

IMAGE PROCESSING TOOLKIT: A FRAMEWORK FOR E-LEARNING IMAGE PROCESSING

Eugene Ageenko, Gaetano La Russa, Vitali Diatchkov

*Department of Computer Science, University of Joensuu, Box 111, FIN-80101 Joensuu, Finland
ageenko@cs.joensuu.fi, larussa@cs.joensuu.fi, vdiatch@cs.joensuu.fi*

ABSTRACT

Teaching image processing is quite a challenging task. This might be the reason why image processing courses are in general of graduate and postgraduate level. According to teaching experiences, an efficient teaching is feasible if a lot of attention is paid to the experimental works of the students. Experimental works should be organized in such a way that students are allowed to put in practice and try out the courses' studied technology during the short time that usually courses grant them. It is clear that better pedagogical results can be achieved if such visualizing tools for the demonstration of the basic aspects of image processing are made available to the teachers. The visualization of the impact that an algorithm has on the visual data (such as e.g. filtering algorithm) is cognitively superior to the plain textual description, which requires learners' use of imagination and interpretative skills, which most of the times are and remain a presumption. These visualizing tools can also be used by the students after their lecture hours. In laboratories students can actively complement the basic lectures by processing and manipulating images and even construct own image processing tools within reasonable time frames. This hand-on experimentation can significantly facilitate the learning of the mathematical concepts of image processing (Naidu, S. et al, 2002). Similarly cognitive processes would benefit from conducted experimentations. Last, but not least, the use Java platform makes these tools widely accessible.

KEYWORDS

Image processing, visualization and experimentation, pedagogical tools, Java.

1. INTRODUCTION

The Image Processing Toolkit (IPT) is an educational tool implemented in Java language (Arnold, K. et al, 2000). The main purpose for designing IPT was to create a simple tool for the visualization of the image processing (IP) algorithms on the base of extendable plug-in architecture. The tool was designed and developed to be simple enough even for the beginners in Java technology to add their own modules to the main application framework. Therefore this tool can be very useful not only for the courses in image processing, as a visualization tool, but also as a framework for the study of Java technology in examples of different image processing methods and algorithms (Gonzalez, R.C. et al, 2002). The dual nature of this application, teaching and learning tool, is the result of the original design that intended to bring visualizing advantages to teachers (i.e. advanced users) as well as to the students (i.e. beginners).

Modern commercial image editing tools (such as Photoshop, Photodraw, Coreldraw, etc.) have complex user interfaces and relatively high prices. Moreover, these advanced high tech tools do not bring any insight on the employed algorithms and are often offered as "black-boxes" software leaving the IP students in absolute state of ignorance. By use of these commercial tools a person can become proficient in the specific use of the tool but not at all in the IP, which of course can be suitable for a *digital designer* but is not suitable for the teaching of IP courses and principles. There has been a substantial effort among the members of this latter community (i.e. teachers and students of IP courses) to create didactical tools for teaching and learning IP. Dozens of created systems are available on the market and their descriptions and literature can be easily found via searching engines in the Internet. Recent review on image processing and computer vision education is available in (Sage and Unser, 2003) and (Bebis, G, et al 2003) respectively.

The Image Processing Toolkit is not a traditional oversized image processing program and neither a complex suite of graphics software. It is a very simple, small educational tool for teachers, in the context of

image processing courses, and for students, in context of studying the object-oriented principles of programming and design. IPT is written in Java language and takes all the advantages of this cross-platform technology and its wide distribution.

On the other hand, IP is also very practical discipline, and seeing the algorithms applied to the images especially in an interactive animated way has dramatic effects on the students. According to authors' experiences, an efficient teaching and learning of IP fundamentals is feasible if much attention is paid to visual representation of the algorithms and the experimental work of the students. The visualization of the impact that an algorithm has on the visual data (such as e.g. filtering algorithm) is cognitively superior to the plain textual description that requires a lot of imagination and interpretative skills. It is clear that better pedagogical results can be achieved if such visualization tools for the demonstration of the basic aspects of image processing are made available to teachers.

