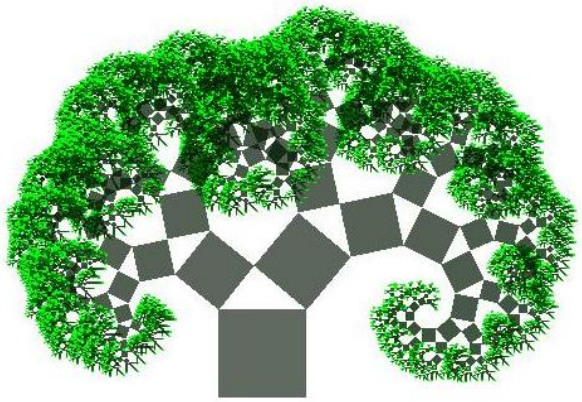
Tutorial: ein Pythagorasbaum als Java-Applet

DominikM



Zusammenfassung

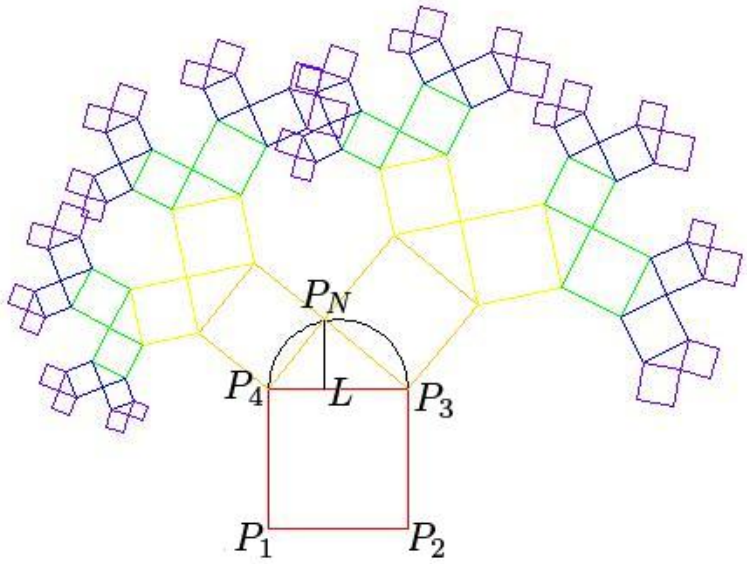
In Teil Eins wird zunächst ein allgemeines Verfahren zum Zeichnen eines Pythagorasbaums beschrieben. Die einzelnen Knackpunkte des Verfahrens werden im Anschluss daran erläutert. Am Ende liegt ein Konstruktionsalgorithmus in Pseudocode vor. Dabei wird besonderen Wert darauf gelegt, dass nur einfache grafischen Funktionen genutzt werden, sodass der Algoritmus auch in Sprachen implementiertbar ist, die nur über wenige grafische Funktionalitäten verfügen.
Im zweiten Teil werden einige Funktionen des *Abstract Window Toolkits* erläutert und eine Art „Feintuning“ vorgenommen. Schließlich folgt die konkrete Implementierung des Algorithmus und die Erstellung des *Java-Applets*.

Inhaltsverzeichnis

1 Geometrische Konstruktion

1.1 Rekursive Konstruktion des Baumes

Als Ursprung des Baums betrachten wir zunächst das untere Quadrat mit den von unten links beginnend gegen den Uhrzeigersinn angeordneten Eckpunkten $P_1(x_1|y_1)$ bis $P_4(x_4|y_4)$. Wir bezeichnen dieses Rechteck als das Quadrat nullter Generation. Man zeichne nun einen Thaleskreis durch die Punkte P_4 und P_3. Nun suche man sich einen beliebigen Punkt $P_N(x_N|y_N)$ auf dem Thaleskreis. Man fälle ein Lot von P_N auf die Strecke $\overline{P_4P_3}$. Sei $L(x_L|y_L)$ der Lotfußpunkt. Das Verhältnis zwischen den Strecken $\overline{LP_3}$ und $\overline{P_4P_3}$ sei k. Man konstruiere nun die beiden Quadrate erster Generation. Eine der Seiten des oben links anliegenden Quadrats ist die Strecke $\overline{P_4P_N}$. Analog ist die Srecke $\overline{P_NP_3}$ eine Seite des anderen Quadrats. Das gleiche Vefahren lässt sich nun beliebig fortsetzen. Dabei ist der Punkt auf dem Thaleskreis so zu wählen, dass das Verhältnis der Strecken gleich k ist.



