Aufgabe 1 Convolution-Filter

Lernziele

1. Vertraut werden mit der Java Syntax.
2. Ein erstes Programm in Java schreiben.
3. Den Umgang mit 2 dimensionalen Arrays in Java üben.
4. Einen nicht trivialen Anwendungsfall für 2 dimensionale Arrays kennenlernen.
5. Den Umgang mit mehrfach geschachtelten Schleifen üben.
6. Den Umgang mit Basisdatentypen / Operatoren / Kontrollstrukturen in Java üben.
7. Vorgegebenen Programmcode lesen und erweitern können.
8. Sich selbständig mit Teilen der Java Bibliothek auseinandersetzen.

Hintergrund

Convolution-Filter (auch Faltungskern oder Faltungsmatrix) sind eine Technik aus der Bildverarbeitung, die Sie aus Werkzeugen wie Photoshop oder Gimp kennen, um Photos (allgemeiner Bilder) nachzubearbeiten oder mit Effekten zu versehen. Die wohl bekanntesten Effekte sind das Weichzeichnen oder Schärfen von Bildern.

Die Kernidee eines Convolution-Filters besteht darin den Farbwert eines Pixels im Bild in Abhängigkeit von der Gewichtung der Farbwerte der Nachbarpixel neu zu berechnen. Die Gewichte und die Größe des Korridors für die Nachbarpixel bestimmen dabei den erzielten Effekt.

Modell eines Convolution-Filters

Convolution-Filter werden als quadratische Matrizen dargestellt, deren Elemente die Gewichte für die Farbwerte der Pixel des Originalbilds enthalten. Die **Matrix** wird wie eine Schablone Pixel für Pixel über das Originalbild geschoben. Die Farbwerte der Pixel, die jeweils unter der Schablone liegen, werden mit den Gewichten der Matrix multipliziert, aufaddiert und schließlich mit einem **Faktor** verrechnet. Mit Hilfe des sogenannten **Bias** können die Ergebniswerte ausgerichtet werden, so dass das Ergebnisbild heller oder dunkler erscheint.

Die Summe der Gewichte der Filtermatrix sollte 1 ergeben, damit die Helligkeit des Originalbildes erhalten bleibt. Die Matrix hat eine ungerade Anzahl von Zeilen und Spalten, damit die Nachbarn gleichmäßig verteilt sind.

Anwendung eines Convolution-Filters am Beispiel

Der einfachste Convolution-Filter, der auch dazu geeignet ist, um die Korrektheit der eigenen Lösung zu überprüfen, ist der Identitäts-Filter. Er überführt das Originalbild in ein gefiltertes Bild mit gleichen Farbwerten wie die Pixel im Originalbild.

Matrix für den Identitätsfilter

Die Matrix für den Identitätsfilter enthält in der Mitte eine 1.0 und für alle Nachbarn das Gewicht 0.0. Werden die Gewichte mit den Farbwerten der Pixel multipliziert, dann fallen die Farbwerte der Nachbarpixel weg und es wird nur der Farbwert des Pixels in der Mitte für die Berechnung verwendet.

0 0 0
0 1.0 0
0 0 0

Faktor

Der Faktor ist 1.0, die Summe der Gewichte ist 1.0, die reziproke Summe also 1.0.

Bias

Der Bias ist 0.0, da das Ergebnis die gleichen Helligkeitswerte wie das Original haben soll.

Technische Repräsentation von Bildern

Ein Bild ist eine Abfolge von Bytes. 4 Bytes in Folge, also ein 32 Bit Integer, werden benötigt um ein Pixel zu repräsentieren. Ein Pixel entspricht einem Farbwert mit Transparenz/Opazität. Die Farbe wird im RGB Farbmodell repräsentiert.

Das unterste Byte des 32 Bit Integer kodiert die Transparenz/Opazität (255 – keine Transparenz/volle Opazität, 0 – volle Transparenz/keine Opazität), das 2'te Byte den Rot-, das 3'te Byte den Grün und das 4'te Byte den Blauanteil.

`0xb01912ff` ist eine Darstellung der Farbinformation, in der jedes Byte in hexadezimaler Schreibweise aufbereitet ist, mit Transparenz = `ff(255)`, Rot = `12(18)`, Grün =`19(25)`, Blau =`b0 (176)`. Die Zahlen in Klammern sind die Dezimalwerte.

