Teaching Image-Processing Programming in Java

Daniel Sage and Michael Unser

Using "student-friendly" ImageJ as a pedagogical tool.

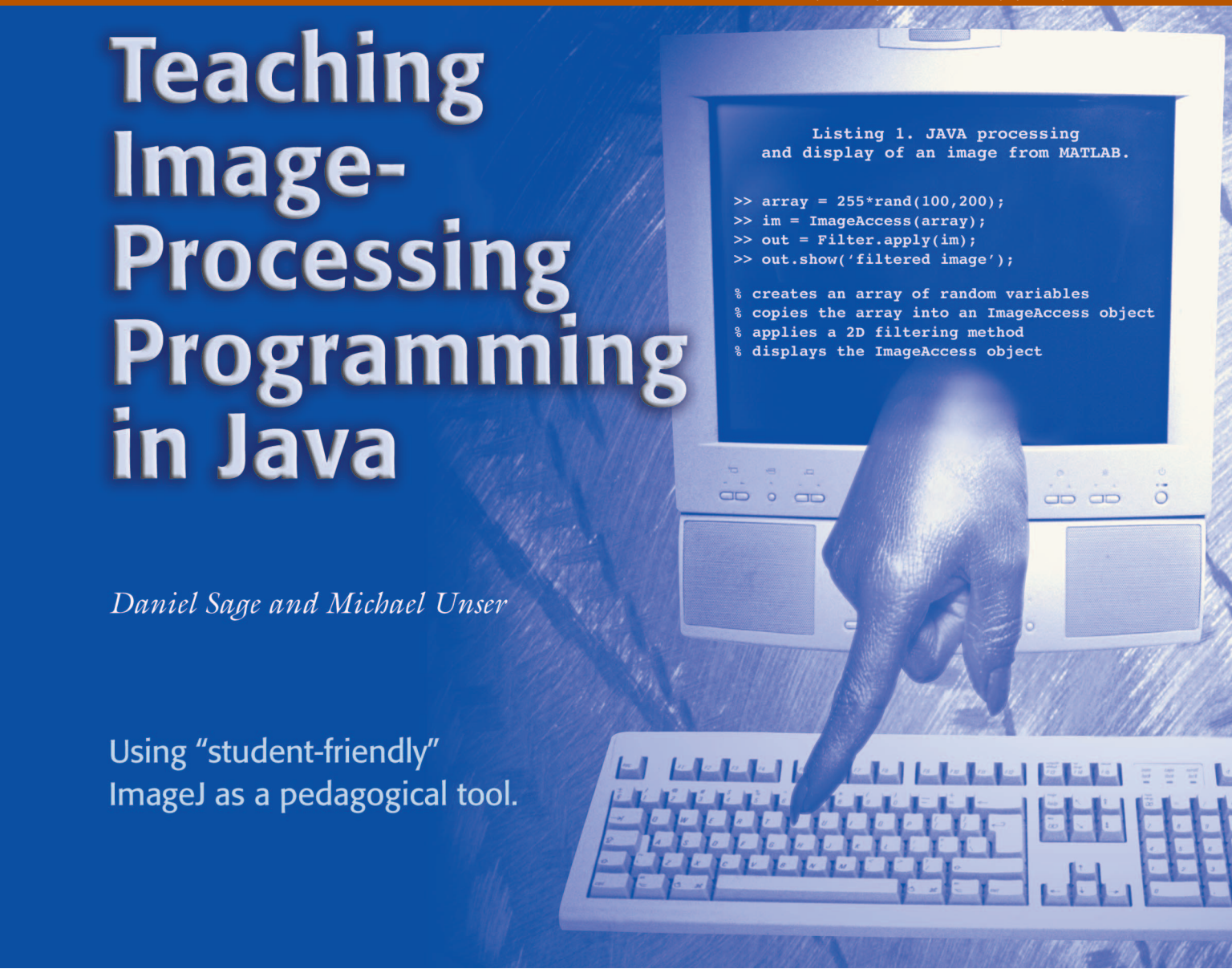


Image processing (IP) can be taught very effectively by complementing the basic lectures with computer laboratories where the participants can actively manipulate and process images. This offering can be made even more attractive by allowing the students to develop their own IP code within a reasonable time frame. After a brief review of existing software packages that can be used for teaching IP, we present a system that we have designed to be as "student friendly" as possible. The software is built around ImageJ, a freely available, full-featured, and user-friendly program for image analysis. The computer sessions are alternated with lectures, typically, a three-hour session at the end of every chapter. The sessions are in the form of assignments that guide the students toward the solution of simple imaging problems. The starting point is typically the understanding and testing of some standard IP algorithm in Java. Next, students are asked to extend the algorithms progressively. This constructive approach is made possible thanks to a programmer-friendly environment and an additional software interface layer that greatly facilitates the developments of plug-ins for ImageJ. Taking into account the fact that our students are not experienced programmers (they typically do not even know Java), we use a "learn by example" teaching strategy, with good success.

Teaching by Doing

Because of the widespread use of imaging, there is an ever-pressing need to train engineers who are proficient with this new technology. This trend is likely to continue as the cost of imaging devices (digital camera, scanners, etc.) keeps declining and as the power of PCs keeps increasing, making sophisticated IP algorithms

1053-588/03/$17.00©2003IEEE

available to a larger base of users and increasing the potential number of applications.

