
ImageJ

Bis vor wenigen Jahren war die Bildverarbeitungs-“Community“ eine relativ kleine Gruppe von Personen, die entweder Zugang zu teuren Bildverarbeitungswerkzeugen hatte oder – aus Notwendigkeit – damit begann, eigene Softwarepakete für die digitale Bildverarbeitung zu programmieren. Meistens begannen solche „Eigenbau“-Umgebungen mit kleinen Programmkomponenten zum Laden und Speichern von Bildern, von und auf Dateien. Das war nicht immer einfach, denn oft hatte man es mit mangelhaft dokumentierten oder firmenspezifischen Dateiformaten zu tun. Die nahe liegendste Lösung war daher häufig, zunächst sein eigenes, für den jeweiligen Einsatzbereich „optimales“ Dateiformat zu entwerfen, was weltweit zu einer Vielzahl verschiedenster Dateiformate führte, von denen viele heute glücklicherweise wieder vergessen sind [155]. Das Schreiben von Programmen zur Konvertierung zwischen diesen Formaten war daher in den 1980ern und frühen 1990ern eine wichtige Angelegenheit. Die Darstellung von Bildern auf dem Bildschirm war ähnlich schwierig, da es dafür wenig Unterstützung vonseiten der Betriebssysteme und Systemschnittstellen gab. Es dauerte daher oft Wochen oder sogar Monate, bevor man am Computer auch nur elementare Dinge mit Bildern tun konnte und bevor man vor allem an die Entwicklung neuer Algorithmen für die Bildverarbeitung denken konnte.

Glücklicherweise ist heute vieles anders. Nur wenige, wichtige Bildformate haben überlebt (s. auch Abschn. 1.5) und sind meist über fertige Programmbibliotheken leicht zugreifbar. Die meisten Standard-APIs, z. B. für C++ und Java, beinhalten bereits eine Basisunterstützung für Bilder und andere digitale Mediendaten.

2.1 Software für digitale Bilder

Traditionell ist Software für digitale Bilder entweder zur Bearbeitung von Bildern oder zum Programmieren ausgelegt, also entweder für den Praktiker und Designer oder für den Programmentwickler.

2.1.1 Software zur Bildbearbeitung

Softwareanwendungen für die Manipulation von Bildern, wie z. B. Adobe Photoshop, Corel Paint u. v. a., bieten ein meist sehr komfortables User Interface und eine große Anzahl fertiger Funktionen und Werkzeuge, um Bilder interaktiv zu bearbeiten. Die Erweiterung der bestehenden Funktionalität durch *eigene* Programmkomponenten wird zwar teilweise unterstützt, z. B. können „Plugins“ für Photoshop¹ in C++ programmiert werden, doch ist dies eine meist aufwändige und jedenfalls für Programmieranfänger zu komplexe Aufgabe.

2.1.2 Software zur Bildverarbeitung

Im Gegensatz dazu unterstützt „echte“ Software für die digitale Bildverarbeitung primär die Erfordernisse von Algorithmenentwicklern und Programmierern und bietet dafür normalerweise weniger Komfort und interaktive Möglichkeiten für die Bildbearbeitung. Stattdessen bieten diese Umgebungen meist umfassende und gut dokumentierte Programm-bibliotheken, aus denen relativ einfach und rasch neue Prototypen und Anwendungen erstellt werden können. Beispiele dafür sind etwa *Khoros/VisiQuest*², *IDL*³, *MatLab*⁴ und *ImageMagick*⁵. Neben der Möglichkeit zur konventionellen Programmierung (üblicherweise mit C/C++) werden häufig einfache Scriptsprachen und visuelle Programmierhilfen angeboten, mit denen auch komplizierte Abläufe auf einfache und sichere Weise konstruiert werden können.

2.2 Eigenschaften von ImageJ

ImageJ, das wir für dieses Buch verwenden, ist eine Mischung beider Welten. Es bietet einerseits bereits fertige Werkzeuge zur Darstellung und interaktiven Manipulation von Bildern, andererseits lässt es sich extrem einfach durch eigene Softwarekomponenten erweitern. ImageJ ist vollständig in Java implementiert, ist damit weitgehend plattformunabhängig und läuft unverändert u. a. unter Windows, MacOS und Linux.

