Turtlegrafik und Fraktale

Ac

Bei der so genannten Turtlegrafik (Schildkröten- bzw. Igel-Grafik) stellt man sich einen kleinen Roboter vor, der sich in einem Koordinatensystem bewegen und dort mittels eines Stiftes (pen) zeichnen kann, nachdem man ihm entsprechende Befehle erteilt hat. Verwendet man bei den Befehlen auch rekursive Elemente, so lassen sich oft verblüffende Grafiken erstellen (z.B. Selbstähnlichkeit; Fraktale).

Die Idee der Turtlegrafik wurde ursprünglich in der Sprache LOGO eingesetzt und später dann in andere Sprachen übertragen, wobei der Befehlssatz teilweise stark erweitert wurde.

Einige Beispiele für Befehle an die Turtle (ursprüngliche Grundbefehle):

FORWARD 70	70 Schritte vorwärts gehen
BACK 42	42 Schritte rückwärts gehen
LEFT 30	um 30° nach links drehen
RIGHT 90	um 90° nach rechts drehen
PENUP	Stift heben
PENDOWN	Stift senken
HOME	zur Mitte des Koordinatensystems gehen
SETPENCOLOR blue	Stiftfarbe auf blau setzen
HIDETURTLE	Turtle verstecken
SHOWTURTLE	Turtle zeigen

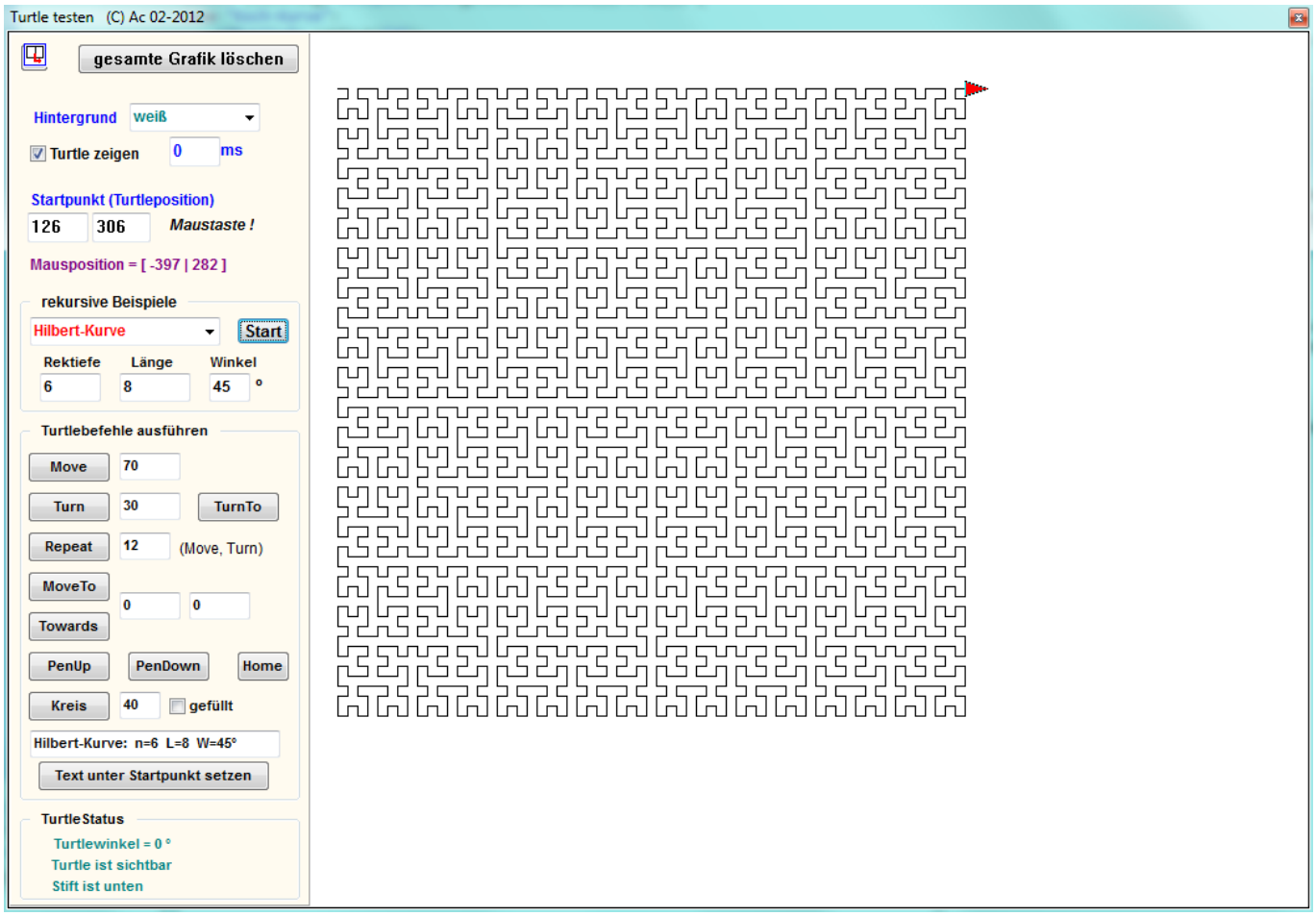
Turtle – Implementierung in speziellen Sprachen