Many universities are meeting this demand by offering a basic course in IP—typically, a two-semester class—that covers all the standard techniques. While IP comes in many gradations, it is typically a topic that is perceived as being rather theoretical. IP 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, IP 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 is a strong incentive for instructors to complement the basic lectures in IP with computer laboratories. A number of initiatives in this area have demonstrated that students gain much in their understanding [1]; they develop in-depth understanding and have a better retention of theoretical material [2]. The students become motivated to study theory if they can experiment with algorithms [3] and visualize the results. Interactive software is generally perceived as a useful tool for complementing textbooks [4], [5].

The purpose of this article is to discuss some of the important issues relating to the use of computer sessions in IP and to present some practical and cost-effective solutions for implementing these ideas in the classroom. In the first part, we discuss the advantages of hands-on experimentation with IP and identify the key points that need to be taken into account to make such an approach successful. We then briefly review the software solutions (both commercial and freeware) that are currently available for teaching IP. In the second part, we get more specific and describe a system (IPLab), which we developed at the Swiss Federal Institute of Technology in Lausanne (EPFL), that is made freely available to the academic community. While our initial motivation was to provide a system where the participants would actively manipulate and process images, we took the challenge further so that we would have the students write their own IP 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 may extend to perform sophisticated IP tasks. Even though a rudimentary knowledge of the Java syntax is required (which can be acquired in a one-hour lesson), 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. There is no need to teach the students how build a complete object-oriented applications [6]. Rather, they are invited to understand some example code, which they then modify to achieve their goals.

Hands-On IP

Even when the lectures include visual demonstrations of IP algorithms, students are often passive. Learning 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.

IP by Direct Image Manipulation

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. The key points here are the following:

▲ basic manipulations to illustrate and reinforce the theoretical concepts treated in the course; visual experimentation with different sets of parameters

▲ use of practical examples to demonstrate IP applications; chaining of simple modules

▲ need for a user-friendly interface to facilitate interaction with the computer; production of results that are visually appealing.

Such experimentation can be achieved easily by using standard IP software.

Programming IP Algorithms

Once the students are accustomed to manipulating images, the challenge is to have them program standard IP algorithms. Our key requirements for this more ambitious level of involvement are as follows.

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

▲ 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. To facilitate programming, we propose a "learning by example" approach: students receive the source code of a basic IP task and are asked to extend and/or complete the algorithm.

▲ The students should not have to worry about data types. The code should be as generic as possible.

▲ 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 nonessential to our purpose (i.e., teaching IP).

▲ The edit-compile-execute programming cycle should be short to see immediate effects on the images when modifying the code.

The two traditional ways to practice IP are through the use of a low-level language (such as C) or a high-level language (such as MATLAB). The low-level language offers the advantage of computational speed, an important factor when dealing with images, but students waste much time with basic input–output operations (reading files, data types, memory allocation, accessing pixels, and displaying images) and rapidly lose

their enthusiasm. A high-level language, on the other hand, offers a rich functionality with a large palette of imaging routines, but tends to hide many important aspects of the algorithm.

Overview of Available Packages

We now give a brief review of the software solutions available to instructors. There has been a substantial effort by members of this community to create didactic tools for teaching IP; a number of systems have been described in the literature and part of them are available on the Internet (cf. the links we are providing in our reference list). Special sessions at conferences and workshops have been organized on this topic [7], [8]; a recent review on computer vision education is also available [9].

The choices of the instructor are usually oriented by the following considerations:

▲ *Scope of the course:* digital signal and IP, mathematical imaging, computer vision, multimedia.

▲ *Background of the students:* electrical engineering or computer science? How proficient are they with programming?

▲ *Level of the course:* undergraduate or graduate level? Note that there are even attempts to introduce IP at the high-school level [10].

▲ *Goals of the interactive tools:* demos for complementing the lectures, practical experimentation with images, or/and programming of algorithms.

▲ *Commercial or freeware:* this is an important consideration both from the ethical and economical point of view.

Most teachers want a plug-and-play system that does not have a steep initial learning curve; they also want an immediate visual feedback of the effect of IP operators [6]. An ideal tool should also be able to solve realistic problems and be relevant for real-world applications [1], [3].

Commercial Packages

Several commercial software packages can be used for setting up IP computer laboratories. The most prominent one is MATLAB of The MathWork Inc. [11], a high-level programming language that is ideally suited for manipulating vectors and matrices. It is widely used in the scientific community for fast prototyping and has been adopted by many universities [12], [13]. MATLAB with its accompanying Image Processing Toolbox is an attractive framework for teaching IP [14]. The interactive nature of MATLAB also encourages "learning by discovery" [15].

Khoros Pro 2001 of Khoral Inc. [16] is an integrated development environment for IP with a special module for teaching known as the "Digital Image Processing Course" [17]. Khoros has earned its place as a pedagogical platform for IP [1], [5] mainly because it offers a visual programming environment coupled with an easy way to link C functions. It also has a large base of users who are willing to exchange their knowledge [18].

IP laboratories have also been developed with other commercial software, including Mathematica [19] of Wolfram Research Inc [20], LabView [21] of National Instruments Corp. [22], and AVS Express [23] of Advanced Visual Systems Inc. [24].

The disadvantage of these commercial products is that they are often expensive and require the sustained availability of a campus-wide license. In many cases, students are not authorized to use the software at home. For these and other reasons, voices have been raised against the use of commercial software, which may conflict with the aims of academic institutions [25]. In addition, many of the packages are platform dependent and the IP operators are often provided as black-box (built-in) routines [23]. Hence, the students do not have access to the core part of the code and cannot visualize intermediate results; this also implies that they cannot easily compare different implementations of an algorithm.

