Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS

David Hauzar

Image Viewing and Manipulation Tool

Department of Software Engineering

Advisor: RNDr. Tomáš Kalibera, Ph.D.
Study Program: Computer Science, General Computer Science

2007

My most sincere thanks go to my advisor, RNDr. Tomáš Kalibera, Ph.D. I thank him for many ideas and suggestions, for great support during the development of this work, and for giving me the opportunity to work with this thesis.

Furthermore, I would like to express my gratitude to my friends, Marcin Kilarski and Archibald Liddell, who reviewed the text.

I hereby declare that I wrote the thesis myself using only the referenced sources. I agree with lending the thesis.

Prague, August 9, 2007 David Hauzar

Contents

Název práce: Nástroj na prohlížení a manipulaci s obrázky
Autor: David Hauzar
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Tomáš Kalibera, Ph.D.
e-mail vedoucího: tomas.kalibera@dsrg.mff.cuni.cz

Abstrakt: Digitální zpracování obrazu zahrnuje mnoho technik užitečných pro opravu a korekci fotografií. Je to například filtrace šumu, zaostřování obrázků, vyvažování barev a mnoho dalších. Cílem práce je navrhnout a implementovat přenositelný program, který by umožnil snadnou integraci existujících implementací těchto technik a uživatelům poskytl sjednocené a snadno použitelné uživatelské rozhraní.

Program obsahuje funkce pro procházení, prohlížení a zpracování obrázků. Mezi pokročilé funkce patří spojení expozice – spojení fotografií stejného objektu pořízených s odlišným nastavením expozice do jedné fotografie s větším dynamickým rozsahem. Nástroj umožňuje aplikovat některé operace na skupinu obrázků. Je to rotace obrázku, změna velikosti obrázku a mediánový filtr. Mechanizmus rozšíření programu zahrnuje podporu pro přidávání nových operací zpracování obrázků, aplikaci těchto operací na skupinu obrázků, přidávání podpory nových grafických formátů, modifikaci a rozšiřování uživatelského rozhraní programu.

Klíčová slova: digitální zpracování obrazu, prohlížení obrázků, vylepšení obrázků, spojení expozice

Title: Image Viewing and Manipulation Tool
Author: David Hauzar
Department: Department of Software Engineering
Advisor: RNDr. Tomáš Kalibera, Ph.D.
Advisor's e-mail address: tomas.kalibera@dsrg.mff.cuni.cz

Abstract: Image processing comprises many useful techniques for fixing and correcting of digital photographs, such as noise filtering, sharpening of images, color balancing, and many others. The aim of the work is to design and implement a portable tool that would allow easy integration of existing implementations of such techniques, providing its users with a unified and easy to use interface.

The tool offers basic functions for image browsing, viewing, and processing. The advanced functions include image bending – manual combining of photographs of the same object taken with different exposures into a single photograph with higher dynamic range. The tool makes it possible to apply some of the operations to a group of images. These operations are image rotation, image rescaling, and median filtering. The extension mechanism of the tool includes support for adding new image processing operations, applying operations to a group of images, extending the range of supported image formats, modifying and extending user interface of the program.

Keywords: digital image processing, image viewing, image enhancement, exposure bending

Chapter 1

Introduction

1.1 Digital Image Processing

William K. Pratt [9]:

> Now, in this beginning of twenty-first century, image processing has become a mature engineering discipline.

Thanks to a high computer efficiency, digital image processing can be used in many applications in both research and industry. Well known are applications in space imagery and medical research.

The example of such application was correcting an error of Hubble telescope optical system [1, 2, 3, 4]. Because of this error, first pictures taken by the telescope were so distorted, that they were unsuitable for any further analysis. Unfortunately, it was physically not possible to fix physically the optical system in following three years. Nevertheless, with the exact knowledge of the error, it was possible to construct an image processing operator that enabled the correction of distorted images. The results were very good and the telescope carried out a large number of observations.

1.2 Digital Image processing and Photographs

Progress in this field has led to the development or improvement of many image processing techniques. It is not surprising that many of such techniques are useful for fixing and correcting digital photographs.

The example of the image processing technique, potentially practical for common users, combines information from more photographies of the same object into the enhanced one. This includes the technique of exposure bending, which serves for enhancing the dynamic range of photographies and, whose implementation is present in this tool. However, it is possible to go even further. With the techniques of image registration, it is possible to automatically fit the images. Next, with the use of a blind deconvolution technique, it is possible to sharpen such images. The noise of such images can be also highly suppressed without the loss of information.

Many techniques also enable the enhancement of a single image. As an example, one can mention noise cleaning techniques, sharpening image techniques, or histogram modification techniques, which seek to enhance the contrast of the image.

It could be interesting, that some digital image improvement techniques are derived from traditional techniques used to enhance images taken on the photography film. The difference is, that enhancing digital photographs is much faster.

1.3 Why Another Image Processing Tool?

Despite the great potential asset of the image improvement techniques of photographs fixing and correcting, more sophisticated techniques are often used only in specialised applications and are not accessible for common users.

The first kind of programs providing the image improvement techniques are image editors like Photoshop. The problem is that such tools are aimed at painting and are too complex for most users. Moreover, the image improvement techniques available in such tools are often oriented to long work with a single image. This is usually not acceptable for users, who would like to enhance many photographies.