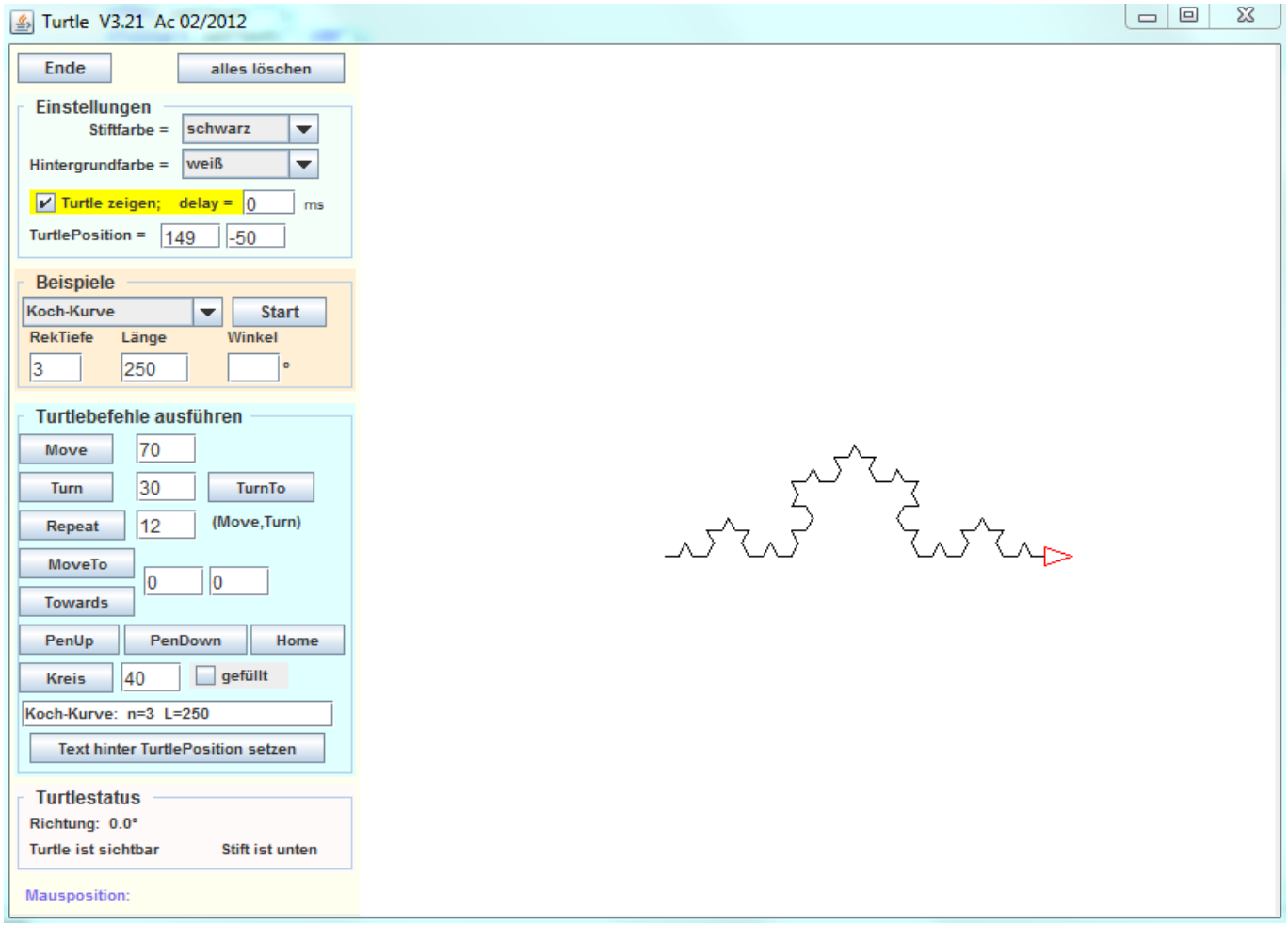
Im folgenden habe ich in den beiden Programmiersprachen Pascal (Delphi-Umgebung) und Java je eine Turtle-Implementierung mit *erweitertem Befehlssatz* vorgenommen.