Sage and Unser say "even when the lectures include visual demonstration of IP algorithms, students are often passive" (Sage and Unser, 2003). The IP visualization and experimentation tools should be used by the students also after their lecture hours. Laboratory assignments (experimentations) should be organized in such a way that students are allowed to try out the courses' studied technology during the short time that courses usually grant them. In laboratories students can actively complement their basic lectures by processing and manipulating images and even construct their own image processing tools within reasonable short time. This hand-on experimentation can significantly facilitate the learning of the mathematical concepts of image processing (Naidu, S. et al, 2002). Similarly cognitive processes would benefit from conducted experimentations. Students would easily become motivated to study theory if they could experiment with algorithms and visualize their results (Sanchez, A. et al, 2001). Interactive software is also generally perceived as a useful tool for complementing text books on IP (Bowyer, K. et al, 2000).

The successful development of the new image-processing plug-ins lies on the full independence of this process from the rest of the application classes. The developer interacts only with 4-5 interfaces and creates the plug-ins without the need of the entire IPT source code. The main tool is being used only as a test framework where the image processing (Bright, D.S., 2004) plug-ins are deployed, tested and used. Accordingly, beginners in Java technology can quickly get into the development of new plug-ins without wasting time in the study of complex class hierarchy of the main framework.

2. STATE-OF-ART IN IMAGE PROCESSING VISUALIZATION

A system called IPLab that is an interesting effort to utilize the ImageJ environment in the creation of the hands-on IP laboratories is proposed in (Sage and Unser, 2003). At first look IPLab is apparently an *extra interface layer* between ImageJ IP framework and the student written image processing applets. This interface layer hides the details of the image implementation (binary, grayscale, or color), which facilitates the comprehension of the ImageJ plug-in architecture and simplifies plug-in development. However this comes at the cost of higher running times, which neglect the benefits of ImageJ. Sage and Unser accentuate the idea of "learning by doing" and stress the effort of the experimentation by the modification of the algorithms. In the laboratory sessions the students are given tasks to test existing IP algorithms written in Java and complete them or extend to new ones using "programming by example" approach. Sage and Unser claim that students that have "rudimentary knowledge of the Java syntax" or "do not even know Java" can master algorithms in "one-hour lesson".

The authors of this article, both proficient in image processing and object-oriented programming teaching, have strong doubt that algorithms illustrated in the article of Sage are feasible to comprehend by such unprepared students. Good skills in Java programming are needed in order to utilize the full power of any given algorithm modification approach. From another point of view, ImageJ can be successfully used alone without an extra interface layer (with a basic knowledge of Java and principles of object-oriented programming). This knowledge will be in one or another way required when student is asked to do changes to the algorithms beyond the simple constant modification. In any case, the changes in the algorithm will require the Java SDK environment downloaded and installed, which might not be possible on public PCs, or being a limitation for remotely located students. The animation, visualization, close observation and other aspects that arise in class teaching are also overlooked in their work. Sage and Unser also claim that "the best way to truly understand an algorithm is obviously to code it and to test it", which according our experience is

not always true. Even though the students remember the algorithm, it does not necessary mean they understand “why” the algorithm works, “what” precisely happens when it is applied and “when” it shall be best used.

One example of the typical task given to our students in JoY (University of Joensuu) as a part of IP class: “providing the satellite photograph of Martian surface, generate the image displaying the safe landing areas for the landing pod”, or “detect the secret message left to us by an ancient civilization”. These tasks require understanding of the image processing algorithms. The result is not only the image showing the correct information for the landing areas but also a sequence of operation that lead to that image.

When comparing the IPT tool to other existing similar tools (as the IPLab), one of the IPT relevant advantages is the possibility to re-operate processed images with various applets and re-iterate such processes. All stages of the processing can be saved and visually compared. Therefore the multiplicity of possible comparisons and experiments, keeping the visual track of transformed images, allow the understanding of parameter values and algorithm variations.

The use of a Java platform makes these tools widely accessible. Not only “Java is a natural language for interactive teaching” (Cheneval, Y., et al, 1998) but also an object-oriented language, “which is desirable for IP programming” (Roman, D. et al, 1998). The IPT toolkit is an extendable, cross-platform, open-interface, plug-in based pedagogical framework with a simple, intuitive user interface that is well appropriate for the listed goals of IP teaching and understanding. The tool works as a concept convener, easing students’ acquisition of complex ideas in Java programming and the image processing area and allowing them to verify algorithms’ validity and functionality in image processing.

