EASY JAVA PROGRAMMING FOR TEACHING IMAGE- PROCESSING

Daniel Sage and Michael Unser
Swiss Federal Institute of Technology, Lausanne EPFL, Switzerland
Biomedical Imaging Group, daniel.sage@epfl.ch, michael.unser@epfl.ch

ABSTRACT

We have designed a series of computer sessions build around **ImageJ** (a public-domain software for image analysis), as a practical complement to a two-semester course in image processing. The students are challenged with simple practical imaging problems as they acquire hands-on practice by experimenting with image-processing operators. In the process, they also learn how to program standard image-processing algorithms in Java. This is made possible thanks to a programmer-friendly environment and a software interface that greatly facilitates the developments of plugins for ImageJ. Since our students have generally not acquired programming skills yet (they typically do not even know Java), we use a learning-by-example teaching strategy, with good success.

1. INTRODUCTION

The basic course in image processing (IP), as taught at the Swiss Federal Institute of Technology Lausanne (EPFL), tends to be perceived as rather theoretical. It is indeed a subject that lends itself quite naturally to a rigorous, mathematical treatment. The mathematics are not difficult but the notation can be intimidating because of the multiple sums and indices. On the other hand, image processing is also a very practical discipline; it is extremely motivating for students to see that the formulas are easily translated into algorithms, often with dramatic visual effects.

Since engineering students are often more interested in applications than in pure theory, there was a strong incentive for us to introduce computer laboratories as a complement to the basic lectures in image processing (semester 7 and 8 in micro-engineering at EPFL). Similar efforts in other universities have demonstrated that students gain a lot in their understanding and intuition of the theory if they can experiment with algorithms and visualize the results [1, 2, 3].

More than just applying operators to images, we took the challenge further so that we would have the students write their own image-processing code down to the pixel level. Of course, we also wanted to give them the benefit of a user friendly interface and of a software platform that they could extend to perform sophisticated image processing tasks. Even though a rudimentary knowledge of the Java syntax is required, we should emphasize that this knowledge comes at essentially no effort from the part of the student and that the laboratories require little programming skills. Rather, the students are invited to understand some example code, which they then modify to achieve their goals.

2. HANDS-ON IMAGE PROCESSING

Even when the lectures include visual demonstrations of image-processing algorithms, the students are often passive. The learning of the mathematical concepts can be facilitated with hands-on experimentation. The first level of involvement is to apply the algorithms to real images and to see the results. The second is to take part in the programming itself and to truly experience how formulas translate into algorithms.

2.1 Image processing at the system level

The usual way to get the students involved is to provide a convivial computer environment that allows them to try out different algorithms and to visualize the results [1, 2, 4]. The key points here are the following:

1) Basic manipulations to illustrate and reinforce the theoretical concepts seen in the course. Visual experimentation with different sets of parameters.
2) Use of practical examples to demonstrate image-processing applications. Chaining of simple modules.
3) Need for a user-friendly interface to facilitate interaction with the computer. Production of results that are appealing visually.

Such experimentation can be achieved easily by using standard image-processing software (Khoros or ImageJ).

2.2 Programming image-processing algorithms

Once the students are accustomed to manipulating images, the challenge is to have them to program simple image-processing algorithms. To the best of our knowledge, this kind of classroom experiment in image processing has not been reported on before. The key requirements here are as follows:

4) The best way to understand an algorithm is obviously to code it and to test it. Students should get the opportunity to implement the most representative algorithms.

5) The exercises should be accessible to inexperienced programmers (very basic knowledge in one language, e.g., C). The assignment should concentrate on IP issues alone.
6) The students should not have to worry about data types. The code should be as generic as possible.
7) The programming should be simple and robust. The graphical user interface and input/output task should be provided to avoid spending time on what is non-essential to our purpose (teaching IP).
8) The edition-compilation-execution programming cycle should be short to see immediate effects on the images when modifying the code.