Auszüge aus der Java-Klasse:

```
public class TurtleAc extends JPanel {
        // Attribute der Turtle
        public double xT; // x-Position (linker Rand = 0)
        public double yT; // y-Position (oberer Rand = 0)
        public double angleT; // Blickrichtung (Winkel bzgl. pos. x-Achse)
        public int xStart, yStart; // Bildschirmkoordinaten; Mitte=(0|0)
        private Color penCol; // Zeichenfarbe
        private Color bgCol; // Hintergrundfarbe
        public boolean penDown; // zeichnen oder nur bewegen
        public boolean isVisible; // dreieckiges Symbol sichtbar ?
        public int delay; // Verzögerung beim Zeichnen der Turtle
        private int maxX, maxY; // Bildschirmgrenzen
        public int midX, midY; // Mittelpunkt des Bildschirms
        private BufferedImage bufOhneTurt;
        private BufferedImage bufMitTurt;
        private Graphics grBufOhneTurt, grBufMitTurt;

        // Initialisierungen der Turtle

        public void init() {
                maxX = getWidth();
                maxY = getHeight();
                bufMitTurt = new BufferedImage(maxX, maxY, BufferedImage.TYPE_INT_RGB);
                bufOhneTurt = new BufferedImage(maxX, maxY, BufferedImage.TYPE_INT_RGB);
                midX = maxX / 2;
                midY = maxY / 2;
```

Auf interaktive Weise lassen sich interessante Grafiken erzeugen , z.B. Kochkurve, Hilbertkurve, Rosetten etc. . Selbstverständlich sind auch einfache geometrische Figuren wie Vielecke möglich.


Turtle testen (C) Ac 02-2012
gesamte Grafik löschen
Hintergrund
weiß
Turtle zeigen
0
ms
Startpunkt (Turtleposition)
126
306
Maustaste !
Mausposition = [-397 | 282]
rekursive Beispiele
Hilbert-Kurve
Start
Rektiefe
Länge
Winkel
6
8
45
°
Turtlebefehle ausführen
Move
70
Turn
30
TurnTo
Repeat
12
(Move, Turn)
MoveTo
Towards
0
0
PenUp
PenDown
Home
Kreis
40
gefüllt
Hilbert-Kurve: n=6 L=8 W=45°
Text unter Startpunkt setzen
TurtleStatus
Turtlewinkel = 0 °
Turtle ist sichtbar
Stift ist unten


Turtle V3.21 Ac 02/2012
Ende
alles löschen
Einstellungen
Stiftfarbe =
schwarz
Hintergrundfarbe =
weiß
Turtle zeigen; delay =
0
ms
TurtlePosition =
149
-50
Beispiele
Koch-Kurve
Start
RekTiefe
Länge
Winkel
3
250
°
Turtlebefehle ausführen
Move
70
Turn
30
TurnTo
Repeat
12
(Move,Turn)
MoveTo
Towards
0
0
PenUp
PenDown
Home
Kreis
40
gefüllt
Koch-Kurve: n=3 L=250
Text hinter TurtlePosition setzen
Turtlestatus
Richtung: 0.0°
Turtle ist sichtbar
Stift ist unten
Mausposition:

Fraktale (selbstähnliche Figuren)

Beispiel: Von Kochsche Kurve (Kochkurve)

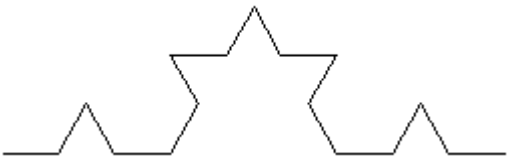
Diese Kurve , die auch Schneeflockenkurve genannt wird, gehört zur Gruppe der sog. Monsterkurven. Sie wird folgendermaßen erzeugt:

Ausgangspunkt ist eine Strecke bestimmter Länge.
Diese Figur der Stufe 0 heißt I n i t i a t o r .

Diese Strecke wird nun durch drei geteilt und das mittlere Stück durch zwei Seiten eines gleichseitigen Dreiecks ersetzt.

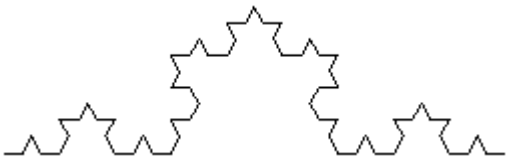
Diese Figur der Stufe 1 heißt G e n e r a t o r .
Es ist der Teil, der für die Erzeugung der Kurve verantwortlich ist.

Im nächsten Schritt wird jede der vier Strecken des Generators durch den Generator selbst ersetzt, allerdings schrumpft seine Größe dabei jeweils auf ein Drittel ! (Stufe 2)



Der letzte Schritt wird fortlaufend wiederholt, so

dass höhere Stufen (Rekursionstiefen) entstehen .



Solche selbstähnlichen Kurven erzeugt man am besten rekursiv und mithilfe der sog. Turtlegrafik, welche Befehle wie move, turn, etc. kennt .

Der Algorithmus (in Java) ist wie folgt:

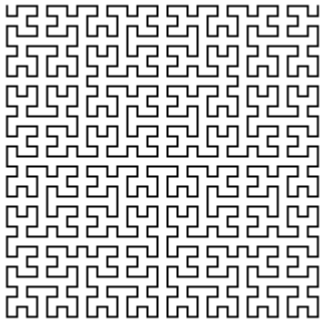
```
void koch(int nrek, double laenge) {
      if (nrek > 0) {
            koch(nrek - 1, laenge / 3);
            tu.turn(60);
            koch(nrek - 1, laenge / 3);
            tu.turn(-120);
            koch(nrek - 1, laenge / 3);
            tu.turn(60);
            koch(nrek - 1, laenge / 3);
      } else
            tu.move(laenge);
}
```

Anmerkung: tu ist eine Instanz der Klasse Turtle .