The standard Java library is already well suited for signal and image processing operations (Lyon D.A., 1999), which frees us from dependency on third party solutions. We especially do not abstract from standard image types because we found it essential to discriminate them. Binary, Grayscale and Color image types are supported and are differentiated by each plug-in, which in turn will process only the image types it supports and accordingly to that type. Indeed morphological operation for binary images and grayscale image are different, as well as filtering algorithms. The execution time of algorithms implemented in standard Java is indeed slower than in ImageJ, however is quite sufficient unless the user operates on images larger than 1 Megapixel, which is quite exceptional for demonstration and experimentation needs.

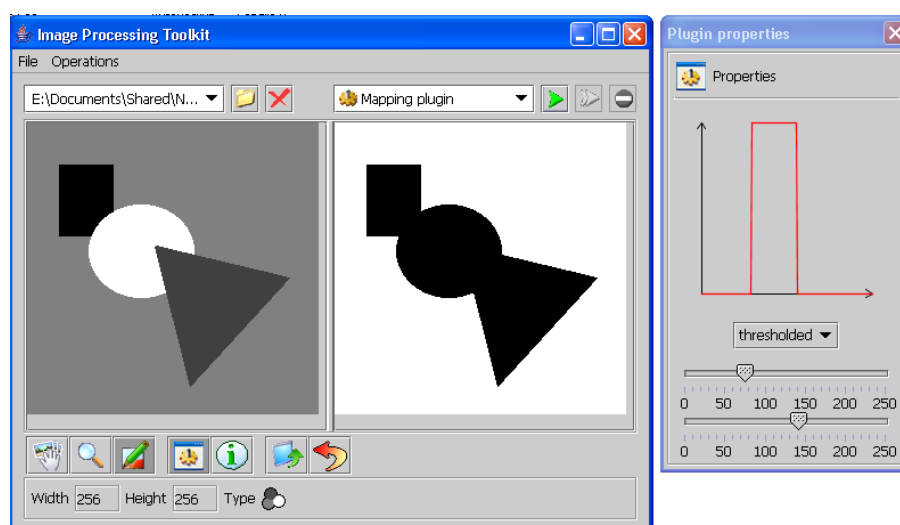


Figure 1. The view on the IPT toolkit, illustrating main application window and the plugin property pane.

3. IPT IN BRIEF

The basic underlying principles that define the IPT interface and its structure can be summarized according the following:

- The IPT does not depend on any third party solutions or systems but is a *standalone tool* using standard Java platform.

- IPT's Java nature makes it a *platform independent tool* and therefore it can be executed on any platform where Java virtual machine is installed.
- The IPT is under GPL license therefore it is *royalty and license free*
- The IPT has *open application interface*, which allows the plug-ins to be made by independent developers. The plug-ins are easily added to the toolkit
- The IPT is a *small sized software*, allowing a fast single file download or web-start. No installation is required
- The IPT *user interface* (with a fast learning curve) is *simple, intuitive and visual*. The point-and-click user interface provides the essence of the functionalities required to teach and study the subject. The IPT does not have a multi-level and context menus requiring only a very simple logic with WYSIWYG appearance. A basic level of computer knowledge is required to operate the tool
- The IPT is meant for teaching and/or learning image processing, therefore its usability and features are *well balanced* for these scopes. For professional image processing and editing other professional commercial tools, such as MathLab or PhotoShop, could be used
- The SDI architecture (one panel with image input and image output and toolbar) allows users to *interactively* apply all available algorithms to the image. The resulting image can replace the original input image with a single mouse click and then being again processed with a new algorithm. The resulting final and intermediate images can be opened in secondary windows for demonstrations, comparisons, detailed observations or re-use
- The produced effects of the algorithm in the output image can be immediately *visualized and compared* with the input image
- The IPT *demonstrates* how algorithms and/or their properties differentiate on the image by giving a comparison of the images in the synchronous panning and zooming
- And last but not least, the IPT allows conducting *experiments* with the algorithms by modifying the algorithm parameters, modifying the images, visualizing the images as numeric 2D arrays (matrices), and modifying the existing or designing new algorithms by using the standard Java language.

