Dr. Ole Klein, Dr. Steffen Müthing
IWR, Universität Heidelberg

Abgabe 12. Januar 2018

Zur Erinnerung:
Sie müssen die jeweiligen Aufgaben nicht vollständig und fehlerfrei bearbeitet haben, um sie ankreuzen zu dürfen. Wenn Sie eine Aufgabe nicht komplett lösen konnten, heißt das weder, dass Sie nicht das Niveau für die Aufgabe haben, noch, dass wir die Aufgabe aus Versehen zu schwer gemacht haben. Vielmehr variieren erfahrungsgemäß nicht nur die Vorkenntnisse unter den Teilnehmern, sondern auch die Lernrate unter den Anfängern. Die Aufgaben sind daher so konzipiert, dass nach Möglichkeit jeder einen Aufgabenteil findet, an dem er oder sie üben kann.

Wenn Sie mit einigen der gestellten Aufgaben Probleme haben, heißt das *nicht*, dass Sie sich Sorgen wegen der Klausuraufgaben machen müssten. Natürlich müssen Sie am Ende der Vorlesungszeit die behandelten Themen beherrschen, die Klausuraufgaben werden aber ein anderes Format haben. Sie sollten sich allerdings mit dem Gedanken anfreunden, kleinere mehrzeilige Programmfragmente auf Papier zu schreiben, da wir bei der großen Anzahl an Teilnehmern leider keine Prüfung am Computer anbieten können.

Hinweis:
Es handelt sich hierbei um den Weihnachtszettel, der entsprechend erst nach der vorlesungsfreien Zeit vorzustellen ist. Ein Teil der Aufgaben ist mit "Bonus" markiert — Sie können diesen nutzen, um Bonuspunkte für die Zulassung zu sammeln oder weiter zu üben um besser im Programmieren zu werden.