Die Klasse `javafx.scene.paint.Color` bietet für alle Pixelinformationen Methoden an (`getRed(),…, getOpacity()`). Die Methoden liefern einen `double` Wert zurück. `Color` rechnet dazu die Integer-Werte im Intervall [0,255] in das Intervall [0,1.0) um. Die nachfolgende Berechnungsvorschrift baut auf der double Darstellung von Farbanteilen auf.

Graphische Veranschaulichung des Verfahrens

Das Originalbild zeigt eine Blumenwiese

Die nachfolgende Matrix ist ein sehr kleiner Ausschnitt aus dem Originalbild. Das rote Rechteck repräsentiert den Identitäts-Filter. Das Pixel, dessen Farbwerte mit dem Filter neu berechnet werden soll, ist rot hinterlegt.

```
0xb01912ff 0xb91b12ff 0xbf190dff 0xca2414ff 0xd02f1dff 0xc82b18ff 0xcb2a18ff
0xba2014ff 0xb51507ff 0xb91401ff 0xc5200cff 0xca2a14ff 0xc2230dff 0xc11f0aff
0xc62916ff 0xbe220cff 0xb81c05ff 0xbb1f08ff 0xc32710ff 0xc3240eff 0xc02006ff
0xbb1902ff 0xc0290eff 0xb52407ff 0xb01f04ff 0xc42913ff 0xca2a14ff 0xc22407ff
0xc72308ff 0xc32e0eff 0xae2503ff 0xa91e01ff 0xba240cff 0xc1210bff 0xc02304ff
0xeb5435ff 0xd74828ff 0xb83111ff 0xa82302ff 0xb62d09ff 0xb52905ff 0xb92604ff
0xff6248ff 0xff7454ff 0xdb5a33ff 0xbd3912ff 0xbc3309ff 0xba2f04ff 0xb92b03ff
0xdd462bff 0xfd7855ff 0xfa8454ff 0xf27942ff 0xe8682bff 0xe66224ff 0xda5218ff
0xd03519ff 0xe75231ff 0xfb7444ff 0xfd8845ff 0xf38536ff 0xff8e3eff 0xfc883dff
0xea3716ff 0xe42707ff 0xfc4c23ff 0xff7e41ff 0xfd9349ff 0xfa994cff 0xfc9e56ff
```

Berechnungsvorschrift

1. Erzeuge ein „leeres“ Ergebnisbild mit gleicher Höhe und Breite des Originalbildes.
2. Zerlege jedes Pixel „unter“ dem Filter in den Rot-, Grün- und Blauanteil (siehe Methoden der Klasse `Color`).
3. Multipliziere für jedes Pixel Rot-, Grün- und Blauanteil mit dem Gewicht der Convolution-Matrix.
4. Summiere die Teilergebnisse von 3. für Rot-, Grün- und Blauanteil separat auf.
5. Multipliziere die Summen mit dem **Faktor** und addiere den **Bias**.
6. Stelle sicher, dass das Ergebnis unter 5.‘tens >= 0 ist (max / abs)
7. Stelle sicher, dass das Ergebnis unter 6’tens <=1 ist.
8. Erzeuge mit den berechneten Rot-, Grün- und Blauwerten und der Transparenz des Pixels in der Mitte des Filters die Farbe für das Pixel in dem Ergebnisbild (siehe Methode `Color.color(…)`).
9. Speichere das Pixel im Ergebnisbild.
10. Schiebe das Rechteck (den Filter) ein Pixel weiter und wiederhole die Schritte 2-10, bis das Originalbild vollständig bearbeitet ist.

Sonstige benötigte Klassen und Funktion zur Bildbearbeitung

Für das Lesen von Pixeln aus dem Originalbild

```
Image blume = new Image(getClass().getResourceAsStream(
                          "Blumenwiese.jpg"));

PixelReader reader = image.getPixelReader();

Color col = reader.getColor(readX, readY);
```

Mit der Methode `getPixelReader()` eines Image-Objekts erhalten Sie einen Reader, der für die Pixel der Bildmatrix die Farbinformation (`Color`) liefert. Dazu müssen der Methode `getColor` der Zeilen und Spaltenindex mitgegeben werden.