Then, there are tools aimed at one specific image improvement function. This approach is good while realizing image improvement functions, which implies complex manual manipulating with images. Once again, this is not interesting for common users. Moreover, there is usually no point in integrating other image improvement functions with these tools.

Next, there are tools designed for image viewing and automatic image processing. It is for example XnView, Google Picasa, or IrfanView. These tools are usually suitable for common users, there are easy to use and suitable for processing a large number of photographies.

Although these tools sometimes offer relatively wide range of functions, it is still only a small part of the potential of image processing techniques. The problem is, that they usually do not provide public SDK or API that would allow easy integration of more complicated image processing techniques.

Chapter 2

Aims of the tool

The tool should bring the advantages of image improvement techniques for common users and allow easy integration of existing implementations of image improvement techniques for programmers.

2.1 Demands on the Tool from Users Viewpoint

The long term plan is to offer all image processing functions that can be helpful to easy to use improvement of photographs. Unfortunately, this is not manageable to maintain by a single programmer. Consequently, functions of the tool must be limited in the initial design. Tool has to offer basic image processing functions exploitable by the majority of users.

The tool should support reading and writing of the most widespread image formats, particularly JPEG and PNG image formats. It should be able to set the quality of the lossy image formats such as JPEG.

Because users will notice the imperfections of photographies while viewing it and they do not often want to spend time by loading a special tool for image processing, the tool should also provide image viewing, browsing and processing functions. The browsing and viewing functions of the tool include browsing images in the filesystem, viewing single images, viewing previous and next images in the directory, zooming images and the support of the full screen mode.

The tool should support basic image processing functions such as image rotation, image cropping, adjusting the contrast and the brightness of the image, the modification of contrast with the curves tool, and simple noise filtering. Other image processing functions should be addible via plugins.

It should be able to select parts of the images so that the image processing functions could work with such selections. In the long term plan, the majority of the image processing functions should be applicable to only the selected part of the image. However, in the initial design, the only image processing function that will work with the selections will be the cropping function, where such functionality is the most useful.

The functionality, which can remarkably save users time, is batch processing. It should be possible to apply various image processing functions to all images in a given directory. It should be possible to set the image format of processed images and set the directory, where processed images will be stored. Batch processing should be supported by all functions which can be applicable to the group of images. In the referring version of the tool, it is the function of image rotation, the function of image rescaling, and the function of noise cleaning.

The tool should support one advanced image processing function which is not common in other image processing tools and which would make the tool more interesting. The function of exposure bending was chosen.

2.2 Demands on the Tool from Extension Programmer Viewpoint

The tool should provide a simple to use and at the same time powerful enough API that would allow the implementation of even complex image improvement functions. Such implementations should be addible without the modification of the source code of the program via additive extensions.

At first, it should be possible to write extensions that could be run by a user clicking on a menu item or on a button. The API of the tool should allow easy integration of such plugins to the menus of the tool.

Next, it is necessary to have the possibility to write extensions that would allow processing a given image. It should be possible to call this plugins from other parts of the tool with various parameters.

Furthermore, the API of the tool should allow to call given image processing operator with given parameters to the group of images and should provide a component which will allow to get the parameters of batch processing from a user.

Extension programmers may need to access the image which a user is currently viewing, process it with a given processing operator, and display the processed image to the user. They should not have to deal with such displaying of the image and with connected problems such as rescaling the image to fit the window, zooming the image, translating the image to the centre of the window, or handling undo and redo operations.

Display operations such as iterative zooming in and zooming out must not affect the quality of processing image. There should also support preview operations without affecting the quality of processing image.

It should be able to access the directory that a user is currently browsing and get images from this directory.

It should be possible to write user interface extensions that would allow to get the input from the user or to display custom user interface component to the user. Moreover, it should be possible to reuse such extensions and the tool should provide user interface components that provide the getting of basic user input from a user.

It should be possible to obtain the selected parts of the image and it should be practicable to add new methods of user driven creating and manipulating with the selections.

Last but not least, it is necessary to allow programmers to add the support of new image formats or to integrate better algorithms that work with existing image formats.

Chapter 3

Structure of the Thesis

In chapter 4, the design of the tool is analysed, discussing possible approaches to fulfill abovementioned requirements. Next, chapter 5 presents the techniques of noise cleaning and exposure bending. Using the tool is described in chapter 6. Chapter 7 provides the description of the architecture of the tool. Chapter 8 is dedicated to the comparison of similar tools and the tool itself is there evaluated. Finally, chapter 9 wraps up the thesis.

Chapter 4

Analysis of the Design of the Tool

4.1 Monolithic or Modular

The basic question is whether the tool should be monolithic or modular. In the monolithic approach, there is no need to define any modules, interactions between modules, and to design interfaces that realise these interactions. This can be beneficial in the initial phase of implementation. Nevertheless, the code would quickly become unmanageable while implementing such complex tool.

The next problem is that the monolithic approach does not allow automated integration of extensions. It would be necessary to modify the source code of the tool while extending it. Hence, it is necessary to recompile the tool every time when an extension is added. This is disadvantageous also for users of the tool.

Consequently, it will not be able to fulfill the demands on the tool listed in Section 2 with the monolithic approach and the modular approach was chosen.

