

Aufgabe 1: Grundlegende Bildoperatoren (Pixelbasiert) (2 Punkte)

Eine große Anzahl an Bildoperatoren arbeitet aus Gründen der Effizienz ausschließlich mit Graustufenbildern. Die Überführung eines Farbbildes in ein Graustufenbild wird dabei durch eine Funktion beschrieben, die für jeweils ein Pixel aus den Farbwerten einen Helligkeitswert berechnet:

$$f_{x,y}(r,g,b) = 0.299 * r + 0.587 * g + 0.114 * b$$

Ein weiterer Filter der nur auf jeweils einem Pixel als Eingabemenge arbeitet ist der Farbinvertierungsfilter. Laden Sie das Vorgabebild „image1.png“. Implementieren Sie die beiden Filter durch folgende Funktionen:

- a) `float getGrayColor(float r, float g, float b):` Implementieren Sie die Graustufenfunktionalität.
 - b) `QColor getInvertColor(float r, float g, float b):` Implementieren Sie einen Invertierungs-FILTER.



(b) Graustufenbild

(c) Invertiertes Bild

Abbildung 1: Original (links) und Ergebnisse von pixelbasierte Bildoperatoren (rechts).

Integrieren Sie Ihre Lösungen in den „Exercise 1“-Bereich des vorgegebenen Programmrahmens.

Aufgabe 2: Erweiterte Bildoperatoren (10 Punkte)

Viele digitale Bilddaten, insbesondere Fotos, weisen Farbfehler auf. Diese äußern sich, z. B. in mangelndem Kontrast, unausgewogener Helligkeit, Farbstichen und Unschärfe. Durch eine digitale Nachbearbeitung können die Bilder meist qualitativ verbessert werden. Konvolutionsfilter sind wichtige Vertreter flächenbezogener Bildoperationen. Durch ihre generische Gestalt ermöglichen Sie die Erzeugung verschiedener visueller Effekte.

Der Laplace-Filter, z. B. ist ein Hochpassfilter, der zur Kantenextraktion dient. Durch die Verstärkung der Kanten in horizontaler, vertikaler und diagonaler Richtung erscheint das gefilterte Bild schärfer gezeichnet (Abbildung 2).

Laden Sie das Vorgabebild „image1.png“ und wenden Sie vier verschiedene Filter in den entsprechenden Funktionen an:

- a) QColor getSharpenColor(const QImage & image, int x, int y): Implementieren Sie einen Laplace-Filter mit einem 3×3 Filterkernel.
 - b) QColor getGaussColor(const QImage & image, int x, int y): Implementieren Sie einen Gauss-Unschärfe-Filter mit einem 3×3 Filterkernel.
 - c) QColor getSobelColor(const QImage & image, int x, int y): Implementieren Sie einen Sobel Kanten-Detektions-Filter mit einem 3×3 Filterkernel. Anmerkung: Der Sobel-Filter soll auf Graustufenwerten operieren!
 - d) QColor getMeanColorDynSize(const QImage & image, int x, int y, int kernelSize): Implementieren Sie den Mean-Filter mit dynamischer Kernelgröße, gegeben als Parameter kernelSize.



Abbildung 2: Links oben: Laplace-Filter. Rechts oben: Gauß-Filter. Links unten: Sobel-Filter. Rechts unten: Mean-Filter.

Wenden Sie die Filterkernel in den Randbereichen des Bildes durch Wiederholung der Randpixel an. Implementieren und verwenden Sie hierzu die Funktion `getPixel()`. Schreiben Sie das Resultat in die vordefinierten RGB-Farbkomponenten um die Bilder auf dem Canvas auszugeben.

Integrieren Sie Ihre Lösungen in den „Exercise 2“-Bereich des vorgegebenen Programmrahmens.

Aufgabe 3: Dithering (8 Punkte)

Dithering bezeichnet Verfahren, die gegebene Bilder so umwandeln, dass sie mit einer gegebenen Menge von Primärfarben möglichst gut approximiert werden. Ein Beispiel hierfür ist die Umwandlung eines Bildes in ein Schwarz-Weiß-Bild zur Ausgabe auf einem Schwarz-Weiß-Drucker (siehe Abbildung 3).

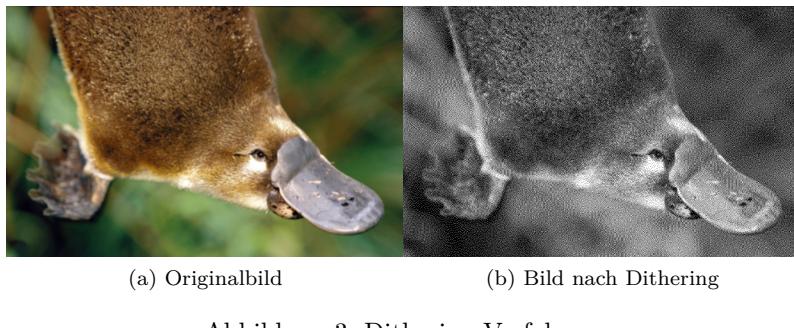


Abbildung 3: Dithering Verfahren.

Laden Sie das Vorgabebild „image1.png“. Nutzen Sie Ihre bereits implementierte Funktion zur Umwandlung des Bildes in ein Graustufenbild. Wandeln Sie dieses danach in ein Schwarz-Weiß-Bild um. Implementieren Sie hierfür das Dithering Verfahren nach Floyd-Steinberg (siehe Abbildung 4). Passen Sie die Funktionen `getDitheringColor(QImage & image, int x, int y)` in `Exercise3` entsprechend an.

| | | |
|------|------|------|
| | P | 7/16 |
| 3/16 | 5/16 | 1/16 |

Abbildung 4: Schema nach Floyd-Steinberg.

