

Einführung in die Programmierung  
WS 2018/19

Übungsblatt 9: Bildbearbeitung

Besprechung: 07.01 - 11.01.2019

Aufgabe 9-1 *Bildbearbeitung*

In dieser Aufgabe sollen Sie ein Programm zur Bildbearbeitung vervollständigen. Dieses soll ein paar grundlegende Bildmanipulationen beherrschen, sodass mittels Konsole eine Bilddatei eingelesen, manipuliert und anschließend die veränderte Version wieder abgespeichert wird. Ein Grundgerüst wird Ihnen bereits zur Verfügung gestellt. Für die volle Funktionalität müssen Sie lediglich die mit TODO gekennzeichneten Methodenrumpfe ergänzen.

- (a) Ein Bild ist im Wesentlichen ein zweidimensionales Array, also ein Array aus Arrays. Jedes Matrixelement heißt Pixel und wird durch eine ganze Zahl repräsentiert. Der hier verwendete Datentyp dafür ist `int`, der 4 Byte enthält. Um einen Pixel als Farbpunkt zu identifizieren, brauchen wir Informationen über seinen Rot-, Grün- und Blauwert. Zusätzlich gibt es eine Transparenz  $\alpha$ . Diese werden derart als Pixelwert gespeichert, dass jeweils 8 Bits für  $\alpha$  und die drei Farbwerte in einem Wert codiert sind:

$\alpha$	$\alpha$	$\alpha$	$\alpha$	$\alpha$	$\alpha$	$\alpha$	$\alpha$	$\alpha$	R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B
----------	----------	----------	----------	----------	----------	----------	----------	----------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

In dem gegebenen Grundgerüst finden Sie bereits die Methode `setColors(...)`. Vervollständigen Sie gemäß der Signatur die Methode `getColors(...)`.

- (b) Vervollständigen Sie die Methode `mirror()`, die das Bild an der vertikalen Mittelachse spiegeln soll.
- (c) Vervollständigen Sie die Methode `rotate(n)`, die das Bild  $n$ -mal um 90 Grad drehen soll. Es obliegt Ihnen, die Drehrichtung zu wählen. Spiegeln Sie aber dabei nicht das Bild! Beachten Sie, dass sich die Dimension des Bilds damit ändern kann.
- (d) Vervollständigen Sie die Methode `invert()`, die jeden Farbwert invertiert. Die Transparenz sollen Sie unverändert lassen. Da eine Farbe einen 8-bit großen Wert zur Darstellung nutzt, bewegen diese sich in einem Intervall von 0 bis 255 einschließlich. Der inverse Farbwert zu  $x$  ist damit  $255 - x$ . Denken Sie insbesondere daran, dass sie die Farbwerte getrennt behandeln müssen und anschließend wieder zusammenfügen und im Bildarray speichern.
- (e) Die Methode `meanFilter(filter, factor)` nutzt eine sogenannte Faltungsmatrix, um Pixel abhängig von ihrer Nachbarschaft zu verändern. In Ihrem Grundgerüst gibt es bereits einige vordefinierte Filter. Die Anwendung dergleichen müssen Sie aber noch implementieren. Vervollständigen Sie die Methode, sodass zu jedem Pixel seine  $3 \times 3$ -Nachbarschaft mit dem durch `filter` gegebenen Filter gewichtet aufsummiert und anschließend durch den `factor` skaliert wird. Auch hier müssen die 3 Farbwerte gesondert betrachtet und anschließend zusammengefügt werden.
- Beispiel: Angenommen, Sie nutzen einen Gauß-Filter mit `filter = [1, 2, 1, 2, 4, 2, 1, 2, 1]` und

einem `factor = 1/16`. Stellen Sie sich die Faltungsmatrix als  $3 \times 3$ -Matrix vor:

$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$
$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$

Für einen Pixel sei sein Rotwert an einer Stelle gegeben. Die für diesen Pixel relevante Nachbarschaft ist ebenfalls  $3 \times 3$  Pixels groß:

32	24	24
32	16	24
32	16	8

**gauss-filter**  
 $\Rightarrow$ 

32	24	24
32	<u>22</u> <u>16</u>	24
32	16	8

Implementieren Sie NUR  $3 \times 3$  Filter. Bei den Randpunkten (die ja keine vollständige Nachbarschaft haben) nehmen Sie an, dass Bildpunkte jenseits des Rands den Wert 0 haben. Sie können dazu die Zugriffsmethode `getPixel(int x, int y)` verwenden.

Das beigefügte `visualization.gif` verdeutlicht die Operation nochmal.

- (f) Zuletzt vervollständigen Sie die Methode `medianFilter()`, die ähnlich wie zuvor die Nachbarschaft betrachtet. Allerdings soll hier keine Summe gebildet werden, sondern der Median aus dem Pixel selbst sowie seinen 4 direkten Nachbarn. Verfahren Sie mit Randpunkten wie zuvor. Sie können Ergebnisse aus der vorherigen Übung verwenden, oder zum Sortieren die Methode `Arrays.sort(array)` verwenden. Der Median liegt dann in der Mitte des Arrays.

Das fertige Programm sollte in der Lage sein, das Testbild `testbild.png` ein wenig zu verbessern, indem etwas Rauschen entfernt und das Bild wieder in seine native Position gebracht wird, zum Beispiel mittels

```
java Bildbearbeitung testbild.png -i -m -rot180 -median
```