¹ www.adobe.com/products/photoshop/

² www.accusoft.com/imaging/visiquest/

³ www.rsinc.com/idl/

⁴ www.mathworks.com

⁵ www.imagemagick.org

Die dynamische Struktur von Java ermöglicht es, eigene Module – so genannte „Plugins“ – in Form eigenständiger Java-Codestücke zu erstellen und „on-the-fly“ im laufenden System zu übersetzen und auch sofort auszuführen, ohne ImageJ neu starten zu müssen. Dieser schnelle Ablauf macht ImageJ zu einer idealen Basis, um neue Bildverarbeitungsalgorithmen zu entwickeln und mit ihnen zu experimentieren. Da Java an vielen Ausbildungseinrichtungen immer häufiger als erste Programmiersprache unterrichtet wird, ist das Erlernen einer zusätzlichen Programmiersprache oft nicht notwendig und der Einstieg für viele Studierende sehr einfach. ImageJ ist zudem frei verfügbar, sodass Studierende und Lehrende die Software legal und ohne Lizenzkosten auf allen ihren Computern verwenden können. ImageJ ist daher eine ideale Basis für die Ausbildung in der digitalen Bildverarbeitung, es wird aber auch in vielen Labors, speziell in der Biologie und Medizin, für die tägliche Arbeit eingesetzt.

Entwickelt wurde (und wird) ImageJ von Wayne Rasband [177] am U.S. *National Institutes of Health* (NIH) als Nachfolgeprojekt der älteren Software *NIH-Image*, die allerdings nur auf MacIntosh verfügbar war. Die aktuelle Version von ImageJ, Updates, Dokumentation, Testbilder und eine ständig wachsende Sammlung beigestellter Plugins finden sich auf der ImageJ-Homepage.⁶ Praktisch ist auch, dass der gesamte Quellcode von ImageJ online zur Verfügung steht. Die Installation von ImageJ ist einfach, Details dazu finden sich in der Online-Installationsanleitung, im IJ-Tutorial [10] sowie auf der ImageJ-Homepage.

ImageJ ist allerdings nicht perfekt und weist softwaretechnisch sogar erhebliche Mängel auf, nicht zuletzt aufgrund seiner Entstehungsgeschichte. Die Architektur ist nicht allzu übersichtlich und speziell die Unterscheidung zwischen den häufig verwendeten *ImageProcessor*- und *ImagePlus*-Objekten bereitet (nicht nur Anfängern) erhebliche Schwierigkeiten. Die Implementierung einzelner Komponenten könnte konsistenter sein und unterschiedliche Funktionalitäten sind oft nicht sauber voneinander getrennt. Auch die fehlende Orthogonalität ist bisweilen ein Problem, d. h., ein und dieselbe Operation kann teilweise auf mehrfache Weise realisiert werden. Dennoch ist ImageJ ein bewährtes, extrem einfach zu erweiterndes Werkzeug, das auch professionell in vielen Bereichen eingesetzt wird und sich zudem hervorragend für die Ausbildung eignet.

Neben ImageJ selbst gibt es zahlreiche Software-Projekte, die direkt auf ImageJ aufbauen oder ImageJ verwenden. Dazu gehört insbesondere *Fiji*⁷ („Fiji Is Just ImageJ“), das eine konsistente Sammlung zahlreicher Plugins, einfache Installation auf unterschiedlichen Plattformen und eine hervorragende Dokumentation bietet. Alle hier gezeigten Programmbeispiele (Plugins) sollten unverändert auch in Fiji ausführbar sein. Eine weitere wichtige Entwicklung ist *ImgLib2*⁸ [170], in dem eine generische Java-Bibliothek zur einheitlichen Repräsentation und Verarbeitung



Wayne Rasband (rechts) bei der 1. ImageJ Konferenz 2006 (Bild: Marc Seil, CRP Henri Tudor, Luxembourg).

⁶ <http://rsb.info.nih.gov/ij/>

⁷ <http://fiji.sc>

⁸ <http://fiji.sc/ImgLib2>

n -dimensionaler Daten speziell für die Bildverarbeitung realisiert wird. ImgLib2 bildet auch das grundlegende Datenmodell von *ImageJ2*,⁹ das eine vollständige und vielversprechende Neuimplementierung von ImageJ zum Ziel hat.

