexe Systeme können in kleinere Aufgaben geteilt und so einfacher gelöst werden. Dieses Aufteilen heisst **Modularisierung**. Jede einzelne Aufgabe kann für sich gelöst werden.

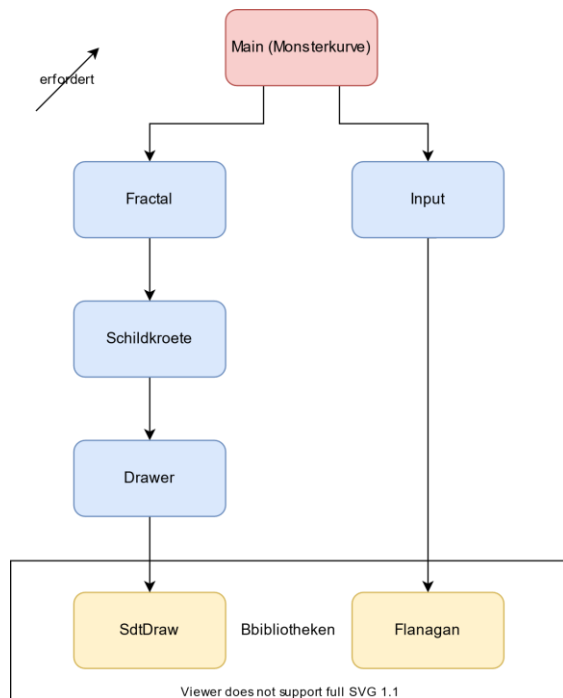


Abb. 5 Modularisierung des Programms

Dieses Programm habe ich folgendermassen modularisiert:

*Main* erfordert das Hilfsmodul *Input*, welches auf die Bibliothek *Flanagan* angewiesen ist. Zudem erfordert *Main* die *Fraktale* (Schneeflocke, Pfeilspitze, Drachenkurve), die das Modul *Schildkroete* verwenden. *Schildkroete* erfordert das Modul *Drawer*, welches unter anderem die Methode *drawLine(Point from, Point to)* bietet. Ausserdem verwendet *Drawer* die Bibliothek *StdDraw*.

Nachdem die Module entworfen wurden, können sie und ihr Zusammenspiel implementiert werden. Dabei muss zunächst ein **Grobalgorithmus** entworfen werden. Der Grobalgorithmus des Main-Programmes ist in Abb. 6 ersichtlich.

