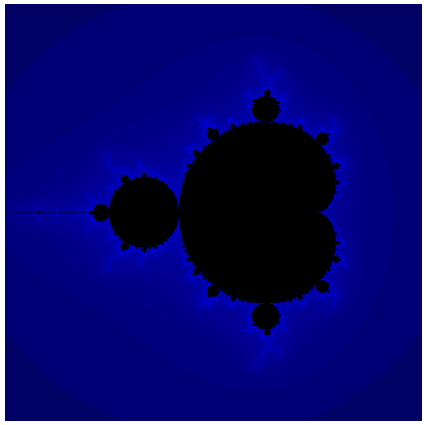
Objektorientierte Programmierung mit C++ (WS 2010)

Dr. Andreas F. Borchert, Tobias Brosch
Institut für Angewandte Informationsverarbeitung
Universität Ulm

Blatt 9: Abgabetermin 22. Dezember 2010 11 Uhr

Mandelbrotmenge



Definition und Grafische Darstellung nach

http://de.wikipedia.org/wiki/Mandelbrot-Menge.

Definition

Die Mandelbrot-Menge $\mathbb{M}$ ist die Menge aller komplexen Zahlen c, für welche die rekursiv definierte Folge komplexer Zahlen $z_0, z_1, \ldots$ mit dem Bildungsgesetz

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

und dem Anfangsglied $z_0 = 0$ beschränkt bleibt, das heißt, der Betrag der Folgenglieder wächst nicht über alle Grenzen. Die grafische Darstellung dieser Menge erfolgt in der komplexen Ebene. Die Punkte der Menge, werden dabei in der Regel schwarz dargestellt und der Rest farbig, wobei die Farbe eines Punktes den Grad der Divergenz der zugehörigen Folge widerspiegelt.

Grafische Darstellung

Am Computer kann die Menge wie folgt dargestellt werden: Jedem Bildpunkt entspricht ein Wert c der komplexen Ebene. Der Computer ermittelt für jeden Bildpunkt, ob die zugehörige Folge divergiert oder nicht. Sobald der Betrag $|z_n|$ eines Folgengliedes den Wert $R = 2$ überschreitet, divergiert die Folge. Die Zahl der Iterationsschritte N gemäß obiger Rekursionsformel, nach denen das erfolgt, kann als Maß für den Divergenzgrad herangezogen werden. Über eine zuvor festgelegte Farbtabelle, die jedem Wert N eine Farbe zuordnet, wird in diesem Fall dem Bildpunkt eine Farbe zugewiesen.

Um in ästhetischer Hinsicht harmonische Grenzen zwischen aufeinanderfolgenden Farben zu erreichen, wird in der Praxis für die Grenze R nicht der kleinste mögliche Wert $R = 2$ gewählt, sondern ein Wert deutlich größer als 2.

Umsetzung

Funktion main

Um nicht selbst Bilddateien schreiben zu müssen, werden wir die Generic Image Library http://www.boost.org/doc/libs/1_45_0/libs/gil/doc/index.html aus der boost-Library verwenden. Es handelt sich hierbei um eine Header-only Bibliothek, d.h. es reicht die Header einzubinden (kein Linken notwendig). Zum Schreiben von png-Dateien, benötigen wir jedoch die Library libpng http://www.libpng.org/, gegen welche wir auch linken müssen (`LDFLAGS += -lpng`). Fügt folgende beiden Includes zu Eurem Programm hinzu:

```
#include <boost/gil/image.hpp>
#include <boost/gil/extension/io/png_io.hpp>
```

Für ein Grauwertbild, gibt es den Datentyp `gray8_image_t`. Definiert den Typ `image_t` als `gray8_image_t`. Legt einen Punkt `dims` vom Typ `image_t::point_t` mittels des Konstruktors an, der zwei Werte vom Typ `int` übergeben bekommt (beide sollen den Wert 1000 haben). Legt anschließend eine Variable `myImage` vom Typ `image_t` an und nutzt den Konstruktor, der die Variable `dims` übergeben bekommt. Dies allokiert Speicher für ein Grauwertbild der Größe 1000×1000.

Typischerweise werden in Algorithmen der Generic Image Library lediglich so genannte Views auf einen Speicherbereich verwendet. Legt daher mit

```
image_t::view_t myView = view(myImage);
```

eine View auf euer Grauwertbild an. Um die einzelnen Grauwertpunkte zu setzen, iteriert mittels zweier for-Schleifen über Eure View (nutzt hierzu den Operator `operator(int x, int y)` einer View, um ein Element der Lokation (x, y) zuzuweisen).

Mittels

```
png_write_view("mandel.png", myView);
```

schreibt Ihr Euer Bild in die Datei „mandel.png“.

Mandel Funktor

Soweit ist noch nichts darüber gesagt, welche Funktion wir innerhalb unserer beiden for-Schleifen verwenden, um den Farbwert unserer Mandelbrot-Visualisierung zu bestimmen. Zunächst sieht es so aus, als ob eine einfache Funktion ausreichend wäre. Häufig stellt man jedoch fest, dass es geschickt ist, wenn die Funktion noch „private Variablen“ hat. In C++ wird daher gerne ein Objekt konstruiert, welches die gleiche Aufrufsyntax wie eine Funktion hat, jedoch ein Objekt ist. Derartige Objekte werden auch häufig Funktoren genannt. In der Umsetzung heißt dies lediglich, dass der Klammeroperator () passend definiert wird.

Implementiert also eine Klasse `Mandel_fn`, welche den Operator `operator()(double x, double y)` implementiert. Ruft diese Methode innerhalb Eurer beiden for-Schleifen, welche über alle Bildpunkte iterieren, mit den Werten von $x \in [-2, 1]$ und $y \in [-1.5, 1.5]$ auf. Die Methode `operator()(double x, double y)`, soll dabei jeweils einen Farbwert vom Typ `gray8_pixel_t` zurückgeben.

Die Zahl der Iterationsschritte wie oben beschrieben könnt Ihr z.B. wie folgt mittels einer privaten Funktion von `Mandel_fn` bestimmen:

```
double get_num_iter(double x, double y) {
        point2<double> Z(0, 0);
        for (int i = 0; i < MAX_ITER; ++i) {
                Z = point2<double> (Z.x * Z.x - Z.y * Z.y + x,
                        2 * Z.x * Z.y + y);
                if (Z.x * Z.x + Z.y * Z.y > 8) {
                        return pow((double) i / MAX_ITER, 0.2);
                }
        }
        return 0;
}
```

Definiert hierzu eine konstante Klassenvariable `MAX_ITER=100`. Das Potenzieren mit 0.2 erfüllt hierbei den Zweck der Reskalierung für eine bessere Darstellung. Beachtet auch den Cast von der Variable `i` auf `double`. Andernfalls fände eine Integer-Division statt, welche in 0 resultiert (probiert es mal aus—beliebter Fehler, der schwer zu finden ist!).

In Eurer Operator-Methode, müsst Ihr jetzt lediglich noch das Ergebnis der Methode `get_num_iter` passend Skalieren (na—wieviele Zahlen lassen sich mit 8-bit darstellen?).

Wenn soweit Alles läuft und Ihr Euer Apfelmännchen bewundert habt, übt nochmals den Umgang mit Templates, indem Ihr den Typ `gray8_pixel_t` der Klasse `Mandel_fn` als Template-Parameter definiert.

Einreichen der Lösung mit:

```
submit cpp 9 Makefile main.C
```

Viel Spaß!