

# 2

## *ImageJ*

Until a few years ago, the image-processing community was a relatively small group of people who either had access to expensive commercial image-processing tools or, out of necessity, developed their own software packages. Usually such home-brew environments started out with small software components for loading and storing images from and to disk files. This was not always easy because often one had to deal with poorly documented or even proprietary file formats. An obvious (and frequent) solution was to simply design a *new* image file format from scratch, usually optimized for a particular field, application, or even a single project, which naturally led to a myriad of different file formats, many of which did not survive and are forgotten today [30,32]. Nevertheless, writing software for *converting* between all these file formats in the 1980s and early 1990s was an important business that occupied many people. Displaying images on computer screens was similarly difficult, because there was only marginal support by operating systems, APIs, and display hardware, and capturing images or videos into a computer was close to impossible on common hardware. It thus may have taken many weeks or even months before one could do just elementary things with images on a computer and finally do some serious image processing.

Fortunately, the situation is much different today. Only a few common image file formats have survived (see also Sec. 1.3), which are readily handled by many existing tools and software libraries. Most standard APIs for C/C++, Java, and other popular programming languages already come with at least some basic support for working with images and other types of media data. While there is still much development work going on at this level, it makes our

job a lot easier and, in particular, allows us to focus on the more interesting aspects of digital imaging.

## 2.1 Image Manipulation and Processing

Traditionally, software for digital imaging has been targeted at either *manipulating* or *processing* images, either for practitioners and designers or software programmers, with quite different requirements.

Software packages for *manipulating* images, such as Adobe Photoshop, Corel Paint and others, usually offer a convenient user interface and a large number of readily available functions and tools for working with images interactively. Sometimes it is possible to extend the standard functionality by writing scripts or adding self-programmed components. For example, Adobe provides a special API<sup>1</sup> for programming Photoshop “plugins” in C++, though this is a nontrivial task and certainly too complex for nonprogrammers.

In contrast to the category of tools above, digital image *processing* software primarily aims at the requirements of algorithm and software developers, scientists, and engineers working with images, where interactivity and ease of use are not the main concerns. Instead, these environments mostly offer comprehensive and well-documented software libraries that facilitate the implementation of new image-processing algorithms, prototypes and working applications. Popular examples are Khoros/VisiQuest,<sup>2</sup> IDL,<sup>3</sup> MatLab,<sup>4</sup> and ImageMagick,<sup>5</sup> among many others. In addition to the support for conventional programming (typically with C/C++), many of these systems provide dedicated scripting languages or visual programming aides that can be used to construct even highly complex processes in a convenient and safe fashion.

In practice, image manipulation and image processing are of course closely related. Although Photoshop, for example, is aimed at image manipulation by nonprogrammers, the software itself implements many traditional image-processing algorithms. The same is true for many Web applications using server-side image processing, such as those based on ImageMagick. Thus image processing is really at the base of any image manipulation software and certainly not an entirely different category.

---

<sup>1</sup> [www.adobe.com/products/photoshop/](http://www.adobe.com/products/photoshop/).

<sup>2</sup> [www.accusoft.com/imaging/visiquest/](http://www.accusoft.com/imaging/visiquest/).

<sup>3</sup> [www.rsinc.com/idl/](http://www.rsinc.com/idl/).

<sup>4</sup> [www.mathworks.com](http://www.mathworks.com).

<sup>5</sup> [www.imagemagick.org](http://www.imagemagick.org).

## 2.2 ImageJ Overview

ImageJ, the software that is used for this book, is a combination of both worlds discussed above. It offers a set of ready-made tools for viewing and interactive manipulation of images but can also be extended easily by writing new software components in a “real” programming language. ImageJ is implemented entirely in Java and is thus largely platform-independent, running without modification under Windows, MacOS, or Linux. Java’s dynamic execution model allows new modules (“plugins”) to be written as independent pieces of Java code that can be compiled, loaded, and executed “on the fly” in the running system without the need to even restart ImageJ. This quick turnaround makes ImageJ an ideal platform for developing and testing new image-processing techniques and algorithms. Since Java has become extremely popular as a first programming language in many engineering curricula, it is usually quite easy for students to get started in ImageJ without spending much time to learn another programming language. Also, ImageJ is freely available, so students, instructors, and practitioners can install and use the software legally and without license charges on any computer. ImageJ is thus an ideal platform for education and self-training in digital image processing but is also in regular use for serious research and application development at many laboratories around the world, particularly in biological and medical imaging.

