ulm university universität
uulm

Institut für Numerische Mathematik
Dr. Andreas F. Borchert und Dr. Michael C. Lehn

12. Januar 2017
Blatt 10

Objektorientierte Programmierung mit C++ (WS 2016/2017)

Abgabe bis zum 19. Januar 2017, 16:00 Uhr

Lernziele:

- Statischer Polymorphismus
- Verarbeitung der Kommandozeile in C++

Aufgabe 12: Darstellung der Mandelbrot-Menge (10 Punkte)

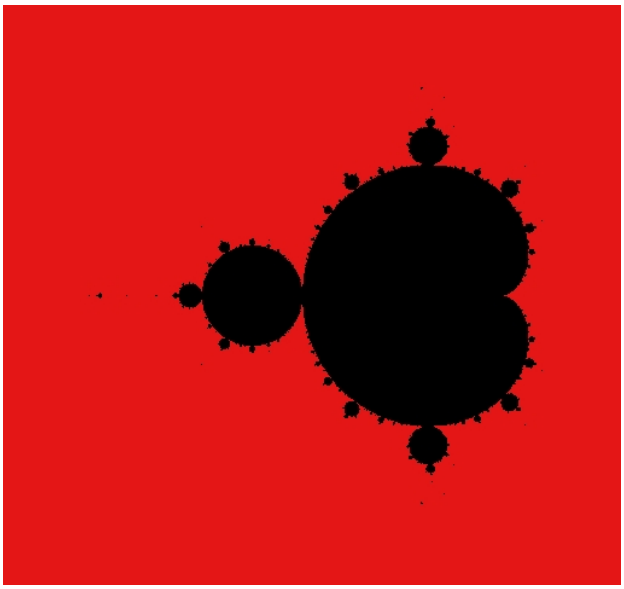
Die Mandelbrot-Menge ist eine von Benoît Mandelbrot entdeckte Teilmenge der komplexen Zahlen, die auf den folgendermaßen definierten Folgen $\{z_n(c)\}_{n\in\mathbb{N}_0}$ für jeden Punkt $c \in \mathbb{C}$ beruht:

$$\begin{aligned} z_0(c) &:= 0 \\ z_{n+1}(c) &:= z_n(c)^2 + c \end{aligned}$$

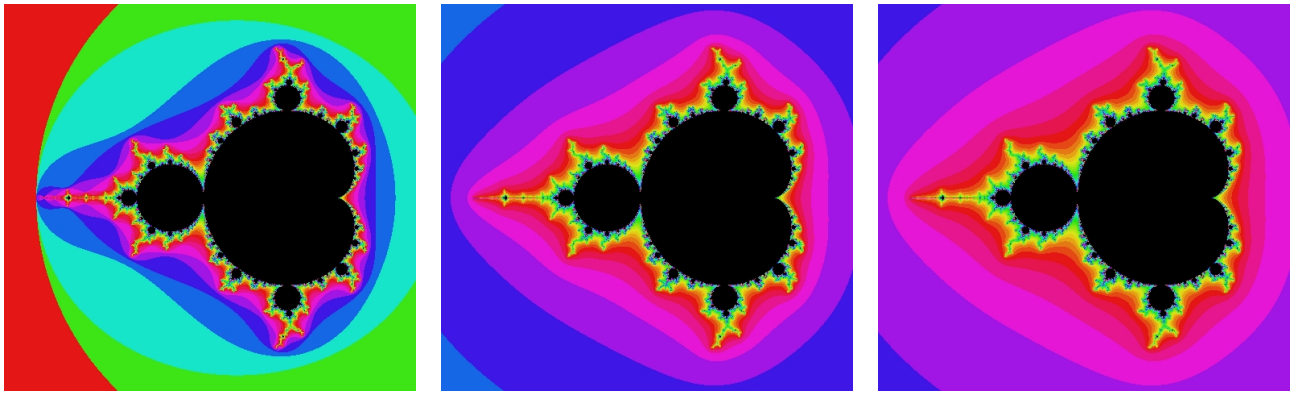
Die Mandelbrot-Menge lässt sich dann wie folgt definieren:

$$M := \{c \in \mathbb{C} \mid z_n(c) \not\to \infty \text{ für } n \to \infty\}$$