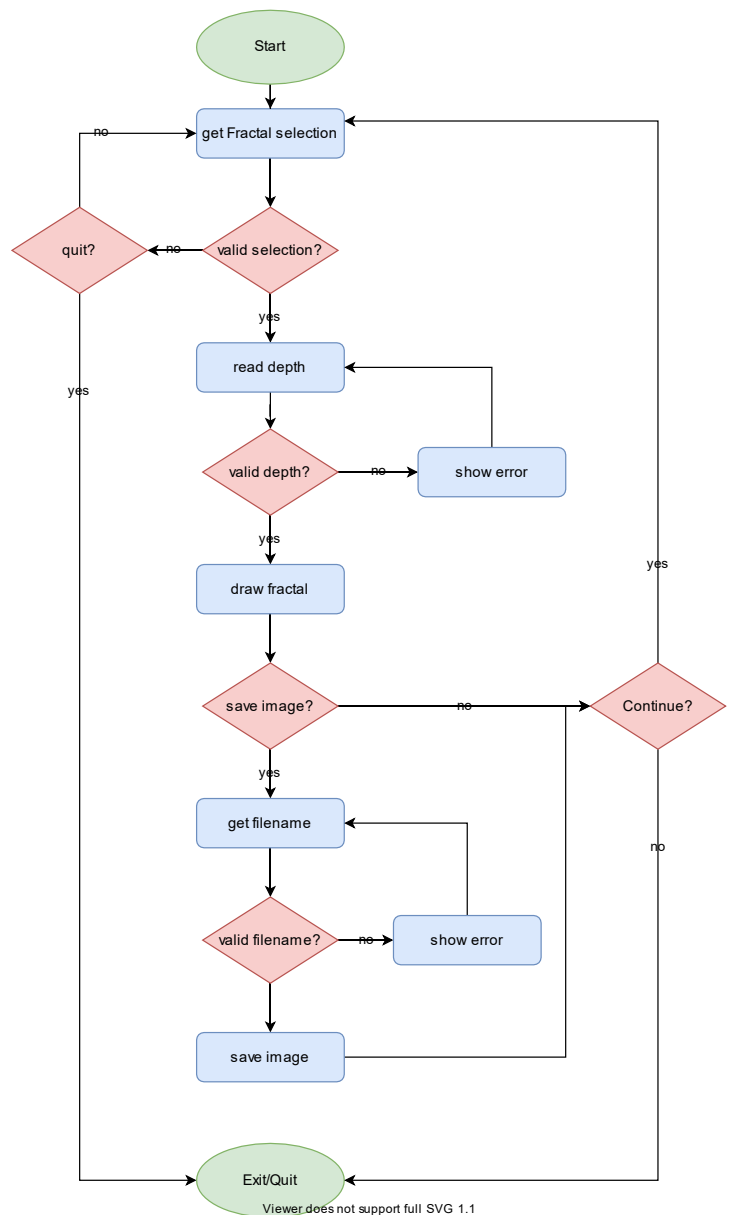


Abb. 6 Grobalgorithmus des Main Programms

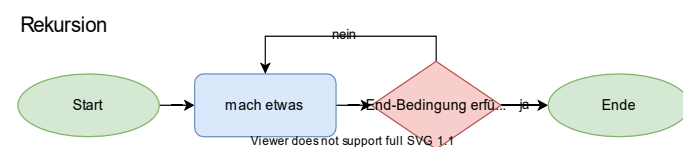


Abb. 7 Ablaufprinzip der Rekursion

Bei allen drei Fraktalen beruht der Grobalgorithmus auf dem allgemeinen Prinzip der Rekursion. Dies bedeutet, dass eine Methode sich selbst immer wieder aufruft, bis eine End-Bedingung erfüllt ist (siehe Abb. 7 Ablaufprinzip der Rekursion).

## 2.4 Entwicklung der Module

Ablaufdiagramme, Beschreibung variablen

## 2.5 Entwicklung der Algorithmen

Wie erwähnt, wird zu Konstruktion aller drei Kurven die Rekursion verwendet. Bei den drei Algorithmen sind die Parameter *depth* (*int*) und *length* (*double*) gegeben. Jene manipulieren die Richtung der *Schildkroete* und rufen ihre eigene Methode wieder mit veränderten Parametern auf. Die Stufe (*depth*) wird um eins verkleinert und die Strecke (*length*) durch einen konstanten Faktor dividiert. Wenn die Stufe den Wert null erreicht, wird die gegebene Strecke gezeichnet und die Methode ruft sich nicht mehr auf. Die End-Bedingung wurde also erfüllt und die Methode beendet.

Das Grundprinzip ist, dass der *Initiator* (anfängliche Strecke) bei jeder Iteration durch den *Generator* ersetzt wird. Der Generator ersetzt also eine gerade Strecke durch eine komplexere Kurve, die eine grössere Anzahl, dafür aber kürzere gerade Strecken aufweist. Diese Strecken werden in der nächsten Stufe wiederum durch den *Generator* ersetzt. So wird dies für jede Stufe wiederholt, bis im Unendlichen die richtige Kurve erreicht wurde.

### 2.5.1 Schneeflocke

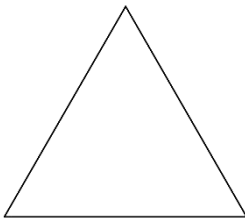


Abb. 10 Schneeflocke 0

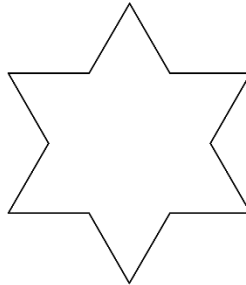


Abb. 9 Schneeflocke 1

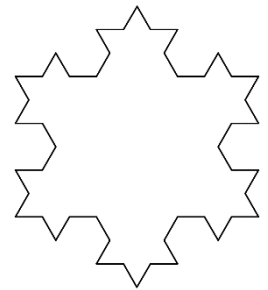


Abb. 8 Schneeflocke 2

Die Schneeflockenkurve ist eine der simpelsten und schönsten Kurven, die rekursiv definierbar sind. Ihr *Initiator* ist in diesem Fall ein gleichseitiges Dreieck. Die nächste Stufe entsteht dadurch, dass jede gerade Strecke der vorherigen Stufe durch drei geteilt und der mittlere Drittel durch zwei Seiten eines gleichseitigen Dreiecks ersetzt wird (siehe Abb. 10, Abb. 9, Abb. 8).

Dieses Verhalten lässt sich mit Quellcode 1 Die Methode *schneeflockeRecursive* (für eine Strecke) beschreiben, allerdings nur für eine Strecke und kein Dreieck.

```
public void schneeflockeRecursive(int depth, double sideLength) {
    if (depth == 0) {
        schildkroete.move(sideLength);
        ...
    } else {
        depth--;
        double dividedSideLength = sideLength / LENGTH_DIVIDER;
        schneeflockeRecursive(depth, dividedSideLength);
        schildkroete.direction(60);
        schneeflockeRecursive(depth, dividedSideLength);
        schildkroete.direction(-120);
        schneeflockeRecursive(depth, dividedSideLength);
        schildkroete.direction(60);
        schneeflockeRecursive(depth, dividedSideLength);
    }
}
```

Quellcode 1 Die Methode *schneeflockeRecursive* (für eine Strecke)

## Grafische Darstellung von Fraktalen

Um ein Dreieck zu erhalten, muss *schneeflockeRecursive* dreimal aufgerufen werden, jedes mal mit einem angepassten Winkel (siehe Quellcode 2 Die Methode *schneeflockeTriangle* (für ein Dreieck)).

```
public void schneeflockeTriangle(int depth, double sideLength) {  
    ...  
    schildkroete.direction(60);  
    schneeflockeRecursive(depth, sideLength);  
    schildkroete.direction(-120);  
    schneeflockeRecursive(depth, sideLength);  
    schildkroete.direction(-120);  
    schneeflockeRecursive(depth, sideLength);  
    ...  
}
```

