Bildverarbeitung mit ImageJ

Dr. Birgit Möller
Institut für Informatik
Martin-Luther-Universität Halle-Wittenberg

Einführung in die Bildverarbeitung, SS 2008

Überblick

ImageJ-Resourcen

- Installation im Linux-Pool:

  `/lehre/ImageJ`

- Installation im Windows-Pool:

  `S:\ImageJ`

- Starten mit:

  ```
  java -cp /lehre/ImageJ/ij.jar ij.ImageJ
  ```

  bzw.

  ```
  java -cp S:\ImageJ\ij.jar ij.ImageJ
  ```

Das Plugin-Konzept

- Grundidee: Bilder sind Objekte,
  die mit Operatoren bearbeitet werden
- jeder Operator wird durch ein **Plugin** realisiert
- eine Hintereinander-Ausführung mehrerer Plugins ergibt komplexere Verarbeitungsstränge
- Automatisierung durch Macro-Sprache
  (für uns erstmal uninteressant)

Die Plugin-Schnittstelle

- jedes Plugin wird durch eine eigene Java-Klasse realisiert
- die Klasse leitet sich von vorgegebenen ImageJ-Interfaces ab
- Plugins werden automatisch beim Start von ImageJ geladen, wenn. . .
  - ihr (Klassen-) Name einen Unterstrich ("_") beinhaltet
  - sie in einem Verzeichnis namens *"plugins"* liegen

Realisierung eigener Plugins

1. Java-Klasse *"my_plugin.java"* schreiben
   (Datei- und Klassenname müssen identisch sein!)

2. Java-Klasse compilieren:

   ```
   javac -cp <ImageJ-Installdir>/ij.jar my_plugin.java
   ```

3. die .class-Datei in einen Ordner *plugins* kopieren,
   z.B. */home/moeller/plugins*

4. ImageJ aufrufen:

   ```
   java -Dplugins.dir=/home/moeller/
        -cp <ImageJ-Installdir>/ij.jar ij.ImageJ
   ```

5. *my_plugin* im Menü 'Plugins' aufrufen

Plugin-Interfaces

- `PlugIn`:
  Basis-Plugin, das keinerlei Eingaben erwartet
- `PlugInFilter`:
  Plugin, das ein Bild als Eingabe bekommt
- `PlugInFrame`:
  Plugin, das in einem eigenen Fenster selbständig läuft

Plugin-Interfaces

- `PlugIn`:
  Basis-Plugin, das keinerlei Eingaben erwartet
- `PlugInFilter`:
  Plugin, das ein Bild als Eingabe bekommt
- `PlugInFrame`:
  Plugin, das in einem eigenen Fenster selbständig läuft

$\Rightarrow$ wir werden von `PlugInFilter` ableiten

Schnittstelle von `PlugInFilter`

Das Interface definiert nur zwei Funktionen:

- `int setup(java.lang.String arg, ImagePlus imp):`

  Setup-Routine, die (1x) beim Laden aufgerufen wird

- `void run(ImageProcessor ip):`

  Hauptroutine, in der das Bild verarbeitet wird, und die mehrfach ausgeführt werden kann

Setup

`int setup(java.lang.String arg, ImagePlus imp):`

- Initialisierung von Klassenvariablen, etc.
- Rückgabewert spezifiziert Eigenschaften des Plugins:
  - `DOES_8G` - 8-bit Grauwertbilder
  - `DOES_ALL` - alle Bildtypen
  - `DOES_RGB` - nur RGB-Farbbilder
  - `DOES_ALL+DOES_RGB`
  - `...`
- Beispiel-Implementierung:

```
public int setup(String arg,ImagePlus imp) {
        return DOES_RGB+DOES_ALL;
}
```

Bildklassen

ImageJ kennt fünf Typen von Bildern:

- 8-bit Grauwertbilder (`GRAY8`)
- 8-bit Farbbilder mit Lookup-Tabelle (`COLOR_256`)
- 16-bit Grauwertbilder (`GRAY16`)
- RGB-Bilder mit 256 Werten pro Kanal (`COLOR_RGB`)
- 32-bit Grauwertbilder (`GRAY32`)

Bildklassen

ImageJ kennt fünf Typen von Bildern:

- 8-bit Grauwertbilder (`GRAY8`)
- 8-bit Farbbilder mit Lookup-Tabelle (`COLOR_256`)
- 16-bit Grauwertbilder (`GRAY16`)
- RGB-Bilder mit 256 Werten pro Kanal (`COLOR_RGB`)
- 32-bit Grauwertbilder (`GRAY32`)

$\Rightarrow$ Wir werden hauptsächlich mit `GRAY8` arbeiten.