1.2 Konstruktion eines Quadrats

Bei der Ausführung des oben beschriebenen Algorithmus ist es notwendig an die Strecken $\overline{P_4P_N}$ und $\overline{P_NP_3}$ Quadrate zu konstruieren. Zu diesem Zweck wollen wir nun ein einfaches Verfahren erläutern.
Man beachte, dass sich der Ursprung des Koordinatensystems in der linken oberen Ecke des Bildschirms befindet und die x- und y-Werte nach rechts beziehungsweise nach unten hin größer werden.
Sei $\overline{PQ}$ eine beliebige Strecke. Die Koordinaten der Punkte P und Q seien x_P, y_P und x_Q, y_Q. Man berechne die Differenzen der Koordinaten in beide Richtungen zu

$$dx = x_Q - x_P \text{ und } dy = y_P - y_Q.$$

Die beiden weiteren Eckpunkte des Quadrats ergeben sich zu

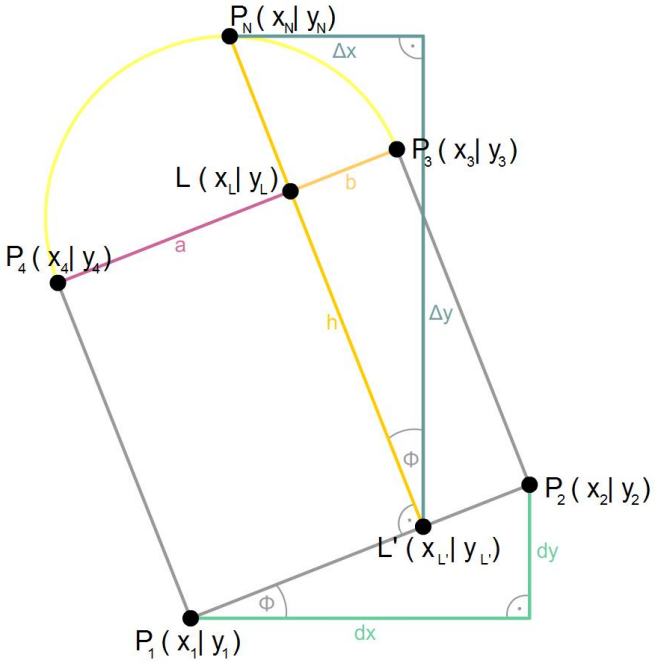
$$R(x_Q - dy|y_Q - dx) \text{ und } S(x_P - dy|y_P - dx).$$

Man verbinde schließlich die Eckpunkte P, Q, R, S und erhalte ein Quadrat.

1.3 Koordinatenbestimmung des Punktes P_N

Unser Ziel ist es nun, eine Formel zu finden, die uns zu einem beliebigem Seitenverhältniss k die Koordinaten des Punktes P_N auf dem Thaleskreis liefert.

Betrachten wir zunächst die allgemeine Situation.



Definieren wir nun das Seitenverhältniss

$$k := \frac{|\overline{LP_3}|}{|\overline{P_4P_3}|} = \frac{|\overline{L'P_2}|}{|\overline{P_1P_2}|} = \frac{b}{a+b}.$$

Dann hat der Punkt $L'(x_{L'}|y_{L'})$ die Koordianten

$$x_{L'} = x_2 - k(x_2 - x_1) = kx_1 + (1-k)x_2 \text{ und } y_{L'} = y_2 + k(y_1 - y_2) = ky_1 + (1-k)y_2.$$

Der Punkt P_N hat die Koordinaten

$$x_N = x_{L'} - \Delta x \text{ und } y_N = y_{L'} - \Delta y.$$

Man beachte hier wieder, dass die y-Werte nach nach unten hin größer werden.