Quellcode 2 Die Methode *schneeflockeTriangle* (für ein Dreieck)

### 2.5.2 Pfeilspitze

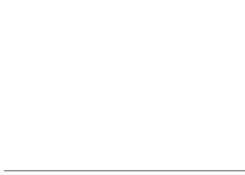


Abb. 13 Pfeilspitze 0



Abb. 12 Pfeilspitze 1

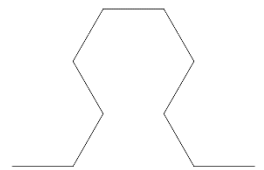


Abb. 11 Pfeilspitze 2

Die Pfeilspitzenkurve (auch Sierpinski-Dreieck genannt) hat eine gerade Strecke als *Initiator*. Ihr *Generator* ist eine dachförmige Figur mit 60 Grad Winkeln, wobei alle Seiten die gleiche Länge haben. Man könnte sagen, es ist die obere Hälfte eines Sechsecks. Allerdings wechselt die Richtung des *Generators* ab, um den gewünschten Effekt zu erhalten (siehe Abb. 13, Abb. 12, Abb. 11). Deshalb wird der zusätzlicher Parameter *richtung* (*boolean*) für die Funktion eingebaut, der von Stufe zu Stufe seinen Wert umkehrt. Die Pfeilspitze kann mit Quellcode 3 Die Methode *pfeilspitze* erstellt werden.

```
public void pfeilspitze(int level, double length, boolean direction) {  
    if (level == 0) {  
        schildkroete.move(length);  
        incrementMoveCount();  
    } else {  
        length /= 2;  
        level--;  
        int degree = direction ? 60 : -60;  
        schildkroete.direction(degree);  
        pfeilspitze(level, length, !direction);  
        schildkroete.direction(-degree);  
        pfeilspitze(level, length, direction);  
        schildkroete.direction(-degree);  
        pfeilspitze(level, length, !direction);  
        schildkroete.direction(degree);  
    }  
}
```

Quellcode 3 Die Methode *pfeilspitze*

## 2.5.3 Drachenkurve



Abb. 16 Drachenkurve 0



Abb. 15 Drachenkurve 1



Abb. 14 Drachenkurve 2

Die Drachenkurve ist besonders speziell. Zum Beispiel deckt die unendlich lange Kurve einen Viertel der Gesamtfläche ab. Sie entsteht so, als würde man einen Streifen Papier beliebig oft zusammenfalten, wieder aufmachen, und bei jedem Falt einen 90 Grad Winkel platzieren. Man könnte sie auch so beschreiben, dass die Kurve der nächsten Stufe die Kurve der vorherigen klont und in einem 90 Grad Winkel daran hängt. Mit Quellcode 4 Die *Methode drache* erhält man dieses Phänomen.

```
public void drache(int level, double length, boolean direction) {
    if (level == 0) {
        schildkroete.move(length);
        ...
    } else {
        int degreeLeft = direction ? -45 : 45;
        int degreeRight = direction ? 90 : -90;
        length /= SQRT_2;
        level--;
        schildkroete.direction(degreeLeft);
        drache(level, length, false);
        schildkroete.direction(degreeRight);
        drache(level, length, true);
        schildkroete.direction(degreeLeft);
    }
}
```

Quellcode 4 Die Methode drache

## 2.6 Testen

Testen ist für jede Applikation essentiell, um geringe als auch signifikante Fehler zu vermeiden. Dabei ist es wichtig, nicht nur die optimalen Eingabe-Werte des Benutzers zu erwarten, sondern auch die ungewöhnlichsten Werte. Diese App wurde mit diversen Werten getestet und so gebaut, dass sie dem Benutzer im Falle falscher Eingaben sowohl konstruktive Fehlermeldungen anzeigt als auch Korrekturmöglichkeiten anbietet.

Ebenfalls wichtig ist die Effizienz der Algorithmen. Diese wurde gemessen und die Resultate werden in 3.2 Effizienz aufgezeigt und analysiert. Dafür wurden die Laufzeit und die Anzahl der Bewegungen der *Schildkroete* in Betracht gezogen.