Es folgen weitere selbstähnliche Kurven:

Hilbertkurve:

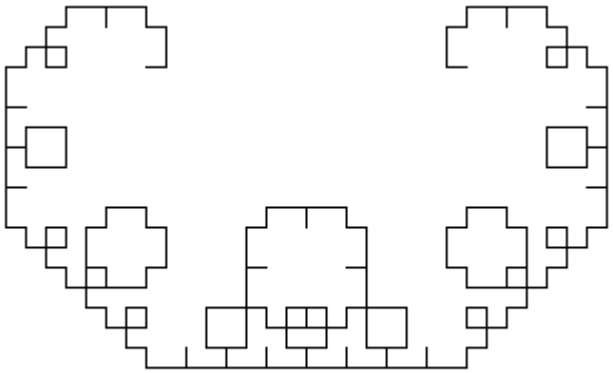
```
void hilbert(int stufe, int vz, int len) {
	if (stufe > 0) {
		tu.turn(-vz * 90);
		hilbert(stufe - 1, -vz, len);
		tu.move(len);
		tu.turn(vz * 90);
		hilbert(stufe - 1, vz, len);
		tu.move(len);
		hilbert(stufe - 1, vz, len);
		tu.turn(vz * 90);
		tu.move(len);
		hilbert(stufe - 1, -vz, len);
		tu.turn(-vz * 90);
	}
}
```



Hilbert-Kurve: n=5 L=5

C-Kurve:

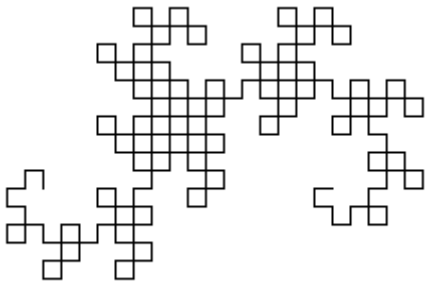
```
void c(int nrek, double laenge) {
	if (nrek > 0) {
		c(nrek - 1, laenge);
		tu.turn(90);
		c(nrek - 1, laenge);
		tu.turn(-90);
	} else
		tu.move(laenge);
}
```



C-Kurve: n=8 L=10

Drachenkurve:

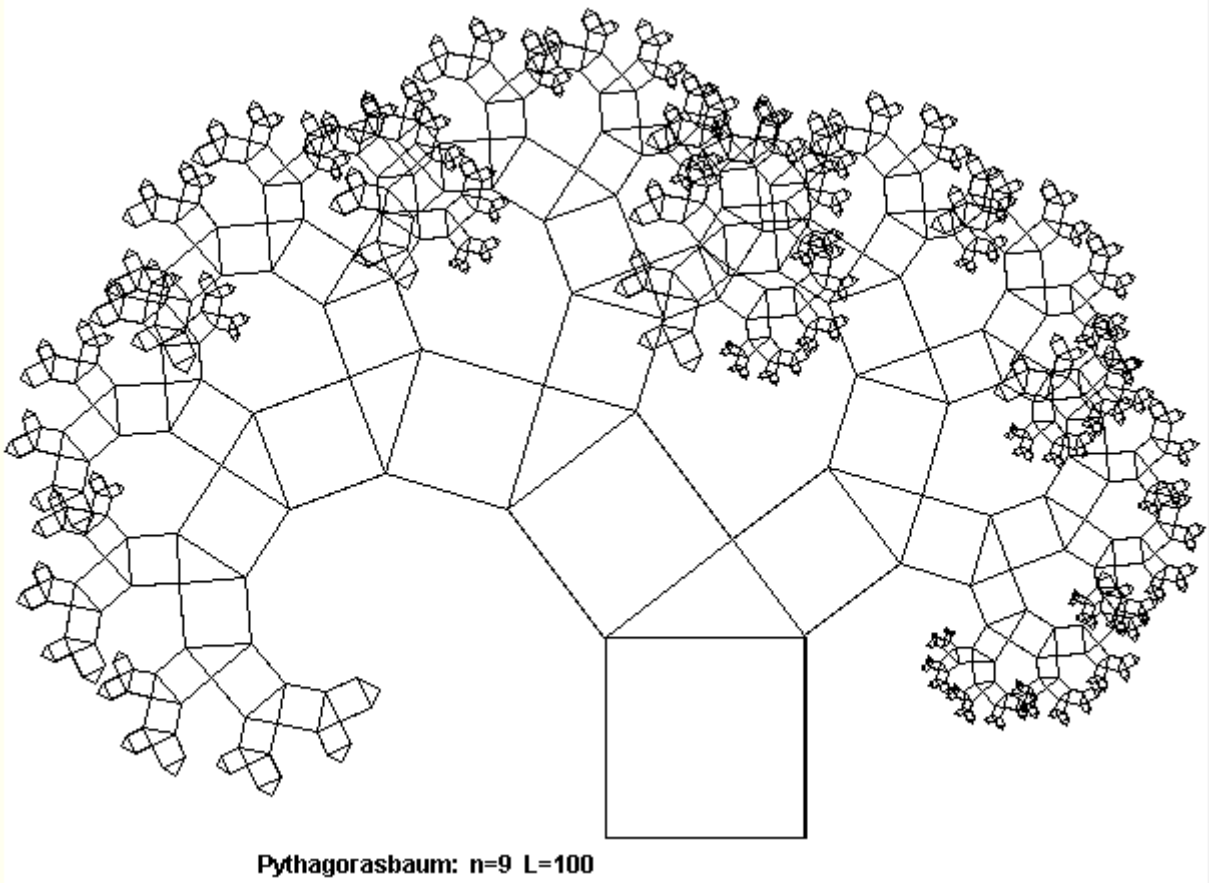
```
double WURZEL2 = Math.sqrt(2);

void drachen(int nrek, int vz, double laenge) {
	if (nrek > 0) {
	   tu.turn(vz * 45);
	   drachen(nrek - 1, -1, laenge / WURZEL2);
	   tu.turn(-vz * 90);
	   drachen(nrek - 1, +1, laenge / WURZEL2);
	   tu.turn(vz * 45);
	} else
		tu.move((int) laenge);
}
```