4.2 First Glance at the Modules

Modules should be separated as possible and the purpose of the modules should be recognisable at first glance. The modules should be easy to use, so good design of their interfaces is fundamental.

It must be possible to enable extending given modules by extensions loaded on the start-up of the tool. Such extensions will be called plugins in further text.

4.3 Extracting Functionality into Plugins

Plugins can play different roles in the architecture of the tool. At first, most of the functionality can be implemented in the kernel of the tool and plugins can serve to extend it. In this approach, it would be possible to work directly with other classes of the kernel, create several independent objects of a given class and call its concrete methods.

The second approach is to design a small kernel of the tool that would include more or less only the extension mechanism and the functionality extract into plugins implemented with the use of the extension mechanism. This solution is naturally more extendable. The Extension mechanism must have enough power to allow the implementation of all the functionality of the tool.

Plugins are written while implementing basic functionality. It allows the extension mechanism to be tested before publishing the interfaces of the extension mechanism to external programmers.

Extracting functionality into plugins typically provides higher re-usability of the source code. The source code of plugins will probably be considerably less dependent on the special purpose classes than corresponding source code in the kernel would be.

The disadvantage of this approach is the higher requirements on the extension mechanism. When solving problems, it is necessary to design a universal API instead of simple tying two classes together. This however has a big risk excessive complexity of the system for the extensions programmer. It is necessary to design the system to automate as many things as possible. As well as this, careful segmentation of the tool to modules and good design of interfaces is essential. Well designed modules allow an extension programmer to study only the part of extension mechanism in which he is most interested.

Conclusion

The architecture with functions in the kernel of the tool is suitable when requirements of the functionality of the tool are exactly given, the re-usability is not the most important and the extending of the tool is exactly specified. In the context of image processing tool, it could be for example only adding the support for new image formats or adding new image processing filters.

For the demands of the analysed tool, the approach with the functionality in the plugins is more fitting.

4.4 Analysis of the Plugins Concept

Plugins types and plugins interfaces

Plugins must allow processing of images, extending the support of image formats, they should also allow the input from the user to be obtained, or display custom user interface component to the user. Plugins should also provide user interface components exploitable in user interface plugins. Next, it would be beneficial, if plugins could also do further general work. Furthermore, it must be possible to integrate a plugin to the menu of the tool.

This list implies that plugins play significantly different roles in the functionality of the tool and consequently could be used in different domains of interest. It would be appropriate to divide plugins into more types with reduced complexity.

In order to allow automatic integration, the plugins of a given type must implement a given interface.

Lifetime and state of the plugins

Assume that each plugin is represented by one object of the class implementing given interface. It must be determined, whether the object of the plugin will be created only once and will live on between its calling or whether the object will be created at each calling.

In the first approach, the object can maintain its state in instance variables and can react differently at each calling. The advantage of this concept is greater flexibility. However, it must be guaranteed, that the instance variables are in the correct state before calling the plugin. If the plugin can be called concurrently, it is also necessary to provide the correct synchronisation of instance variables.

Plugins which are represented by such objects allow realising asynchronous communication in the tool by receiving messages via the mechanism of backward calling. The asynchronous communication is described in Section 4.4.

Due to these benefits, the approach with plugins represented by an object which lives on between its callings was chosen.

It is possible to go even further into the design of the state of the plugins. Previously designed persistent object holds only one state shared by all callers. The caller of the plugin does not know anything about callings of this shared plugin object by other entities and it is consequently not possible to rely on that the plugin in the same state after the previous calling. It would be possible to solve this problem by simulating independent objects by one object of the plugin. The plugins object could assign each caller a unique identifier and could store the independent state for each identifier. The caller could then pass this identifier when calling the plugin.

The extension mechanism of the tool does not contain a direct support of independent states of plugins. However, because the objects representing the plugins live on between callings, it is possible to construct such plugins.

Plugins Communication

It should be possible to call a plugin from any other plugin. A plugin will be identified by its type and the unqualified name of its main class. It is necessary to determine the form of the calling.

The caller will pass the identifier of the plugin or more generally the service he demands and additional information in the parameters. The calling should be mediated by the kernel of the tool. The kernel will choose the correct plugin which offers given service and call it. It can handle additional functions before and after calling the plugin for example creating a new thread for the plugin.

The previous concept provides a unified way of calling the plugins that should be sufficient in most cases. However, it could be also possible to obtain the object representing given plugin and invoke its custom method. This would allow plugins to provide richer interfaces.

With the concept of plugins as permanent objects which was described earlier, there is also possibility to implement indirect, asynchronous communication controlled by events. Consequently, it is possible to create a plugin that will do an arbitrarily complex operation when such an event occurs. This means the plugins programmer has a very strong mechanism in automating the functionality of the tool. The concept of asynchronous communication is developed more in Section 4.6

4.5 Installation, Uninstallation and Updating of Plugins

Due to the fact that plugins can call other plugins, it is necessary to guarantee that pulgins which are needed by another plugins will not be uninstalled. Consequently, the dependencies between plugins need to be defined.

It would be good to allow programmers to divide the implementation of one plugin to more class files. This is easily manageable when plugins will be in jar archives. Detecting the name of the main class of the plugin can be solved simply by introducing the convention that the main class has the same name as the jar archive.