ImageJ was (and still *is*) developed by Wayne Rasband [34] at the U.S. National Institutes of Health (NIH), originally as a substitute for its predecessor, NIH-Image, which was only available for the Apple Macintosh platform. The current version of ImageJ, updates, documentation, the complete source code, test images, and a continuously growing collection of third-party plugins can be downloaded from the ImageJ Website.<sup>6</sup> Installation is simple, with detailed instructions available online, in Werner Bailer’s programming tutorial [3], and in the authors’ *ImageJ Short Reference* [5].

To give a structured orientation on ImageJ, this short reference<sup>7</sup> is grouped into different task areas and concentrates on the key functionalities. Some specific rarely used functions were deliberately omitted, but they can of course be found in the ImageJ documentation and the (online) source code.

### 2.2.1 Key Features

As a pure Java application, ImageJ should run on any computer for which a current Java runtime environment (JRE) exists. ImageJ comes with its own Java runtime, so Java need not be installed separately on the computer. Under

---

<sup>6</sup> <http://rsb.info.nih.gov/ij/>.

<sup>7</sup> Available at [www.imagingbook.com](http://www.imagingbook.com).



**Figure 2.1** Wayne Rasband (right), author of ImageJ, at the 1st ImageJ Conference 2006 (picture courtesy of Marc Seil, CRP Henri Tudor, Luxembourg).

the usual restrictions, ImageJ can be run as a Java “applet” within a Web browser, though it is mostly used as a stand-alone application. It is sometimes also used on the server side in the context of Java-based Web applications (see [3] for details). In summary, the key features of ImageJ are:

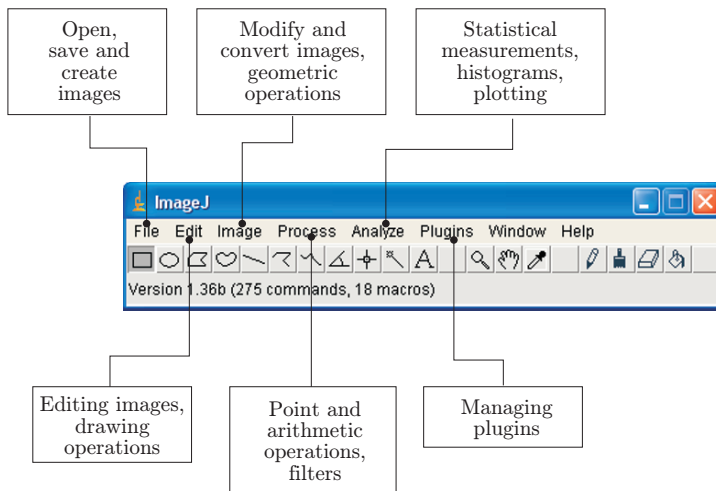
- A set of ready-to-use, interactive tools for creating, visualizing, editing, processing, analyzing, loading, and storing images, with support for several common file formats. ImageJ also provides “deep” 16-bit integer images, 32-bit floating-point images, and image sequences (“stacks”).
- A simple plugin mechanism for extending the core functionality of ImageJ by writing (usually small) pieces of Java code. All coding examples shown in this book are based on such plugins.
- A macro language and the corresponding interpreter, which make it easy to implement larger processing blocks by combining existing functions without any knowledge of Java. Macros are not discussed in this book, but details can be found in ImageJ’s online documentation.<sup>8</sup>

### 2.2.2 Interactive Tools

When ImageJ starts up, it first opens its main window (Fig. 2.2), which includes the following menu entries:

- **File**: opening, saving and creating new images.
- **Edit**: editing and drawing in images.
- **Image**: modifying and converting images, geometric operations.

<sup>8</sup> <http://rsb.info.nih.gov/ij/developer/macro/macros.html>.



**Figure 2.2** ImageJ main window (under Windows XP).

- **Process:** image processing, including point operations, filters, and arithmetic operations between multiple images.
- **Analyze:** statistical measurements on image data, histograms, and special display formats.
- **Plugin:** editing, compiling, executing, and managing user-defined plugins.