Noncommercial C-Based Solutions

Chronologically, the first group of noncommercial offerings is based on the C language (later on also C++) [3], [2]. In [6], the authors argue that the C language is the closest to being universal—it is the choice of many IP and numerical-analysis libraries. Some libraries have been developed in academia specifically to provide support for IP teaching [26], [27]. The C language gives fast execution code and the students really need to worry about the "hard-core" part of the algorithms. According to [2], students should absolutely know how to handle pointers, which can represent a time-consuming and frustrating task. An interesting class library for IP (CLIP) [6] was developed to handle memory management tasks—with a small overhead time—and to do other technical and common operations through a small user interface which is easy to learn. Of course, there are also other proposals based on less common programming languages such as Python [28], Lisp [29] (which uses an unfamiliar syntax and is less adapted to teaching), and Tcl/Tk (the CVIPTools frameworks [30]).

Noncommercial Java-Based Solutions

Recently, more and more programmers are turning to Java for writing IP software that is platform independent. Java has also other advantages that are discussed in the next section. Below, we give an overview of available Java packages that can be used for pedagogical purposes, even though not all of them were developed with that specific goal in mind. All these packages are freely available on the Internet.

▲ NeatVision provides an image-analysis and software development environment [31]. Many of its algorithms are based on a reference book [32]. It has a nice user interface. It is strongly oriented towards computer vision as opposed to signal processing.

▲ Java Vision Toolkit (JVT) is a software library for

Table 1. Comparison of the computation times for a 3 × 3 filter: built-in ImageJ routines versus ours (ImageAccess). Experimental conditions: 512 × 512 pixels image (byte), Java Virtual Machine JRE 1.1.8, Pentium III/500 MHz.

	Computation time
Built-in ImageJ smooth operator (3 × 3 filtering)	35 ms
Built-in ImageJ convolve operator (3 × 3 filtering)	200 ms
Our separable implementation of 3 × 3 filtering with mirror boundary conditions	150 ms
Our nonseparable implementation of 3 × 3 filtering with mirror boundary conditions	325 ms

machine vision and IP applications [33], [34]. Only a few sessions are available, and the package is rather rudimentary.

▲ ImageJ, a powerful, full-featured IP program developed at the National Institutes of Health [35], is used routinely by biologists worldwide to assist them with the processing and analysis their images. ImageJ also has an extensive library of plug-ins developed by users.

▲ Hypertext Image Processing Reference (HIPR) is a collection of IP resources to illustrate and try-out standard IP operators using interactive applets [36], [37].

▲ Java Image and Graphics Library (JIGL) is an IP library, but without a graphical user interface [38].

▲ IPlab with ImageAccess is a collection of documented IP laboratories (downloadable sessions including handouts for the student and software) that was designed by us to take advantage of the features of ImageJ. An important addition is the "ImageAccess" software layer that greatly facilitates the programming of plug-ins, making it accessible to students.

Many of these IP frameworks may be used equivalently as a foundation for creating interesting IP laboratories. According to us, the availability of a graphical user interface is an important prerequisite to make the software attractive and easy to use for the students. In these packages, especially the most comprehensive ones, the programming environment offered to the user is often rather general and technical. This is the reason why we developed a "student-friendly" intermediate interface layer, called ImageAccess, to be described in the section "ImageAccess: The Interface Layer." Even though it was originally designed for ImageJ, it can be ported to other frameworks as well. Presently, in addition to ImageJ, it supports applets for the Web and can cooperate with the Java Virtual Machine integrated into MATLAB. It is thus also possible to call Java IP routines directly as MATLAB functions.

We believe that making the tools and student sessions available to the community through the Internet not only assists and inspires others to design and share their own classes but also provides the authors with valuable feedback for further enhancements. We will now give a more detailed description of the system that we are promoting and comment on our experience in using these tools for teaching IP.

The IPLAB/ImageJ Combination

Our goal in developing IPLab was to offer an environment where the students could implement the algorithms literally as they are seen in the course [39]. It was also an attempt to combine the advantages of low-level and high-level languages by borrowing the best from both philosophies.

Specifically, we have chosen to base our system on:

▲ Java as the programming language

▲ ImageJ [35], one of the most comprehensive IP freeware available, for a graphical user interface which provides convivial interaction with the full functionality of an IP application

▲ ImageAccess, a "student-friendly" software layer that we have developed to meet the requirements listed earlier; it simplifies and robustifies the access to pixel data without having to worry about technicalities and the interfacing with ImageJ

▲ sample source code to enable students to extend the algorithm progressively and make them learn by example.

Java

We have chosen to develop our pedagogical tool in Java. The main arguments in favor of using this language are: 1) Java is platform neutral, hence well adapted to the diversity of the students community; 2) Java is free; and 3) Java is network ready. This makes it possible to develop remote teaching and virtual laboratories [40], even though this is not the way we work—we prefer to maintain contact with our students.

Some authors claim that Java is a natural language for interactive teaching [41] and that it is ready for signal and image processing applications [42]. Java is an object-oriented language which is desirable for IP programming [43].

For our part, we add the following arguments:

▲ Java is robust with a good handling of errors and garbage collection; this eliminates the main source of bugs and crashes.