Installation and uninstallation

As plugins will be detected in the start-up of the tool, the installation of a new plugin will consist in detecting the plugins type, checking the dependencies of the plugin and copying it to the given directory if all dependencies were fulfilled.

Uninstallation of the plugin will consist of checking dependencies and deleting the jar file of the plugin.

Updating

allows the existing functionality of the tool to be improved. For example, updated user interface plugin can provide more functionality or a better look and feel.

However, because a plugin is identified only by its type and unqualified name of its main class, the new version of the plugin must have the same interface as the old version and must be usable in the same way.

Hence, updating the plugin will consist only of the uninstallation of the plugin and performing new installation of such a plugin. Besides from this, it is necessary to solve non standard situations caused by plugin dependencies. Different versions of the plugin have different implementations and can call different plugins. It is therefore necessary to check whether the new version of plugin has the unfulfilled dependencies. If it has, then it cannot be installed. If it has not, the old version will be uninstalled. To perform uninstallation, a check of dependencies will be omitted. Thereinafter, the new version of the plugin will be installed.

Further issues

It can be necessary to perform other actions while installing and uninstalling specific types of plugins. For example integrating a plugin to the menus of the tool or removing the plugin from the menus.

Unexpectedly terminating the tool while performing installation, uninstallation and especially updating the plugins can lead to a not well defined state of tool. Therefore recovery from these states should be performed.

4.6 Functions of the Kernel of the Tool

The rest of the section provides an overview of the basic demands on the functionality of the kernel of the tool.

Managing plugins

The kernel must provide loading of plugins and calling of plugins. Calling of the plugins is analysed in Chapter 4.4.

Shared data structures

It is necessary to provide access to the image which the user is currently viewing or to the directory which the user is currently browsing. This data will be stored in shared data structures in the kernel of the program. Plugins should also have the possibility to react to changes in such structures via the mechanism of backward calling.

Hence, arbitrary complex action can be automatically performed when shared data structure is changed. This gives the possibility to implement the architecture of the tool with functionality in the plugins. The particular solution is described in Chapter 7.

Managing menus

The kernel must provide the ability to obtain the data needed for creating menus of the tool and provide the interface for accessing it.

The data can be stored in XML text files. This enables the automated integration of plugins to the tool menus. A plugin specifies to which place on which menu it should be placed and it will be automatically integrated to the specified menus by changing the given text files. If these files are accessible by plugins, the integration can be done by plugins.

API of the tool should provide automated creating of the menu component of given GUI toolkit from menu data. Once again, this functionality can be provided by plugins.

Batch processing

In general, batch processing includes obtaining the parameters of processing, obtaining the plugin that provides the given processing function, and processing each image of a given set of images with use of this plugin.

All this functionality can be provided by plugins. However, the kernel of the tool should offer the data structure that keeps to the parameters of batch processing.

Other plugins should for example provide user interface component that allows obtaining the batch processing data.

Localisation of the tool

It would be advantageous to store localised strings in text files. This would allow easy localisation of the tool. The kernel should manage the initialisation of the localisation and offer method for getting localised strings.

The problem is, that because of potential collisions, the automatic modification of localisation files is barely manageable. Consequently, programmers of plugins which will be not distributed with the tool and will be installed by the user will not be able to use the localisation provided by the tool and will have to solve it themselves.

Further functionality

The kernel must include the data structure for working with the image. Kernel should also include classes, which can make the programming of the functionality of the tool easier, such as utility classes, user interface components, or various data structures.

4.7 Programming Language

One of the main demands on the programming language and used libraries was the support of multiple platforms. Also because of plugins concept, the straightforward support of loading classes unknown during the compilation was very important. As well it was necessary to implement relatively complex functionality in short time, so programming languages with garbage collector were preferred.

Java programming language was chosen. It enables to create code which runs wide spectrum of platforms and it provides the Reflection API, or the support of multi-threading and synchronisation. Moreover, it has garbage collector and provides libraries with rich functionality including the collections framework or the support of Observer design pattern. It is possible to call native code compiled for given platform that enables particularly optimalisation of the computationally costly operations with big image data.

4.8 Other Plugins Mechanisms

The rest of the section provides an overview of two plugin mechanisms in Java. It will be discussed, whether it would be appropriate to adopt the mentioned mechanisms in the tool. However, there are more plugin mechanisms available. For example the JPlugin [6].

Extension mechanism of ImageJ

ImageJ [5] is a public domain Java image processing tool inspired by NIH Image for the Macintosh designed to process medical images.

The tool has a powerful and easy to use extension mechanism specialised for extending the image processing program. These declare as many interesting plugins written for ImageJ. There is even possibility to use ImageJ library outside ImageJ. The advantage of this approach is particularly the adoption of an existing plugin mechanism and reuse of the plugins written for ImageJ.

On the other hand, it was put emphasis on distinctly different demands while designing ImageJ, the tool for processing medical images, than while designing this tool. There is no support of browsing and viewing images, no support of batch processing, there is not sufficient support of integrating plugins to the menu, the support of user interface is not satisfactory enough.

Consequently, it would not be possible to use this plugin mechanism directly. While the plugins concept is nice and it is easy to write ImageJ plugins, the sources of kernel classes are highly complicated and sometimes badly designed, so it would be complicated to modify and extend ImageJ's plugin mechanism.