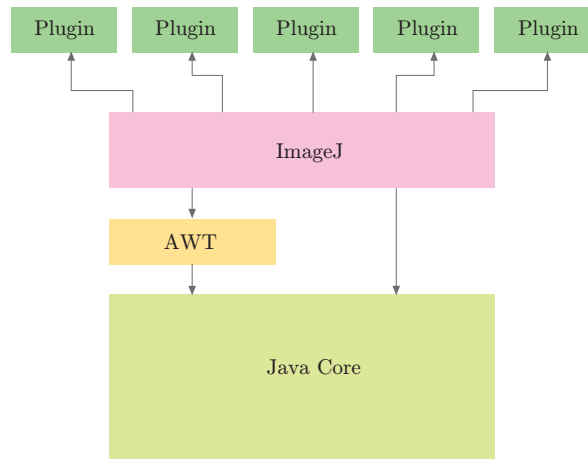
The current version of ImageJ can open images in several common formats, including TIFF (uncompressed only), JPEG, GIF, PNG, and BMP, as well as the formats DICOM<sup>9</sup> and FITS,<sup>10</sup> which are popular in medical and astronomical image processing, respectively. As is common in most image-editing programs, all interactive operations are applied to the currently *active* image, i.e., the image most recently selected by the user. ImageJ provides a simple (single-step) “undo” mechanism for most operations, which can also revert modifications effected by user-defined plugins.

### 2.2.3 ImageJ Plugins

Plugins are small Java modules for extending the functionality of ImageJ by using a simple standardized interface (Fig. 2.3). Plugins can be created, edited, compiled, invoked, and organized through the Plugin menu in ImageJ’s main window (Fig. 2.2). Plugins can be grouped to improve modularity, and plugin

<sup>9</sup> Digital Imaging and Communications in Medicine.

<sup>10</sup> Flexible Image Transport System.



**Figure 2.3** ImageJ software structure (simplified). ImageJ is based on the Java core system and depends in particular upon Java’s Advanced Windowing Toolkit (AWT) for the implementation of the user interface and the presentation of image data. Plugins are small Java classes that extend the functionality of the basic ImageJ system.

commands can be arbitrarily placed inside the main menu structure. Also, many of ImageJ’s built-in functions are actually implemented as plugins themselves.

Technically speaking, plugins are Java classes that implement a particular interface specification defined by ImageJ. There are two different kinds of plugins:

- **PlugIn**: requires no image to be open to start a plugin.
- **PlugInFilter**: the currently active image is passed to the plugin when started.

Throughout the examples in this book, we almost exclusively use plugins of the second type (**PlugInFilter**) for implementing image-processing operations. The interface specification requires that any plugin of type **PlugInFilter** must at least implement two methods, `setup()` and `run()`, with the following signatures:

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

When the plugin is started, ImageJ calls this method first to verify that the capabilities of this plugin match the target image. `setup()` returns a vector of binary flags (packaged as a 32-bit `int` value) that describes the plugin’s properties.

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

This method does the actual work for this plugin. It is passed a single argument *ip*, an object of type `ImageProcessor`, which contains the image to be processed and all relevant information about it. The `run()` method returns no result value (`void`) but may modify the passed image and create new images.

### 2.2.4 A First Example: Inverting an Image

Let us look at a real example to quickly illustrate this mechanism. The task of our first plugin is to invert any 8-bit grayscale image to turn a positive image into a negative. As we shall see later, inverting the intensity of an image is a typical *point operation*, which is discussed in detail in Chapter 4. In ImageJ, 8-bit grayscale images have pixel values ranging from 0 (black) to 255 (white), and we assume that the width and height of the image are  $M$  and  $N$ , respectively. The operation is very simple: the value of each image pixel  $I(u, v)$  is replaced by its inverted value,

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

for all image coordinates  $(u, v)$ , with  $u = 0 \dots M-1$  and  $v = 0 \dots N-1$ .