Um diese Menge zu bestimmen, ist es hilfreich zu wissen, dass $|z_n(c)| \leq 2$ gilt für alle $c \in M, n \in \mathbb{N}_0$. Wenn numerisch festgestellt werden soll, ob $c \in M$, dann wird die Folge berechnet, bis entweder ein n gefunden wird mit $z_n(c) > 2$ oder n eine vorgegebene Grenze $n_{\max}$ überschreitet. Das lässt sich in eine Grafik umsetzen, bei der jeder Punkt unterschiedlich markiert wird, je nachdem, ob die Grenze erreicht wird oder nicht. Folgendes Diagramm demonstriert dies, wobei die Punkte schwarz gefärbt sind, bei denen $n_{\max}$ überschritten wurde:



Es bietet sich aber an, abhängig von der Zahl der benötigten Iterationen unterschiedliche Farben zu verwenden. Diese Vorgehensweise ist unter dem Namen „Escape-Time-Algorithmus“ bekannt. Bei dieser Vorgehensweise ist es sinnvoll, den Algorithmus nicht sofort beim Erreichen von $z_n(c) > 2$ abzubrechen, sondern bei $z_n(c) > 2^N$. Je höher N ist, umso feiner werden die Farbabstufungen. Hier sind Beispiele für $N = 1, 8$ und 16:



Wenn die n die vorgegebene Grenze $n_{\max}$ überschreitet, ist eine schwarze Farbe zu wählen. In anderen Fällen wird ein Wert t bestimmt mit $t \in \mathbb{R}, t \geq 0$. Im einfachsten Fall entspricht t einfach dem Wert n, der beim Abbruch der Schleife erreicht wird. Bei stufenlosen Übergängen wird hier aber noch ein Wert hinzuaddiert aus dem Bereich $[0, 1)$. Wenn daran Interesse besteht, gibt es hier Hinweise dazu. Dies ist für die Aufgabenstellung jedoch optional, obwohl es in den folgenden Beispielen Verwendung findet.

Für die Farben werden Farbpaletten verwendet, die jeweils aus einer endlichen Folge $\{c_i\}_{i=0,\ldots,C-1}$ besteht. Wir könnten dann für den Wert t eine Farbe zwischen den Farbpunkten $c_{\lfloor t \rfloor \bmod C}$ und $c_{\lfloor t+1 \rfloor \bmod C}$ auswählen, indem wir zwischen den Farbpunkten linear mit $t \bmod 1$ interpolieren.

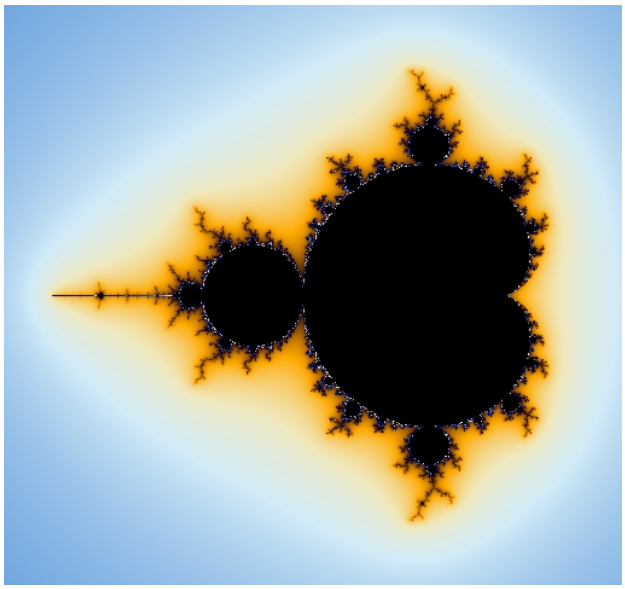
Da die Zahl der Iterationen recht hoch sein kann und insbesondere bei tieferen Blicken in die Mandelbrot-Menge $n_{\max}$ deutlich höher anzusetzen ist, lassen sich zu rasche Farbüberläufe kaum noch vermeiden. Deswegen kann bei der Auswahl der Farben aus der Palette auch $\hat{t}$ anstelle von t zum Einsatz kommen mit $\hat{t} := \log(t)/\log(n_{\max})$.

