

Medientechnik

Übung – Java-Bildbearbeitung

Java2D - Bildbearbeitung

- Bilder laden
- Bildfilter
 - Fertige Filter in Java2D
 - Eigenen Filter implementieren
 - Fertige Filter von Drittanbietern

Bilder laden

Einlesen von Bilddateien umfasst komplexe Algorithmen

- Decodieren des Bildformats
- Einlesen lokal aus Datei oder über eine URL
- Berücksichtigung von langsamen Festplatten- und Netzzugriffen
- » Observer-Modell: Anzeigefunktion wird wieder aufgerufen, wenn Daten nachgeladen sind

Java: Diverse Möglichkeiten zum Laden eines Bildes (Image)

- Standard-AWT-Methode (MediaTracker)
- Swing-Methode (ImageIcon)

Laden eines BufferedImage:

- ImageIO
- Spezielle Codecs (Sun-JPEG-Codec meist in Standardinstallation enthalten)

BufferedImage laden

Beispiele mit ImageIO oder Spezial-Codec für JPEGs

```
public BufferedImage loadImage(String filename) throws IOException{  
    return ImageIO.read(new File(filename));  
}
```

```
import com.sun.image.codec.jpeg.*;  
  
public BufferedImage loadImage(String filename) throws IOException, ImageFormatException{  
    InputStream in = new FileInputStream(file);  
    JPEGImageDecoder decoder = JPEGCodec.createJPEGDecoder(in);  
    BufferedImage img = decoder.decodeAsBufferedImage();  
    in.close();  
    return img;  
}
```

Fehlermeldung bei Sun-Importen

- aufgrund einer Eclipse-Einstellung werden teilweise Fehler (“Access restriction...”) bei Verwendung der Sun Pakete angezeigt
- Lösung:
 - Window → Preferences → Java → Compiler → Errors/Warnings → Deprecated and restricted API
 - auf „Warning“ oder „Ignore“ stellen

Java - Bildfilter

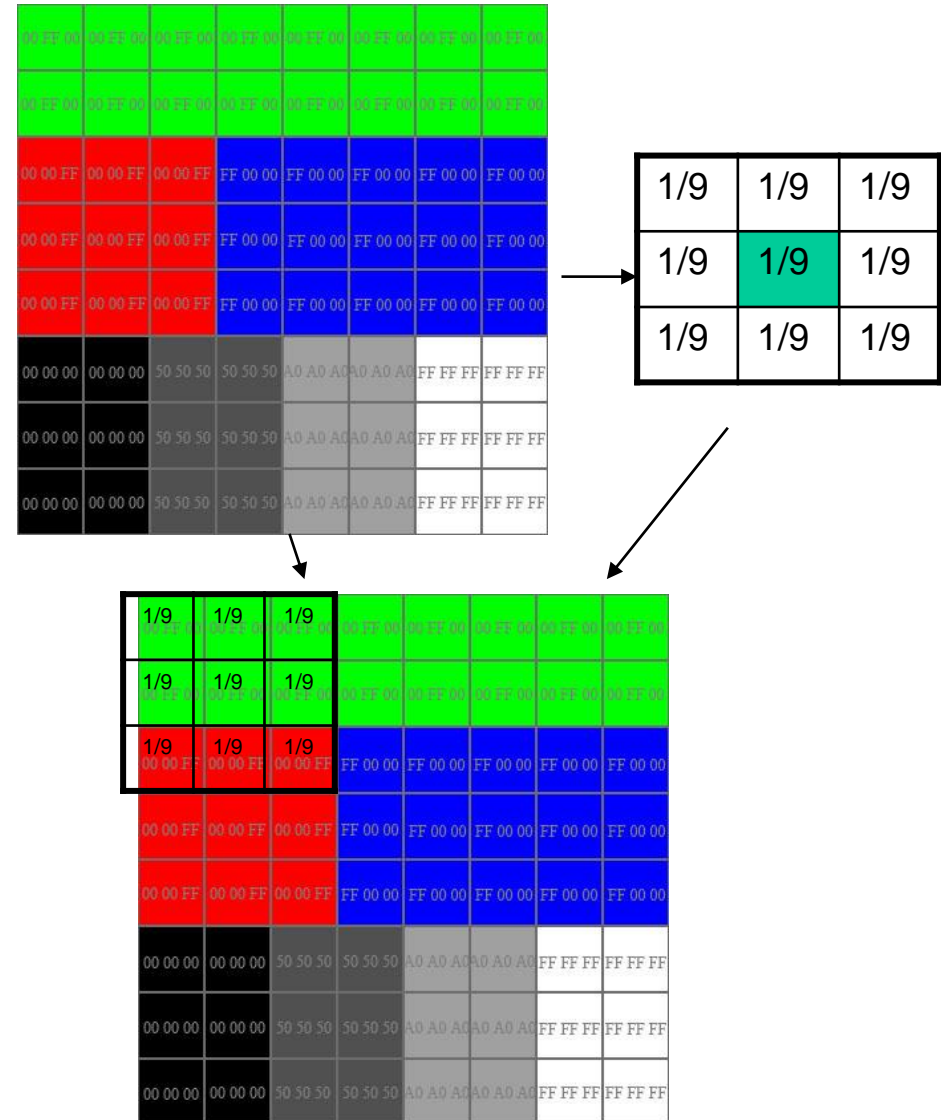
Klasse	Hilfsklassen	Effekte	"in place"?(src = dst)
ConvolveOp	Kernel	Weich- und Scharfzeichnen, Kantenerkennung	nein
Affine TransformOp	java.awt.geom. AffineTransform	Geometrische Transformationen	nein
LookupOp	LookupTable, ByteLookupTable, ShortLookupTable	Inversion, Farbtrennung, Aufhellung, Thresholding	ja
RescaleOp		Aufhellen, Abdunkeln	ja
Color ConvertOp	java.awt.Color. ColorSpace	Farbraum-konversion	ja

ConvolveOp

- Matrix wird über das Bild gelegt
- Matrixwert und Pixelwert werden multipliziert
- Anschließend werden alle Werte addiert

=> Neuer Pixelwert des Mittelpunktes der Matrix

```
float[] values = {
    1/9f, 1/9f, 1/9f,
    1/9f, 1/9f, 1/9f,
    1/9f, 1/9f, 1/9f
};
Kernel kernel = new Kernel(3,3,values);
ConvolveOp cOp= new ConvolveOp(kernel);
cOp.filter(srcImg, dstImg);
```



Eigener Filter

Theorie: Wie kann man Bilder auf Pixelbasis manipulieren?