2.2.1 Features

Als reine Java-Anwendung läuft ImageJ auf praktisch jedem Computer, für den eine aktuelle Java-Laufzeitumgebung (Java *runtime environment*, „jre“) existiert. Bei der Installation von ImageJ wird ein eigenes Java-Runtime mitgeliefert, sodass Java selbst nicht separat installiert sein muss. ImageJ kann, unter den üblichen Einschränkungen, auch als *Java-Applet* innerhalb eines Web-Browsers betrieben werden, meistens wird es jedoch als selbstständige Java-Applikation verwendet. ImageJ kann sogar serverseitig, z. B. für Bildverarbeitungsoperationen in Online-Anwendungen eingesetzt werden [10]. Zusammengefasst sind die wichtigsten Eigenschaften von ImageJ:

- Ein Satz von fertigen Werkzeugen zum Erzeugen, Visualisieren, Editieren, Verarbeiten, Analysieren, Öffnen und Speichern von Bildern in mehreren Dateiformaten. ImageJ unterstützt auch „tiefe“ Integer-Bilder mit 16 und 32 Bits sowie Gleitkommabilder und Bildfolgen (sog. *stacks*).
- Ein einfacher Plugin-Mechanismus zur Erweiterung der Basisfunktionalität durch kleine Java-Codesegmente. Dieser ist die Grundlage aller Beispiele in diesem Buch.
- Eine Makro-Sprache und ein zugehöriger Interpreter, die es erlauben, ohne Java-Kenntnisse bestehende Funktionen zu größeren Verarbeitungsfolgen zu verbinden. ImageJ-Makros werden in diesem Buch nicht eingesetzt; die zugehörigen Details finden sich online.¹⁰

2.2.2 Interaktive Werkzeuge

Nach dem Start öffnet ImageJ zunächst sein Hauptfenster (Abb. 2.1), das mit folgenden Menü-Einträgen die eingebauten Werkzeuge zur Verfügung stellt:

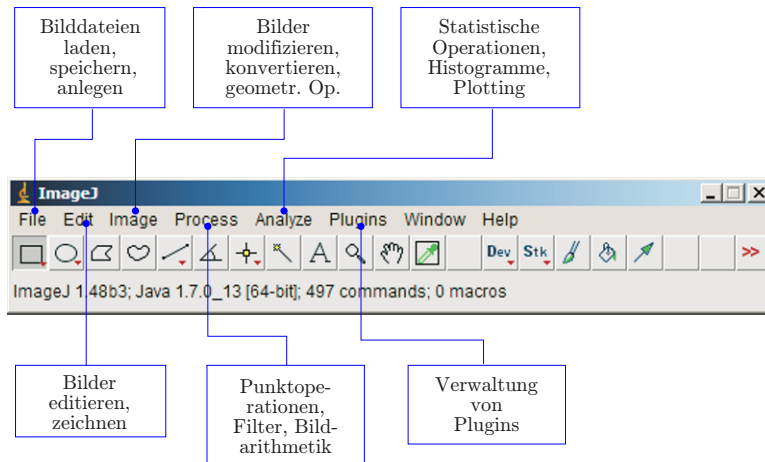
- **File:** zum Laden und Speichern von Bildern sowie zum Erzeugen neuer Bilder.
- **Edit:** zum Editieren und Zeichnen in Bildern.
- **Image:** zur Modifikation und Umwandlung von Bildern sowie für geometrische Operationen.

⁹ <http://developer.imagej.net/>. Um Verwechslungen zu vermeiden, wird in diesem Zusammenhang das „klassische“ ImageJ häufig als „ImageJ1“ bzw. „IJ1“ bezeichnet.

¹⁰ <http://rsb.info.nih.gov/ij/developer/macro/macros.html>.

Abbildung 2.1

Hauptfenster von ImageJ (unter Windows).