▲ Java is syntactically close to C and easy to learn if we provide examples and templates for the methods.

▲ Java is reasonably fast: applying a 3 × 3 convolution filter takes only a fraction of a second on a 512 × 512 pixel 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 that plays a major role on the Web.

ImageJ and Plug-Ins

Our IP system is based on a public-domain software: ImageJ. As a result, it can run on any platform with a Java Virtual Machine (Mac, Windows, and various flavors of Unix). The application and its source are freely available. The author, Wayne Rasband, is with the National Institutes of Health, Bethesda, Maryland, USA [35].

ImageJ has an open architecture that allows extensibility by the addition of Java plug-ins, and we take advantage of this functionality for adding our educational modules. Java also provides a mechanism for loading the plug-ins dynamically without having to restart the application after each modification of the code; this functionality offers a fast and comfortable way to edit-compile-execute a program.

Since the programming of ImageJ plug-ins was not originally meant for novice programmers, we have made this process much more transparent and robust for the student. We provide the function templates and their corresponding commands under the "plug-ins" menu. The templates typically take the form of a dialog box, enabling the user to change the parameters of his algorithm. The other key component is our "student-friendly" software layer called ImageAccess (see below), which greatly facilitates the programming of IP algorithms.

Table 2. Cost of the overhead of the access (due to ImageAccess) compared to the cost of the IP algorithm itself for the separable and the nonseparable implementation of an averaging filter. The access time includes data conversion, the copy of pixel values, and implementation of the boundary conditions. Experimental conditions: 512 × 512 pixels image (byte), Java Virtual Machine JRE 1.1.8, Processor Pentium III/500 MHz.

	Separable implementation		Nonseparable implementation	
Kernel Size	Algorithm	Access	Algorithm	Access
3 × 3 averaging	75 ms	75 ms	75 ms	250 ms
5 × 5 averaging	150 ms	75 ms	175 ms	405 ms
7 × 7 averaging	200 ms	75 ms	250 ms	640 ms
9 × 9 averaging	235 ms	75 ms	375 ms	910 ms
11 × 11 averaging	250 ms	75 ms	500 ms	1,280 ms
13 × 13 averaging	295 ms	75 ms	655 ms	1,690 ms

Sample Source Code to Enable "Learning by Example"

The students who participate in the IP laboratories do not necessarily know Java. Hence, we always provide them with an example of a Java method that does an operation 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 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. A sample two-hour session on morphological filtering (handout + software listing) can be viewed at http://bigwww.epfl.ch/teaching/iplabsite/trial.html; the solution can also be run on the Web.

ImageAccess: The Interface Layer

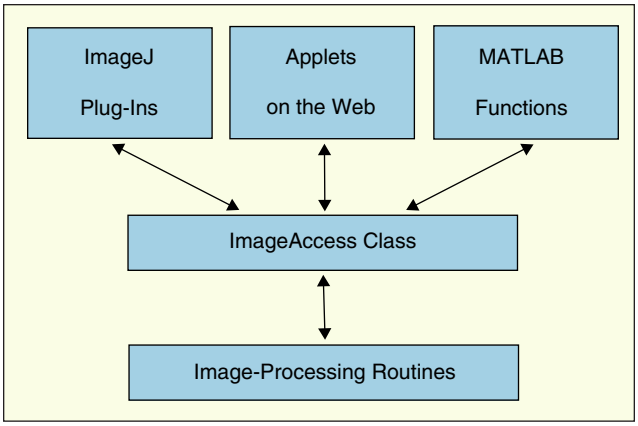
Simplified Image Data Access

The key component of our system is the Java class, ImageAccess, that provides a high-level and foolproof interface that lets students safely manipulate images. We have designed it by applying two well-known principles of software development.

▲ *Abstraction.* For the user, an image is simply an instance of the ImageAccess class. The pixel data is always retrieved and stored in "double" format, independently of the underlying ImageJ image type. In this way, students do not have to worry about rounding, truncation, or conversion of pixel data. Moreover, pixel data can be accessed "anywhere" through the use of consistent mirror symmetric boundary conditions. For example, when a student wants to retrieve a 3 × 3 block of an image centered on the upper left corner (0, 0), the interface layer provides a full block with "outside" pixel values that are correctly extrapolated. This frees the student from having to worry about what happens at the boundaries and results in more pleasant results (no border artifacts in the output). The aim of applying abstraction is to let the source code express the original algorithm more clearly. The full documentation of the class is available at: http://bigwww.epfl.ch/teaching/iplabsite/Docs/index.html.

▲ *Encapsulation.* The fact of working with ImageAccess objects prevents the students from having to worry about implementation details. 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.

Conceptually, there is a clear pedagogical advantage in separating the IP code (algorithm) as much as possible from the access to the pixels. For our purpose, 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 IP routines in ImageJ are faster than ours but also significantly more complicated. Our additional layer leads to an overhead, as illustrated in Tables 1 and 2. Note that in the case of a separable algorithm where rows and columns are processed in succession, the cost of the access is fixed (e.g., 75 ms), irrespective of the type of processing. For nonseparable processing, the access cost is more important: it increases proportionally to the number of pixels in the local neighborhood. We consider the overhead an acceptable price to pay for the substantial simplifications in algorithm transcription. Thanks to this layer, an algorithm can be translated into Java almost literally. This is in contrast with ImageJ's own operators, which need to be implemented for each data type (e.g., byte, 32 bits).