In particular, no Java programming is required in order to experiment with the algorithms, especially when a student can design complex algorithms using already existing components as building blocks. In this way the toolkit can be already accessible for undergraduate and high-school students. However the knowledge of Java language will facilitate the understanding of fundamental building blocks (IP algorithms and plug-ins) and design new ones. The programming of such algorithms itself also facilitates the learning of Java and can therefore be used as a laboratory work for Java or OOP classes.

Each plug-in can be customized via its *properties' pane* (see Figure 1) enabling experimentation with the plug-in. Multiple views of the images can be engaged. Views support properties to enlarge the image content and *synchronously display* the images to ease demonstration and comparison.

A processed image can be easily reused and another operation continued (forming the *sequence of operation*).

At any step the image can be opened in a separate window to form a chain of the action for the visual demonstration (see Figure 2) or for the reuse of the images in further operations. As shown before, views can be enlarged and synchronously panned for detailed observation.

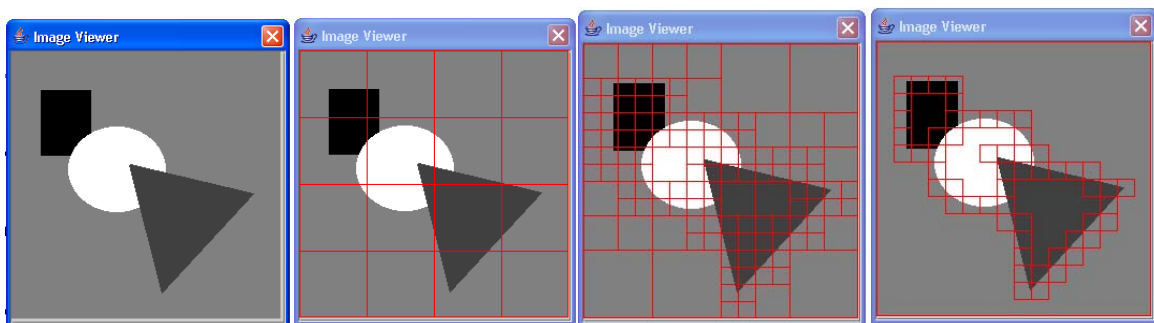


Figure 2. Example of Split-and-merge animation. Original image, after a first-step, after a split sequence, final result.

4. THE IPT FRAMEWORK

4.1 The toolkit architecture

The Java technology gives very powerful opportunities to develop complex projects and applications by decomposing them into independent parts that are communicating between each other by the means of well defined interfaces (Rasband, W., 2004). The architecture of the IPT toolkit conforms to the well known design patterns and paradigms in object-oriented programming. The IPT, a standalone Java application, consists of the (see Figure 1):

- main framework with all user interface (UI) components, operations and plug-ins management routines,
- the plug-ins – modules that implement image processing algorithms and methods, and finally
- XML configuration file, which describes the following properties of the IPT: (set of plug-ins, set of favorite images (absolute paths on local drive), set of recent images, etc.)

Plug-in based architecture of the IPT lets to easily extend existing functionality with new implemented image processing algorithms. Plug-in is a Java class that implements special interfaces or extends appropriate abstract class. The architecture of plug-ins consists of:

- `ia.ImagePlugin` – the interface;
- `ia.ImageProcessingPlugin` – abstract class for one step processing plug-ins;
- `ia.ImageProcessingAnimatedPlugin` – abstract class for animated plug-ins;
- `ia.FrameworkCallback` – callback class.

The first class is a prototype for a simple one-step plug-in where image processing is performed in one step (e.g. simple filtering operation). It receives an image as the input parameter and returns the processed image which is subsequently displayed in the application. The second abstract class is the base class for the animated plug-ins, in which an image processing operation is performed iteratively. Each of the iterations makes some changes to the image resulting in animation in the main application. The plug-in management is a task of main framework that has different UI elements such as menu or buttons to control workflow of plug-ins execution.