Nun wollen wir Δx und Δy bestimmen. Die Höhe h berechnet sich zu

$$h := |\overline{L'P_N}| = |\overline{L'L}| + |\overline{LP_N}| = |\overline{P_1P_2}| + |\overline{LP_N}|.$$

Nach dem Höhensatz gilt

$$\left|\overline{LP_N}\right|^2 = ab$$

und somit

$$h = \left|\overline{P_1P_2}\right| + \sqrt{ab}.$$

Mittels Substitution von

$$a = (1-k)\left|\overline{P_1P_2}\right| \text{ und } b = k\left|\overline{P_1P_2}\right|$$

in die obere Gleichung ergibt sich

$$h = \left|\overline{P_1P_2}\right| + \sqrt{k(1-k)\left|\overline{P_1P_2}\right|^2} = \left|\overline{P_1P_2}\right|\left(1+\sqrt{k(1-k)}\right).$$

Anwendung von Trigonometrie liefert nun

$$\sin\phi = \frac{\Delta x}{h} = \frac{\Delta x}{\left|\overline{P_1P_2}\right|\left(1+\sqrt{k(1-k)}\right)} \text{ und } \sin\phi = \frac{dy}{\left|\overline{P_1P_2}\right|}.$$

Mittels Gleichsetzen dieser beiden Gleichungen erhält man schließlich

$$\Delta x = \left(1+\sqrt{k(1-k)}\right)dy.$$

Analog erhält man aus

$$\cos\phi = \frac{\Delta y}{h} = \frac{\Delta y}{\left|\overline{P_1P_2}\right|\left(1+\sqrt{k(1-k)}\right)} \text{ und } \cos\phi = \frac{dx}{\left|\overline{P_1P_2}\right|}$$

die Gleichung

$$\Delta y = \left(1+\sqrt{k(1-k)}\right)dx.$$

Somit sind die Koordinaten des Puntes P_N gegeben durch

$$\begin{aligned} x_N &= kx_1 + (1-k)x_2 - \left(1+\sqrt{k(1-k)}\right)dy, \\ y_N &= ky_1 + (1-k)y_2 - \left(1+\sqrt{k(1-k)}\right)dx. \end{aligned}$$

gegeben.

1.4 Vorläufiger Konstruktionsalgorithmus und Endlosproblematik

Mit den oben angestellen Überlegungen ist es nun möglich einen ersten Algorithmus anzugeben.

```
00:   ZeichnePythagorasbaum(Punkt P1, P2){

01:       dx = x2 - x1;
02:       dy = y1 - y2;

03:       x3 = x2 - dy;
04:       y3 = y2 - dx;
05:       x4 = x1 - dy;
06:       y4 = y1 - dx;
07:       NeuerPunkt(P3, x3, y3); //Punkt P3(x3|y3) anlegen
08:       NeuerPunkt(P4, x4, y4);

09:       PolygonZeichnen(P1, P2, P3, P4);

10:       k = 0.5;
11:       xN = k * x1 + (1-k) * x2 - (1 + Wurzel(k * (1-k)) * dy;
12:       yN = k * y1 + (1-k) * y2 - (1 + Wurzel(k * (1-k)) * dx;
13:       NeuerPunkt(PN, xN, yN);

14:       ZeichnePythagorasbaum(P4, PN);
15:       ZeichnePythagorasbaum(PN, P3);

16:   }
```

Beim Aufruf der Funktion *ZeichnePythagorasbaum* müssen zwei Parameter angegeben werden. Dabei handelt es sich zu Beginn um die beiden Eckpunkte der unteren Linie des Quadrates nullter Generation.
In den Zeilen eins bis neun wird gemäß 1.2 ein Quadrat an diese untere Linie konstruiert.
Es folgt zunächst die Angabe des Seitenverhältnis k, dann die Berechnung der Koordinaten x_N, y_N und die Konstruktion des Punktes P_N.
Schlussendlich werden die beiden nächsten Quadrate an das bestehende Quadrat konstruiert. Dies geschieht duch rekursiven Aufruf der Methode *ZeichnePythagorasbaum*, wobei als Parameter nun diejenigen Punkte übergeben werden, welche die Eckpunkte der Unterseite der neuen Quadrate darstellen.

Der Algortihmus kann so aber nicht verwendet werden, da er immer weitere Aufrufe der Funktion *ZeichnePythagorasbaum* durchführt und somit nicht terminiert. Folglich ist ein Abbruchkriterium nötig, um welches der Algorithmus im folgenden Unterkapitel erweitert wird.