- **Process:** für typische Bildverarbeitungsoperationen, wie Punktoperationen, Filter und arithmetische Operationen auf Bilder.
- **Analyze:** für die statistische Auswertung von Bilddaten, Anzeige von Histogrammen und spezielle Darstellungsformen.
- **Plugin:** zum Bearbeiten, Übersetzen, Ausführen und Ordnen eigener Plugins.

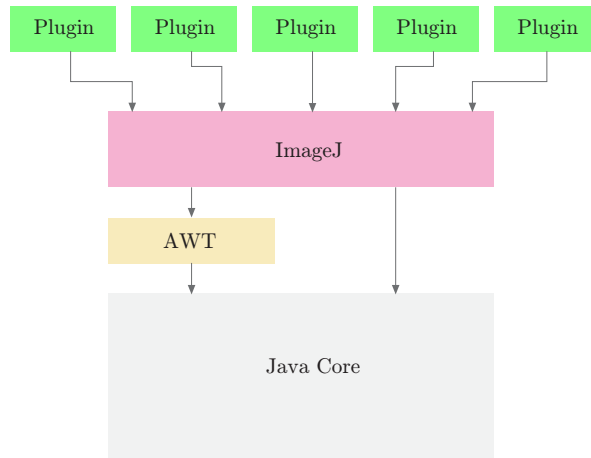
ImageJ kann derzeit Bilddateien in mehreren Formaten öffnen, u.a. TIFF (nur unkomprimiert), JPEG, GIF, PNG und BMP, sowie die in der Medizin bzw. Astronomie gängigen Formate DICOM (Digital Imaging and Communications in Medicine) und FITS (Flexible Image Transport System). Wie in ähnlichen Programmen üblich, werden auch in ImageJ alle Operationen auf das aktuell selektierte Bild (current image) angewandt. ImageJ verfügt für die meisten eingebauten Operationen einen „Undo“-Mechanismus, der (auf einen Arbeitsschritt beschränkt) auch die selbst erzeugten Plugins unterstützt.

2.2.3 ImageJ-Plugins

Plugins sind kleine, in Java definierte Softwaremodule, die in einfacher, standardisierter Form in ImageJ eingebunden werden und damit seine Funktionalität erweitern (Abb. 2.2). Zur Verwaltung und Benutzung der Plugins stellt ImageJ über das Hauptfenster (Abb. 2.1) ein eigenes **Plugin-Menü** zur Verfügung. ImageJ ist modular aufgebaut und tatsächlich sind zahlreiche eingebaute Funktionen wiederum selbst als Plugins implementiert. Als Plugins realisierte Funktionen können auch beliebig in einem der Hauptmenüs von ImageJ platziert werden.

Abbildung 2.2

Software-Struktur von ImageJ (vereinfacht). ImageJ basiert auf dem Java-Kernsystem und verwendet insbesondere Javas AWT (Advanced Windowing Toolkit) als Grundlage für das User Interface und die Darstellung von Bilddaten. Plugins sind kleine Java-Klassen mit einer einfachen Schnittstelle zu ImageJ, mit denen sich die Funktionalität des Systems leicht erweitern lässt.



Programmstruktur

Technisch betrachtet sind Plugins Java-Klassen, die eine durch ImageJ vorgegebene Interface-Spezifikation implementieren. Es gibt zwei verschiedene Arten von Plugins:

- **Plugin** benötigt keinerlei Argumente, kann daher auch ohne Beteiligung eines Bilds ausgeführt werden;
- **PluginFilter** wird beim Start immer ein Bild (das aktuelle Bild) übergeben.

Wir verwenden in diesem Buch fast ausschließlich den zweiten Typ – **PluginFilter** – zur Realisierung von Bildverarbeitungsoperationen. Eine Plugin-Klasse, die auf dem **PluginFilter-Interface** basiert, muss zumindest die folgenden beiden Methoden enthalten:

```
int setup (String args, ImagePlus im)
```

Diese Methode wird bei der Ausführung eines Plugin von ImageJ als erste aufgerufen, vor allem um zu überprüfen, ob die Spezifikationen des Plugin mit dem aktuellen Bild zusammenpassen. Die Methode liefert einen 32-Bit **int**-Wert, dessen Bitmuster die Eigenschaften des Plugin beschreibt.

```
void run (ImageProcessor ip)
```

Diese Methode erledigt die tatsächliche Arbeit des Plugin. Der einzige Parameter **ip** (ein Objekt vom Typ **ImageProcessor**) verweist auf das zu bearbeitende Bild und relevante Informationen dazu. Die **run**-Methode liefert keinen Rückgabewert (**void**), kann aber das übergebene Bild verändern und auch neue Bilder erzeugen.

