


## Praktikum Computergrafik, Blatt 2

### **\*\* Aufgabe 1** (Die Mandelbrot-Menge)

In Moodle  finden Sie die Datei `mandelbrot.zip` mit den Rahmenprogrammen:

**MandelbrotPanel.java:** JPanel-Klasse mit main-Routine zur Darstellung, muss nicht editiert werden, Sie können da aber mit der Anzahl der Iterationsschritte (`maxiter`) und dem Farbschema (`colorScheme`) experimentieren.

Mit den Tasten des numerischen Ziffernblocks kann man in der Darstellung navigieren (Anleitung im Code).

**Mandelbrot.java:** Das ist die Klasse, in der Sie den Mandelbrotalgorithmus implementieren müssen.

Ich entschuldige mich für den kriminell vor Parametern überbordenden Konstruktor. Ist der Trennung von Boilerplate- und zu implementierenden Code geschuldet.

**MandelbrotTest.java:** JUnit4-Test, muss nicht editiert werden. Identisch mit dem vom APA-Server ausgeführten Test.

**colorscheme:** Ein Package mit Farbschemata, mit denen man die Mandelbrotmenge und ihre Umgebung einfärben kann. Sie können sich da auch selber was einfallen lassen. Am schönsten ist `ColorSchemeHue`.

a) Implementieren Sie in `Mandelbrot.java` zunächst die Methode

```
double transformPx(int px)
```

die ein  $x_{\text{LKOS}}$  in ein  $x_{\text{GKOS}}$  umrechnet. D.h. die Funktion soll die lineare Funktion implementieren, die u.a. 0 auf `xMin` und `width` auf `xMax` abbildet. Stellen Sie dazu die in der Vorlesung besprochene Gleichung zur Berechnung  $x_{\text{LKOS}} \rightarrow x_{\text{GKOS}}$  um.

Implementieren Sie dann das analoge

```
double transformPy(int py)
```

für die  $y$ -Werte.

b) In `render` wird in einer Doppelschleife über alle Pixel iteriert. Für jeden Pixel wird mit den beiden Methoden, die Sie gerade implementiert haben, aus dem  $x$ -Wert der Real- und aus dem  $y$ -Wert der Imaginärteil einer komplexen Zahl  $c$  berechnet.

Nun sollen Sie eine Iteration einbauen, die die komplexen Werte  $z_n$  nach folgender Gesetzmäßigkeit berechnet:

$$z_0 = 0$$
$$z_{n+1} = z_n^2 + c$$

Dies wird solange durchgeführt, bis die maximale Zahl von Iterationen `maxIter` erreicht wurde, oder bis  $|z_n|^2 \geq 4$ .

Zur Erinnerung (?):

- Die imaginäre Einheit  $i$  ist definiert über

$$i^2 = -1$$

- Eine komplexe Zahl  $z$  ist definiert als

$$z = x + i \cdot y$$

mit  $x \in \mathbb{R}$  dem Realteil,  $y \in \mathbb{R}$  dem Imaginärteil.

- Die Addition zweier komplexer Zahlen  $z_1 = x_1 + iy_1$  und  $z_2 = x_2 + iy_2$

$$z_1 + z_2 = (x_1 + x_2) + i(y_1 + y_2)$$

ist also eine komplexe Zahl mit Realteil  $x_1 + x_2$  und Imaginärteil  $y_1 + y_2$ .

- Der Betrag  $|z|$  einer komplexen Zahl  $z = x + iy$ :

$$|z| = \sqrt{x^2 + y^2}$$

- Das Quadrat  $z^2$  einer komplexen Zahl  $z = x + iy$


$$z^2 = (x^2 - y^2) + i(2xy)$$

ist also eine komplexe Zahl mit Realteil  $x^2 - y^2$  und Imaginärteil  $2xy$ .

- c) *Optional*: In der momentanen Version ist der Farbton von der Zahl der durchlaufenen Iterationen abhängig. Wurde `maxIter` erreicht, so werden die Punkte schwarz gefärbt. Für `maxIter = ∞` wäre dies die fraktale Mandelbrotmenge.

Experimentieren Sie mit alternativen ColorSchemes, die Sie in `MandelbrotPanel` setzen können (s.o.).

