Dokumentation SVG-Fraktale

Einreichung für den Bundeswettbewerb Informatik

Aufgabe 4

Die gewählte Grundfigur ist die erste vorgegebene, der Sierpínski-Teppich.

„Wähle eine der auf www.bundeswettbewerb-informatik.de bereit gestellten Grundfiguren oder denke dir selbst eine Grundfigur aus. Schreibe ein Programm, das die Grundfigur oder eine bereits berechnete Verfeinerungsstufe aus einer SVG-Datei einliest und eine neue SVG-Datei erzeugt, die die nächste Stufe beschreibt. Die SVG-Dateien sollen von gängigen Web-Browsern (z. B. Firefox) angezeigt werden können.
Als Programmiersprache zur Verarbeitung von XML-Dialekten wie SVG eignet sich XSLT, aber auch andere Sprachen sind zugelassen. Es genügt, wenn eine Ausführung des Programm, nur einen Verfeinerungsschritt vornimmt.“

Installation

Schnellstart: Auf CD liegt in dem Verzeichnis „*schnellstart*“ ein von der CD lauffähiger Webserver bei. Kopieren sie den Ordner „schnellstart“ auf die Festplatte (z.B. Desktop) und klicken Sie in dem kopierten Verzeichnis auf „*schnellstart.exe*“. Eine portable Firefox-Version wird starten und die Präsentationsseite des SVG-Generators anzeigen. Sobald Sie Firefox schließen wird der mitgelieferte Server wieder herunterfahren.

Manuell: Sofern Sie das Programm auf bereits installierten Webservern prüfen möchten, benötigen Sie jenen Webserver (wie z.B. Apache HTTP) mit der PHP-Erweiterung. In das „htdocs“-Verzeichnis des Servers kopieren Sie den Ordners „*svg-fractals*“ von der beigelegten CD. Sofern Sie den Server gestartet haben, können Sie das Programm mit einem aktuellen Webbrowser (empfohlen werden mindestens Mozilla Firefox 4.0 oder Chrome 4.0) unter der Adresse *http://localhost/svg-fractals/index.php* erreichen.

CD-Inhalt

- schnellstart/
  - schnellstart.exe
  - server2go/
- svg-fractals/
  - carpet.svg.php
  - CarpetAnalyzer.php
  - CarpetGenerator.php
  - CarpetUtils.php
  - index.php
- docs/
  - carpet.svg (Beispielsdatei für Input-Funktion)
  - Dokumentation.pdf

Mathematische Erkenntnisse

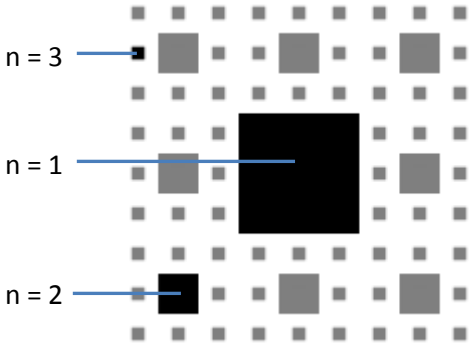
Die folgenden Erkenntnisse wurden bei der Problemlösung von mir persönlich entwickelt und erfordern daher keine Quellenangabe.

In den folgenden Beschreibungen sowie im Quelltext werden die Verfeinerungsstufen eines Quadrates „Stages“ genannt.

Formeln:

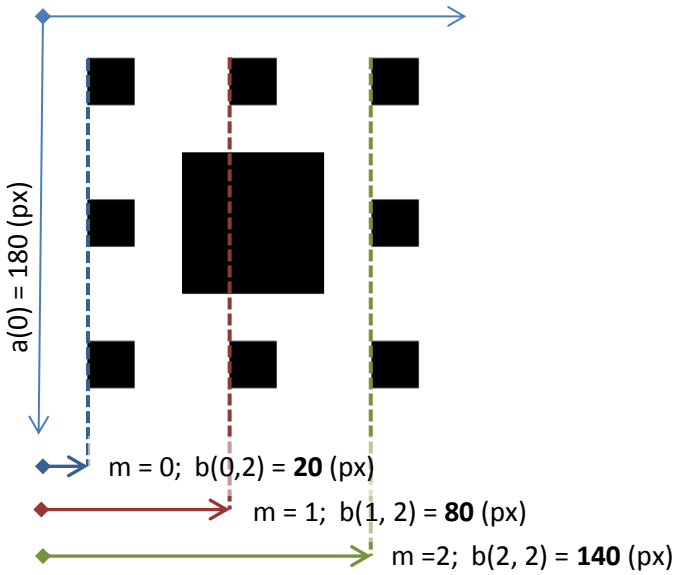
- Seitenlängen der Quadrate: a(n)= a(0)/3^n
  - a(0) = Seitenlänge der Grundfläche
  - n = Stage eines Quadrates
    (Stage, siehe Zeichnung am rechten Seitenrand)
- Abstand zum Grundflächenrand (Koordinaten):
  - b(m,n) =a (n)+m·a(n-1)
  - m = x-tes Quadrat – 1

