

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	<span style="color: red;">22</span> <span style="color: red;">16</span>	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
```

*Lösungsvorschlag:*

```

1  import java.awt.image.*;
2  import java.io.*;
3  import javax.imageio.*;
4  import java.util.Random;
5  import java.util.Arrays;
6
7  /**
8   * Die Klasse Bildbearbeitung laedt eine Bilddatei und fuehrt
9   * abhaengig von gewaehlten Optionen eine Reihe von
10  * Bildmanipulationen aus.
11  */
12  public class Bildbearbeitung {
13      private int [][] pixels;
14
15      /**
16       * Konstruktor fuer die Klasse Bildbearbeitung, die eine Bilddatei einliest
17       * und das zweidimensionale Pixel-Array pixels befuellt.
18       * @param file Einzulesende Bilddatei
19       */
20      private Bildbearbeitung(String file) {
21          try {
22              BufferedImage img = ImageIO.read(new File(file));

```

```

23         pixels = new int [img.getWidth()][img.getHeight()];
24         for(int i = 0; i < dimX(); i++)
25             for(int j = 0; j < dimY(); j++)
26                 pixels[i][j] = img.getRGB(i, j);
27     } catch (IOException e) {}
28 }
29
30 private int dimX(){
31     return pixels.length;
32 }
33
34 private int dimY(){
35     if(dimX() == 0)
36         return 0;
37     return pixels[0].length;
38 }
39
40 /**
41  * Diese Funktion schreibt den Inhalt des Pixelarrays in die
42  * Datei 'ausgabe.png'. Falls diese Datei nicht vorhanden ist,
43  * wird sie angelegt. Eine vorhandene Datei wird eventuell ueberschrieben!
44  */
45 private void save(String option){
46     BufferedImage img = new BufferedImage(dimX(), dimY(), 3);
47     for(int i = 0; i < dimX(); i++)
48         for(int j = 0; j < dimY(); j++)
49             img.setRGB(i, j, pixels[i][j]);
50
51     try {
52         File outputfile = new File("ausgabe-" + option + ".png");
53         ImageIO.write(img, "png", outputfile);
54     } catch (Exception e) {}
55 }
56
57 /**
58  * Diese Funktion nimmt einen ARGB-Wert und wandelt ihn in
59  * ein vierelementiges Array um, dass die einzelnen Bytes
60  * als int-Komponenten enthaelt.
61  * @param ARGB Integre Darstellung eines Pixels mit 4 Byte
62  * Information: alpha-rot-gruen-blau
63  * @return 4-elementiges Array [alpha, red, green, blue]
64  */
65 private int[] getColors(int ARGB) {
66     // TODO nur das hier?
67     int alpha = (ARGB >> 24) & 255;
68     int red = (ARGB >> 16) & 255;
69     int green = (ARGB >> 8) & 255;
70     int blue = ARGB & 255;
71     int[] array = {alpha, red, green, blue};
72     return array;
73 }
74
75 /**
76  * Ein vierelementiges Array mit kleinen (< 1 byte) int-Werten
77  * wird zu einem einzigen 4 byte Integer zusammengesetzt.
78  * @param array 4-elementiges Array
79  * @return Eine Integerdarstellung einer Farbe in ARGB-Format.
80  */
81 private int setColors(int[] array) {

```