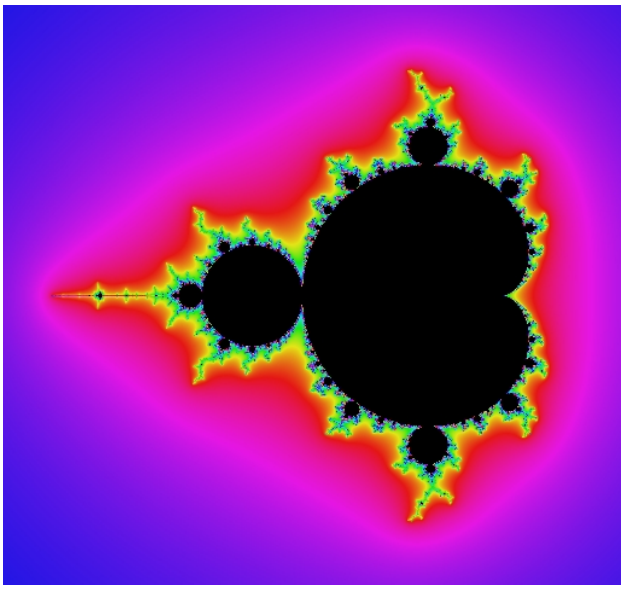
Um das Tempo der Farbüberläufe weiter zu beeinflussen, könnte noch ein Farbprogressions-Parameter p ins Spiel kommen, bei dem bei der Indizierung der Farbpalette tp statt t verwendet wird.

Für die Farben können unterschiedliche Farbräume sinnvoll sein. Der HSV-Farbraum (HSV steht für *hue*, *saturation*, *value*) erlaubt das einfache Arbeiten mit Regenbogenfarben, indem etwa *saturation* und *value* konstant bleiben, aber *hue* aus dem Bereich zwischen 0 und 360 gewählt wird. Der RGB-Farbraum wird gerne verwendet, um klassische Farbreihenfolgen zu verwenden, etwa die Tag-und-Nacht-Farben-Folge (25, 7, 26), (9, 1, 47), (4, 4, 73), (0, 7, 100), (12, 44, 138), (24, 82, 177), (57, 125, 209), (134, 181, 229), (211, 236, 248), (241, 233, 191), (248, 201, 95), (255, 170, 0), (204, 128, 0), (153, 87, 0), (106, 52, 3), (66, 30, 15).