Integrieren Sie Ihre Lösungen in den „Exercise 3“-Bereich des vorgegebenen Programmrahmens.

Aufgabe 4: Mandelbrot-Menge (10 Punkte)

Visualisieren Sie eine Mandelbrot-Menge. Die Funktion ist durch die rekursiv definierte Folge $\{a_n\}$ mit $a_{n+1} = a_n^2 + z$ definiert, wobei das Anfangsglied durch $a_0 = 0$ und z durch den aktuell zu untersuchenden Punkt (Pixel) gegeben ist.

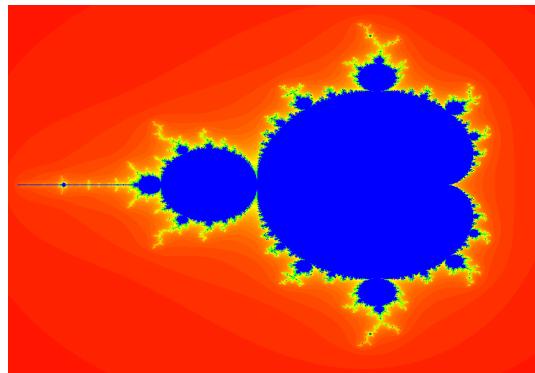


Abbildung 5: Ein Beispiel für die Mandelbrot-Menge.

- Implementieren Sie zunächst die Funktion `computeIterations(float cx, float cy)` in `Exercise4a` und `Exercise4b`: Berechnen Sie für den durch (cx, cy) gegebenen Bildpunkt die Anzahl der Iterationsschritte i zur schrittweisen Berechnung von $|z_{n+1}|^2 = x_{n+1}^2 + y_{n+1}^2$. Verwenden Sie als Abbruchkriterium $|z_{n+1}|^2 \leq square_{max}$ mit $square_{max} \geq 4$. Brechen Sie weiterhin ab, wenn die maximale Anzahl an Iterationen i_{max} (z. B. 100) überschritten ist. Bestimmen Sie nun abhängig von $\frac{i}{i_{max}}$ einen Farbwert in der Funktion `chooseColor(int value, int max)` (Abbildung 5).
- Durch die Verwendung räumlicher Datenstrukturen kann der Bildaufbau iterativ verfeinert werden. Insbesondere für rechenintensive Visualisierungen ergibt sich aus diesen „Vorschau“-Ergebnissen ein Vorteil für die Usability.

Erweitern Sie Ihre Lösung um eine Quadtree-basierte Berechnung und Darstellung der Mandelbrot-Menge. Implementieren Sie dazu die Funktion `drawRecursive(QPainter & painter, int x, int y, int w, int h, int level)` in `Exercise4b`, die für den Bereich $B = (cx_0, cy_0) \times (cx_1, cy_1)$ genau einen Farbwert der Mandelbrot-Menge an den Koordinaten des Mittelpunktes von B berechnet. Die Eckpunkte von B werden anhand von $level$ bestimmt (Abbildung 6). Ermitteln Sie rekursiv die Farbwerte der Subbereiche für $level - 1$. Brechen Sie die Rekursion ab, wenn $level = 0$ erreicht ist.

Rufen Sie nun die Funktion `drawRecursive` auf. Inkrementieren Sie das Level automatisch um 1 nach jeweils 0,5 Sekunden unter Verwendung eines `QTimer`. Zeichnen Sie das Ergebnis für dieses Level. Wenn $Level > 12$ erreicht ist, beginnen Sie wieder mit $level = 0$.

Integrieren Sie Ihre Lösungen in den „Exercise 4a/b“-Bereichen des vorgegebenen Programmrahmens.

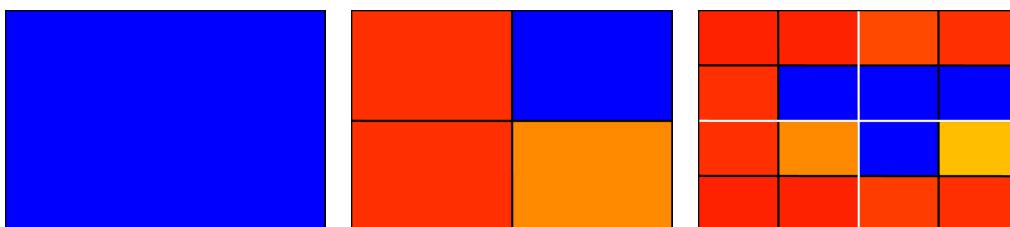


Abbildung 6: Beispiele für Quadtree-basierte Darstellung der Mandelbrot-Menge (Links: Level 0, Mitte: Level 1, Rechts: Level 2).

Allgemeine Hinweise:

- Die Aufgaben sollen maximal zu zweit bearbeitet werden; Ausnahmen müssen mit den Übungsleitern abgesprochen werden.
- Bitte reichen Sie Ihre Lösungen bis Dienstag, den 29.04.2014, um 13:15 Uhr ein.
- Tragen Sie Ihre Matrikelnummern in die Quellcode-Dateien ein. Danach packen Sie Ihre Lösung und geben der Zip-Datei einen Namen nach folgendem Schema:
cg1_blatt1_matrikelnummer1_matrikelnummer2.zip

Beachten Sie, dass nur die vollständigen Quelltexte, Projektdateien und Mediadateien geschickt werden sollen. Senden Sie uns keinesfalls ausführbare Dateien oder bereits kompilierte Binär- oder Temporärdateien (*.obj, *.pdb, *.ilk, *.ncb etc.)! Testen Sie vor dem Verschicken, ob die Projekte aus den Zipdateien fehlerfrei kompiliert und ausgeführt werden können.