1.5 Konstruktionsalgorithmus für einen Baum mit n Generationen

Es bietet sich an, zu Beginn eine Generation festzulegen, bis zu dieser der Baum gezeichnet werden soll. Diese Angabe wird nun beim Funktionsaufruf als dritter Parameter *stufe* übergeben. Zu Beginn der Funktion wird nun geprüft, ob *stufe*≥ 0 ist. Trifft dies zu, so wird zunächst *stufe* dekrementiert und die in 1.4 erläuterte Befehlsabfolge ausgeführt, wobei zu beachten ist, dass beim weiteren rekursiven Aufruf von *ZeichnePythagorasbaum* der dekrementierte Wert von *stufe* übergeben wird. Sind alle gewünschten Generationen gezeichnet, so gilt *stufe*$= 0$ und es werden keine weiteren Aufruf von *ZeichnePythagorasbaum* mehr durchgeführt. Somit terminiert der Algorithmus wie gewünscht.
Da der Algorithmus keine speziellen grafischen Funktionen benötigt, sondern lediglich vorraussetzt, dass Rechtecke gezeichnet werden können, ist dieser auch in den meisten Sprachen implementiertbar, die nur über wenige grafische Funktionalitäten verfügen.

```
00:   AUFRUF: ZeichnePythagorasbaum(Punkt P1, P2, n);

01:   ZeichnePythagorasbaum(Punkt P1, P2, stufe){

02:      if (stufe >= 0) {

03:         stufe--;

04:         dx = x2 - x1;
05:         dy = y1 - y2;

06:         x3 = x2 - dy;
07:         y3 = y2 - dx;
08:         x4 = x1 - dy;
09:         y4 = y1 - dx;
10:         NeuerPunkt(P3, x3, y3);
11:         NeuerPunkt(P4, x4, y4);

12:         PolygonZeichnen(P1, P2, P3, P4);

13:         k = 0.5;
14:         xN = k * x1 + (1-k) * x2 - (1 + Wurzel(k * (1-k)) * dy;
15:         yN = k * y1 + (1-k) * y2 - (1 + Wurzel(k * (1-k)) * dx;
16:         NeuerPunkt(PN, xN, yN);

17:         ZeichnePythagorasbaum(P4, PN, stufe);
18:         ZeichnePythagorasbaum(PN, P3, stufe);

19:      }
20:   }
```

2 Implementierung in Java

2.1 AWT und Applet

Das *Abstract Window Toolkit* (AWT) ist eine von Java angebotene Bibliothek zur einfachen Entwicklung von grafischen Oberflächen. Diese stellt unter anderem Grafik-, Farb- und Punktobjekte, sowie Funktionalitäten zum Zeichnen von Polygonen zur Verfügung, um nur diejenigen zu nennen, die wir bei dieser Applikation benötigen.
Mittels

```
import java.awt.*;
```

werden sämtliche Inhalte von *java.awt* zur Verfügung gestellt.

Nun wird eine Klasse *Pythagorasbaum* erstellt. Da es sich dabei um eine Applet handelt, wird die Klasse von *java.applet.Applet* abgeleitet.
Dies geschieht durch eine Erweiterung der Signatur:

```
public class Pythagorasbaum extends java.applet.Applet { ... } .
```

Anschließend wird die Funktion zum Zeichnen des Baumes deklariert:

```
void ZeichnePythagorasbaum(Graphics g, Point p1, Point p2, int stufe) .
```

Zu beachten ist, dass im Weiteren ein Grafikobjekt als Parameter der Funktion benötigt wird.

Beim Start des Applets wird die Methode *paint* in der Oberklasse aufgerufen. Durch Überschreiben der Methode in der Unterklasse *Pythagorasbaum* kann die Methode zum Zeichnen verwendet werden:

```
public void paint(Graphics g) {
   int stufe = 5;
   Point p1 = new Point(250, 355);
   Point p2 = new Point(300, 355);
   ZeichnePythagorasbaum(g, p1, p2, stufe);
} .
```

Innerhalb der Methode *paint* bietet es sich außerdem an, die jüngste Generation festzulegen und die beiden Eckpunkte der unteren Linie des Ausgangsquadrats anzugeben. Die Instanzierung der Punkte erfolgt mittels Angabe der jeweiligen x- und y-Koordinaten. Auf die Koordinaten der Punkte kann anschließend mittels $p1.x$ beziehungsweise $p1.y$ zugegriffen werden.