▲ *1. The same IP routines are used to create a plug-in for ImageJ, to interface with MATLAB, or to build a demonstration applet.*

Interfacing with the Web

Programming in Java offers the interesting opportunity to easily port applications to the Web, through the mechanism of applets. To easily create stand-alone applets based on the same IP source code, our interface layer also comes in an "ImageAccess for Applets" flavor, which can be used exactly the same way by the programmer but does not make use of ImageJ internally anymore. In this way, we can easily generate and distribute IP demonstration applets at a very low development cost. The same IP code can therefore be reused in a plug-in or in an applet (see Figure 1). Note that such applets are also used to provide on-line examples for the students (some on-line examples can be found at http://bigwww.epfl.ch/demo/).

Interfacing with MATLAB

Recent versions of MATLAB integrate a Java Virtual Machine. Therefore, it becomes possible to run Java IP routines directly from the MATLAB command window or from a MATLAB function. The level of integration is surprisingly high so that Java objects, such as ImageAccess ones, can be handled in a transparent way. Listing 1 illustrates the call of IPlab commands (here, a two-dimensional (2-D) filter followed by an image display) from MATLAB; the data is transferred through the object "im," which contains a copy of the image array used in MATLAB.

Examples

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

Digital Filter

We compare two implementations of a digital filter using a nonseparable (cf. Listing 2) and a separable algorithm (cf. Listing 3).

The separable implementation offers many advantages in terms of computation time and modularity. The code in Listing 3, which is generic for the most of part, clearly shows the two loops, the first one that scans the rows and the second one that scans the columns. The only specific parts are the one-dimensional (1-D) routines difference3() and average3(), which can be modified easily to yield other separable filters.

Listing 1. JAVA processing and display of an image from MATLAB.

```
>> array = 255*rand(100,200);     % creates an array of random variables
>> im = ImageAccess(array);       % copies the array into an ImageAccess object
>> out = Filter.apply(im);        % applies a 2D filtering method
>> out.show('filtered image');    % displays the ImageAccess object
```

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

```
public ImageAccess filter2D_NonSeparable(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    ImageAccess output = new ImageAccess(nx, ny);
    double block[][] = new double[3][3];
    double value = 0.0;
    for (int x=0; x<nx; x++) {
          for (int y=0; y<ny; y++) {
                input.getNeighborhood(x, y, block);
                value = (block[2][0] - block[0][0] + block[2][1] -
                      block[0][1] + block[2][2] - block[0][2]) / 6.0;
                output.putPixel(x, y, value);
          }
    }
    return output;
}
```

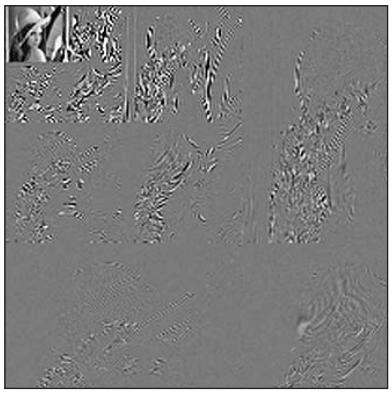
Listing 3. Example of a separable filtering template (vertical edge detector) given to the students.

```
public ImageAccess filter2D_Separable(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    ImageAccess output = new ImageAccess(nx, ny);
    double rowin[]  = new double[nx];
    double rowout[] = new double[nx];
    for (int y=0; y<ny; y++) {
          input.getRow(y, rowin);
          difference3(rowin, rowout);
          output.putRow(y, rowout);
    }
    double colin[]  = new double[ny];
    double colout[] = new double[ny];
    for (int x=0; x<nx; x++) {
          output.getColumn(x, colin);
          average3(colin, colout);
          output.putColumn(x, colout);
    }
    return output;
}

private void average3(double in[], double out[]) {
    int n = in.length;
    out[0] = (2.0 * in[1] + in[0]) / 3.0;
    for (int k=1; k<n-1; k++) {
          out[k] = (in[k-1] + in[k] + in[k+1]) / 3.0;
    }
    out[n-1] = (2.0 * in[n-2] + in[n-1]) / 3.0;
}

private void difference3(double in[], double out[]) {
    int n = in.length;
    out[0] = 0.0;
    for (int k=1; k<n-1; k++) {
          out[k] = (in[k+1] - in[k-1])/2.0;
    }
    out[n-1] = 0.0;
}
```

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 (nonseparable and separable implementation). By mastering those examples, they get a rather complete exposure to the topic of linear filtering.

Wavelet Transforms

Another interesting example is the implementation of a separable wavelet transform in 2-D (c.f. Figure 2). The students have three hours to program the transform and to apply it to various IP tasks (simple coding by zeroing out nonsignificant coefficients and noise reduction by soft-thresholding). To simplify their task, we give the templates of separable routines for the analysis part (Listing 4); we ask them to code the 1-D Haar transform and to write the synthesis part (both 1-D and 2-D) from scratch.



▲ *2. Haar wavelet transform of Lena (three iterations across scale).*

As far as the students are concerned, this is perhaps one of the most impressive sessions they go through. The great majority of them are capable of completing the full assignment; the 1-D routines split_1D and merge_1D for the Haar transform are rather easy—two liners—and the wavelet synthesis is the same as the analysis, but the other way around.

Classroom

A three-hour laboratory session is

Listing 4. Code for the analysis part of the wavelet transform.