### 3. Resultate

#### 3.1 Grafisch dargestellte Fraktale

Die grafischen Resultate der Fraktale bei verschiedenen Stufen werden im folgenden Abschnitt präsentiert:

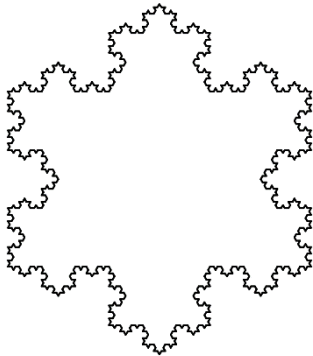


Abb. 19 Schneeflocke 4

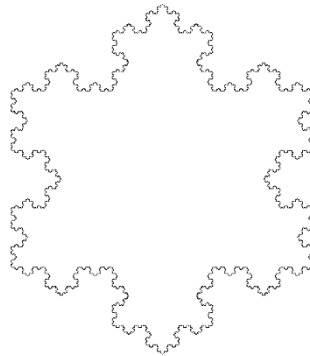


Abb. 17 Schneeflocke 6

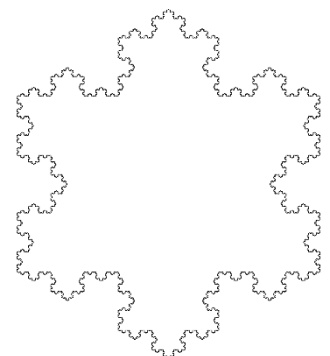


Abb. 18 Schneeflocke 8

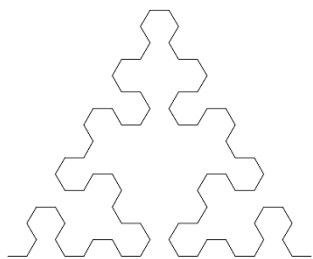


Abb. 24 Pfeilspitze 4

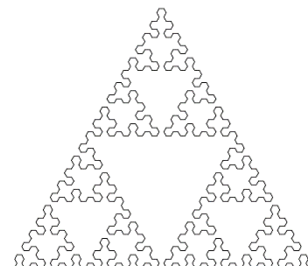


Abb. 20 Pfeilspitze 6

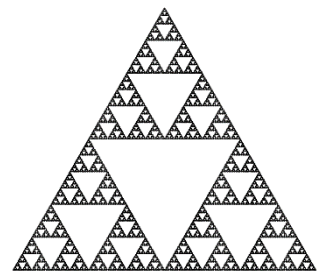


Abb. 25 Pfeilspitze 8

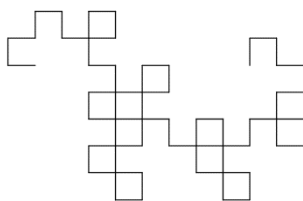


Abb. 23 Drachenkurve 6

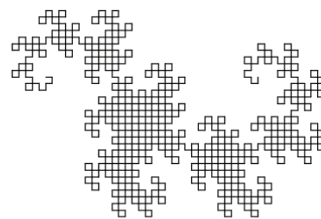


Abb. 22 Drachenkurve 10

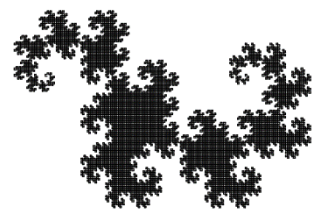


Abb. 21 Drachenkurve 14

#### 3.2 Effizienz

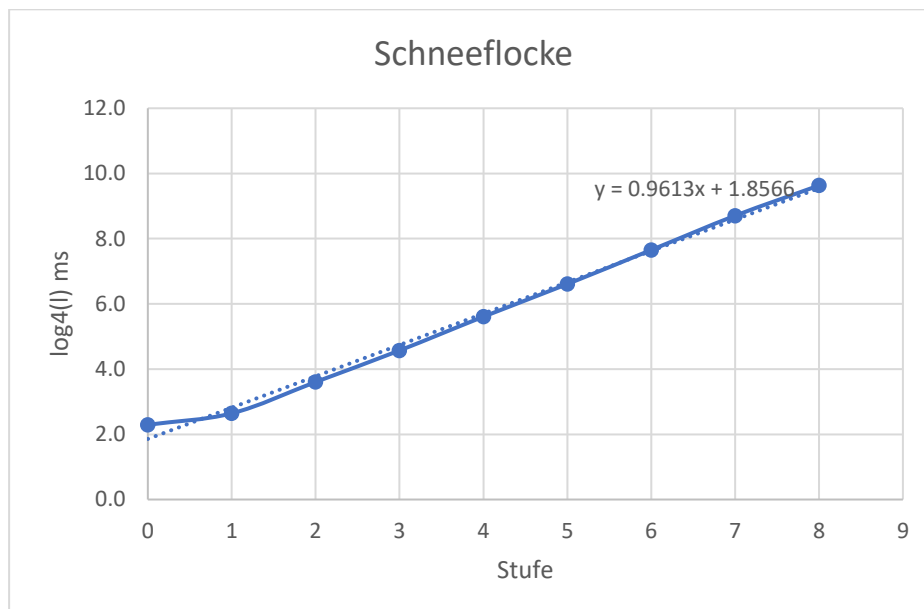
Bei vielen Applikationen spielt die Effizienz eine kleine Rolle. Ob das Programm 20 oder 50 Millisekunden beansprucht, ist für den Benutzer von geringer Bedeutung. Wenn es aber eine ganze Sekunde oder gar mehrere Minuten dauert, muss auf die Effizienz/Laufzeit geachtet werden. Bei meinem Programm ist dies der Fall: Schon bei einer Stufe 5 bemerkt man eine deutlich erhöhte Laufzeit, was zwar in einer schönen Animation der Kurve resultiert aber nicht erwünscht ist. Bei einer Stufe von acht bei der Schneeflockenkurve werden sogar einige Minuten benötigt, um die Grafik zu vollenden.