Ein Polygon kann mit Java-AWT durch Angabe seiner Eckpunkte festgelegt werden. Zum Zeichnen stehen die Methoden des Grafikobjekts *drawPolygon* und *fillPolygon* zur Verfügung, wobei Ersteres ausschließlich die Seiten und Letzteres auch die Inhalt des Polygons zeichnet.
Zum Erstellen und Zeichnen eines Quadrats kann beispielsweise der folgende Code verwendet werden.

```
Polygon quadrat = new Polygon();
quadrat.addPoint(p1.x, p1.y);
quadrat.addPoint(p2.x, p2.y);
quadrat.addPoint(p3.x, p3.y);
quadrat.addPoint(p4.x, p4.y);
g.drawPolygon(quadrat);
```

Eine Instanz der Klasse *Color* stellt eine Farbe dar, deren Aufbau sich nach dem RGB-Modell richtet. Die Angabe eines Farbtons erfolgt durch Angabe der Rot-, Grün und Blauanteile. Die Instanzierung erfolgt mittels

```
Color gruen = new Color(0.0f, 1.0f, 0.0f, 1.0f);
```

wobei die ersten drei Parameter die Rot-, Grün- und Blauanteile von 0 bis 100% angeben und der letzte optionale Parameter die Farbintensität bestimmt. Es sind jeweils Werte vom Typ Float zwischen 0 und 1 zu verwenden. Alternativ können auch Integerwerte zwischen 0 und 255 verwendet werden, wie sie beim RGB-System üblich sind.
Um dem Grafikobjekt die oben angegebene Farbe zuzuweisen kann der Befehl

```
g.setColor(gruen);
```

benutzt werden.

2.2 „Feintuning“: Farb- und Parameterwahl

Da Stamm, Äste und Blätter in der Realität unterschiedliche Farbtöne aufweisen, bietet es sich an, die Farbe in Abhängigkeit der Rekursionstiefe zu variieren. Dabei ist zu beachten, dass die einzelnen Farbanteile, sowie die Intensität bei allen Aufrufen der Methode zwischen 0 und 1 liegen.
In späteren Applet wird dazu der Rot- und Blauanteil konstant gleich Null gehalten. Grünanteil und Intensität nehmen mit zunehmender Rekursionstiefe ab, sodass der Stamm des Baumes einen fast gräulichen Ton erhält und die Blätter in unterschiedlichen Grüntönen gezeichnet werden, wodurch diese leicht plastisch wirken. Es bietet sich hier an, die Intensität schwächer abklingen zu lassen als den Grünanteil, da so eine intensivere Darstellung des Stamms ereicht werden kann.
Natürlich sind auch andere Einfärbemethoden möglich. Zum Beispiel kann man mittels einfacher Abfragen jeder Rekursionstiefe eine andere Farbe zuweisen.
Möchte man eine bunten Baum erhalten, so kann man jedem Quadrat eine zufällige Farbe zuordnen.
Dies ist mittels

```
g.setColor(new Color( (int) (Math.random() * 255),
                      (int) (Math.random() * 255),
                      (int) (Math.random() * 255)));
```

möglich.

Außerdem besteht die Möglichkeit, das Seitenverhältnis k in Abhängigkeit der Rekursionstiefe zu verändern oder mittels

```
k = Math.random();
```

zufällig festzulegen oder zu variieren.
Bei festem Seitenverhältnis erhält man im Fall $k = 0.5$ einen symmetrischen Baum.
Außerdem besteht mittels

```
g.fillRect(x, y, width, height);
```