ImageJ classes are also tightly knotted with ImageJ image processing engine. This engine is relatively efficient and certainly noteworthy, but it does not offer such complexity as dedicated processing engines such as JAI [13].

It was decided not to adopt the extension mechanism of ImageJ. However, the plugins concept of ImageJ strongly involved the design of the tool.

Java Plugin Framework

Java Plugin Framework [7] provides a runtime engine that dynamically discovers and loads plugins. Using JPF directly would be not suitable, because Java Plugin Framework is general framework and therefore cannot offer desirable degree of automation. Nevertheless, it would be possible to use JPF to implement more specialised plugin mechanism that would satisfy the demands stated earlier.

4.9 Image Processing in the Tool

Library for image processing

Photographs include millions of pixels and their processing can easily became a bottle neck. Consequently, one of the most important characteristics of image processing library is efficiency. Next important characteristics are easy to use and the amount of included image processing functions.

JAI [13] image processing library was chosen. It supports accelerating of common image processing operations using native code of given platform, it offers wide spectrum of functions, and it has presentable documentation. Furthermore, it has wide community of users and it is supported by the Sun. However, JAI is not part of standard Java distribution, so it is necessary to distribute it independently. JAI library is relatively complex and plugins programmers should not be forced to use it. There should be the possibility to use less complex image processing solutions, especially AWT imaging which is standard part of Java. On the other hand, the usage of JAI should be as easy as it is possible.

The image data structure

The image data structure must primarily store the image data. Then, it must store the information about selected areas in the image – regions of interest. The basic support of regions of interest includes the possibility of adding new regions of interest to an image and getting information about regions of interest from an image. Next issues include the support of modifying existing regions of interest by user. Then, there must be solved the problem how to modify regions of interests when there was performed geometrical modification of the image. Next, plugins would have the possibility to observe the regions of interest.

Other demands on the image data structure is storing additional information like EXIF [8] information.

Chapter 5

Analysis of Selected Image Processing Functions

The image processing functions that the tool should support were mentioned in Section 2.1. In this chapter it is described in more detail the function of noise cleaning and the function of exposure bending.

5.1 Noise Cleaning

In [9] is pointed:

> An image may be subject to noise and interference from several sources, including electrical sensor noise, photographic grain noise, and channel errors.

Consequently, an undesirable noise, perceptible particularly on photographs of night scenes where setting long exposure time and high sensibility is necessary, is one of the most common defects of digital photographs. Hence, it is appropriate to offer functions for noise suppression.

Noise can be reduced by statistical filtering techniques or the application of ad-hoc noise cleaning techniques. The statistical filtering is suitable particularly if the additional information about the distortion of the image is known. More information about statistical filtering is in [9].

If no information about the distortion of an image is available, noise cleaning techniques are usually more congruous. Many noise cleaning techniques stands on the observation, that image noise caused by a noisy sensor or channel transmission errors usually consists of isolated pixels that are not spatially correlated. Consequently, erroneous pixels are markedly different from their neighbours.

Linear noise cleaning

Owing to the fact, that an image noise is not spatially correlated, it usually has a higher spatial frequency spectrum than the other image elements. Consequently, the noise can be reduced by simply removing the high frequencies. This can be done by the convolution or equivalently in the Fourier domain by low pass filter. In the discrete case, the convolution of the image I with M rows and N columns, and the kernel K with J rows and K columns is given by:

$$O[m,n] = \sum_{j=1}^{J} \sum_{k=1}^{K} I[m-j, n-k] K[j,k]$$

Information about convolution and low pass filter are in [9].

Unfortunately, high frequencies also contain the information about the edges of an image. Consequently, linear noise cleaning techniques tend to smooth the image highly. This is discussed in [9].

Nonlinear noise cleaning

Nonlinear techniques aim to not blur the edges of images and to provide better compromise between noise smoothing and not loosing of image detail.

The example of nonlinear cleaning technique is **outlier**. Each pixel is compared to the average of its eight neighbours. If the magnitude of the difference is greater than some threshold level, the pixel is judged noisy, and it is replaced by its neighbourhood average.

The next example is the **median filter**. In the one dimensional case, median filter is performed using a sliding window consisting of an odd number of samples. The centre pixel in the window is replaced by the median of the pixels in the window.

The **bilateral filter** works like convolution based filters in the areas with constant colour, but it lowers the contribution of pixels whose values differ greatly from the centre sample.

The last discussed filter is the **rotating mask filter**. The neighbourhood of the pixel is divided into sub-regions. For each sub-region is computed the mean and dispersion. The output value is the mean of the sub-region with minimal dispersion.

Noise cleaning technique present in the tool

The implementation of median filter was integrated to the tool. It is simple and consequently easy to implement, and it gives relatively good results in suppressing the noise of digital cameras. The capabilities of median filter are discussed in [9].

The tool enables to determine the level of noise cleaning by choosing the size of the window. The median filter is also applicable to the group of images.

The further improvement of median filter implementation in the tool could include better support for adjusting the window of the median filter – user would specify the initial size of the window. Then, the preview of the filtered image would be displayed. If the filtering was insufficient, user could press the “More filtering” button, which would enlarge the size of the mask and display the preview. If the filtering was excessive, user could press the “Less filtering” button, which would decrease the size of the mask and display the preview.