Die **absolute Seitenlänge** eines Quadrates der Stage *n* lässt sich abhängig von der Grundseite der Zeichenfläche a(0) berechnen. Vom BWINF sind 180 Pixel vorgegeben, also ist a(0) = 180(px).
Die genaue Bestimmung der „Stages“ wird in der Zeichnung rechts verdeutlicht.
→ a(n)= a(0)/3^n



Quadrate der Stages 1 bis 3

Der **Abstand eines Quadrates** der Stage *n* zum Seitenrand lässt mit der oben beschrieben Formel berechnen. m besagt hier, dass das Quadrat das sovielte Quadrat in der Reihe vom Seitenrand ist. Von m wird immer 1 subtrahiert. Folglich gilt für das 1. Quadrat m = 0, für das 3. Quadrat gilt m = 2. In der unteren Zeichnung wird dies grafisch verdeutlicht.
→ b(m,n) =a (n)+m·a(n-1)

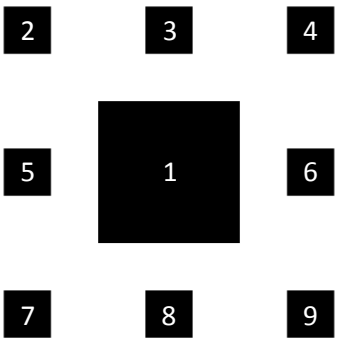


Lösungsidee

Da die Elemente (Quadrate) dieses Fraktales (Sierpínski-Teppich) in einem Raster angeordnet sind, kann man diese Zeilen- und Spaltenweise generieren lassen. Die Rastergröße orientiert sich an den kleinsten Quadraten der Stage.

Algorithmus

Das Programm („*CarpetGenerator.php*“) durchläuft in verschachtelten Schleifen die Verfeinerungsstufen (große Quadrate zuerst, kleine zuletzt) und errechnet in jeder Stage die Koordinaten und die Seitenlänge der Quadrate. Hierbei arbeitet es wie beim Schreiben der deutschen Sprache: von links nach rechts und dann von oben nach unten.
In der Zeichnung rechts sind die Quadrate in der Reihenfolge ihrer Generierung beschriftet.



Generierungsreihenfolge

Die Analyse

Schreibe ein Programm, das die Grundfigur oder eine bereits berechnete Verfeinerungsstufe aus einer SVG-Datei einliest und eine neue SVG-Datei erzeugt, die die nächste Stufe beschreibt.

In der Aufgabe ist gefordert, dass SVG-Dateien eingelesen werden können. Die Analyse dieser Dateien übernimmt die Klasse „CarpetAnalyzer.php“. Sie nimmt (im Konstruktor) als Parameter den Dateinamen einer .svg-Datei an. Nach der Analyse gibt sie folgende Werte zurück:

- carpet_size – Die Seitenlänge der Grundfläche (= a(0))
- stages – Anzahl der Stages, die in diesem Teppich berechnet wurden

Die carpet_size wird aus dem Attribut „height“ (oder „width“, da Bild quadratisch ist äquivalent) der SVG ermittelt. Der „stages“-Wert wird durch Analyse aller <rect>-Objekte festgestellt. Durch Umstellung der oben beschrieben Formel zur Berechnung der Seitenlängen, a(n)= a(0)/3^n, lässt sich die Stage eines Quadrates durch Angabe der Seitenlänge berechnen. Die nach n (Stage) umgestellte Formel lautet: $n = \log_3(a(0)/a(n))$
Mithilfe dieser Formel wird zu jedem der gegebenen <rect>-Objekte die Stage notiert. Zuletzt wird der höchste errechnete Stage-Wert gesucht, dieser ist dann der Rückgabewert „stage“.