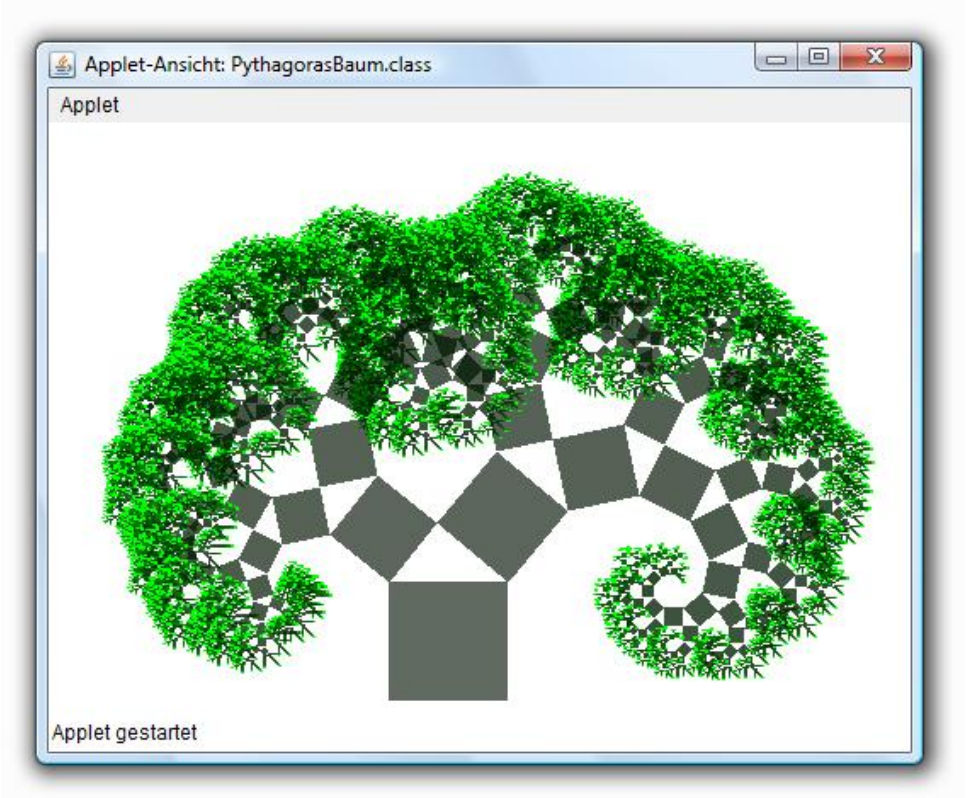
die Möglichkeit das Quadrat nullter Generation durch Zeichnen eines zweiten Rechtecks nach unten hin zu vergrößern, sodass der Stamm länger wird.

2.3 Java Applet zum Pythagorasbaum

Mit dem Algorithmus aus 1.5 und den in 2.1 und 2.2 erläuterten Techniken und Überlegungen kann nun das Applet programmiert werden.

```
00:   import java.awt.*;

01:   public class Pythagorasbaum extends java.applet.Applet {

02:      public void ZeichnePythagorasbaum(Graphics g, Point p1, Point p2,
                                           int stufe, double k) {

03:        if (stufe >= 0) {

04:          g.setColor(new Color(0.0f, 1.0f / (stufe + 1.0f),
                                  0.0f, 25.0f /(stufe + 25.0f)));
05:          stufe--;

06:          int dx = p2.x - p1.x;
07:          int dy = p1.y - p2.y;

08:          Point p3 = new Point(p2.x - dy, p2.y - dx);
09:          Point p4 = new Point(p1.x - dy, p1.y - dx);

10:          Polygon quadrat = new Polygon();
11:          quadrat.addPoint(p1.x, p1.y);
12:          quadrat.addPoint(p2.x, p2.y);
13:          quadrat.addPoint(p3.x, p3.y);
14:          quadrat.addPoint(p4.x, p4.y);
15:          g.fillPolygon(quadrat);

16:          int xN = (int) (k * p1.x + (1 - k) * p2.x
                        - (1 + Math.sqrt(k * (1 - k))) * dy);
17:          int yN = (int) (k * p1.y + (1 - k) * p2.y
                        - (1 + Math.sqrt(k * (1 - k))) * dx);
18:          Point pN = new Point(xN, yN);

19:          ZeichnePythagorasbaum(g, pN, p3, stufe, k);
20:          ZeichnePythagorasbaum(g, p4, pN, stufe, k);
21:        }
22:      }

23:      public void paint(Graphics g) {
24:        int stufe = 15;
25:        double k = 0.6;
26:        Point p1 = new Point(200, 340);
27:        Point p2 = new Point(270, 340);
28:        ZeichnePythagorasbaum(g, p1, p2, stufe, k);
29:      }
30:   }
```

Das fertige Applet sieht nun wie folgt aus.



2.4 Alternative Varianten

Alternativ befindet sich links ein symmetrischer Pythagorasbaum mit den Generationen 0 bis 10, verlängertem Stamm und zufällig generierten Farben.
Rechts ist ein Baum mit Generationen 0 bis 5 und zufällig gewählten Seitenverhältnissen zu sehen, der stufenweise gemäß der Abfolge im Lichtspektrum eingefärbt ist.