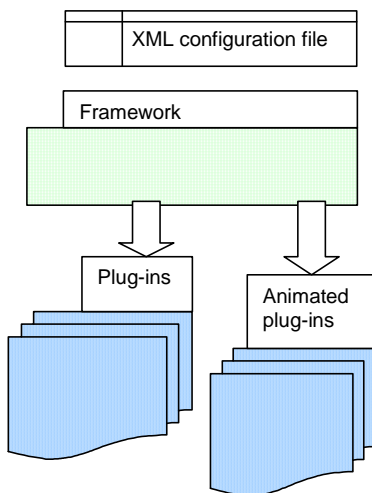


Figure 3. Toolkit architecture.

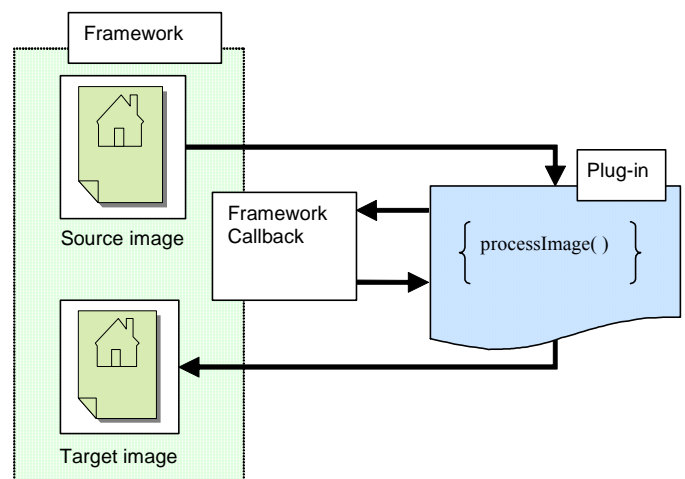


Figure 4. Plug-in workflow.

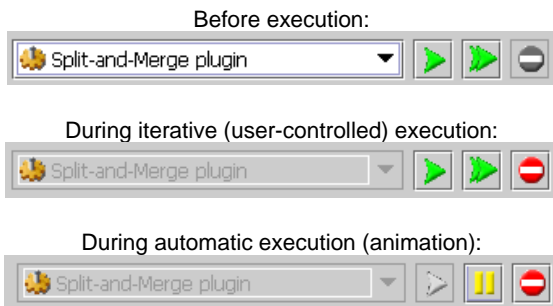


Figure 5. The view on the toolbar of the animated plug-in before, and during execution.

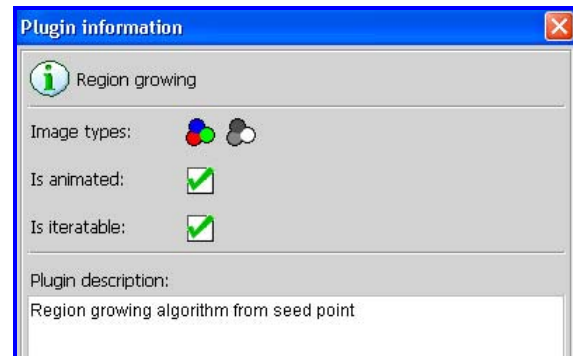


Figure 6. Plug-in information pane displaying supported image types, plug-in type and plug-in description.

4.2 The framework

The IPT is a standalone Java application because of needs to read and write to local disk. The Swing library is a heart of UI components. It is a cross-platform with light-weight UI library used in the IPT. The basic management of UI components lies on the IPTFrameworkPanel class. It is responsible for the creating all user interface components and communication infrastructure of the IPT.

The image input/output operations are relied on the javax.imageio library that is a standart image I/O library from Java 1.4. The most spread image formats are accessible from this library for the reading/writing operations. The wrapper class for all images to be loaded to the IPT framework is a standard BufferedImage class. All loaded images are presented using BufferedImage class and it is a start point to access different image information as color model or pixel data.

The IPT framework follows to MVC (Model-View-Controller) design pattern (Budd, T., 2001). It means that representation code is separated from the controlling code and data model. Part of the management routines are put in the IPTController class and are being called using set of static methods from any part of the framework. The Model-View-Controller architecture is a special design pattern created to model the real-world applications. It consists of the three:

- The model that simulates an object from a real world to the give degree of details
- The view that represents the model for the user.
- The controller that controls the model parameters (is the response of the use back to the model).