Für das Schreiben von Pixeln in das Ergebnisbild

```
WritableImage filteredImage = new WritableImage((int) width,(int) height);

PixelWriter writer = filteredImage.getPixelWriter();

writer.setColor(posX,poxY,color);
```

Zunächst muss ein beschreibbares Bild der gleichen Größe des Originalbildes erzeugt werden. Mit der Methode `getPixelWriter()` dieses Image-Objekts erhalten Sie einen Writer, der Farbinformation der Pixel in das Ergebnisbild schreibt. Dazu müssen der Methode `setColor` der Zeilen und Spaltenindex sowie die Farbe des Ergebnisbildes mitgegeben werden. Achten Sie darauf, dass das Pixel auf derselben Position steht, wie im Originalbild!

Programmieraufgabe

Ihre Aufgabe wird es sein,

1. für die vorgegebenen Filterobjekte ***IDENTITY***, ***BLUR***, ***STRONG_BLUR***, ***MOTION_BLUR***, ***EDGE_DETECT***, ***HORIZONTAL_EDGES***, ***VERTICAL_EDGES***, ***SHARPEN***,***SUBTLE_SHARPEN***, ***EMBOSS*** der abstrakten Klasse `Filter` in der Methode `generateFilter()` die jeweils passende Filtermatrix zu erzeugen.

    a. Wenn Sie Muster in den Filtermatrizen vorfinden, dann erzeugen Sie den Inhalt der Matrix programmatisch und **nicht** durch Verwendung eines Array-Literals.

    b. Informieren Sie sich im Web über die einzelnen Filter.

2. in der abstrakten Klasse `Filter` private Instanz-Variablen für die Filtermatrix, den Faktor und den Bias einzuführen.

3. in der abstrakten Klasse `Filter` die beiden Konstruktoren zu implementieren. Führen Sie die Implementierung des Default-Konstruktors auf den Konstruktor mit einem Parameter zurück! Wie das geht, steht im 1'ten Script.

4. in der abstrakten Klasse `Filter` die `public` Methode `apply(Image)` für alle möglichen Filtertypen generisch zu implementieren. Die Methode `apply` setzt obige Berechnungsvorschrift um.

5. in der abstrakten Klasse `Filter` die `private` Methode `calculateFactor()` zu implementieren, die aus der Summe der Gewichte der Matrix den Multiplikator für die Teilsummen der Farbwerte bestimmt. Sollte die Teilsumme 0.0 werden, dann ist der Faktor auf 1.0 zu setzen.

Der Quelltext der abstrakten Klasse `Filter` ist in großen Teilen vorgegeben und darf nur an den Stellen modifiziert werden, an denen ein TODO steht.

Der vorgegebene Programmcode benutzt Konzepte von Java, die Sie für die Lösung der Aufgabe noch nicht verstehen müssen.

Die Klasse `ConvolutionMain` und die XML Datei `Convolution.fxml` dürfen Sie nicht verändern. Die Klasse `ConvolutionController` müssen Sie verändern, wenn Sie andere Bilder oder weitere Filter verwenden wollen.

Erläuterung der Klassen und Dateien im mitgelieferten Projekt

`Convolution.fxml` enthält den Aufbau des GUI's (in JavaFX auch Szenengraph genannt) in FXML Notation. Die Datei wurde mit dem SceneBuilder erzeugt und darf auf keinen Fall geändert werden.

`ConvolutionMain` baut das Fenster auf und liest den Szenengraph aus der fxml Datei. Dabei werden für die Elemente des XML Datei Java-Objekte für die GUI-Elemente wie Dropdown-Boxen oder ImageViews erzeugt. Gleichzeitig werden alle mit @FXML annotierten Instanz-Variablen der Klasse `ConvolutionController` mit den GUI-Objekten initialisiert. Die Verbindung zwischen fxml

und Controller-Klasse sind die `fx:id` Attribute. Die GUI-Objekte werden in die Instanz-Variablen gleichen Namens geschrieben.