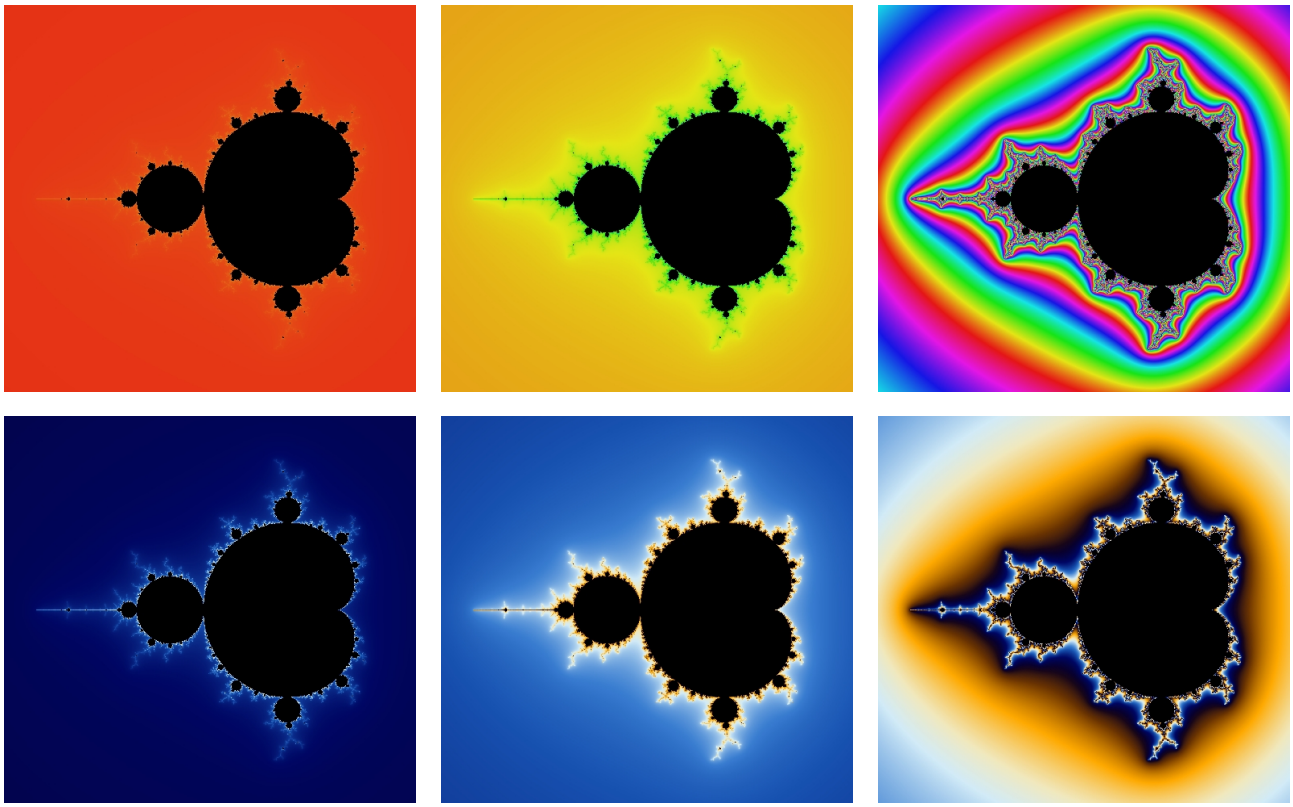
Für jeden der Farbräume ist eine geeignete Interpolationsfunktion zu verwenden. Bei beiden Farbräumen werden jeweils die einzelnen Komponenten individuell linear interpoliert. Der einzige Sonderfall ist die *hue*-Komponente beim HSV-Farbraum. Wenn der *hue*-Wert des zweiten Farbpunkts kleiner oder gleich dem ersten *hue*-Wert ist, dann sollte der *hue*-Wert des zweiten Farbpunkts temporär um 360 erhöht werden. Nach der Interpolation ist das Resultat wieder modulo 360 zu nehmen.

Die beiden folgenden Bilder zeigen die Mandelbrot-Menge bei fließenden Farbübergängen links mit RGB (und den oben genannten Tag-und-Nacht-Farben) und rechts mit HSV (eine Farbfolge mit nur mit dem einen Element $(0, 0.9, 0.9)$, das aber wie oben beschrieben eine sinnvolle Interpolation mit sich selbst ermöglicht und dann entsprechend dem Wert von t den *hue*-Bereich von 0 bis 360 abdeckt).





Die obigen Bilder arbeiteten mit den voreingestellten Farbkonfigurationen. Die folgenden drei Bilder zeigen unter Verwendung des gleichen HSV-Farbraums die Resultate mit den Farbprogressions-Parameter p für die Werte $0.1, 0.5$ und 20. Darunter sind drei Bilder mit dem RGB-Farbraum und den Farbprogressions-Parametern $10, 20$ und 100:



Der Mandelbrot-Algorithmus sollte im Rahmen der Aufgabenstellen statisch-polymorph entwickelt werden, so dass er für beliebige Gleitkommazahlen-Typen als auch für unterschiedliche Farbräume verwendet werden kann.

Die Wahl des Gleitkommazahlen-Typs ermöglicht etwa die Verwendung von **long double** oder von *mpf_class* aus der C++-Schnittstelle der GMP-Bibliothek. Allerdings verlängern Gleitkommazahlen mit höherer Genauigkeit deutlich die benötigte Rechenzeit – dies gilt insbesondere bei Nutzung der GMP-Bibliothek. Hier reicht es, wenn dieser Parameter beim Hauptprogramm zur Übersetzzeit festlegen läßt. Alle Funktionen außerhalb von *main* () sollten hier aber mit einem entsprechenden Template-Parameter arbeiten.