2.2.4 Beispiel-Plugin: „inverter“

Am besten wir sehen uns diese Sache an einem konkreten Beispiel an. Wir versuchen uns an einem einfachen Plugin, das ein 8-Bit-Grauwertbild in-

vertieren, also ein Positiv in ein Negativ verwandeln soll. Das Invertieren der Intensität ist eine typische Punktoperation, wie wir sie in Kap. 4 im Detail behandeln. Unser Bild hat 8-Bit-Grauwerte im Bereich von 0 bis zum Maximalwert 255 sowie eine Breite und Höhe von M bzw. N Pixel. Die Operation ist sehr einfach: Der Wert jedes einzelnen Bildelements $I(u, v)$ wird umgerechnet in einen neuen Pixelwert

$$I(u, v) \leftarrow 255 - I(u, v),$$

der den ursprünglichen Pixelwert ersetzt, und das für alle Bildkoordinaten $u = 0, \dots, M-1$ und $v = 0, \dots, N-1$.

2.2.5 Plugin `My_Inverter_A`

Wir benennen unser erstes Plugin „`My_Inverter_A`“ und das ist sowohl der Name der Java-Klasse wie auch der Name der zugehörigen Quelldatei.¹¹ Die vollständige Auflistung des Java-Codes für dieses Plugin findet sich in Prog. 2.1. Das Unterstreichungszeichen „_“ im Namen ist wichtig, da ImageJ in diesem Fall die Klasse beim Startup automatisch in das Plugin-Menü aufnimmt. Das Programm enthält nach dem Importieren der notwendigen Java-Pakete die Definition einer einzigen Klasse `My_Inverter_A`. Diese „implementiert“ das in ImageJ definierte *Interface* `PlugInFilter` und muss daher zumindest die Methoden `setup()` und `run()` enthalten.

Die `setup()`-Methode

Vor der eigentlichen Ausführung des Plugin, also vor dem Aufruf der `run()`-Methode, wird die `setup()`-Methode vom ImageJ-Kernsystem aufgerufen, um Informationen über das Plugin zu erhalten. In unserem Beispiel wird nur der Wert `DOES_8G` (eine 32-Bit `int`-Konstante, die im Interface `PlugInFilter` definiert ist) zurückgegeben, was anzeigt, dass dieses Plugin 8-Bit-Grauwertbilder (8G) verarbeiten kann. Die Parameter `args` und im der `setup()`-Methode werden in diesem Beispiel nicht benutzt (s. auch Aufg. 2.7).

Die `run()`-Methode

Wie bereits erwähnt, wird der `run()`-Methode ein Objekt `ip` vom Typ `ImageProcessor` übergeben, in dem das zu bearbeitende Bild und zugehörige Informationen enthalten sind. Zunächst werden durch Anwendung der Methoden `getWidth()` und `getHeight()` auf `ip` die Dimensionen des aktuellen Bilds abgefragt. Dann werden alle Bildkoordinaten in zwei geschachtelten `for`-Schleifen mit den Zählvariablen `u` und `v` horizontal bzw. vertikal durchlaufen. Für den eigentlichen Zugriff auf die Bilddaten werden zwei weitere Methoden der Klasse `ImageProcessor` verwendet:

¹¹ `My_Inverter_A.java`

Programm 2.1

ImageJ-Plugin zum Invertieren von 8-Bit-Grauwertbildern. Das Plugin implementiert das Interface `PlugInFilter` und enthält die dafür erforderlichen Methoden `setup()` und `run()`. Das eigentliche Bild wird der `run()`-Methode des Plugins mit dem Parameter `ip` als Objekt vom Typ `ImageProcessor` übergeben. ImageJ nimmt an, dass das Plugin das übergebene Bild verändert und aktualisiert nach Ausführung des Plugins die Bildanzeige automatisch. Programm Prog. 2.2 zeigt eine alternative Implementierung basierend auf dem `PlugIn`-Interface.

```

1 import ij.ImagePlus;
2 import ij.plugin.filter.PlugInFilter;
3 import ij.process.ImageProcessor;
4
5 public class My_Inverter_A implements PlugInFilter {
6
7     public int setup(String args, ImagePlus im) {
8         return DOES_8G; // this plugin accepts 8-bit grayscale images
9     }
10
11     public void run(ImageProcessor ip) {
12         int M = ip.getWidth();
13         int N = ip.getHeight();
14
15         // iterate over all image coordinates (u,v)
16         for (int u = 0; u < M; u++) {
17             for (int v = 0; v < N; v++) {
18                 int p = ip.getPixel(u, v);
19                 ip.putPixel(u, v, 255 - p);
20             }
21         }
22     }
23
24 }

```

`int getPixel (int u, int v)`

Liefert den Wert des Bildelements an der Position (u, v) oder null, wenn (u, v) außerhalb der Bildgrenzen liegt.

`void putPixel (int u, int y, int v)`

Setzt das Bildelement an der Position (u, v) auf den neuen Wert a . Die Anwendung hat keine Auswirkungen auf das Bild, wenn (u, v) außerhalb der Bildgrenzen liegt.

Beide Methoden überprüfen die übergebenen Koordinaten und Pixelwerte genau, um allfällige Fehler zu vermeiden, und sind dadurch zwar „idiotensicher“ aber naturgemäß auch relativ langsam. Wenn man sicher sein kann, dass alle besuchten Bildelemente innerhalb der Bildgrenzen liegen und die eingefügten Pixelwerte garantiert im zulässigen Wertebereich des verwendeten Bildtyps liegen (wie in Prog. 2.1), dann kann man sie durch schnellere Zugriffsmethoden ersetzen, wie `get()` und `set()` anstelle von `getPixel()` und `putPixel()` (siehe Prog. 2.2). Am effizientesten ist es, überhaupt keine Methoden zum Lesen bzw. Schreiben zu verwenden, sondern direkt auf die Elemente des Pixel-Arrays zuzugreifen. Details zu diesen und anderen Methoden finden sich online in der API-Dokumentation zu ImageJ.¹²

¹² <http://rsbweb.nih.gov/ij/developer/api/index.html>

Programm 2.2 zeigt eine alternative Implementierung des Inverter-Plugin basierend auf dem `PlugIn`-Interface, das grundsätzlich kein aktuelles Bild erfordert und lediglich eine `run()` Methode vorsieht. In diesem Fall wird das aktuelle Bild nicht direkt übergeben, sondern mit der (statischen) Methode `IJ.getImage()` ermittelt. Falls aktuell kein Bild verfügbar ist, wird durch `getImage()` automatisch eine Fehlermeldung angezeigt und die Ausführung des Plugin abgebrochen. Der Test auf den passenden Bildtyp (`GRAY8`) und die zugehörige Fehlerbehandlung muss hier allerdings explizit durchgeführt werden. Über den (hier nicht verwendeten) Parameter `args` kann der `run()` Methode eine beliebige Zeichenkette mit zusätzlichen Informationen zur Steuerung des Plugin übergeben werden.

2.2.7 PlugIn oder PlugInFilter?

Die Verwendung von `PlugIn` oder `PlugInFilter` ist weitgehend Geschmacksache, zumal beide Varianten Vor- und Nachteile aufweisen. Für Plugins, die kein vorhandenes Bild benötigen, sondern Bilder aufnehmen, erzeugen, oder sonstige Operationen durchführen, ist das `PlugIn`-Interface die natürliche (wenn auch nicht einzige) Wahl. Hingegen sollte das Interface `PlugInFilter` insbesondere dann verwendet werden, wenn durch das Plugin ein vorhandenes Bild bearbeitet, d. h. modifiziert wird. Aus Gründen der Einfachheit und Konsistenz wird `PlugInFilter` durchgehend als Basis für praktisch alle in diesem Buch beschriebenen Plugins verwendet.