The model is usually invisible (except program code). The view is visible on the screen as a graphical image, graphs, diagrams, etc. The controls are visible as GUI elements including scrollbars, buttons, etc. The model is frequently changed by the controller (for example when user interacts with the controller by the mean of its GUI). As a consequence, this action initiates the changes in the view. The model itself exists independently of the view and controller that can be varied from the application to application.

The IPT framework uses data wrapper classes where different configuration information is kept, but also other Java objects. MVC architecture is utilized as follows (see Figure 8):

- *Model* – ImageObjectStorage class;
- *View* – IPTViewerPanel, IPTZoomedImagePopupWindow classes, representing the graphical view of the image to the user.
- *Controller* – The IPTController class. It has a set of methods to get access from one independent part of toolkit to another and additional helper static methods.
- *Processing components*: ImagePlugin, ImageAnimationThread
- *GUI components*: IPTFrameworkPanel, IPTMenuBar, IPTStatusBar, IPTMainToolbar (see Figure 7 for the GUI component layout).

The GUI components build user interface and dispatch user events. Controller processes events and calls necessary routines and actions, after updates view. Processing components are components where buisness

logic is encapsulated – plugins, algorithms, workflow management, etc. The View is based upon components visualizing the results to the user and is being updated by the Controller.

Plug-in design and management in main IPT framework is based on open interfaces. Each plug-in gives set of methods to be called by framework as `ImageProcessingPlugin.processImage`. In the same time framework provides an open interface `ia.FrameworkCallback` for plug-ins with a set of methods to be called from them. Thus the dual communication model is realized in IPT’s plug-in architecture: as framework can call methods from plug-ins to manage its behavior and process images as plug-ins can call methods from framework to get specific additional information such as seed point for algorithm.

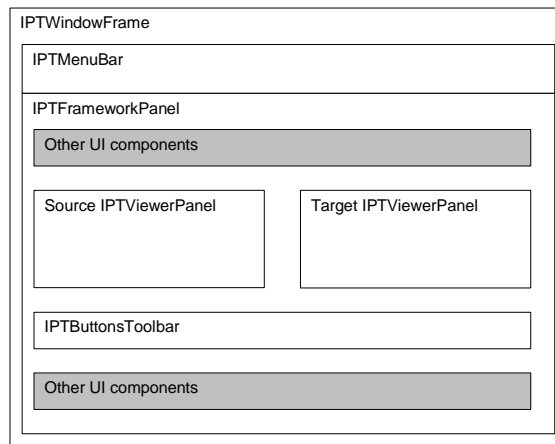


Figure 7. GUI components layout.

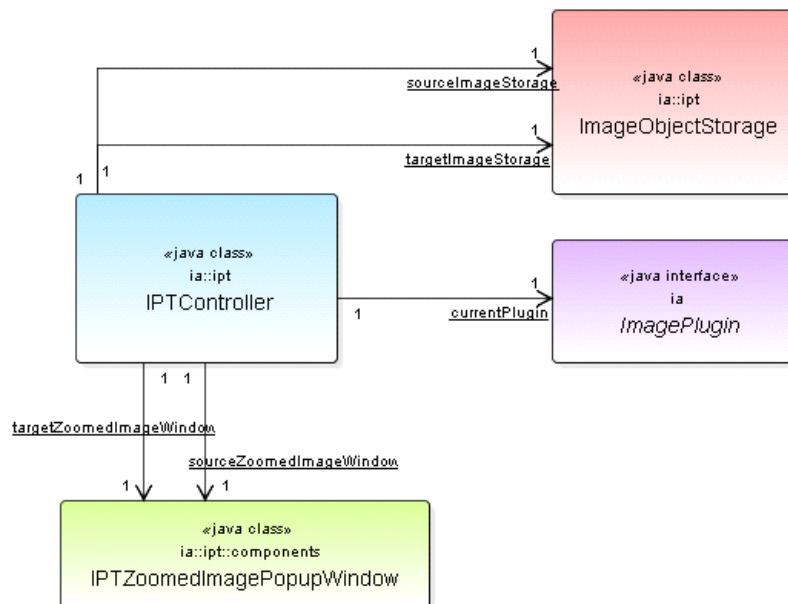


Figure 8. IPT Framework architecture.