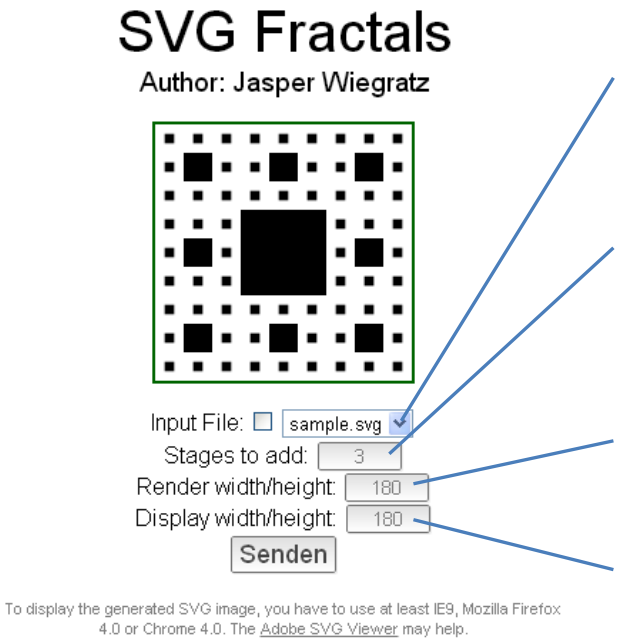
Dateien die analysiert werden sollen, werden im Arbeitsverzeichnis des Programms gesucht. Dies ist bei der manuellen Installation das Verzeichnis „htdocs/svg-fractals/“ und bei der automatischen Installation „schnellstart\server2go\htdocs\svg-fractals“. Kopieren Sie hier die zu analysierenden .svg-Dateien hin, sie können dann später auf dem Formular ausgewählt werden.

Programmaufbau

Das Programm besteht aus 3 Dateien:

1. **index.php** – Dies ist eine reine Präsentationsschicht, die automatisch vom Browser aufgerufen wird und das SVG-Bild (siehe 2.) einbindet sowie das Formular zur Anpassung der Generator-Parameter darstellt.
2. **carpet.svg.php** – Dies ist eine PHP-Seite, die für den Browser als .svg interpretiert wird, indem die MIME Typ-Angabe im HTTP-Header von „*text-html*“ zu „*image/svg+xml*“ geändert wird. Außerdem ruft die Seite den Generator (3.) auf.
3. **CarpetAnalyzer.php** – Diese Klasse analysiert eine SVG-Datei auf die Werte „stages“ und „canvas_size“. Ihr Verhalten wird weiter oben erläutert.
4. **CarpetGenerator.php** – Die Generator-Klasse erzeugt ein PHP-Array mit den Informationen aller Quadrate und gibt diese später in SVG-Notation zurück. Der Algorithmus ist in dieser Klasse implementiert (`generateCarpet()`).
5. **CarpetUtils.php** – Diese Klasse enthält Methoden, die global vom Programm benutzt werden. Dazu zählen die oben beschriebenen Formeln als Methoden sowie eine Verzeichnis-Auflistungs-Methode (benötigt für das Formular).

Erklärung des Formulars



SVG-Auswahl: Hier kann eine SVG-Datei ausgewählt werden, deren Stages ermittelt und um „Stages to add“ erhöht werden soll.

Stage-Anzahl: Diese Anzahl an Stages wird zu der unter „Input File“ gewählten SVG-Datei hinzugefügt. Ist „Input File“ deaktiviert (Checkbox aus), wird eine neue Datei mit der gegeben Stage-Anzahl erstellt.

Render-Größe: Dies ist die Länge der Grundseite in px, mit der der Teppich berechnet werden soll. Da SVG-Bilder beliebig skalierbar sind, hat dieser Wert keinen direkten Einfluss auf die Darstellung.

Anzeige-Größe: Dies ist die Breite/Höhe, mit der das SVG auf dieser Seite abgebildet werden soll.

Hinweis zur Speichernutzung

Bei der Ausführung des Fraktal-Generators ist besonders auf die „Stage“-Größe zu achten, also die Anzahl der Verfeinerungsschritte. Da die Anzahl neuer Elemente („unsichtbare“, überlagerte Objekte eingeschlossen) zunehmend schnell steigt, wird bei größeren Verfeinerungsstufen erheblich mehr Arbeitsspeicher benötigt. Da PHP standardmäßig ein Speicherlimit von 128MB gesetzt hat, wird das Programm ab einschließlich 7 Stages aufgrund des Speicherbedarfs abbrechen.