`ConvolutionController` verbindet die GUI-Elemente mit der Applikationslogik. Wenn auf dem GUI eine Aktion ausgelöst wird, oder ein Wert geändert wurde, z.B. durch Auswahl eines neuen Bildes oder Filters, dann werden in den ImageViews die Originalbilder ausgetauscht, oder das Ergebnis der Anwendung eines Filters angezeigt. Dazu registriert der Controller sogenannte Listener bei den GUI-Elementen, die die Applikationslogik implementieren.

`ConvolutionController` ist auch verantwortlich für das Aufrufen der `apply` Methode der Filter-Objekte und übergibt dabei das aktuell geladene Bild (Typ `Image`).

Sie starten die Anwendung – natürlich erst, wenn Sie die Aufgabe gelöst haben – indem Sie die Klasse `ConvolutionMain` selektieren und als Java-Applikation ausführen.

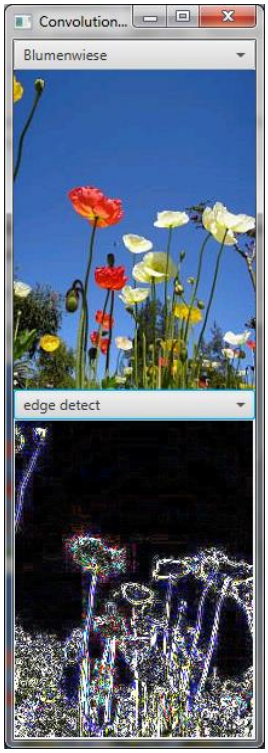
Überprüfen der Lösung

Überprüfen Sie die Korrektheit Ihrer Lösung mit dem Bild `Blumenwiese.jpg` und den nachfolgenden Filtern. Die Filtermatrix steht jeweils unter den Bildern. Der Bias ist, wenn nicht explizit angegeben immer 0.0.

IDENTITY


```
0.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 0.0
```

EDGE_DETECT

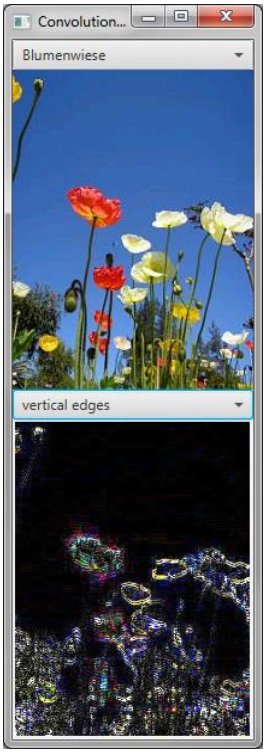



```
-1.0 -1.0 -1.0
-1.0  8.0 -1.0
-1.0 -1.0 -1.0
```

HORIZONTAL_EDGES


```
0.0  0.0 0.0 0.0 0.0
0.0  0.0 0.0 0.0 0.0
-1.0 -1.0 2.0 0.0 0.0
0.0  0.0 0.0 0.0 0.0
0.0  0.0 0.0 0.0 0.0
```

VERTICAL_EDGES




```
0.0 0.0 -1.0 0.0 0.0
0.0 0.0 -1.0 0.0 0.0
0.0 0.0  4.0 0.0 0.0
0.0 0.0 -1.0 0.0 0.0
0.0 0.0 -1.0 0.0 0.0
```

BLUR


```
0.0 0.2 0.0
0.2 0.2 0.2
0.0 0.2 0.0
```

STRONG_BLUR


```
0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 1.0 1.0 1.0 0.0 0.0
0.0 1.0 1.0 1.0 1.0 1.0 0.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0
0.0 1.0 1.0 1.0 1.0 1.0 0.0
0.0 0.0 1.0 1.0 1.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0
```

MOTION_BLUR


```
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
```

SHARPEN


```
-1.0 -1.0 -1.0
-1.0  9.0 -1.0
-1.0 -1.0 -1.0
```

SUBTLE_SHARPEN


```
-1.0 -1.0 -1.0 -1.0 -1.0
-1.0  2.0  2.0  2.0 -1.0
-1.0  2.0  8.0  2.0 -1.0
-1.0  2.0  2.0  2.0 -1.0
-1.0 -1.0 -1.0 -1.0 -1.0
```

EMBOSS mit Bias 0.5


```
-1.0 -1.0 0.0
-1.0  0.0 1.0
 0.0  1.0 1.0
```