4.3 The plug-in

The plug-in architecture gives the capability to develop and add new image processing algorithms to the IPT without any changes in the source code of the main framework. Simultaneously, the main framework can be separately updated resulting e.g. in a new updated user interface, extra functionality or new plug-in

capabilities accessible to all plug-ins automatically. The process of plug-in development is entirely independent from the main framework and can be performed by any person familiar with basics of Java language (Flanagan, D., 1999). Sample plugins are supplied with the framework for illustrative purposes.

Plug-in can:

- 1) process the image;
- 2) return type of images it accepts for processing and resulting type of the image;
- 3) support animation of the algorithm;
- 4) provide short description of the plug-in and the algorithm it visualizes;
- 5) modify parameters of the algorithm using own control pane.

The main processing methods of the plug-in are:

- `public BufferedImage processImage(BufferedImage sourceimage);`
- `public BufferedImage processIterationImage (BufferedImage sourceimage, BufferedImage targetimage, int count);`

The image processing algorithm shall be implemented solely in these methods depending on whether the plug-in is animated or not. The main framework communicates with the plug-in using these methods as well as few other methods used to retrieve informative about the plug-in (such as its name, description, capabilities, supported image types and such. The common part between the framework and plug-ins is the interface `ia.ImagePlugin`, through which all communication is performed. The plug-in has its own component with different user interface controls for the management and tuning parameters of processing algorithm. The visualization of this component is also performed by calling the method `JPanel createGUI()` of the plug-in common interface. The Figure 4 outlines the workflow of non-animated plug-in execution: source image is passed into the main processing method which returns target filtered image.

An abstract class constitutes the base class for the animated plug-ins, in which an image processing operation is performed iteratively. Each of the iteration makes some changes to the image resulting in animation in the main application. The plug-in management is a task of main framework that has different UI elements such as menu or buttons to control workflow of plug-ins execution.

5. THE USABILITY OF IPT

The simple user interface provides intuitive understanding of the functions of the IPT, allowing an easy usage of its integrated tools. As it was mentioned above, the educational purpose of this tool is to help teachers of image processing courses to visualize various image filter algorithms. Another additional advantage that teachers of Java programming technology can have of this tool is by using it as a framework for getting students' practical experience feedbacks in developing image processing plug-ins. Novices can study Java programming language and verify their developing know-how and understanding of different image processing algorithms. In addition the IPT gives the proper framework for testing and experimenting in image processing algorithms, providing those practical visual feedbacks that easily can be compared to similar algorithms' outcomes. Each plug-in has a UI property component where different controls for the tuning of the algorithm parameters are placed. Researchers can therefore combine different algorithms and methods trying to obtain new effects. The interfaces and technologies are open, cross-platform and at the current time there some few dozens of implemented plug-ins and image processing algorithms that can be used as examples for a personalized creative development (Barrows, H.S. et al, 1996).

The intuitive user interface allows, via the use of few buttons, the management and execution of plug-ins. RGB, grayscale and binary images are supported in the IPT but obviously different plug-ins are intended for different image types. There are built-in capabilities to convert one type of image to another one as, for example, in the case it would be needed to process a RGB image by "threshold" plug-in but only grayscale types of images is supported.

A list of the most recent images is kept in the configuration file and displayed in the menu. The tool provides a function to bookmark own favorite images.

6. CONCLUSIONS

In this paper, the authors present an image processing toolkit capable of demonstrating, in an intuitive visual form, all main connections that exists between an algorithm and its visual data. The toolkit, based on a Java platform, has proved to be pedagogical software capable of demonstrating basic and advanced concepts of IP. The advantage of such a toolkit is twofold: the teachers can convey in a more intuitive way various complex concepts related to the image processing; while students can quickly get into the topic by perceiving how changing the parameters of an algorithm affects the images. As mentioned before, the toolkit can also be used by the students outside lecture and lab hours, allowing them to conduct their own experiments and consequently learn-by-doing, soon becoming enough experienced to create their own IP plug-ins. When using this toolkit, students have been able to process and manipulate images and construct their own image processing tools within short times.