3. THE PROPOSED APPROACH

In accordance with the list of goals above, we have chosen to base our system on an existing software package, **ImageJ** [5], and we have selected Java as programming language. The two keys concepts that are specific to our approach are: 1) the principle of "learning by example", and 2) the use of a "programmer-friendly" software layer that provides the interface with ImageJ.

3.1. Java

There are previous examples of image-processing laboratories based on Matlab, Khoros or C/C++. For our part, we have chosen Java. Some papers claim that Java is a natural language for interactive teaching [6] and that it is ready for image processing [7]. Our main arguments are

1) Java is robust with a good handling of errors and garbage collection; this eliminates the main source of bugs and crashes.
2) Java is syntactically close to C and easy to learn if we provide examples and templates for the methods.
3) Java is reasonably fast: a 3*3 convolution filter takes approximatively 150 ms on a 512*512 pixels image; this means that the students get almost immediate feedback.

Another argument not to be neglected is the "hype" factor; students are attracted by Java, a modern and fashionable language which plays a major role on the Web.

3.2. Learning by example

The students who take the IP laboratories do not necessarily know Java. Hence, we always provide them with an example of a Java method that does an operation that is very similar to the assignment. In particular, we make sure that the example uses the same type of syntax (loops, assignments, mathematical functions) as required for the solution. In addition, we do structure their code by providing empty templates that need to be filled in. This means that a good portion of the assignment can usually be implemented by simple modifications of the example.

3.3. ImageJ and plugins

Our IP system is based on a public-domain software: ImageJ. ImageJ is a general-purpose image-processing program; it is the Java offspring of the well-known NIH Image software. As a result, it can run on any platform with a Java Virtual Machine (Mac, Windows, Linux, etc…). The application and the source are freely available. The author, Wayne Rasband, is with the National Institutes of Health, Bethesda, Maryland, USA.

ImageJ has an open architecture that allows extensibility by addition of Java plugins [8]; these can be compiled and loaded dynamically. We take advantage of this functionality for adding our educational plugins.

Since the programming of ImageJ plugins was not originally meant for novice programmers, we have made this process much more transparent and robust for the student. In particular, we provide under the "Plugins" menu the function templates and their corresponding commands. They typically take the form of a dialog box, enabling the user to change the parameters of his algorithm. The other key component is our "programmer-friendly" software layer (cf. below) which handles the access to the pixel values.

4. INTERFACE LAYER

We have developed a class, named ImageAccess, that provides a high-level and foolproof way of accessing the pixels of an image in ImageJ. The access is independent of the image type. The data retrieved by the methods of the ImageAccess class are always in "double" format. Hence, the image-processing code is written once only in double (best precision); the type conversion is handled automatically.

The typical way to program is to retrieve an image block by using a method that begins with get...(). The block is processed and the result is written in the image using a put...() method. The block can be a single pixel, a row, a column, a 3*3 or a 5*5 neighborhood window.

For locations outside the image, the methods of the ImageAccess class return pixel values by applying mirror boundary conditions. For example, when a student wants to retrieve a 3*3 block of an image centered on (0,0), the interface layer provides the block with mirror conditions. This frees the programmer from having to worry about what happens at the boundaries. It produces simpler code and results in more pleasant results (no frame or border artifacts on the output).

Conceptually, there is a clear advantage in separating the image-processing code (algorithm) from the access of the pixels, since the latter is a technical part that depends on the language, the platform, or the frame grabber. However this is not the approach taken in ImageJ because is has a computational cost associated with it. As a result, the typical image-processing routines in ImageJ are faster than ours but also significantly more complicated. Our method of access leads to an overhead as documented in

Table 1 and Table 2. We consider this as an acceptable price to pay for substantial simplifications in algorithm transcription. Thanks to this layer, an algorithm can be translated into Java almost literally, not to mention that the code is independent of the data type. This is in contrast with ImageJ's own operators which need to be implemented for each data type (e.g., byte, 32 bits).