Die Wahl des Farbraums ist mit dem ersten Kommandozeilenargument zu bestimmen. Also etwa mit „-RGB“ für den RGB-Farbraum und „-HSV“ für den HSV-Farbraum. Sobald das erste Argument verarbeitet ist, kann dann mit dem entsprechendem Template-Parameter die restliche Verarbeitung erfolgen.

Um das Resultat zu visualisieren, empfiehlt sich die Verwendung der GDK-Pixbuf-Bibliothek. Wie dies funktioniert, zeigt *gdk-pixbuf-demo.cpp*. Die notwendigen Übersetzungsoptionen zur Verwendung der GDK-Pixbuf-Bibliothek liefern die folgenden beiden Kommandos, die entsprechend im *Makefile* eingesetzt werden können, um die in der linken Spalte genannten Parameter zu ergänzen:

CFLAGS	`pkg-config -cflags gdk-pixbuf-2.0`
LDLIBS	`pkg-config -libs gdk-pixbuf-2.0`

Da die GDK-Pixbuf-Bibliothek nur den RGB-Farbraum unterstützt, müssen Farbwerte aus anderen Farbräumen in RGB-Farben konvertierbar sein. Dies geht am besten mit einer Konvertier-Methode, die bei allen Farbraum-Klassen zur Verfügung steht (auch bei dem RGB-Farbraum, wo die Operation trivial ist). C-Code für die Konvertierung aus dem HSV-Farbraum in dem RGB-Farbraum steht von Nan C. Schaller zur Verfügung.

Das Programm soll als Argumente von der Kommandozeile neben dem Farbraum-Parameter zumindest folgende Parameter einlesen:

Option	Beschreibung	Standardwert
`-x`	$\Re(c)$ des Bildmittelpunkts c	-0.7
`-y`	$\Im(c)$ des Bildmittelpunkts c	0
`-d`	Horizontaler Durchmesser des abzubildenden Bereichs	3.0769
`-w`	Breite in Pixeln	512
`-h`	Höhe in Pixeln	480
`-i`	Maximale Anzahl der Iterationen	1000
`-p`	Farbprogression	30 bei RGB, 3 bei HSV

Falls Optionen weggelassen werden, sollen im Programm Standardwerte festgelegt sein, die dafür verwendet werden. Außerdem soll beim Aufruf mit der Option „-?“ eine „Usage“-Meldung ausgegeben werden, d. h. eine Auflistung der verfügbaren Optionen mit ihren Bedeutungen.

Erwartet wird ein strukturierter Umgang mit den verfügbaren Parametern. Definieren Sie eine geeignete Datenstruktur, die einen Parameter beschreibt und den Umgang damit erleichtert. Für die verfügbaren Parameter soll dann eine entsprechende Tabelle angelegt werden, die mit Hilfe einer Initialisierung gefüllt wird. Das Verarbeiten der Parameter soll dann tabellengesteuert erfolgen.

Diese Aufgabe ist etwas umfangreicher als frühere Aufgabenblätter. Es lohnt sich daher, sie mit Hilfe eines Teams zu lösen. So lässt sich die Aufgabenstellung aufteilen:

- *main*(), Kommandozeilenverarbeitung
- Klassen für die Farbräume, Unterstützung der Konvertierung von HSV in RGB, Hilfsfunktionen für die Bestimmung Farbe aus einer Farbpalette
- Mandelbrot-Algorithmus
- Generierung der Grafik mit Hilfe der GDK-Pixbuf-Bibliothek

Es wäre nett, wenn Sie ein Resultat, das Ihnen gefällt, zusammen mit den Parametern Ihrem Ergebnis beifügen würden. Wir erstellen dann eine kleine Galerie mit den eingereichten Bildern.

Sie können wie üblich Ihre Lösung wieder einreichen:

```
thales$ tar cvf Mandelbrot.tar *.?pp *.jpg Makefile
thales$ submit cpp 12 Mandelbrot.tar
```

Viel Erfolg!