Übung 1 *Die Mandelbrot-Menge*

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

```
void mandelbrot (Canvas& canvas, double threshold,
                 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 $z = (x, y), x \in \mathbb{R}, y \in \mathbb{R}$:

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

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:

- Erzeugen Sie zunächst eine Klasse `IterationResult`, die einen `Point` (den letzten berechneten der Bahn) und ein **`int`** (die Anzahl an durchgeführten Iterationen) speichert.
- 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`.
- Benutzen Sie die Funktion `iterate` um die Pixel im übergebenen `canvas` zu füllen.
- Punkte c mit "beschränkten" Bahnen sollen schwarz dargestellt werden (Zahlenwert 0).
- 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?

(Grundlagen)

Übung 2 *Die Julia-Mengen*

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.

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). 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.
- 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).

(Grundlagen + Fortgeschritten (jeweils Bonus))

Übung 3 *Einfache automatische Bildbearbeitung*

Die Bilder, die Sie im Rahmen dieses Übungsblattes erstellen, haben einen kleinen Nachteil: da der Logarithmus direkt eingetragen wird, gibt es eine große Lücke in den Grauwerten zwischen reinem Schwarz und dem ersten auftretenden anderen Grauwert. Außerdem haben viele kleine Bereiche ein extrem breites Spektrum an hellen Grauwerten, je nachdem, wie viele Schritte nun exakt gebraucht wurden. Diese beiden Effekte zusammen sorgen dafür, dass der nutzbare Helligkeitsbereich eingeschränkt wird, und entsprechend wirken die entstehenden Bilder etwas kraftlos und matt.

Wir wollen daher eine einfache Reskalierung entwerfen, die gegen einzelne Pixel robust ist: einzelne Ausreißer unter den Pixeln sollen keinen großen Einfluss auf das Ergebnisbild haben, und auch größere komplett schwarze oder komplett weiße Bereiche, wie sie bei der Mandelbrot-Menge auftauchen, sollen die nutzbare Bandbreite nicht negativ beeinflussen. Dazu gehen wir wie folgt vor:

- Der bereitgestellte Header `pgm.hh` enthält eine Funktion zum Einlesen von Bildern, nutzen Sie diese um Zugriff auf die Bilddaten zu bekommen.
- Es wird ein Histogramm der im Bild auftretenden Graustufen erstellt. Verwenden Sie dazu ihren Histogramm-Code aus den vorherigen Aufgaben, selbstverständlich mit den nötigen Anpassungen.

- Es werden zwei Grenzen m_{min} und m_{max} berechnet, so dass $n\%$ der auftretenden Pixelwerte kleiner als m_{min} und $n\%$ der Werte größer als m_{max} sind. Dabei ist n eine kleine natürliche Zahl, für die Sie eine gute Wahl durch Experimentieren finden sollen. Die beiden Grenzen können Sie mit Hilfe des Histogramms berechnen, indem Sie die einzelnen Werte passend aufsummieren.
- Alle Grauwerte bis *einschließlich* $m_{\text{min}} + 1$ werden auf 0 (schwarz) gesetzt, alle ab *einschließlich* $m_{\text{max}} - 1$ werden auf 255 (weiß) gesetzt. Das Verschieben der Grenzen um 1 garantiert, dass wir auch dann reskalieren, wenn ein deutlicher Prozentsatz des Bildes komplett schwarz oder weiß ist (wie zum Beispiel bei der Mandelbrot-Menge).
- Zwischen den neuen Schwarz- und Weißwerten wird linear interpoliert und das Ergebnis in ein neues Bild gespeichert.

(Grundlagen (Bonus))

Übung 4 *Die Mandelbrot-Menge für Fortgeschrittene*

Lesen Sie sich die Aufgabenstellung für Programmieranfänger durch, und lösen Sie die dort beschriebene Aufgabe mit den folgenden Abänderungen:

- Die Rekurrenzbeziehung sollte frei wählbar sein, statt fest auf $z_{k+1} = z_k^2 + c$ verdrahtet zu sein. Benutzen Sie dafür entweder Vererbung und virtuelle Funktionen (dynamischer Polymorphismus) oder Templates (statischer Polymorphismus). Überlegen Sie sich einige weitere Funktionen, die zu interessanten Ergebnissen führen, oder suchen Sie danach im Internet (komplexe Polynome sind naheliegende Kandidaten, man kann jedoch z.B. auch trigonometrische Funktionen einbauen).
- Die Grauwerte sollten im Bereich 0 bis 255 liegen und diesen auch komplett ausnutzen, d.h. die kleinste Anzahl an Iterationen bekommt den Wert 1 zugeordnet, die größte, die kleiner als `maxIt` ist, den Wert 255.
- Statt einer Ausgabe in eine Bilddatei sollen Sie die SDL-Bibliothek [2] verwenden, um das Bild auf dem Bildschirm anzuzeigen. Installationsanweisungen für diese Bibliothek finden Sie, sofern nötig, im zugehörigen Wiki [3]. Wenn Sie die von uns verteilte virtuelle Maschine verwenden, können Sie die Bibliothek und die Entwicklungspakete mit folgendem Befehl installieren:

```
1 [ipk@localhost ~]$ sudo dnf install SDL2-devel
```

  Nach erfolgreicher Installation reicht es, den Header `SDL2/SDL.h` einzubinden und das Programm mit dem Flag `-lSDL2` zu kompilieren. Machen Sie sich mit der Funktionsweise dieser Bibliothek vertraut, um die Aufgabe lösen zu können.
- Es ist nicht nötig, besonders effizienten Code zu schreiben. Eine einfache aber ausreichende Implementierung lässt sich zum Beispiel mit den Funktionen `SDL_SetRenderDrawColor` und `SDL_RenderDrawPoint` schreiben.

Bonus-Votierpunkt: Produzieren Sie statt der Grayscale-Bilder Farbbilder, indem Sie die benötigte Anzahl Iterationen auf eine Color Map abbilden. Verwenden Sie dazu den HSV-Farbraum, weisen Sie die Anzahl der Komponente "Hue" zu, und rechnen Sie in den für die Darstellung benötigten RGB-Farbraum um. Setzen Sie dabei die beiden anderen für die Umrechnung benötigten Komponenten auf ihr Maximum, und wählen Sie eine passende periodische Abbildung auf "Hue", die die Struktur der Graphik angemessen abbildet.

(Fortgeschritten)

[2] `https://www.libsdl.org/`
[3] `https://wiki.libsdl.org/Installation`