```

82     int alpha = array[0] << 24;
83     int red = array[1] << 16;
84     int green = array[2] << 8;
85     int blue = array[3];
86     return alpha | red | green | blue;
87 }
88
89 /**
90  * Das Bild wird hier rotiert und n*90 Grad.
91  * @param n Anzahl der Vierteldrehungen.
92  */
93 private void rotate(int n) {
94     // TODO
95     if(n <= 0)
96         return;
97
98     int [][] pixelsTemp = new int [dimY()][dimX()];
99     for(int j = 0; j < dimY(); j++){
100         for(int i = 0; i < dimX(); i++) {
101             pixelsTemp[j][dimX()-1-i] = pixels[i][j]; //ODER naechste Zeile
102             //pixelsTemp[dimY()-1-j][i] = pixels[i][j];
103         }
104     }
105     pixels = pixelsTemp;
106     rotate(n-1);
107 }
108
109 /**
110  * Die Farben werden invertiert: Farbe = (255-Farbe)
111  */
112 private void invert() {
113     // TODO
114     for(int i = 0; i < dimX(); i++) {
115         for(int j = 0; j < dimY(); j++){
116             int [] array = getColors(pixels[i][j]);
117             //array[0] = (byte)(255-array[0]);
118             array[1] = (255-array[1]);
119             array[2] = (255-array[2]);
120             array[3] = (255-array[3]);
121             pixels[i][j] = setColors(array);
122         }
123     }
124 }
125
126 /**
127  * Das Bild wird vertikal gespiegelt
128  */
129 private void mirror() {
130     // TODO
131     for(int i = 0; i < dimX()/2; i++) {
132         int [] col = pixels[i];
133         pixels[i] = pixels[dimX()-i-1];
134         pixels[dimX()-i-1] = col;
135     }
136 }
137
138 /**
139  * Hilfsfunktion zum Zugriff, die Randpunkten gueltige
140  * nullwertige Nachbarn zuweist.

```

```

141     */
142     private int getPixel(int i, int j) {
143         if(i < 0 || j < 0 || i >= dimX() || j >= dimY())
144             return 0;
145         return pixels[i][j];
146     }
147
148     /**
149      * Diese Funktion betrachtet zu jedem Pixel seine Nachbarschaft,
150      * summiert gewichtet diese Menge auf und skaliert sie
151      * @param filter 3x3 Umgebungsgewichte
152      */
153     private void meanFilter(double[] filter, double factor) {
154         int[][] pixelsTemp = new int[dimX()][dimY()];
155         for(int i = 0; i < dimX(); i++) {
156             for(int j = 0; j < dimY(); j++){
157                 double[] sum = {0.,0.,0.,0.};
158                 int[] sumInt = new int[4];
159                 for(int col = 0; col < 4; col++){
160                     sum[col] += filter[0]*getColors(getPixel(i-1,j-1))[col];
161                     sum[col] += filter[1]*getColors(getPixel(i-1,j))[col];
162                     sum[col] += filter[2]*getColors(getPixel(i-1,j+1))[col];
163                     sum[col] += filter[3]*getColors(getPixel(i,j-1))[col];
164                     sum[col] += filter[4]*getColors(getPixel(i,j))[col];
165                     sum[col] += filter[5]*getColors(getPixel(i,j+1))[col];
166                     sum[col] += filter[6]*getColors(getPixel(i+1,j-1))[col];
167                     sum[col] += filter[7]*getColors(getPixel(i+1,j))[col];
168                     sum[col] += filter[8]*getColors(getPixel(i+1,j+1))[col];
169                     sumInt[col] = (int)(sum[col]*factor);
170                 }
171                 pixelsTemp[i][j] = setColors(sumInt);
172             }
173         }
174         pixels = pixelsTemp;
175     }
176
177
178     /**
179      * Gaussfilter
180      */
181     private void gauss(){
182         double[] filter = {1,2,1,2,4,2,1,2,1};
183         meanFilter(filter, 1.0/16.0);
184     }
185
186     /**
187      * Blurfilter/ Lowpassfilter
188      */
189     private void lpf(){
190         double[] filter = {1,1,1,1,1,1,1,1,1};
191         meanFilter(filter, 1.0/9.0);
192     }
193
194     /**
195      * Hochpassfilter 1
196      */
197     private void hpf1(){
198         double[] filter = {0,-1,0,-1,4,-1,0,-1,0};
199         meanFilter(filter, 1.0);

```