Drachen-Kurve: n=8 L=150

Pythagorasbaum:

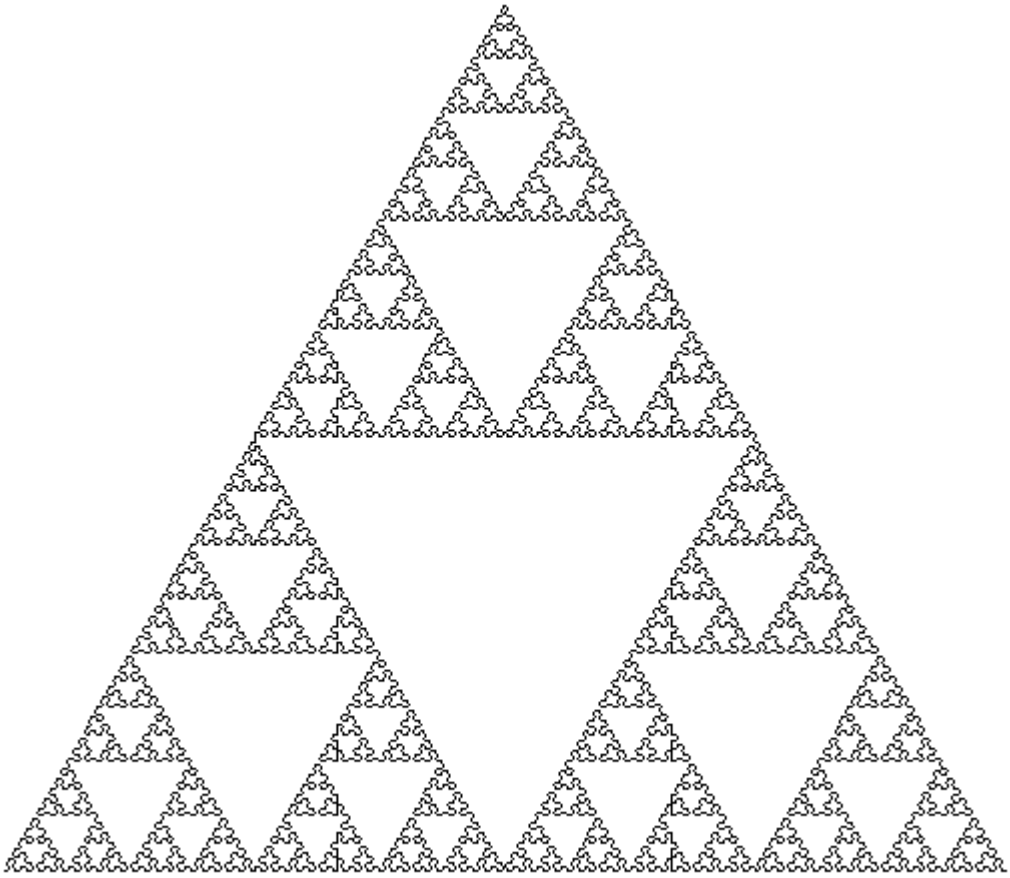
```
void pythBaum(int nrek, double laenge) {
	double alpha = 36.86989765, beta = 53.13010235;
	if (nrek == 0)
		tu.move(laenge);
	else {
		tu.move(laenge);
		tu.turn(90);
		tu.move(laenge);
		tu.turn(90);
		tu.move(laenge);
		tu.turn(90);
		tu.move(laenge);
		tu.turn(180);
		tu.move(laenge);
		tu.turn(-beta);
		if (nrek > 0)
			pythBaum(nrek - 1, 0.8 * laenge);
		else
			tu.move(0.8 * laenge);
		tu.turn(-90);
		if (nrek > 0)
			pythBaum(nrek - 1, 0.6 * laenge);
		else
			tu.move(0.6 * laenge);
		tu.turn(-alpha);
		tu.move(laenge);
		tu.turn(90);
	}
}
```



