Dr. Ole Klein, René Heß
IWR, Universität Heidelberg

Aufgabenblatt 8

Allgemeine Hinweise:

- Für die Aufgaben auf diesem Übungsblatt müssen Sie am 22.01. votieren.
- Den Quellcode der Programme können Sie von der Vorlesungs-Homepage https://conan.iwr.uni-heidelberg.de/teaching/ipk_ws2020/ herunterladen.

Aufgabe 1: Die Mandelbrot-Menge (1 Punkt)

In einer der vorherigen Aufgaben haben Sie eine Klasse `Canvas` (deutsch: Leinwand) erstellt. In dieser Aufgabe wollen wir die von Ihnen erstellte Klasse für eine konkrete Anwendung benutzen.

Die Mandelbrot-Menge ist die Menge aller komplexen Zahlen $c \in \mathbb{C}$, für die die Funktion $f_c(z) = z^2 + c$, als Iterationsvorschrift aufgefasst und mit $z_0 = 0$ gestartet, Bahnen erzeugt, die ganz innerhalb eines Kreises mit endlichem Radius bleiben [1]. Eng verbunden ist diese Menge mit den Julia-Mengen, bei denen stattdessen alle Startpunkte z_0 gesucht sind, die bei einem fest gewählten $c \in \mathbb{C}$ solche Bahnen erzeugen. Diese beiden Mengen wollen wir im folgenden visualisieren, wobei wir mit der Mandelbrot-Menge beginnen.

Hinweis: Da diese Aufgabe auf Ihren bisherigen Arbeiten aufbaut, wird Ihnen auf der Homepage eine Beispielimplementierung für `Canvas` und `Point` zur Verfügung gestellt, so dass Sie in jedem Fall eine geeignete Grundlage haben, falls Ihre Implementierung nicht ganz vollständig sein sollte. Falls Sie diese Implementierung verwenden, greifen Sie auf die einzelnen Pixel mit einer Syntax zu, die wie ein Funktionsaufruf aussieht:

```
1    // Canvas erzeugen
2    Canvas canvas(...);
3    // Wert für Pixel (3,83) ausgeben
4    std::cout << canvas(3,83) << std::endl;
5    // Wert für Pixel (42,31) auf 94 ändern
6    canvas(42,31) = 94;
```

Schreiben Sie eine Funktion

```
1  void mandelbrot (Canvas& canvas, double threshold,
2                   int maxIt, std::string filename)
```

die einen `canvas` mit einer Visualisierung der Mandelbrot-Menge füllen und damit eine Bilddatei erzeugen soll. Dabei ist `threshold` ein Radius, ab dem Punkte als "unendlich weit weg" gelten (sonst wäre das Problem nicht in endlicher Zeit lösbar), und `maxIt` ist die Anzahl an maximal durchzuführenden Iterationen. Bahnen von Testpunkten $c \in \mathbb{C}$, die nach `maxIt` Iterationen noch einen euklidischen Abstand kleiner als `threshold` vom Ursprung haben, gelten als beschränkt. Die Iterationsvorschrift von oben heißt, geschrieben für ein Paar $z_k = (x_k, y_k), x_k \in \mathbb{R}, y_k \in \mathbb{R}$:

$$x_{k+1} = x_k^2 - y_k^2 + c_1$$
$$y_{k+1} = 2 \cdot x_k y_k + c_2$$

Dabei ist $c = (c_1, c_2)$ der Punkt, für den die Bahn berechnet werden soll, und $z_0 = (0, 0)$. Gehen Sie bei der Implementierung wie folgt vor:

[1] https://de.wikipedia.org/wiki/Mandelbrot-Menge

(a) Erzeugen Sie zunächst eine Klasse `IterationResult`, die einen `Point` (den letzten berechneten der Bahn) und ein `int` (die Anzahl an durchgeführten Iterationen) speichert.

Alle Member-Variablen sollen `private` sein. Schreiben Sie die nötigen Accessor-Methoden, um auf den Punkt und die Iterationen zuzugreifen bzw. diese zu verändern.

(b) Schreiben Sie eine Funktion

```
IterationResult iterate (Point z, Point c, double threshold, int maxIt)
```

die die obige Iterationsvorschrift für einen Punkt c durchführt, bis entweder der Punkt weiter als `threshold` vom Ursprung entfernt ist oder `maxIt` Iterationen durchgeführt wurden, und das Ergebnis zurück gibt. Verwenden Sie intern die Darstellung in Komponentenschreibweise, d.h. berechnen Sie das Paar (x_{k+1}, y_{k+1}) jeweils aus dem Paar (x_k, y_k), und speichern Sie die Punkte dabei jeweils in einem `Point`.

(c) Schreiben Sie die Funktion `mandelbrot()`. Diese soll die Funktion `iterate()` benutzen, um die Pixel im übergebenen `canvas` zu füllen.