*The plugin class: `My_Inverter`*

We decide to name our first plugin “`My_Inverter`”, which is both the name of the Java class and the name of the source file that contains it (Prog. 2.1). The underscore character (“\_”) in the name causes ImageJ to recognize this class as a plugin and to insert it automatically into the menu list at startup. The Java source code in file `My_Inverter.java` contains a few `import` statements, followed by the definition of the class `My_Inverter`, which implements the `PlugInFilter` interface (because it will be applied to an existing image).

*The `setup()` method*

When a plugin of type `PlugInFilter` is executed, ImageJ first invokes its `setup()` method to obtain information about the plugin itself. In this example, `setup()` only returns the value `DOES_8G` (a static `int` constant specified by the `PlugInFilter` interface), indicating that this plugin can handle 8-bit grayscale images (Prog. 2.1, line 8). The parameters `arg` and `im` of the `setup()` method are not used in this case (see also Exercise 2.4).

```

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

```

**Program 2.1** ImageJ plugin for inverting 8-bit grayscale images (file `My_Inverter.java`).

### The `run()` method

As mentioned above, the `run()` method of a `PlugInFilter` plugin receives an object (`ip`) of type `ImageProcessor`, which contains the image to be processed and all relevant information about it. First, we use the `ImageProcessor` methods `getWidth()` and `getHeight()` to query the size of the image referenced by `ip` (lines 12–13). Then we use two nested `for` loops (with loop variables `u`, `v` for the horizontal and vertical coordinates, respectively) to iterate over all image pixels (lines 16–17). For reading and writing the pixel values, we use two additional methods of the class `ImageProcessor`:

`int getPixel (int u, int v)`

Returns the pixel value at position (*u*, *v*) or zero if (*u*, *v*) is outside the image bounds.

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

Sets the pixel value at position (*u*, *v*) to the new value *a*. Does nothing if (*u*, *v*) is outside the image bounds.

Details on these and other methods can be found in the ImageJ reference [5] (available online at the books support site).



If we are sure that no coordinates outside the image bounds are ever accessed (as in `My_Inverter` in Prog. 2.1) and the inserted pixel values are guaranteed not to exceed the image processor's range, we can use the slightly faster methods `get()` and `set()` in place of `getPixel()` and `putPixel()`, respectively. The most efficient way to process the image is to avoid read/write methods altogether and directly access the elements of the corresponding pixel array.<sup>11</sup>

### *Editing, compiling, and executing the plugin*

The source code of our plugin should be stored in a file

`My_Inverter.java`

located within `<ij>/plugins/`<sup>12</sup> or an immediate subdirectory. New plugin files can be created with ImageJ's `Plugins→New...` menu. ImageJ even provides a built-in Java editor for writing plugins, which is available through the `Plugins→Edit...` menu but unfortunately is of little use for serious programming. A better alternative is to use a modern editor or a professional Java programming environment, such as Eclipse,<sup>13</sup> NetBeans,<sup>14</sup> or JBuilder,<sup>15</sup> all of which are freely available.

For compiling plugins (to Java bytecode), ImageJ comes with its own Java compiler as part of its runtime environment.<sup>16</sup> To compile and execute the new plugin, simply use the menu

`Plugins→Compile and Run...`

Compilation errors are displayed in a separate log window. Once the plugin is compiled, the corresponding `.class` file is automatically loaded and the plugin is applied to the currently active image. An error message is displayed if no images are open or if the current image cannot be handled by that plugin.

At startup, ImageJ automatically loads all correctly named plugins found in the `<ij>/plugins/` directory (or any immediate subdirectory) and installs them in its `Plugins` menu. These plugins can be executed immediately without any recompilation. References to plugins can also be placed manually with the

`Plugins→Shortcuts→Install Plugin...`

---

<sup>11</sup> See Sec. 7.6 of the ImageJ Short Reference [5].

<sup>12</sup> `<ij>` denotes ImageJ's installation directory, and `<ij>/plugins/` is the default plugins path, which can be set to any other directory.

<sup>13</sup> [www.eclipse.org](http://www.eclipse.org).

<sup>14</sup> [www.netbeans.org](http://www.netbeans.org).

<sup>15</sup> [www.borland.com](http://www.borland.com).

<sup>16</sup> Currently only for Windows; for MacOS and Linux, consult the ImageJ installation manual.

command at any other position in the ImageJ menu tree. Sequences of plugin calls and other ImageJ commands may be recorded as macro programs with `Plugins→Macros→Record`.