Hier sind die Resultate der Effizienz-Messungen, wobei  $s$  die Anzahl der Bewegungen der Schildkröte ist und  $l$  die Laufzeit.

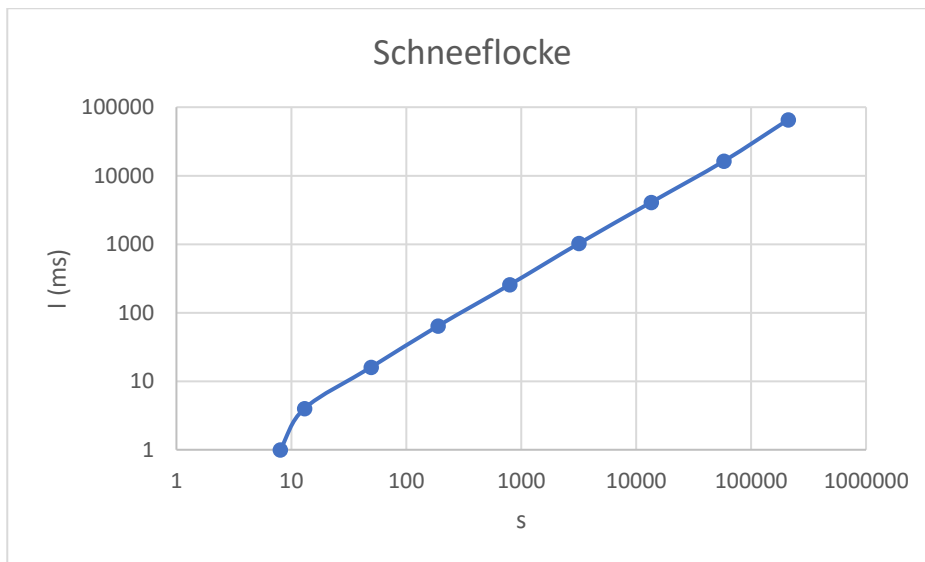
## Schneeflocke

Stufe	$l$ (ms)	$s$	$\log_4(l)$	$\log_4(s)$	$l/s$
0	8	1	2.3	0	24.00
1	13	4	2.6	1	9.75
2	49	16	3.6	2	9.25
3	188	64	4.6	3	8.83
4	793	256	5.6	4	9.29
5	3171	1024	6.6	5	9.29
6	13510	4096	7.7	6	9.90
7	58065	16384	8.7	7	10.63
8	209928	65536	9.6	8	9.61

Tabelle 1 Effizienz-Messungen Schneeflocke



Figur 1  $\log_4$ (Laufzeit) zur Stufe von Schneeflocke



Figur 2 Laufzeit zu s mit logarithmischen Skalen von Schneeflocke

In Tabelle 1 Effizienz-Messungen Schneeflocke kann man einen exponentiellen Zuwachs der Laufzeit mit der Stufe erkennen. Dies sieht man auch daran, dass der  $\log_4(l)$  proportional zu Stufe ist. In Figur 1  $\log_4(\text{Laufzeit})$  zur Stufe von Schneeflocke hat die Regressionsgerade fast eine Steigung  $a = 1$ , was vermuten lässt, dass  $l \sim 4^n$ , wenn  $n$  die Stufe ist.

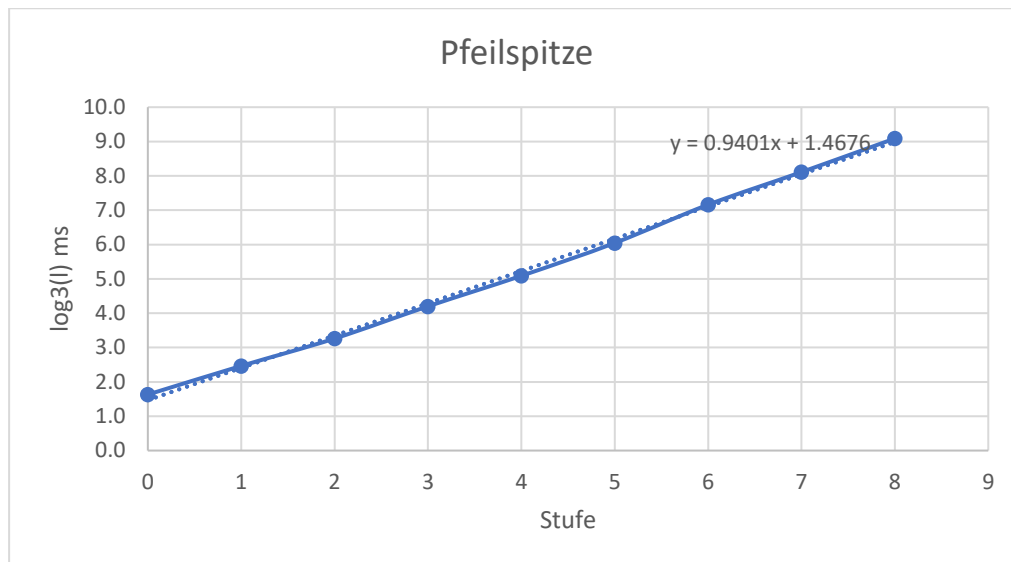
Ebenfalls ersichtlich ist, dass die Laufzeit proportional zu Anzahl Bewegungen der Schildkroete ist, obwohl die Strecken der Schildkroete kleiner sind. Offensichtlich spielt beim Zeichnen einer Linie die Länge der Linie keine Rolle.