5.2 Exposure Bending

The exposure bending technique enables combining of photographs of the same object taken with different exposures into a single photography with higher dynamic range. The motivation and using of exposure bending is described in Section 6.8.

Two types of limitations of the dynamic range have to be considered. The first one is the limitation of the dynamic range of the camera. The second one the limitation of the dynamic range of the resulting image and common monitors and printers. Because of the limitation of resulting image, the dynamic range increase must be defined as the process of correctly reproducing the highlights and shadows of a high dynamic range scene in the resulting image.

It was considered, that the exposure bending method provided by the tool should aim to add the information missing in the underexposed or overexposed areas in the image, not to distort the details of other parts of image, and to preserve natural transitions between the dark and bright areas. By contrast, mapping of high dynamic range image into the image with substantially lower dynamic range image must lead to distortion of details. Moreover, the results of such mapping can look unnaturally. Consequently, the exposure bending method provided by the tool should correct primarily the limitations of the camera. Because the dynamic range of two photographs of high dynamic range scene – one well-exposed in dark areas and overexposed in bright areas and the second one well-exposed in bright areas and underexposed in dark areas – is usually higher than the dynamic range of 8 bit image, bending of only two images is considered.

Assume that such two images – *image one*, well-exposed in the bright areas of the scene and underexposed in the dark areas and *image two*, well-exposed in the dark areas and overexposed in the bright areas – should be bended into enhanced one.

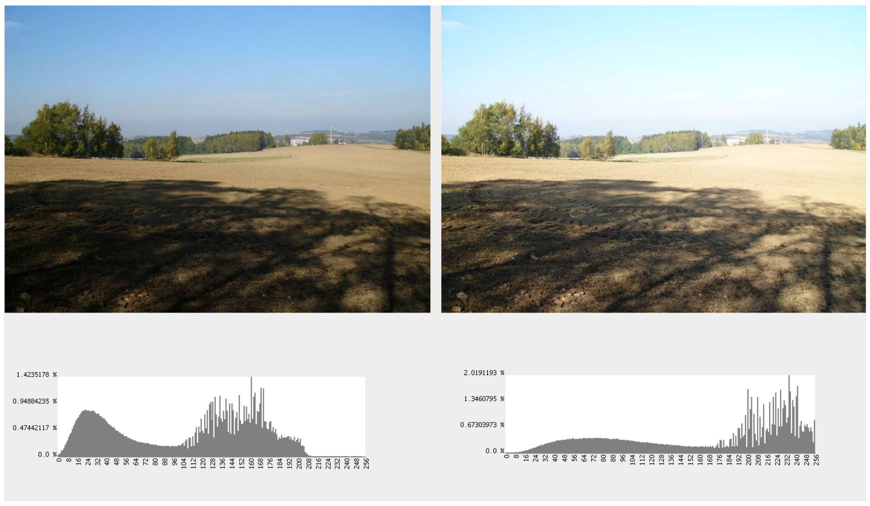


Figure 5.1: *Image one* is well-exposed in the bright areas of the scene and underexposed in the dark areas (left); *Image two* is well-exposed in the dark areas and overexposed in the bright areas (right).

Averaging

If two images mentioned earlier are averaged, the information from *image one* will appear in the bright areas of the resulting image and the information from *image two* will appear in the dark areas.

Unfortunately, this information will be distorted by mostly irrelevant information from the other image. Consequently, the contrast in the image will be decreased. Moreover, resulting image will be highly blured in areas where two images are not correctly fitted.

Dynamically weighted averaging

Dynamically weighted averaging assigns higher weight to the pixels of this image that is well-exposed in given area. It therefore suppresses both the loss of contrast and blurring in unfitted areas.

Mask of the image

The weights of the pixels of averaged images can be given by a mask. The mask of the image is a grey scale image with the same dimension as the image. The brightness value of a given pixel in the mask determines the transparency of the corresponding pixel in the image. Black colour means that the pixel is fully transparent, white colour means, that the pixel is opaque. If the mask is applied to one image and the second image is placed under this image, these two images will be averaged with weights given by the brightness values of the mask.

Masks for weighted averaging

Hence, the mask should hold the information where the images are well-exposed. One possible approach is to convert *image two* to grey scale image and the resulting image apply as a mask to *image one. Image two* is overexposed in the bright areas so the mask will be white or nearly white in such areas. Consequently, in such areas will be *image one* opaque. The mask will be darker in the dark areas and therefore, *image one* – underexposed in such areas – will be more transparent there. Finally, *image two* will be placed under the *image one.*

This approach almost eliminates the loss of the contrast and blurring in the bright areas of the scene. The suppression of such effects is smaller in the dark areas and particularly in the areas with middle brightness. The effect of reduction of contrast and blurring resulting image could be reduced by modification of the brightness of the mask using for example curves tool described in Section 6.7.

Another approach should be to create the mask as the average of the brightness values of bended images. The effect of the reduction of the contrast and blurring the images can be then suppressed by enhancing the contrast of such mask.

Blurring the mask

Simply applying masks presented earlier usually does not give good results. Such masks induce too sharp transitions between the images in the borders between dark and bright areas. This causes the creation of undesirable artifacts and blurring of resulting image in such areas if bended images are not exactly fitted.

It is therefore necessary to blur the mask and make such transitions smoother. Gauss blurring is fitting for this purpose. Higher blurring radius is needed when the images are worse fitted and there are sharp borders between dark and bright areas. The blurring radius of 120 gives usually good results in most cases.