The high-level, data-handling routines analysis() and split() are given to the students. Their assignment is to write the code (also shown here) for split_1D() (Haar decomposition) and to implement the 2-D inverse transform completely.

```
public ImageAccess analysis(ImageAccess input, int nbScale) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    ImageAccess output = input.duplicate();
    ImageAccess buffer;
    for (int i=0; i<nbScale; i++) {                 // From fine to coarse loop
          buffer = new ImageAccess(nx, ny);         // Create the buffer
          ouput.getSubImage(0, 0, buffer);          // Get the buffer
          buffer = split(buffer);                   // Split the buffer
          output.putSubImage(0, 0, buffer);         // Put the buffer
          nx = nx / 2;
          ny = ny / 2;
    }
    return output;
}

private ImageAccess split(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    ImageAccess output= new ImageAccess(nx, ny);
    double rowin[]  = new double[nx];
    double rowout[] = new double[nx];
    for (int y=0; y<ny; y++) {
          input.getRow(y, rowin);
          split_1D(rowin, rowout);
          output.putRow(y,rowout);
    }
    double colin[] = new double[ny];
    double colout[] = new double[ny];
    for (int x=0; x<nx; x++) {
          output.getColumn(x, colin);
          split_1D(colin, colout);
          output.putColumn(x,colout);
    }
    return output;
}

private void split_1D(double in[], double out[]) {
    int n = in.length / 2;
    double sqrt2 = Math.sqrt(2.0);
    int k1;
    for (int k=0; k<n; k++) {
          k1 = 2 * k;
          out[k]   = (in[k1] + in[k1+1]) / sqrt2;
          out[k+n] = (in[k1] - in[k1+1]) / sqrt2;
    }
}
```

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. As backup, we usually provide reference versions of the assigned algorithms as executable code (bytecode) to make sure that all students can undertake the experimental part of the assignment under equal conditions; of course, they are greatly encouraged to run their own code and make sure that they get the same results. The sessions take place in two computer rooms with 30 Windows 2000 machines in each. There are typically three teaching

assistants per room for technical assistance. At the end of the session, the students submit their results (source code and processed images) on the Web. The images are checked automatically, and the assistants proofread the source code. The students get back their corrected assignments the next week.

The sessions that are currently available are:

- ▲ Introduction—Understanding of the Fourier transform
- ▲ Digital filtering and applications
- ▲ Morphological operators and applications
- ▲ Edge detection and applications
- ▲ Wavelet transforms
- ▲ Geometric transformation and interpolation
- ▲ Tomography and filtered backprojection
- ▲ Deconvolution
- ▲ Texture.

These session assignments are also available on the Web: http://bigwww.epfl.ch/teaching/iplabsite/. Some examples of results with user interfaces are shown in Figure 3.

Before the introduction of the laboratories, our optional IP course normally attracted 15 to 20 students. With the third edition of the laboratories (term 2002/2003), the number of students went up to 45, a good indication of success. The feedback from the students has also been extremely positive.

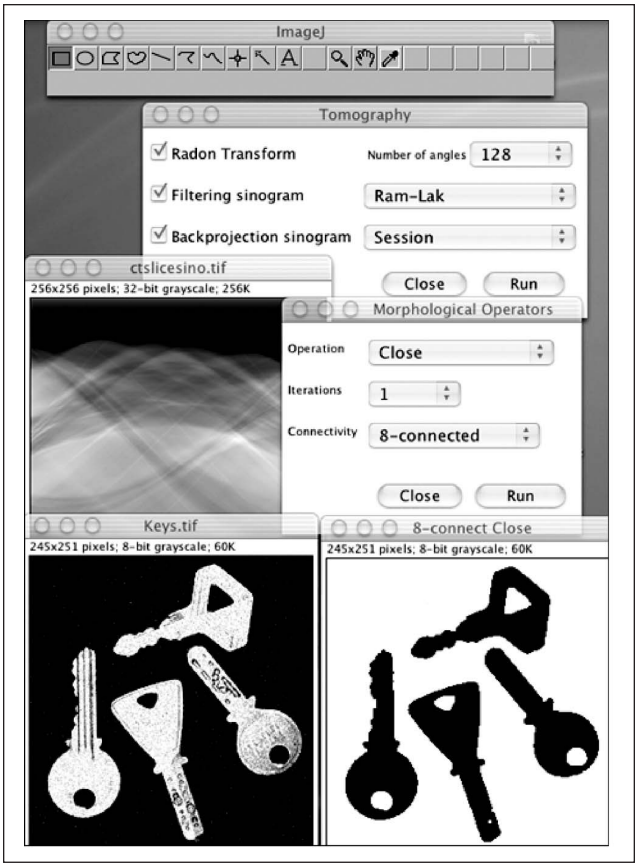
The combination of "ImageJ" and our interface layer is also used by the students who choose to complete a practical semester or diploma project in our laboratory, which is fully equipped with Macintosh computers. Here, the students develop their new IP algorithms using ImageJ and a user-friendly Java integrated development environment (IDE); at the end, they can easily produce a demonstration applet, which is then made available on the Internet.

The students value the fact that the software tools are all freely available on the Web. After downloading ImageJ and a Java development kit (JDK), they are ready to work at home.

Discussion and Conclusion

The proposed computer laboratories are a perfect complement to a theoretical course on IP. Students get active, hands-on practice in IP that will be valuable to them later in the workplace. They also learn how to implement IP algorithms. The computer sessions increase their interest in the course; students like to interact with images and become much more involved as soon as they see some practical relevance. The programming experience raises their curiosity and often stimulates them to do their own experiments. The overall reaction of our students has always been very positive.