## Pfeilspitze

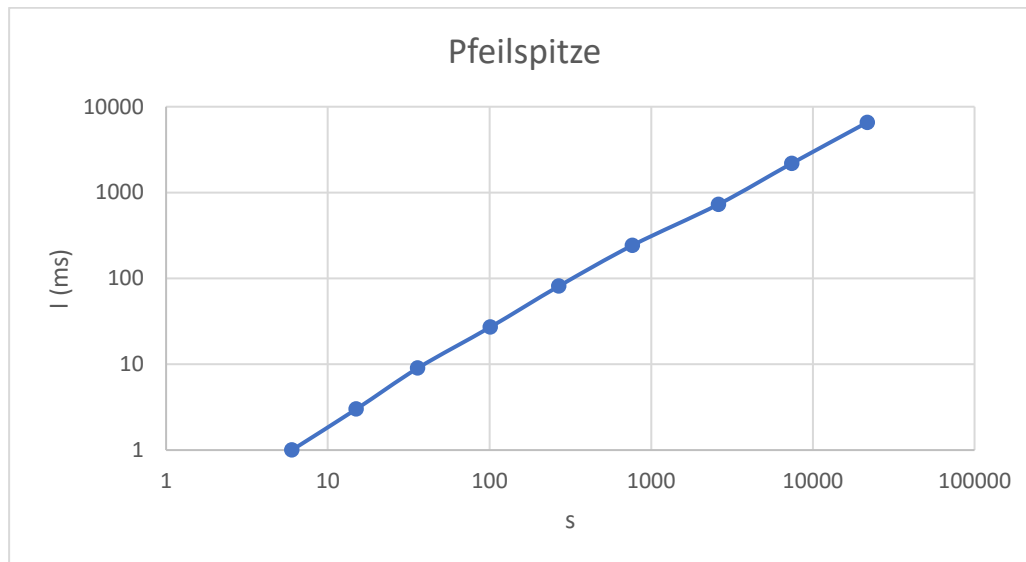
Stufe	l (ms)	s	log3(l)	log3(s)	l/s
0	6	1	1.6	0	6.00
1	15	3	2.5	1	5.00
2	36	9	3.3	2	4.00
3	101	27	4.2	3	3.74
4	268	81	5.1	4	3.31
5	767	243	6.0	5	3.16
6	2608	729	7.2	6	3.58
7	7408	2187	8.1	7	3.39
8	21683	6561	9.1	8	3.30

Tabelle 2 Effizienz-Messungen Pfeilspitze

## Grafische Darstellung von Fraktalen



Figur 3  $\log_3(\text{Laufzeit})$  zur Stufe von Pfeilspitze



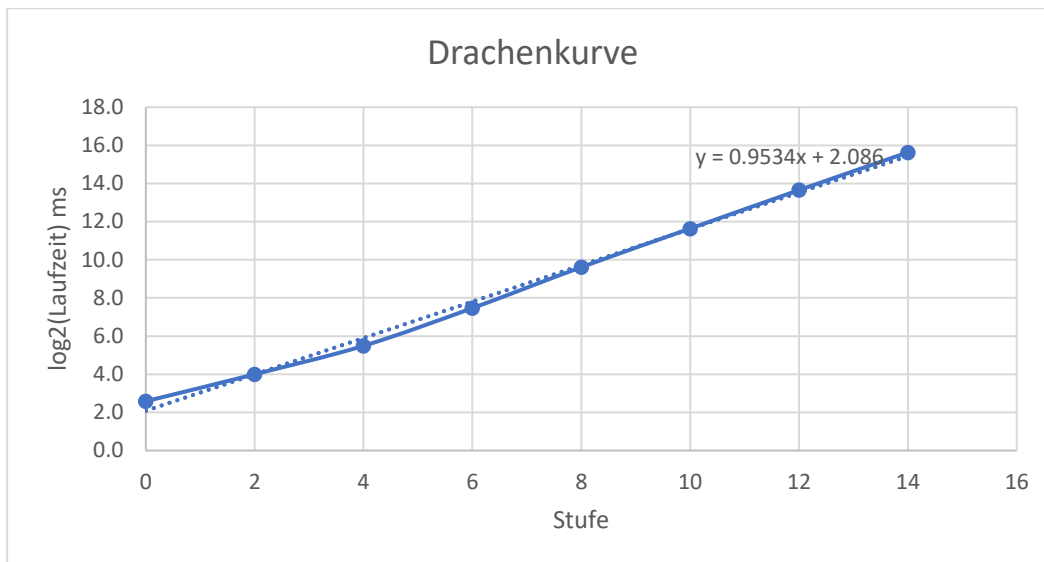
Figur 4 Laufzeit zu  $s$  mit logarithmischen Skalen von Pfeilspitze

Bei der Pfeilspitze sind die Ergebnisse sehr ähnlich zu denen der Schneeflocke. Allerdings ist hier  $\log_3(l)$  proportional zur Stufe was das Verhältnis  $l \sim 3^n$  vermuten lässt.