Quelltext (Auszug CarpetGenerator.php):

```php
<?php

/**
 * @author Jasper Wiegratz
 */
class CarpetGenerator {

  //Defining standard parameters
  private $stages = 5;
  private $canvas_size = 180;
  private $canvas_dsize = 180;

  /**
   * @param Array $args Should contain the keys 'canvas_size' and 'stages'
(default: 5)
   *        canvas_size: Side length of the canvas (defaults to 180px)
   *        stages: Amount of refinements
   */
  public function __construct($args) {
    if(isset($args['stages'])) $this->stages = $args['stages'];
    $this->canvas_size = $args['canvas_size'];
    $this->canvas_dsize = $args['canvas_dsize'];
  }

  /**
   * @param int $stage Stage (of refinement)
   * @param int $canvas_size Size of the (SVG) canvas
   * @return int Side length of square of given stage
   */
  private function getSquareSidelength($stage, $canvas_size){
    //<size> / 3^<stage>
    //pow(number $base, number $exp ) = $base^$exp
    return($canvas_size/pow(3, $stage));
  }

  /**
   * @param int $stage Stage (of refinement)
   * @param int $canvas_size Size of the (SVG) canvas
   * @param int $n $n-th element in one row
   * @return int Coordinate (one dimension) of n-th square of given stage
   */
  private function getSquarePosition($stage, $canvas_size, $n){
    $sq_a = $this->getSquareSidelength($stage, $canvas_size);
    return($sq_a + 3*($n-1)*$sq_a);
  }

  /**
   * @return Array with $ret[$stage][$square]
   */
  private function generateCarpet() {

    $carpet = array();
    $cs = $this->canvas size;
    for($stage = 1; $stage <= $this->stages; $stage++){
      //Executed per stage
      //sq_a: square sidelength
      $sq_a = $this->getSquareSidelength($stage, $cs);
```

```
      $sq_per_row = pow(3, $stage-1);
      $sq_total = $sq_per_row * $sq_per_row;

      for($x_row = 1; $x_row <= $sq_per_row; $x_row++){
        //Executed per x-row
        $x row coord = $sq a;
        for($y_row = 1; $y_row <= $sq_per_row; $y_row++){
          //Executed per y-row on x-row => x/y coordinate
          $carpet[$stage][] = array(
          'x' => $this->getSquarePosition($stage, $cs, $y_row),
          'y' => $this->getSquarePosition($stage, $cs, $x_row),
          's' => $this->getSquareSidelength($stage, $cs)
          );
        }
      }
    }
    return($carpet);
  }

  private function generateSVG($carpet){
    $svg = '';
    $ca = $carpet;
    foreach($ca as $stageno=>$ca_stage){
      //Executed per stage
      $svg .= "\n<!-- Stage: $stageno -->";
      foreach($ca_stage as $sq_array){
        //Executed per square in stage
        //sq: square
        (float) $x = $sq_array['x']; //first x-coordinate of square
        (float) $y = $sq_array['y']; //first y-coordinate of square
        (float) $s = $sq array['s']; //square size
        $svg .= "\n<rect x=\"$x\" y=\"$y\" width=\"$s\" height=\"$s\" />";
      }
    }
    return($svg);
  }

  /**
   * @return String Generated squares as SVG markup
   */
  public function generateCarpetSVG() {
    $carpet = $this->generateCarpet();
    $svg = $this->generateSVG($carpet);
    return($svg);
  }

}
?>
```

Produkt (SVG-Auszug):

```
<?xml version="1.0" encoding="iso-8859-1"?>
<svg width="180px" height="180px" xmlns="http://www.w3.org/2000/svg">

<!-- Stage: 1 -->
<rect x="60" y="60" width="60" height="60" />
<!-- Stage: 2 -->
<rect x="20" y="20" width="20" height="20" />
<rect x="80" y="20" width="20" height="20" />
<rect x="140" y="20" width="20" height="20" />
<rect x="20" y="80" width="20" height="20" />
<rect x="80" y="80" width="20" height="20" />
<rect x="140" y="80" width="20" height="20" />
<rect x="20" y="140" width="20" height="20" />
<rect x="80" y="140" width="20" height="20" />
<rect x="140" y="140" width="20" height="20" />
<!-- Stage: 3 -->
<rect x="6.6666666666667" y="6.6666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="26.666666666667" y="6.6666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="46.666666666667" y="6.6666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="66.666666666667" y="6.6666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="86.666666666667" y="6.6666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="106.66666666667" y="6.6666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="126.66666666667" y="6.6666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="146.66666666667" y="6.6666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="166.66666666667" y="6.6666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="6.6666666666667" y="26.666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="26.666666666667" y="26.666666666667" width="6.6666666666667"
height="6.6666666666667" />
<rect x="46.666666666667" y="26.666666666667" width="6.6666666666667"
height="6.6666666666667" />

<!-- Restliche Quadrate entfernt -->

</svg>
```

(Die Elemente der 3. bis 5. Ebene wurden aufgrund des hohen Platzbedarfs weggelassen)