Thresholding

The process of exposure bending with use of thresholding is based on the idea that *image two* is well-exposed in the dark areas and in the middle tones, but overexposed in the bright areas and would be enhanced by adding the information in the bright areas from *image one*. Similar technique is used for bending photographs taken with the analogue camera and photography film.

The bending process used in this technique is special case of dynamically weighted averaging. Mask representing the weights of bended images is obtained by the thresholding operation. *Image two* is thresholded into two levels. Parts of the *image two* that are correctly exposed will be in the mask black and parts of *image two* that are overexposed will be in the mask white. This mask will be applied to *image two*. Well-exposed areas will be opaque and consequently not affected by *image one*. Overexposed areas will be fully transparent, so only *image one* will be visible there.

Naturally, this procedure would create sharp transitions between bended images. Because of different exposition, these transitions would look unnaturally. Moreover, if the images were not correctly fitted, there would appear undesirable artifacts in such transitions. It is therefore necessary to blur the black and white mask highly. The resulting mask will be grey near transitions between black and white colour and the latter effect will be suppressed.

The advantage of thresholding technique is, that almost only the areas in the transitions between bended images are influenced. Other areas are not affected neither by the effect of blurring nor by the effect of reduction of the contrast.

Consequently, the method of thresholding is well suitable for bending more than two images. The areas with middle brightness could be taken from one image, the dark areas could be taken from the second image, and the bright areas could be taken from the third image.

The disadvantage is, that the transitions between bended images can look less naturally than while using masks mentioned earlier. This effect is even worse when specifying unsuitable threshold level and insufficient blurring radius.

Next disadvantage is that this method cannot be easily fully automated. User must specify the thresholding level. Even though specifying the thresholding level is intuitive – a user specifies what information will be taken from one image and what information from the other – and with use of the preview of the thresholding operation would be probably manageable for most of the users, it requires certain effort and also some practise to get best results.

HDR exposure bending

In a HDR (High Dynamic Range) image is possible to store the information about full dynamic range of the scene.

According to [10], the exposure bending process with the use of HDR image consists of merging bended photographs into the HDR image and tone mapping. The tone mapping process compresses the tonal range of an HDR image of the scene in order to reveal its details in highlights and shadows.

HDR exposure bending is significantly less straightforward and thus more difficult to implement than the techniques mentioned earlier. Moreover, it assumes, that the dynamic range of the resulting image is smaller than the dynamic range of the merged image. Consequently, it is beneficial particularly when bending many images.

The method implemented in the tool

The method of dynamically weighted averaging using the mask taken from the brighter image applied to the darker image was implemented to the tool.

It produces naturally looking images for most of the scenes and it does not require specifying any input from a user. The effect of loss of the contrast or blurring in the dark areas and the areas with middle brightness is usually not noticeable. Moreover, this effect is usually less intrusive than unnaturally looking transitions between bended images caused by improperly chosen thresholding level and blur radius while using thresholding. Last but not least, implementation of this method is relatively easy.

Possible improvements

The method of creating the mask as the average of the brightness values of bended images and enhancing its contrast would probably give better results. However, it was not implemented from capacity reasons and its implementation was left for future work.

Next, the thresholding exposure bending method can be implemented. Even though yet implemented method gives good results while bending two images and it is more suitable for beginners, the thresholding method can give better results in some cases. Moreover, the thresholding method is more suitable for bending more than two images and it would be good to offer this functionality to advanced users.

The possibility of manipulating with the brightness of the mask using the curves tool can be considered. In fact, with adjusting the mask using the curves tool, it is possible to get even the thresholding mask. However, manipulating with the mask using the curves tool is probably too laborious and maybe blind to users. Users would have to know what the mask is and to what image it will be applied. Hence, the asset of manipulating with the mask using the curves tool might be verified before offering such functionality to users.

Chapter 6

Users Guide

6.1 Installation

Package content

PhotoJ **binary package** is suitable for users that want to run the tool as well for plugin developers. It contains:

- Jar files of the program providing native acceleration on Windows and Linux, and jar file of the program without the native acceleration runnable on all the platforms where JRE is accessible.
- Runnable scripts for Windows and Linux.
- JAI [13] library. Versions with the support of native acceleration on Windows and Linux, and version without the support of native acceleration.

PhotoJ **source files package** is suitable for users that want to compile the tool and for the develepers of the kernel of the tool. It contains:

- Source files of the tool.
- Ant scripts for compilation of the tool with the support of native acceleration on Windows and Linux, and without the support of native acceleration.
- Runnable scripts for Windows and Linux (need appropriate jar files to be created)
- JAI [13] library. Versions with the support of native acceleration on Windows and Linux, and version without the support of native acceleration.

System Requirements

The tool requires Java JRE SE version 1.6. It can be downloaded from [11]. It is expected, that tool will run with newer versions as well.

Installation

To install the tool, unpack the archive with the tool. If archive with binary package was chosen, follow the instructions in Section 6.2.

Compilation

There are prepared three Ant [12] scripts in the buildfiles directory: `build_windows.xml` provides compilation with the support of accelerated JAI operations on Windows, `build_linux.xml` provides compilation with the support of accelerated JAI operations on Linux, and `build_no_acceleration.xml` compiles the program with non-acclelerated version of JAI, which runs in all platforms, where JRE is available. To compile the tool with the support of acceleration in other platforms, new ant script needs to be created.