### *Displaying and “undoing” results*

Our first plugin in Prog. 2.1 did not create a new image but “destructively” modified the target image. This is not always the case, but plugins can also create additional images or compute only statistics, without modifying the original image at all. It may be surprising, though, that our plugin contains no commands for displaying the modified image. This is done automatically by ImageJ whenever it can be assumed that the image passed to a plugin was modified.<sup>17</sup> In addition, ImageJ automatically makes a copy (“snapshot”) of the image before passing it to the `run()` method of a `PlugInFilter`-type plugin. This feature makes it possible to restore the original image (with the `Edit→Undo` menu) after the plugin has finished without any explicit precautions in the plugin code.

## 2.3 Additional Information on ImageJ and Java

In the following chapters, we mostly use concrete plugins and Java code to describe algorithms and data structures. This not only makes these examples immediately applicable, but they should also help in acquiring additional skills for using ImageJ in a step-by-step fashion. To keep the text compact, we often describe only the `run()` method of a particular plugin and additional class and method definitions, if they are relevant in the given context. The complete source code for these examples can of course be downloaded from the book’s supporting Website.<sup>18</sup>

### 2.3.1 Resources for ImageJ

The short reference in [5] contains an overview of ImageJ’s main capabilities and a short description of its key classes, interfaces, and methods. The complete and most current API reference, including source code, tutorials, and many example plugins, can be found on the official ImageJ Website. Another great source for any serious plugin programming is the tutorial by Werner Bailer [3].

---

<sup>17</sup> No automatic redisplay occurs if the `NO_CHANGES` flag is set in the return value of the plugin’s `setup()` method.

<sup>18</sup> [www.imagingbook.com](http://www.imagingbook.com).

### 2.3.2 Programming with Java

While this book does not require extensive Java skills from its readers, some elementary knowledge is essential for understanding or extending the given examples. There is a huge and still-growing number of introductory textbooks on Java, such as [2, 11, 13] and many others. For readers with programming experience who have not worked with Java before, we particularly recommend some of the tutorials on Sun's Java Website.<sup>19</sup> Also, in Appendix B of this book, readers will find a small compilation of specific Java topics that cause frequent problems or programming errors.

## 2.4 Exercises

### Exercise 2.1

Install the current version of ImageJ on your computer and make yourself familiar with the built-in functions (open, convert, edit, and save images).

### Exercise 2.2

Write a new ImageJ plugin that reflects a grayscale image horizontally (or vertically) using `My_Inverter.java` (Prog. 2.1) as a template. Test your new plugin with appropriate images of different sizes (odd, even, extremely small) and inspect the results carefully.

### Exercise 2.3

Create an ImageJ plugin for 8-bit grayscale images of arbitrary size that paints a white frame (with pixel value 255) 10 pixels wide *into* the image (without increasing its size). Make sure that this plugin also works for very small images.

### Exercise 2.4

Write a new ImageJ plugin that shifts an 8-bit grayscale image horizontally and cyclically until the original state is reached again. To display the modified image after each shift, a reference to the corresponding `ImagePlus` object is required (`ImageProcessor` has no display methods). The `ImagePlus` object is only accessible to the plugin's `setup()` method, which is automatically called before the `run()` method. Modify the definition in Prog. 2.1 to keep a reference and to redraw the `ImagePlus` object as follows:

```
1 public class XY_plugin implements PlugInFilter {
2
3     ImagePlus im;    // instance variable of this plugin object
4
5     public int setup(String arg, ImagePlus im) {
6         this.im = im;    // keep a reference to the image im
7     }
8 }
```

<sup>19</sup> <http://java.sun.com/docs/books/tutorial/>.

```
7     return DOES_8G;
8 }
9
10 public void run(ImageProcessor ip) {
11     ... // use ip to modify the image
12     im.updateAndDraw(); // use im to redisplay the image
13     ...
14 }
15
16 } // end of class XY_plugin
```



<http://www.springer.com/978-1-84800-190-9>

Principles of Digital Image Processing

Fundamental Techniques

Burger, W.; Burge, M.J.

2009, XIV, 261 p. With online files/update., Softcover

ISBN: 978-1-84800-190-9