The IPT is an advanced educational tool using intuitivity and concept correlation to concretize abstract concepts related to image processing algorithms. IPT, even though at its experimental phase, has already proved to be a powerful tool in image processing courses and also an efficient learning framework environment for students wishing to study Java technology. The use of IPT improves the mastering of application design in object oriented environment and provides new tutoring/learning approaches to this field.

The IPT is an extendable, cross-platform with a simple user interface application that is well appropriate for the listed goals of image processing understanding. In concrete the tool has worked as a concept convener, easing students' acquisition of complex ideas in Java technology and image processing area and allowing them to verify algorithms' validity and functionality in image processing. It is expected that the under-development refined version will be a solid cornerstone in the image processing courses utilizing visual tools (Ageenko, E., 2005).

REFERENCES

- Ageenko, E., 2005, Interactive Environment for Algorithm Visualization, Demonstration and Experimentation in Image Processing, (Research Project home-page), <http://cs.joensuu.fi/pages/ageenko/research/edu/ipt.htm>
- Arnold, K., Gosling, J., Holmes, D., 2000, *The Java™ Programming Language*, 3rd ed., Addison-Wesley.
- Barrows, H.S. et al, 1996. Computer-Supported Problem-Based Learning: A Principled Approach to the Use of Computers in Collaborative Learning, In T. Koschmann (Ed.) *CSCLE: Theory and Practice of an Emerging Paradigm*, Lawrence Erlbaum Associates, Mahwah, New Jersey. pp. 83-124.
- Bebis, G., Egbert, D., Shah, M., 2003, "Review of computer vision education," *IEEE Transactions on Education*, vol. 46, no. 1, pp. 2–21.
- Bowyer, K., Stockman, G., and Stark, L., 2000, "Themes for improved teaching of image computation," *IEEE Transactions on Education*, vol. 43, no. 2, pp. 221–223.
- Bright, D.S., 2004. Digital Image Processing with NIH Image (Mac) /Scion Image (PC) /ImageJ, 2004. *Report*. Surface and Microanalysis Science Division, National Institute of Standards and Technology, Gaithersburg, MD 20899-8371, <http://www.nist.gov/lispix/imlab/labs.html>
- Budd, T., 2001, *An Introduction to Object-Oriented Programming*, 3rd ed. Addison Wesley.
- Cheneval, Y., et al, 1998, "Interactive DSP education using Java," in Proc. *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP'98)*, Seattle, WA, vol. 3, pp. 1905–1908.
- Flanagan, D., 1999. 3rd edition, November 1999. *Java in a Nutshell: A Desktop Quick Reference (Java Series)*, 648 pages, O'Reilly & Associates.
- Gonzalez, R.C. et al, 2002. *Digital image processing*, 2nd edition, Addison-Wesley.
- Lyon D.A., 1999, *Image Processing in Java*. Upper Saddle River, NJ: Prentice-Hall.
- Naidu, S. et al, 2002. "The Experience of Practitioners with Technology-Enhanced Teaching and Learning", *Journal of IEEE Education Technology & Society*, Vol. 5 N.1, January 2002, 23-34
- Rasband, W., 2004. *ImageJ*, Research Services Branch, National Institutes of Health, USA, <http://rsb.info.nih.gov/ij/>
- Roman, D., Fischer, M., and Cubillo, J. "Digital image processing — An object-oriented approach," *IEEE Transactions on Education*, vol. 41, no. 4, pp. 331–333, 1998.
- Sage, D., Unser, M., 2003, "Teaching Image-Processing Programming in Java", *IEEE Signal Processing Magazine*, vol. 20, no. 6, pp. 43-52, November 2003.
- Sanchez, A., Velez, J.F., and Moreno, A.B., 2001, "Introducing algorithm design techniques in undergraduate digital image processing courses," *Int. Journal on Pattern Recognition and Artificial Intelligence*, vol. 15, no. 5, pp. 789–803.