Pythagorasbaum: n=9 L=100

Pfeilspitzenkurve:

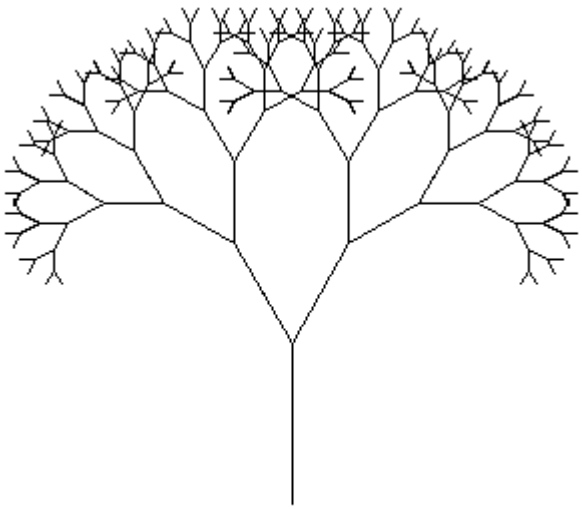
```
void pfeilSpitze(int nrek, int vz, double laenge) {
      if ((nrek > 0) && (laenge > 1)) {
            tu.turn(60 * vz);
            pfeilSpitze(nrek - 1, -vz, laenge / 2);
            tu.turn(-60 * vz);
            pfeilSpitze(nrek - 1, vz, laenge / 2);
            tu.turn(-60 * vz);
            pfeilSpitze(nrek - 1, -vz, laenge / 2);
            tu.turn(60 * vz);
      } else tu.move(laenge);
}
```



Pfeilspitzenkurve: n=8 L=500

Binärbaum:

```
public void binBaum(int nrek, int winkel, double laenge) {
      if (nrek > 0) {
         tu.move(laenge);
         tu.turn(winkel / 2);
         binBaum(nrek - 1, winkel, laenge / 1.4);
         tu.turn(-winkel);
         binBaum(nrek - 1, winkel, laenge / 1.4);
         tu.turn(winkel / 2);
         tu.move(-laenge);
      }
}
```

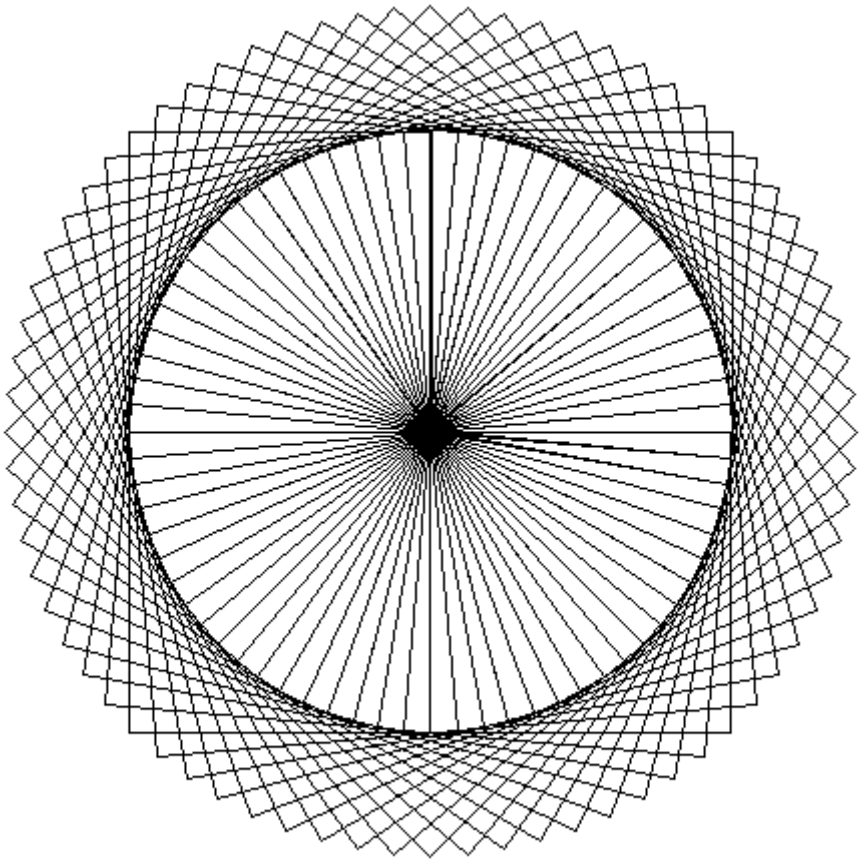


Binärbaum: n=8 L=80 φ=60°

Jetzt noch einige mit der Turtle erzeugte nicht fraktale Figuren:

Quadratrosette:

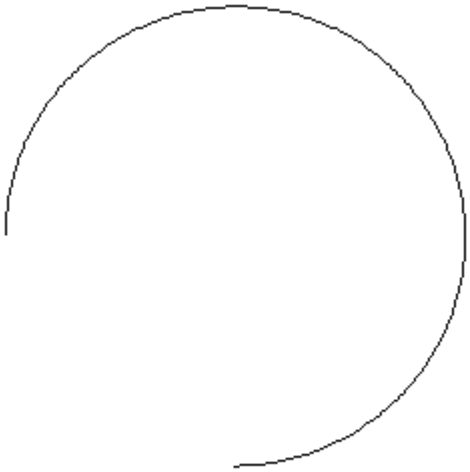
```
void quadrat(int strecke) {
	for (int k = 1; k <= 4; k++) {
		tu.move(strecke);
		tu.turn(90);
	}
}

void quadratRosette(int strecke, int winkel) {
	if (winkel == 0)
		return;
	for (int k = 1; k <= 360 / winkel; k++) {
		quadrat(strecke);
		tu.turn(winkel);
	}
}
```



Quadratrosette: L=150 φ=5°

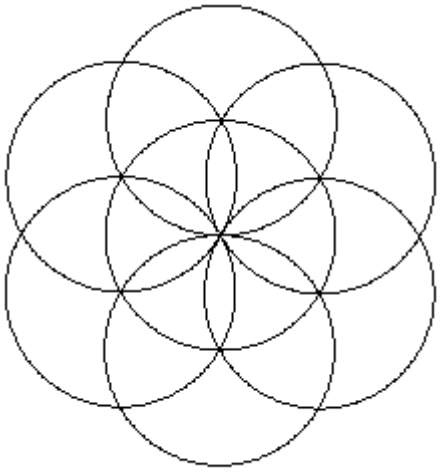
Linksbogen und Siebenkreis:

```
void linksBogen(int strecke, int winkel) {
	for (int k = 1; k <= winkel; k++) {
		tu.move(strecke);
		tu.turn(1);
	}
}
```



Linksbogen: L=2 φ=270°

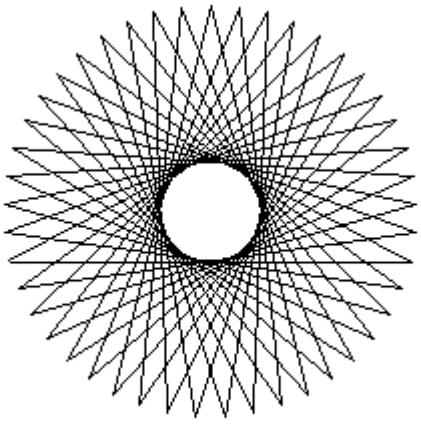
```
void siebenkreis(int strecke) {
	for (int k = 1; k <= 6; k++) {
		linksBogen(strecke, 60);
		// Sechstelkreis
		tu.turn(-60);
		linksBogen(strecke, 360);

		// Kreis
		tu.turn(60);
	}
}
```



Siebenkreis: L=1

Sternvieleck:

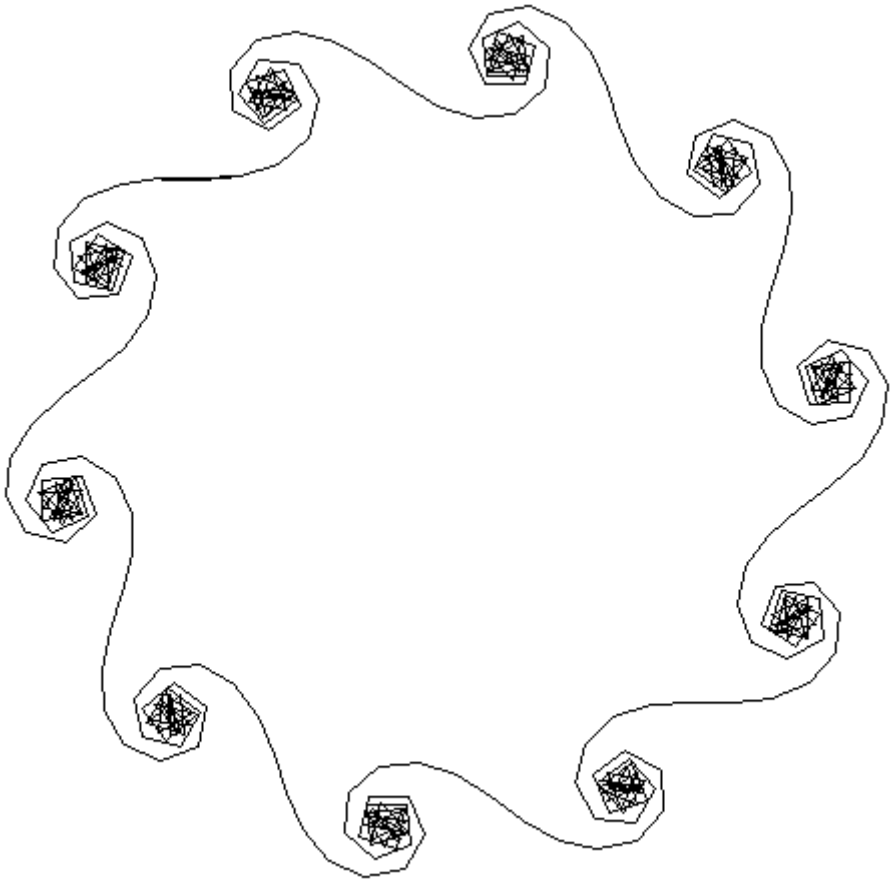
```
void sternVieleck(int strecke, int ablenkWinkel) {
	int gesamtWinkel = 0;
	do {
		tu.move(strecke);
		tu.turn(ablenkWinkel);
		gesamtWinkel = gesamtWinkel + ablenkWinkel;
	} while (gesamtWinkel % 360 != 0);
}
```