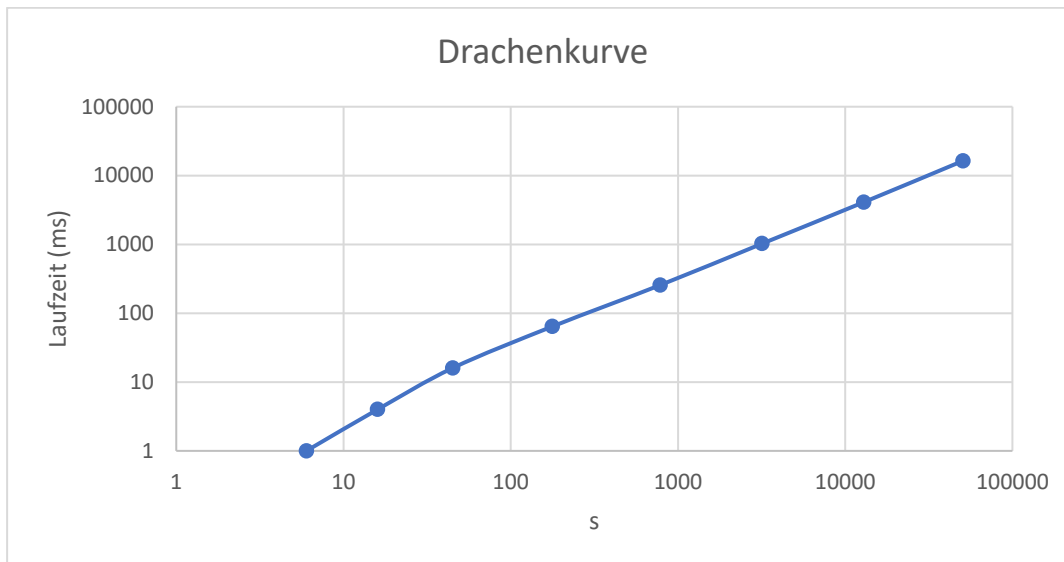
## Drachenkurve

Stufe	Laufzeit l (ms)	s	$\log_2(\text{Laufzeit})$	$\log_2(s)$	Laufzeit/s
0	6	1	2.6	0	6.00
2	16	4	4.0	2	4.00
4	45	16	5.5	4	2.81
6	177	64	7.5	6	2.77
8	783	256	9.6	8	3.06
10	3182	1024	11.6	10	3.11
12	12902	4096	13.7	12	3.15
14	50729	16384	15.6	14	3.10

Tabelle 3 Effizienz-Messungen Drachenkurve



Figur 5  $\log_2(\text{Laufzeit})$  zur Stufe von Drachenkurve



Figur 6 Laufzeit zu  $s$  mit logarithmischen Skalen von Drachenkurve

Bei der Drachenkurve sind die Ergebnisse wieder ähnlich zu den oberen. Allerdings ist hier  $\log_2(l)$  proportional zur Stufe was das Verhältnis  $l \sim 2^n$  vermuten lässt.

## 3.3 Funktionen des Main-Programms

Beim Ausführen des Programmes wird der Benutzer gefragt, ob welche Monsterkurve (Schneeflocke, Pfeilspitze oder Drachenkurve) er gezeichnet haben möchte. Darauf hin, kann er die gewünschte Stufe eingeben. Sollte dies kleiner als null sein, wird er auf diesen Fehler hingewiesen und kann die Stufe neu eingeben. Danach beginnt das Programm mit der Zeichnung der Kurve und fragt den Benutzer nach der Vollendung, ob er das Bild speichern möchte. Falls ja, wird ein neues Eingabefenster gezeigt, in dem der Dateiname eingegeben werden kann. Jetzt beginnt der Vorgang wieder von vorne und der Benutzer kann eine neue Kurve auswählen.

## 4. Diskussion

### Flächendrachenkurve

blabla

### 4.1 Optimierungsmöglichkeiten

Die Resultate der Effizienz (siehe 3.2 Effizienz) zeigen eine drastische Erhöhung der Laufzeitdauer mit einer erhöhten Stufe. Genauer gesagt verhält sie sich exponentiell zur Stufe.  $n$  die Stufe, so ist bei der Schneeflocke  $l \sim 4^n$ , bei der Pfeilspitze  $l \sim 3^n$  und bei der Drachenkurve  $l \sim 2^n$ . Die Laufzeit ist proportional zur Anzahl Bewegungen der Schildkroete, bzw. zur Anzahl Strecken.