Bildklassen (2)

- Basis-Bildklasse: `ImagePlus`
- zu jedem Bild gehört außerdem ein `ImageProcessor`

  $\Rightarrow$ Achtung! Unschönes Software-Design!
- normalerweise wird mit `ImageProcessor` gearbeitet
- Zugriff auf den `ImageProcessor`:

```
ImagePlus myImg;
ImageProcessor myProcessor=
                    myImg.getProcessor();
```

Bildklassen (3)

- neue Bilder erzeugen:

```
ImagePlus img=
NewImage.createRGBImage("Title", w, h, 1,
                        NewImage.FILL_BLACK);
```

- wichtige Funktionen von `ImageProcessor`:
  - `int getWidth(), int getHeight()`
  - `int getPixel(int x, int y),`
    `void putPixel(int x, int y, int value)`
  - `void invert(), void smooth(),`
    `void add(int value)`
  - ...und noch viele mehr (siehe Tutorial)

Pixelzugriffe

a) über Zugriffsfunktionen:
   - `int getPixel(x, y)`
   - `int [] getPixel(x, y, null)`
   - `void putPixel(x, y, value)`

b) direkte Bearbeitung des Pixel-Arrays:
   - `<type> [] getPixels()`

- Typ des Arrays bei b) abhängig vom Bildtyp
- Grauwertbilder basieren auf `byte`-Arrays
  $\Rightarrow$ Wertebereich von `byte` in Java -127 bis 128
- Umwandlung:

```
int pix= pixels[i] & 0xff;
pixels[i] = (byte) pix;
```

Pixelzugriffe

a) über Zugriffsfunktionen:
   - `int getPixel(x, y)`
   - `int [] getPixel(x, y, null)`
   - `void putPixel(x, y, value)`

b) direkte Bearbeitung des Pixel-Arrays:
   - `<type> [] getPixels()`

- Typ des Arrays bei b) abhängig vom Bildtyp
- Grauwertbilder basieren auf `byte`-Arrays
  $\Rightarrow$ Wertebereich von `byte` in Java -127 bis 128
- Umwandlung:

```
int pix= pixels[i] & 0xff;
pixels[i] = (byte) pix;
```

Pixelzugriffe (2)

- die Arrays sind eindimensional, zeilenweise aneinandergehängt...

```
int [] pixels= img.getPixels();
for (int y= 0; y< img.getHeight(); ++y)
    for (int x= 0; x< img.getWidth(); ++x)
        int pos= y*img.getWidth() + x;
        int val= pixels[pos] & 0xff;
        ...
        pixel[pos]= (byte) neuerWert;
```

Pixelzugriffe (3)

- Farbbilder liefern 1D `int`-Arrays zurück
- ein `int` kodiert jeweils alle Farbkanäle:

```
int red= (int)(pixels[i] & 0xff0000) >> 16;
int green= (int)(pixels[i] & 0x00ff00) >> 8;
int blue= (int)(pixels[i] & 0x0000ff);
...
pixels[i]= ((red & 0xff)<<16) +
           ((green & 0xff)<<8) +
           (blue & 0xff);
```

Ausgaben

- ImageJ kapselt die Kommandozeile
- Fehlermeldungen:

```
IJ.error("Ein Fehler ist aufgetreten!");
```

- Nachrichtenfenster:

```
IJ.showMessage("Das Ergebnis ist 42!");
```

Weitere Dokumentation:

- ImageJ-Homepage:
  `http://rsb.info.nih.gov/ij/`
- ImageJ Documentation Portal:
  `http://imagejdocu.tudor.lu/`
- ImageJ-Tutorial - siehe Homepage zur Vorlesung:
  `http://www2.informatik.uni-halle.de/agprbio/AG/Lehre/EBV_SS08/infos.html`