## **\*\* Aufgabe 2** (Polygon-Rasterung mit Scanline-Algorithmus)

In Moodle  finden Sie die Datei `scanline.zip` mit den Rahmenprogrammen:

**PolygonRastererPanel.java**: `JPanel`-Klasse mit `main`-Routine zur Darstellung, muss nicht editiert werden. Sie können aber im Konstruktor verschiedene Testfälle einstellen (`zigzag`, `cg`, `stars`).



zigzag



cg



stars

**PolygonRasterer.java:** Diese Klasse sollte den Scanline-Algorithmus implementieren.

**Edge.java:** Datenstruktur für eine Kante im Scanline-Algorithmus.

**Polygon.java:** Hilfsklasse, durch die sich leicht Polygone erstellen lassen und die eine Liste von Edges liefert.

**Star.java:** Erstellt ein Sternen-Polygon.

**PolygonRastererTest.java:** JUnit4-Test, muss nicht editiert werden. Identisch mit dem vom APA-Server ausgeführten Test.

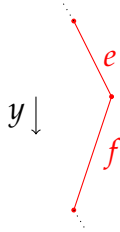
Setzen Sie den Scanline-Algorithmus in der Methode `rasterize` der Klasse `PolygonRasterer` um.

Dazu ein paar Anmerkungen:

- Zur Vereinfachung nehmen wir an, dass sich Polygone immer komplett im Viewport befinden, den Rand also nicht schneiden.
- Das im Vorlesungsskript erwähnte Sortieren der Kanten in der ET nach  $y_{\min}$  dürfen Sie aus Gründen der Vereinfachung weglassen. Sie können die ET für die Suche nach Kanten, für die  $y_{\min}$  erreicht wurde, komplett durchsuchen.
- Für das Kopieren oder Verschieben der Kanten von der ET in die AET bei Erreichen von deren  $y_{\min}$  müssen Sie möglicherweise den Copy-Konstruktor einsetzen. Zumindest bei mir wurde das Panel zweimal gezeichnet und beim zweiten Mal müssen die  $x_{\text{Schnitt}}$ -Werte der Kanten richtig initialisiert sein.
- Das Sortieren der AET nach  $x_{\text{Schnitt}}$  kann man z.B. mit `LinkedList.sort` und einem `Comparator` machen.
- Das Zeichnen der horizontalen Liniensegmente von Schnittpunkt zu Schnittpunkt darf mit `Graphics.drawLine` erfolgen. (Sie müssen nicht Ihren Bresenham anwerfen.)
- Das Entfernen von Kanten aus der AET bei Erreichen von  $y_{\max}$  kann man zweistufig machen. Erst die zu entfernenden Kanten in einer Liste sammeln und dann mit deren Hilfe und `removeAll` aus der AET werfen. Vielleicht fällt Ihnen aber was eleganteres ein.

## Fragen zum Praktikum (prüfungsrelevant)

- 1.) Der Scanline-Algorithmus zum Polygon-Rastern muss an der Ecke zwischen zwei aufeinander folgenden Kanten, an denen der  $y$ -Wert monoton steigt, aufpassen, dass er keine inkonsistente Zahl von Kanten in der AET hat.



Ohne Anpassungen wäre  $y_{\max}$  der Kante  $e$  gleich mit  $y_{\min}$  der Kante  $f$  und beide Kanten könnten zeitgleich in der AET landen. Daher reduzieren wir den  $y_{\max}$ -Wert der Kante  $e$  um 1.

Welches möglicherweise unerwünschte Verhalten handelt man sich damit ein?

- 2.) Eine alternative Idee, um die Konsistenz der Kantenzahl in der AET sicherzustellen, wäre die folgende: Ziehe das Entfernen von AET-Kanten, bei den  $y_{\max}$  erreicht wurde, zu der Stelle vor, wo Kanten mit erreichtem  $y_{\min}$  aus der ET in die AET hinzugefügt werden:

```

for all  $y \leftarrow 0, 1, \dots$  do
    Kanten mit  $y_{\min} = y$  werden in AET aufgenommen.
    Entferne Kanten mit  $y_{\max} = y$  aus AET.
    Sortiere AET bzgl.  $x_{\text{Schnitt}}$ .
    Fülle Pixel zwischen Paaren von  $x$ -Koordinaten
      aus der AET.
    Aktualisiere für alle Kanten der AET die
       $x_{\text{Schnitt}}$ -Werte:  $x_{\text{Schnitt}} \leftarrow x_{\text{Schnitt}} + 1/m$ 
end for
  
```

Welche Risiken und Nebenwirkungen ergeben sich hierdurch?

- 3.) Der Scanline-Algorithmus zum Polygonrastern füllt nach der „Even-Odd“-Regel.

Wie würden die folgenden sich komplett im Viewport befindlichen und sich schneidenden Polygone gefüllt werden?