	Total	Algorithm	Access	Overhead
1) ImageJ routine	35 ms	-	-	-
2) Separable	97 ms	24 ms	72 ms	1 ms
3) Non-separable	597 ms	43 ms	498 ms	56 ms

Table 1. Comparison of the computation time for a 3*3 averaging filter on a 512*512 pixels image (byte) with three implementations: 1) using the standard ImageJ routine, 2) using the ImageAccess class and a separable implementation of the filter and 3) using the ImageAccess class and a non-separable implementation of the filter. The access time includes data conversion, the copy of pixel values and the implementation of the boundary conditions.

	Algorithm	Access
Separable 3*3 averaging	24 ms	72 ms
Separable 5*5 averaging	48 ms	72 ms
Separable 7*7 averaging	54 ms	72 ms

Table 2. Cost of the overhead of the ImageAccess for a 3*3, 5*5, and 7*7 averaging filter on a 512*512 pixels image, compared to the cost of the image-processing algorithm itself.

While designing our interface, we have also considered the possibility of creating stand-alone applets that can be posted and run through the Web. Thus, we have written two versions of the ImageAccess class, one for ImageJ and one for the applets. The second gives access to the pixels of an image within an applet. In this way, we can easily generate image processing demonstration applets at very low development cost. The same image-processing code can therefore be included in a plugin or in an applet. Note that such applets are also used to provide on-line examples for the students.

5. EXAMPLES

In this section, we present two examples that illustrate the ease with which image-processing algorithms can be programmed using our access interface. The code is relatively straightforward; it is essentially a literal translation of the textbook versions of the algorithm.

5.1. Digital filter

We compare two implementations of a digital filter using a non-separable (cf. Listing 1) and a separable algorithm (cf. Listing 2).

Listing 1. Example of a non-separable filtering template (vertical edge detector) given to the students.

```
void filter2D(ImageAccess img) {

   int nx = img.getWidth();
   int ny = img.getHeight();
   int type = img.getType();

   ImageAccess out;
   out = new ImageAccess(nx, ny, type);
   double block[][];
   double p = 0.0;

   for (int x=0; x<nx; x++) {
      for (int y=0; y<ny; y++) {
        block = img.getNeighborhood3x3(x, y);
        p = ( block[2][0] - block[0][0] +
              block[2][1] - block[0][1] +
              block[2][2] - block[0][2] ) / 6.0;
        out.putPixel(x, y, p);
      }
   }
   img.copyPixels(out);
}
```

Listing 2. Example of a separable filtering template (vertical edge detector) given to the students.

```
void filter2D_Separable(ImageAccess img) {

   int nx = img.getWidth();
   int ny = img.getHeight();

   // Row processing
   double rowin[]  = new double[nx];
   double rowout[] = new double[nx];
   for (int y=0; y<ny; y++) {
      img.getRow(y, rowin);
      difference3(rowin, rowout);
      img.putRow(y, rowout);
   }

   // Column processing
   double colin[]  = new double[ny];
   double colout[] = new double[ny];
   for (int x=0; x<nx; x++) {
      img.getColumn(x, colin);
      average3(colin, colout);
      img.putColumn(x, colout);
   }
}
```

The separable implementation offers many advantages in terms of computation time and modularity. The code, which is essentially generic, clearly shows the two loops, the first one scanning the rows and the second one scanning the columns. The only specific parts are the 1D routines `difference3` and `average3` which can be easily modified to yield other separable filters.

In practice, we give these two templates as examples to the students and ask them to program other digital filters such as a horizontal edge detector and a 5*5 moving-average filter (non-separable and separable implementation). By mastering those examples, they get a rather complete exposure to the topic of linear filtering.

5.2. Wavelet transforms

Another interesting example is the implementation of a separable wavelet transform in 2D. The students have 2 hours to program the transform and to apply it to various image-processing tasks (simple coding by zeroing out non-significant coefficients, and noise reduction by soft-thresholding). To simplify their task, we give the templates of separable routines for the analysis part and we only ask them to code the Haar transform and to write the synthesis part from scratch.