Editieren, Übersetzen und Ausführen des Plugins

Die Java-Datei des Plugins muss im Verzeichnis `<ij>/plugins`¹³ von ImageJ oder in einem Unterverzeichnis davon abgelegt werden. Neue Plugin-Dateien können über das `Plugins▷New...` von ImageJ angelegt werden. Zum Editieren verfügt ImageJ über einen eingebauten Editor unter `Plugins▷Edit...`, der jedoch für das ernsthafte Programmieren kaum Unterstützung bietet und daher wenig geeignet ist. Besser ist es, dafür einen modernen Editor oder gleich eine komplette Java-Programmierungsumgebung zu verwenden (z. B. *Eclipse*¹⁴, *NetBeans*¹⁵ oder *JBuilder*¹⁶).

Für die Übersetzung von Plugins (in Java-Bytecode) ist in ImageJ ein eigener Java-Compiler als Teil der Laufzeitumgebung verfügbar. Zur Übersetzung und nachfolgenden Ausführung verwendet man einfach das Menü

¹³ `<ij>` steht für das Verzeichnis, in dem ImageJ selbst installiert ist.

¹⁴ www.eclipse.org

¹⁵ www.netbeans.org

¹⁶ <http://www.embarcadero.com/products/jbuilder>

Programm 2.2

Alternative Implementierung des Inverter-Plugins auf Basis des `PlugIn` Interface). Im Unterschied zu Prog. 2.1 enthält dieses Plugin nur *eine* Methode `run()`. Das aktuelle Bild (`im`) wird hier zunächst als `ImagePlus`-Objekt mithilfe der Methode `IJ.getImage()` ermittelt, anschließend auf den passenden Bildtyp überprüft und eine Referenz auf den zugehörigen `ImageProcessor` (`ip`) entnommen. Der `args`-Parameter der `run()`-Methode bleibt in diesem Fall unberücksichtigt, der Rest ist identisch zu Prog. 2.1, außer dass die (etwas schnelleren) Zugriffsmethoden `get()` und `set()` verwendet sind. Man beachte, dass in diesem Fall die Darstellung des modifizierten Bilds am Ende nicht automatisch sondern durch den expliziten Aufruf `updateAndDraw()` erfolgt.

```

1 import ij.IJ;
2 import ij.ImagePlus;
3 import ij.plugin.PlugIn;
4 import ij.process.ImageProcessor;
5
6 public class My_Inverter_B implements PlugIn {
7
8     public void run(String args) {
9         ImagePlus im = IJ.getImage();
10
11         if (im.getType() != ImagePlus.GRAY8) {
12             IJ.error("8-bit grayscale image required");
13             return;
14         }
15
16         ImageProcessor ip = im.getProcessor();
17         int M = ip.getWidth();
18         int N = ip.getHeight();
19
20         // iterate over all image coordinates (u,v)
21         for (int u = 0; u < M; u++) {
22             for (int v = 0; v < N; v++) {
23                 int p = ip.get(u, v);
24                 ip.set(u, v, 255 - p);
25             }
26         }
27
28         im.updateAndDraw(); // redraw the modified image
29     }
30 }

```

Plugins▷Compile and Run...,

wobei etwaige Fehlermeldungen über ein eigenes Textfenster angezeigt werden. Sobald das Plugin in den entsprechenden `.class`-File übersetzt ist, wird diese Datei automatisch geladen und das Plugin auf das aktuelle Bild angewandt. Eine Fehlermeldung zeigt an, falls keine Bilder geöffnet sind oder das aktuelle Bild nicht den Möglichkeiten des Plugins entspricht.

Im Verzeichnis `<ij>/plugins/` angelegte, korrekt benannte Plugins werden beim Starten von ImageJ automatisch als Eintrag im **Plugins**-Menü installiert und brauchen dann vor der Ausführung natürlich nicht mehr übersetzt zu werden. Plugin-Einträge können manuell mit

Plugins▷Shortcuts▷Install Plugin..

auch an anderen Stellen des Menübaums platziert werden.

Digitale Bildverarbeitung

Eine algorithmische Einführung mit Java

Burger, W.; Burge, M.J.

2015, XXIII, 803 S. 375 Abb. in Farbe., Hardcover

ISBN: 978-3-642-04603-2