- Spezielle Datenstruktur speichert Werte der einzelnen Pixel
- Zugriff und Manipulation einzelner Pixel durch Bit-Operatoren (<<, >>, &, |)

00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00
00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00	00 FF 00 00
00 00 FF	00 00 FF	00 00 FF	FF 00 00	FF 00 00	FF 00 00	FF 00 00	FF 00 00
00 00 FF	00 00 FF	00 00 FF	FF 00 00	FF 00 00	FF 00 00	FF 00 00	FF 00 00
00 00 FF	00 00 FF	00 00 FF	FF 00 00	FF 00 00	FF 00 00	FF 00 00	FF 00 00
00 00 00	00 00 00	50 50 50	50 50 50	A0 A0 A0	A0 A0 A0	FF FF FF	FF FF FF
00 00 00	00 00 00	50 50 50	50 50 50	A0 A0 A0	A0 A0 A0	FF FF FF	FF FF FF
00 00 00	00 00 00	50 50 50	50 50 50	A0 A0 A0	A0 A0 A0	FF FF FF	FF FF FF

Eigener Filter

Beispiel Java, *BufferedImage*, Typ RGB:

- Zugriff auf einen einzelnen Pixel über:
 - int getRGB(int x, int y)
- Laden der Pixelwerte in ein Array (rgbArray) über:
 - int[] getRGB(int startX, int startY, int w, int h, int[] rgbArray, int offset, int scansize)

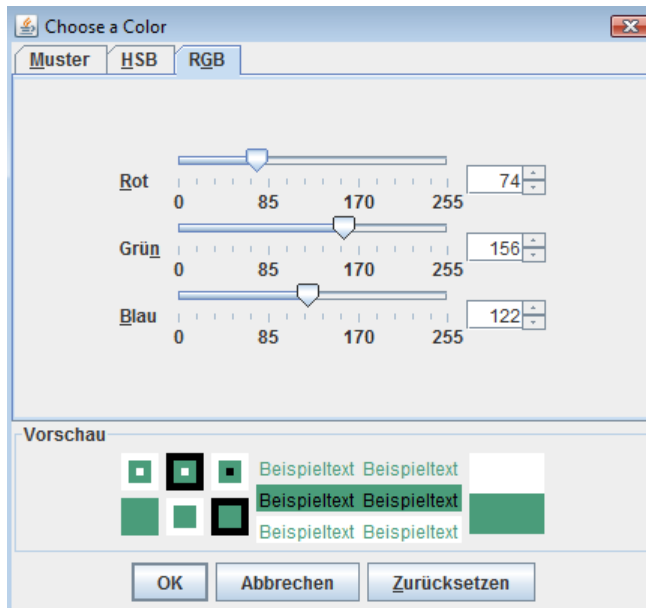
Eigener Filter – Manipulation eines Pixels

Dezimal: (74, 156, 122)

Hexadezimal: (4A, 9C, 7A)

⇒ 0x4A9C7A

⇒ $4889722_{10} = 1001010\ 10011100\ 01111010_2$



Wie kommt man z.B. an den Grün-Wert?

```
int p = srcImg.getRGB(x, y);
```

//Sei p = 4889722 der Wert des Pixels an Position(x/y)

```
int red = (p >> 16) & 0xff;
```

```
int green = (p >> 8) & 0xff;
```

```
int blue = (p) & 0xff;
```

Logische Operatoren

```
int green = (p >> 8) & 0xff;
```

`p >> 8` (Bitshift um 8 Stellen nach rechts)

```
01001010 10011100 01111010 >> 8 =  
00000000 01001010 10011100
```

`(p >> 8) & 0xff` (logisches Und, filtert nur die Werte der letzten 8 Bits heraus)

```
01001010 10011100  
&    11111111  
=    100111002 = 0x9C
```

Zusammensetzen einzelner Farbwerte mit |
(logisches Oder)

```
int newRGBValue = (red<<16)|(green<<8)|blue;
```

Weitere Filter: Beispiel JH Labs

Color Adjustment Filters

Distortion and Warping Filters

Effects Filters

Texturing Filters

Blurring and Sharpening Filters

Edge Detection

Transitions

Alpha Channel Filters



[TwirlFilter](#) - Distort an image by twisting



[WarpFilter](#) - A general grid image warp



[WaterFilter](#) - Simulate water ripples

Effects Filters



[BlockFilter](#) - Mosaic or pixellate an image



[BorderFilter](#) - Add a border



[ChromeFilter](#) - Simulate chrome



[ColorHalftoneFilter](#) - Color halftoning effect.



[CrystallizeFilter](#) - Make an image look like stained glass



[EmbossFilter](#) - Simple embossing

JH Labs Filter

- jar herunterladen
<http://www.jhlabs.com/ip/filters/download.html>
- In Projekt einbinden
 - neuer Ordner “lib”, jar reinkopieren
 - Project Properties -> Java Build Path -> Add External Jar
- Filter aussuchen und verwenden