As far as the students are concerned, this is perhaps one of the most impressive session they go through. The great majority of them are capable of completing the full assignment (the 1D routines `split` and `merge` for the Haar transform are rather easy—2 liners—and the wavelet synthesis is the same as the analysis, but the other way around).

6. CLASSROOM

A laboratory session, which is two-hours long, is typically devoted to one chapter of the course. The assignment is given one week in advance; it contains a programming part and an experimental part (where the desired results are processed images). The sessions take place in 2 computer rooms with 30 Windows NT machines in each. There are typically 4 teaching assistants per room for technical assistance. At the end of the session, the students submit their results (source code + processed images) on the Web. There is also a make-up session for those who run out of time, but this is an option that is rarely used. The images are checked automatically and the source code is proofread by the assistants. The students get back their corrected assignments the next week.

Our current sessions are
1) Introduction—Applications of the Fourier transform.
2) Filtering and applications.
3) Morphological operators and applications.
4) Edge detection and applications.
5) Recursive filtering and interpolation.
6) Wavelet transforms.
7) Tomographic reconstruction (in progress).
8) Image analysis, segmentation (in progress).

Some subjects of sessions are available on the web: http://bigwww.epfl.ch/teaching/iplabsite/.

Before the introduction of the laboratories, our optional image-processing course normally attracted 20-25 students. With the second edition of the laboratories (term 2000/2001), the number of students went up to 40-55 which is a good indication of success. The feedback from the students has also been extremely positive.

The students value the fact that that our software tools are freely available on the Internet. They can download ImageJ and a Java compiler, and they are ready to work at home.

7. DISCUSSION

The proposed computer laboratories are a perfect complement to a theoretical course on image processing. They provide students with hands-on experience. They also teach them to implement image processing algorithms. Students like to interact with images and become much more interested in the course as soon as they see some practical relevance. The programming experience also raises their curiosity. Their overall reaction has been very positive.

As designers of the laboratories, we are still astonished by the robustness of Java and ImageJ. The system is quite stable and appears to be immune to the student' programming errors—much more so than any other language or system that we have tested before. We have not yet experienced a significant crash.

The laboratory is entirely based on ImageJ which is a professional-level image-processing software. The good news is that the software is freely available on the Web, and is still evolving. Hence, the students can walk away from the course with an image-processing system that is fully operational; they are also given the tools to develop their own plugins.

8. REFERENCES

[1] J.A. Robinson, "A software system for laboratory experiments in image processing," *IEEE Transactions on Education*, vol. 43, no 4, pp. 455-459, November 2000.

[2] L. Balmelli, S. Ayer, Y. Cheneval, M. Vetterli, "A framework for interactive courses and virtual laboratories," *IEEE Second Workshop on Multimedia Signal Processing*, pp. 501-506, 1998.

[3] M. Sonka, E.L. Dove, S.M. Collins, "Image systems engineering education in an electronic classroom," *IEEE Transactions on Education*, vol. 41, no. 4, pp. 263-272, November 1998.

[4] M. Lindenbaum, Y.Y. Zeevi, "Education in image sciences and engineering at the Technion," *Proceedings of the 1996 IEEE International Conference on Image Processing (ICIP'96),* Lausanne, Switzerland, vol. 1, pp. 447-448, 1996.

[5] W. Rasband, ImageJ Web site, *http://rsb.info.nih.gov/ij/.*

[6] Y. Cheneval, L. Balmelli, P. Prandoni, J. Kovacevic, M. Vetterli, "Interactive DSP education using Java," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'98),* Seattle, WA, USA, vol. 3, pp. 1905-1908, May 1998.

[7] D.A. Lyon, *Image processing in Java*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[8] W. Bailer, "Writing ImageJ plugins—A tutorial," *http://rsb.info.nih.gov/ij/docs/index.html,* November 2000.