- Punkte c mit "beschränkten" Bahnen sollen schwarz dargestellt werden (Zahlenwert 0). Beschränkt heißt in diesem Fall, dass die Iteration nach `maxIt` Iterationen beendet wird.
- Punkte mit "unbeschränkten" Bahnen sollen eine Graustufe zugeordnet bekommen, die um so heller ist, je länger es dauert, bis der betrachtete Punkt "entkommt". Wählen Sie daher den Logarithmus (`std::log()`) der Iterationszahl als Grauwert, und multiplizieren Sie diesen mit 100, um genügend Abstufungen zu erhalten.

Benutzen Sie beim Testen Ihres Programms zunächst eine kleine Auflösung, bis Sie sicher sind, dass Ihr Code im wesentlichen das richtige tut. Als Zentrum ihres Bildes sollten Sie dabei den Punkt $(-1, 0)$ verwenden. Erstellen Sie dann ein hochaufgelöstes Bild, indem Sie für die horizontale Auflösung eine Pixelanzahl von 4000 wählen, sowie 3000 für die vertikale. Variieren Sie die Werte für `threshold` und `maxIt`, bis Sie mit dem Ergebnis zufrieden sind. Als Startwert können Sie jeweils mit 1000 beginnen und dann variieren. Welche Effekte haben größere und kleinere Werte jeweils?

Aufgabe 2: Die Julia-Mengen (1 Punkt)

Diese Aufgabe ist eine Fortsetzung der vorhergehenden. Statt der Mandelbrot-Menge visualisieren wir eine der zugehörigen Julia-Mengen. Schreiben Sie dazu eine Funktion

```
void julia (Point c, Canvas& canvas, double threshold,
            int maxIt, std::string filename)
```

Diese Funktion füllt einen gegebenen `canvas` ähnlich wie die bereits von Ihnen geschriebene Funktion `mandelbrot`, allerdings ist jetzt c eine Konstante und der Startpunkt z_0 wird variiert. Berechnen Sie dazu jeweils die Koordinaten eines Pixels, nutzen Sie diese Koordinaten für den Punkt z_0, und füllen Sie wie bisher die Pixel mit Grauwerten, indem Sie den Logarithmus der Iterationsanzahl berechnen.

- Benutzen Sie für das Erstellen des Bildes den Wert $c = (-0.8, 0.156)$. Dieser Punkt soll gleichzeitig als Konstante für die Iterationsvorschrift und als Zentrum der Leinwand dienen.
- Sie können auch gerne mit anderen Punkten experimentieren — je nachdem, ob der Startpunkt c in der Mandelbrot-Menge liegt, oder auf ihrem Rand, in der Nähe, oder weit von ihr entfernt, entstehen völlig unterschiedliche Julia-Mengen.

Aufgabe 3: Geglättete Fraktale (1 Bonuspunkt)

Sowohl bei der Mandelbrot-Menge als auch bei Ihren Julia-Mengen entstehen unschöne Abstufungen, weil die benötigte Anzahl an Iterationen immer eine natürliche Zahl ist. Man kann zeigen, dass man durch die Definition

$$\tilde{k} := k - \log_2\left(\frac{\ln(|z_k|)}{\ln(t)}\right)$$

eine Art "kontinuierliche Anzahl Iterationen" definieren kann, die zusätzlich mit berücksichtigt, wie weit die Iteration im letzten Schritt über den Radius `threshold` hinaus schießt. Dabei ist in obiger

Formel k der Index der letzten Iteration, z_k ihr Punkt und $|z_k|$ sein komplexer Betrag (also seine euklidische Norm $\sqrt{x^2+y^2}$). t ist der Wert von `threshold`, und $\tilde{k}$ eine reelle Zahl, die ein neues Maß dafür ist, wie lange es dauert, bis die Bahn den vorgeschriebenen Kreis verlässt.

- Modifizieren Sie Ihre Funktionen `mandelbrot()` und `julia()` so, dass sie einen optionalen weiteren Parameter `smooth` haben. Wenn `smooth` wahr ist, soll $\tilde{k}$ statt k verwendet werden, sonst die ursprüngliche Definition. Für $\log_2$ stellt C++ die Funktion `std::log2()` bereit.
- Einen Parameter können Sie in C++ optional machen, indem Sie mit einem '=' getrennt angeben, was der Wert sein soll, wenn nichts angegeben ist. Hinter einem optionalen Parameter dürfen dabei aber nur Parameter folgen, die ebenfalls optional sind, und bei einem Aufruf, bei dem ein optionaler Parameter übergeben wird, müssen auch alle Parameter davor explizit angegeben werden, selbst wenn sie ebenfalls optional sind, also z.B.

```
1 // This function has one required parameter problem and one
2 // optional parameter verbose
3 void solve_problem(Problem& problem, bool verbose = false);
4
5 solve_problem(p);
6 solve_problem(p,true);
```

- Diese Modifikation ist der Grund, warum Sie eine Klasse namens `IterationResult` erstellen sollten, die Zugriff auf den letzten berechneten Punkt z_k ermöglicht.
- Stellen Sie sicher, dass die neue Versionen jeweils deutlich glattere Bilder erzeugen, und dass es auch weiterhin möglich ist, die ursprünglichen Funktionsaufrufe zu verwenden (die dann implizit diese Glättung deaktivieren).