As designers of IPLab, we are still astonished by the robustness of Java and ImageJ. The system is quite stable and appears to be robust against the student's programming errors—much more so than any other language or system that we have tested before. Up to now, we have not yet experienced a single crash due to bugs in plug-ins.



▲ *3. Examples of user interface and results for sessions 3 and 7.*

The laboratories are entirely based on ImageJ. The students can walk away from the course with an IP system that is operational. Using the ImageAccess interface layer, they can easily program both ImageJ plug-ins or Internet applets. The system that we have described may also appeal to practitioners as it offers a simple, full-proof way of developing professional level IP software.

Acknowledgments

We would like to thank Dimitri Van De Ville for the editorial advice and Wayne Rasband for his helpful and valuable remarks and for his generosity in making ImageJ freely available to the community. We are also thankful to EPFL students and assistants who have tested the system and have given us feedback to improve it.

Daniel Sage received the master of sciences (DEA) and Ph.D. degrees in control and signal processing in 1986 and 1989, respectively, from the INPG, Grenoble, France. From 1989 to 1998, he was a consulting engineer developing vision systems for quality control. He was head of the Industrial Vision Department at Attexor S.A. In 1998, he joined the Biomedical Imaging Group at the Swiss Federal Institute of Technology Lausanne (EPFL) as the head of software development. He is also involved in the development of methods for computer-assisted teaching.

Michael Unser received the M.S. (summa cum laude) and Ph.D. degrees in electrical engineering in 1981

and 1984, respectively, from the Swiss Federal Institute of Technology (EPFL) in Lausanne, Switzerland. From 1985 to 1997, he was with the Biomedical Engineering and Instrumentation Program, National Institutes of Health, Bethesda, Maryland. He is now a professor and director of the Biomedical Imaging Group at the EPFL. His main research area is biomedical image processing. He has a strong interest in sampling theories, multiresolution algorithms, wavelets, and the use of splines for image processing. He is the author of over 100 published journal papers in these areas. He is involved in various editorial activities; these include associate editor-in-chief of *IEEE Transactions on Medical Imaging* and editor-in-chief of the *Wavelet Digest*. He received the 1995 Best Paper Award and the 2000 Magazine Award from the IEEE Signal Processing Society. He is a Fellow of the IEEE.

References

[1] M. Sonka, E.L. Dove, and S.M. Collins, "Image systems engineering education in an electronic classroom," *IEEE Trans. Educ.*, vol. 41, no. 4, pp. 263–272, 1998.

[2] E. Fink and M. Heath, "Image-processing projects for an algorithms course," *Int. J. Pattern Recognition Artificial Intell.*, vol. 15, no. 5, pp. 859–868, 2001.

[3] A. Sanchez, J.F. Velez, and A.B. Moreno, "Introducing algorithm design techniques in undergraduate digital image processing courses," *Int. J. Pattern Recognition Artificial Intell.*, vol. 15, no. 5, pp. 789–803, 2001.

[4] K. Bowyer, G. Stockman, and L. Stark, "Themes for improved teaching of image computation," *IEEE Trans. Educ.*, vol. 43, no. 2, pp. 221–223, 2000.

[5] G.W. Donohoe and P.F. Valdez, "Teaching digital image processing with Khoros," *IEEE Trans. Educ.*, vol. 39, no. 2, pp. 137–142, 1996.

[6] J.A. Robinson, "A software system for laboratory experiments in image processing," *IEEE Trans. Educ.*, vol. 43, no. 4, pp. 455–459, 2000.

[7] "Curriculum advances in digital imaging systems," in *Proc. IEEE Int. Conf. Image Processing (ICIP'96)*, Lausanne, Switzerland, 1996.

[8] "Undergraduate education and image computation," in *Proc. IEEE Computer Vision and Pattern Recognition (CVPR'00)*, Hilton Head Island, SC, 2000.

[9] G. Bebis, D. Egbert, and M. Shah, "Review of computer vision education," *IEEE Trans. Educ.*, vol. 46, no. 1, pp. 2–21, 2003.

[10] Center for Image Processing in Education (CIPE) [Online}. Available: http://www.cipe.com/

[11] The MathWorks Inc. (MATLAB), Natwick, MA {Online]. Available: http://www.mathworks.com/

[12] B.M. Dawant, "MATLAB-supported undergraduate image processing instruction," in *Proc. SPIE Medical Imaging 1998*, vol. 3338, 1998, pp. 276–284.

[13] H.J. Trussell and M.J. Vrhel, "Image display in teaching image processing. I. Monochrome images," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP'00)*, Istanbul, Turkey, 2000, vol. 6, pp. 3518–3521.

[14] C.S. Zuria, J.M. Ramirez, D. Baez-Lopez, and G.E. Flores-Verdad, "MATLAB based image processing lab experiments," in *Proc. IEEE Frontiers in Education Conf. (FIE'98)*, Tempe, AZ, 1998, pp. 1255–1258.

[15] S.L. Eddins and M.T. Orchard, "Using MATLAB and C in an image processing lab course," in *Proc. IEEE Int. Conf. Image Processing (ICIP'94)*, Austin, TX, 1994, vol. 1, pp. 515–519.