Dieses exponentielle Verhalten lässt sich erklären: Bei der **Schneeflocke** beispielsweise besteht der Generator aus vier Seiten. Mit jeder Erhöhung der Stufe müssen also viermal mehr Strecken gezeichnet werden, deshalb die Formel  $s = 4^n$ . Bei der **Pfeilspitze** besteht der Generator aus drei Strecken, also müssen pro Stufe dreimal so viele Strecken gezeichnet werden, deshalb die Formel  $s = 3^n$ . Bei der **Drachenkurve** ist es gleich, nur ersetzt der Generator eine durch zwei Strecken. Und weil  $s \sim l$ , muss  $l = k * 4^n$  für die Schneeflocke,  $l = k * 3^n$  für die Pfeilspitze und  $l = k * 2^n$  für die Drachenkurve gelten.

Logisch gesehen ist es folglich unmöglich, die Effizienz zu steigern und eine geringere Laufzeit zu erhalten. Dies lässt sich leider nicht ändern.

### 4.2 Rückblick auf das Problem

Was für Schwierigkeiten?

Hätte besser angehen können?

## 6. Quellenverzeichnis

Kürzel: o.T.= ohne Tag, o.M.= ohne Monatsangabe , o.J.= ohne Jahr, o.O.= ohne Ortsangabe,

### Text

#### F

Flanagan, M.: „Michael Thomas Flanagan’s Java Scientific Library“. 25.06.2020, URL: <https://www.ee.ucl.ac.uk/~mflanaga/java/> (abgerufen 14.05.2021)

#### P

Princeton University: „Department of Computer Science“. o.T., o.M., 2021, URL: <https://www.cs.princeton.edu/> (abgerufen 14.05.2021)

#### R

Radons, G.: „Fraktale“. o.T., o.M., o.J., URL: <https://www.spektrum.de/lexikon/physik/fraktale/5252> (abgerufen 14.5.2021)

#### U

Ullenboom, C.: Java ist eine Insel. Das umfassende Handbuch. Bonn: Rheinwerk Computing, 2012 (10th edition).

### Abbildungen

#### W

Welt: „Mandelbrot bewies die Schönheit der Mathematik“. 17.10.2010, URL: <https://www.welt.de/wissenschaft/article10361079/Mandelbrot-bewies-die-Schoenheit-der-Mathematik.html> (abgerufen 14.5.2021)

## 7. Abbildungsverzeichnis

Abb. 1 Mandelbrot Fraktal (Vater der Fraktale: Mandelbrot bewies die Schönheit der Mathematik - WELT, 2010) .....	2
Abb. 2 Drachenkurve bei Stufe 13 .....	2
Abb. 3 Schneeflocke bei Stufe 7 .....	2
Abb. 4 Pfeilspitze bei Stufe 9 .....	2
Abb. 5 Modularisierung des Programms .....	4
Abb. 6 Grobalgorithmus des Main Programms .....	4
Abb. 7 Ablaufsprinzip der Rekursion .....	4
Abb. 8 Schneeflocke 2 .....	5
Abb. 9 Schneeflocke 1 .....	5
Abb. 10 Schneeflocke 0 .....	5
Abb. 11 Pfeilspitze 2 .....	6
Abb. 12 Pfeilspitze 1 .....	6
Abb. 13 Pfeilspitze 0 .....	6
Abb. 16 Drachenkurve 2 .....	7

Abb. 14 Drachenkurve 1 .....	7
Abb. 15 Drachenkurve 0 .....	7
Abb. 17 Schneeflocke 6 .....	8
Abb. 18 Schneeflocke 8 .....	8
Abb. 19 Schneeflocke 4 .....	8
Abb. 20 Pfeilspitze 6 .....	8
Abb. 21 Drachenkurve 14 .....	8
Abb. 22 Drachenkurve 10 .....	8
Abb. 23 Drachenkurve 6 .....	8
Abb. 24 Pfeilspitze 4 .....	8
Abb. 25 Pfeilspitze 9 .....	8

## 8. Figurenverzeichnis

Figur 1 $\log_4$ (Laufzeit) zur Stufe von Schneeflocke .....	9
Figur 2 Laufzeit zu s mit logarithmischen Skalen von Schneeflocke .....	10
Figur 3 $\log_3$ (Laufzeit) zur Stufe von Pfeilspitze .....	11
Figur 4 Laufzeit zu s mit logarithmischen Skalen von Pfeilspitze .....	11
Figur 5 $\log_2$ (Laufzeit) zur Stufe von Drachenkurve .....	12
Figur 6 Laufzeit zu s mit logarithmischen Skalen von Drachenkurve .....	12

## 9. Tabellenverzeichnis

Tabelle 1 Effizienz-Messungen Schneeflocke .....	9
Tabelle 2 Effizienz-Messungen Pfeilspitze .....	10
Tabelle 3 Effizienz-Messungen Drachenkurve .....	11

## 10. Quellcodeverzeichnis

Quellcode 1 Die Methode schneeflockeRecursive (für eine Strecke) .....	5
Quellcode 2 Die Methode schneeflockeTriangle (für ein Dreieck) .....	6
Quellcode 3 Die Methode pfeilspitze .....	6
Quellcode 4 Methode drache .....	7

## 11. Anhang

### 11.1 Weitere Diagramme

### 11.2 Gesamter Quellcode