```

200     }
201
202     /**
203      * Hochpassfilter 2
204      */
205     private void hpf2(){
206         double[] filter = {-1,-1,-1,-1,9,-1,-1,-1,-1};
207         meanFilter(filter,1.0);
208     }
209
210     /**
211      * Medianfilter
212      */
213     private void medianFilter(){
214         //TODO
215         int [][] pixelsTemp = new int [dimX()][dimY()];
216         for(int i = 0; i < dimX(); i++) {
217             for(int j = 0; j < dimY(); j++){
218                 int [] values = new int [4];
219                 for(int col = 0; col < 4; col++){
220                     int [] arrayMedian = new int [5];
221                     arrayMedian[0] = getColors(getPixel(i,j))[col];
222                     arrayMedian[1] = getColors(getPixel(i-1,j))[col];
223                     arrayMedian[2] = getColors(getPixel(i+1,j))[col];
224                     arrayMedian[3] = getColors(getPixel(i,j-1))[col];
225                     arrayMedian[4] = getColors(getPixel(i,j+1))[col];
226                     Arrays.sort(arrayMedian);
227                     values[col] = arrayMedian[2];
228                 }
229                 pixelsTemp[i][j] = setColors(values);
230             }
231         }
232         pixels = pixelsTemp;
233     }
234
235     /**
236      * Fuegt auf n Pixeln Rauschen hinzu.
237      */
238     private void jitter(int n) {
239         Random random = new Random();
240         for(int i = 0; i < n; i++) {
241             int x = random.nextInt(dimX());
242             int y = random.nextInt(dimY());
243             int [] colors = getColors(getPixel(x,y));
244             colors[1] = random.nextInt(256);
245             colors[2] = random.nextInt(256);
246             colors[3] = random.nextInt(256);
247             pixels[x][y] = setColors(colors);
248         }
249     }
250
251     public static void main(String [] args) {
252
253         int argCount = args.length;
254
255         if(argCount == 0) {
256             System.out.println("Verwendung: java Bildbearbeitung <FILE> [-options]");
257             System.out.println("wobei options Folgendes umfasst:");
258             System.out.println("\t -rot90 90 Grad rotieren");

```

```

259         System.out.println("\t -rot180 180 Grad rotieren");
260         System.out.println("\t -rot270 270 Grad rotieren");
261         System.out.println("\t -i Farben invertieren");
262         System.out.println("\t -m Horizontal spiegeln");
263         System.out.println("\t -lpf Verwaschen");
264         System.out.println("\t -median Median-Filter");
265         System.out.println("\t -gauss Gauss-Filter");
266         System.out.println("\t -hpf1 Hochpassfilter1");
267         System.out.println("\t -hpf2 Hochpassfilter2");
268         System.out.println("\t -jitter Verrauscht das Bild");
269         return;
270     }
271
272     // load image as specified in first argument args[0]
273     Bildbearbeitung bild = new Bildbearbeitung(args[0]);
274
275     bild.save("raw");
276     for(int i = 1; i < argCount; i++) {
277         String option = args[i];
278         System.out.println("Processing: " + option);
279         if(option.equals("-rot90"))
280             bild.rotate(1);
281         else if(option.equals("-rot180"))
282             bild.rotate(2);
283         else if(option.equals("-rot270"))
284             bild.rotate(3);
285         else if(option.equals("-i"))
286             bild.invert();
287         else if(option.equals("-m"))
288             bild.mirror();
289         else if(option.equals("-gauss"))
290             bild.gauss();
291         else if(option.equals("-median"))
292             bild.medianFilter();
293         else if(option.equals("-lpf"))
294             bild.lpf();
295         else if(option.equals("-jitter"))
296             bild.jitter(50000);
297         else if(option.equals("-hpf1"))
298             bild.hpf1();
299         else if(option.equals("-hpf2"))
300             bild.hpf2();
301
302         bild.save(""+i);
303     }
304
305     bild.save("done");
306 }
307 }

```