[16] Khoros Pro 2001 Integrated Development Environment, Khoral Inc., Albuquerque, NM [Online]. Available: http://www.khoral.com/

[17] R. Jordan and R. Lotufo, *Digital Image Processing (DIP) with Khoros Pro 2001*, visited in February 2001.

[18] R. Lotufo and R. Jordan, "Hands-on digital image processing," in *IEEE Proc. Frontiers in Education Conf. (FIE'96)*, Salt Lake City, UT, 1996, pp. 1199–1202.

[19] M. Jankowski, Digital image processing with Mathematica [Online]. Available: http://www.usm.maine.edu/~mjkcc/docs/dip/

[20] Mathematica, Wolfram Research, Inc, Champaign, IL [Online]. Available: http://www.wolfram.com/

[21] U. Rajashekar, G.C. Panayi, F.P. Baumgartner, and A.C. Bovik, "The SIVA demonstration gallery for signal, image, and video processing education," *IEEE Trans. Educ.*, vol. 45, pp. 323–335, Nov. 2002.

[22] LabVIEW, National Instruments Corp., Austin, TX [Online]. Available: http://www.ni.com/

[23] D.S. Sohi and S.S. Devgan, "Application to enhance the teaching and understanding of basic image processing techniques," in *Proc. IEEE Southeastcon 2000*, Nashville, TN, 2000, pp. 413–416.

[24] Advanced Visual Systems Inc. (AVS/Express), Waltham, MA [Online]. Available: http://www.avs.com/software/soft_t/avsxps.html

[25] M. Jackson, D.I. Laurenson, and B. Mulgrew, "Supporting DSP education using Java," in *Proc. IEE Symp. Engineering Education: Innovations in Teaching, Learning and Assessment*, London, UK, 2001, pp. 1–6.

[26] A. Jacot-Descombes, M. Rupp, and T. Pun, "LaboImage 4.0: Portable window based environment for research in image processing and analysis," in *Proc. SPIE Symp. Electronic Imaging Science and Technology, Image Processing: Implementation and Systems*, San Jose, CA, 1992.

[27] F. DePiero, "SIPTool: The signal and image processing tool—An engaging learning environment," in *Proc. IEEE Frontiers in Education Conf. (FIE'01)*, 2001, vol. 3, pp. F4C–1–5.

[28] A. Goncalves Silva, R. De Alencar Lotufo, and R. Campos Machado, "Toolbox of image processing for numerical Python," in *Proc. IEEE Brazilian Symp. Computer Graphics and Image Processing*, Florianopolis, Brazil, 2001, pp. 402.

[29] S.L. Tanimoto and J.W. Baer, "Programming at the end of the learning curve: Lisp scripting for image processing," in *Proc. IEEE Symp. Human-Centric Computing Languages and Environments*, Stresa, Italy, 2001, pp. 238–239.

[30] S.E. Umbaugh, *Computer Vision and Image Processing: A Practical Approach Using CVIPtools.* Englewood Cliffs, NJ: Prentice-Hall, 1998.

[31] P.F. Whelan, NeatVision, Vision Systems Group at Dublin City University, Dublin, Ireland. Available: http://www.neatvision.com/

[32] P.F. Whelan and D. Molloy, *Machine Vision Algorithms in Java: Techniques and Implementation.* New York: Springer-Verlag, 2001.

[33] M.W. Powell and D. Goldgof, "Software toolkit for teaching image processing," *Int. J. Pattern Recognition and Artificial Intell.*, vol. 15, no. 5, pp. 833–844, 2001.

[34] M.W. Powell, Java Vision Toolkit (JVT), Univ. of South Florida, FL. Available: http://marathon.csee.usf.edu/~mpowell/jvt/

[35] W. Rasband, ImageJ, National Institutes of Health, Bethesda, MD. Available: http://rsb.info.nih.gov/ij/

[36] R.B. Fisher and K. Koryllos, "Interactive textbooks: Embedding image processing operator demonstrations in text," I*nt. J. Pattern Recognition and Artificial Intell.*, vol. 12, no. 8, pp. 1095–1123, 1998.

[37] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, Hypermedia Image Processing Reference (HIPR). Available: http://www.dai.ed.ac.uk/HIPR2/

[38] B. Morse, Java Image and Graphics Library (JIGL), Brigham Young University, Provo, UT. Available: http://rivit.cs.byu.edu/jigl/

[39] D. Sage and M. Unser, "Easy Java programming for teaching image-processing," in *Proc. IEEE Int. Conf. Image Processing (ICIP'01),* Thessaloniki, Greece, 2001, vol. 3, pp. 298–301.

[40] D.Y. Wang, B. Lin, and J. Zhang, "JIP: Java image processing on the Internet," *Proc. SPIE Color Imaging: Device-Independent Color, Color Hardcopy, and Graphic Arts*, vol. 3648, pp. 354–364, Dec. 1998.

[41] Y. Cheneval, L. Balmelli, P. Prandoni, J. Kovacevic, and M. Vetterli, "Interactive DSP education using Java," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP'98)*, Seattle, WA, 1998, vol. 3, pp. 1905–1908.

[42] D.A. Lyon, *Image Processing in Java*. Upper Saddle River, NJ: Prentice-Hall, 1999.

[43] D. Roman, M. Fischer, and J. Cubillo, "Digital image processing—An object-oriented approach," *IEEE Trans. Educ.*, vol. 41, no. 4, pp. 331–333, 1998.