Sternvieleck: L=200 φ=152°

Knäuel:

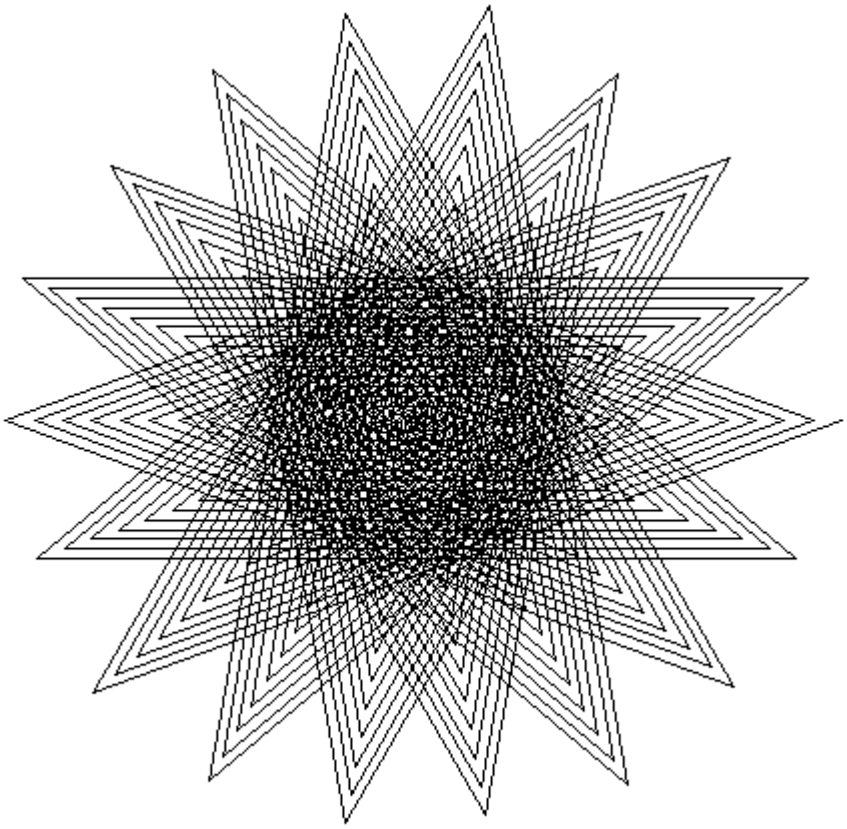
```
void vSchritt(int strecke, int winkel) {
	tu.move(strecke);
	tu.turn(winkel);
}

void knaeuel(int nrek, int strecke, int winkel, int ablenkWinkel) {
	if (nrek >= 1) {
		vSchritt(strecke, winkel);
		knaeuel(nrek - 1, strecke, winkel + ablenkWinkel, ablenkWinkel);
	} // Achtung vor Stacküberlauf !!
}
```



Knäuel: n=2000 L=20 φ=4°

Polygonspirale:

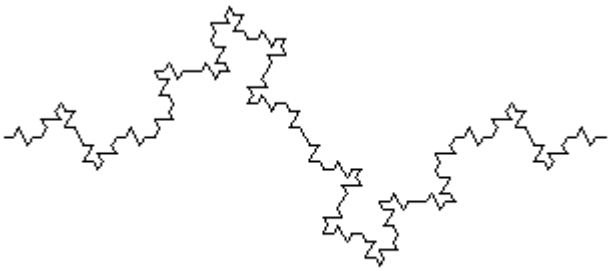
```
	void polygonSpirale(double maxStrecke, int winkel, double zuwachs) {
		double strecke = 1.0;
		while (strecke < maxStrecke) {
			tu.move(strecke);
			tu.turn(winkel);
			strecke += zuwachs;
		}
	}
```



Polygonspirale: maxstrecke=400 φ=140° zuwachs=1,5

Zackenkurve:

```
void zackenkurve(int stufe, double laenge) {
	double teillaenge = laenge / 4.0;
	if (stufe > 0) {
	   zackenkurve(stufe - 1, teillaenge);
	   tu.turn(60);
	   zackenkurve(stufe - 1, teillaenge);
	   tu.turn(-120);
	   zackenkurve(stufe - 1, teillaenge);
	   zackenkurve(stufe - 1, teillaenge);
	   tu.turn(120);
	   zackenkurve(stufe - 1, teillaenge);
	   tu.turn(-60);
	   zackenkurve(stufe - 1, teillaenge);
	} else
		tu.move(laenge);
}
```



Zackenkurve: n=3 L=300