To compile the program, copy chosen ant file from directory `buildfiles/` to main directory of the program and run:

```
$ ant
```

To create jar files, run:

```
$ ant jar
```

According to buildfile that was chosen, file `photoj.jar`, `photoj_windows.jar`, or `photoj_linux.jar` will be created and all the plugins will be packed into its own package.

To display all options of given ant script, run:

```
$ ant -v -projecthelp
```

6.2 Running the Program

Windows: Go to the directory where the tool was unpacked and enter:

```
$ photoj.vbs
```

Linux: Go to the directory where the tool was unpacked and enter:

```
$ sh ./photoj.sh
```

Other platforms: Go to the directory where the tool was unpacked and enter:

```
$ java -Xmx256m -jar photoj.jar
```

Supported arguments

```
[path to a directory or image file] [interpolation quality]
```

path to directory or image file If a directory is passed, the tool will start browsing given directory.

If an image file is inserted, the tool will start viewing given file.

If no path is inserted, the tool will start browsing the last directory when it was while previous running the tool. If the program is started for the first time, it will start browsing in the home directory of the user.

interpolation quality It is possible to specify an interpolation quality used for given types of processing. The quality is given by the numbers from one to three. One is the lowest quality, three is the highest quality.

The interpolation quality settings have following syntax:

```
[permanent quality
  [display quality
    [preview quality
      [default quality]]]]
```

permanent quality The quality of interpolation of image operations applied to the images that can be saved to disk or can have an impact to further operations.

display quality The quality of interpolation of image operations that transforms an image in order to display the image to the user. It is for example zooming of the image.

preview quality The quality of interpolation used when performing preview of operations.

default quality The quality of interpolation used when it is not specified whether the operation is permanent, whether it serves to display image, or whether it serves for preview of the operation.

Setting the amount of RAM memory

It is possible to change the amount of RAM memory accessible to the tool. Setting at least 256MB of memory is recommended.

If `photoj.vbs` or `photoj.sh` is used to start the tool, change value of the item `memory` in `photoj.vbs` or `photoj.sh`. If java is used directly to start the tool, change the value `256` in `-Xmx256m` argument.

6.3 Browsing Filesystem

Filesystem can be browsed in a browsing mode. If there are images in current directory, the thumbnails of images are displayed. By clicking on the thumbnail of given image, the viewing mode is started. The following operations are accessible from the menu of the browsing mode window:

File menu:

- *Exit* – quits the tool.

Tools menu:

- *Exposure bending* – runs the exposure bending tool. Exposure bending is described in Section 6.8. Using exposure bending in the tool is described in Section 6.9.
- *Image fitting* – runs the image fitting tool. Image fitting tool enables to fit two images. This is useful when bending the images. Fitting images is described in Section 6.9.

Plugins menu:

- *Install plugin* – allows installing new plugin and updating existing plugin.

  The jar file of the plugin that should be installed can be chosen using displayed dialog. Next, the description of the plugin is displayed and it is possible to confirm or cancel the installation. After confirming, he installation begins. If the plugin is already installed, it is possible to update it.
- *Uninstall plugin* – allows listing installed plugins and uninstalling of such plugins.

  To uninstall the plugin, select the plugin's type and than select the plugin. Description of the plugin is displayed. The plugin can be uninstalled by pressing uninstall button. After doing so, it is checked, whether the uninstallation of the plugin would not violate any dependencies. If it would not, the plugin will be uninstalled.

Batch processing menu:

- *Rotate* – enables rotating all the images in current directory by an arbitrary angle.
- *Rescale* – provides rescaling all the images in current directory to given dimension.
- *Noise suppression* – enables applying noise suppression filters to all the images in current directory.
  - *Median filter* – provides the median noise suppression filter. The level of noise suppression can be set by adjusting mask size. Setting higher values will cause greater noise suppression, but also greater image distortion.

6.4 Viewing Image

An image can be viewed in a viewing mode. The following operations are accessible from the menu of the viewing mode window:

File menu:

- *Save* – saves changes to currently viewed image.
- *Save as* – saves currently viewed image in specified image format to specified directory and switches to this image and this directory.
- *Exit* – quits the tool.

Edit menu:

- *Undo* – performs an undo of previous operation.
- *Redo* – perfroms a redo of previous undo.

View menu:

- *Next image* – starts viewing next image in the directory.
- *Previous image* – starts viewing previous image in the directory.
- *Zoom in* – zooms in the image.
- *Zoom out* – zooms out the image.
- *Fit viewing window* – rescales the image to fill the area of the viewing window.
- *Original scale* – displays the image in the original scale.
- *Toggle full screen* – toggles full screen mode.
- *Browsing mode* – switches to the browsing mode.

Selections menu:

- *Lock in selecting / Unlock selecting* – locking in selecting disables modification of selections. Unlocking selecting enables the modification of selections.

Transform menu:

- *Rotate left* – rotates the image left with the angle of 90 degrees.
- *Rotate right* – rotates the image right with the angle of 90 degrees.
- *Rotate* – rotates the image with arbitrary angle. Provides the preview of the rotation and enables applying this operation in batch to all images in actual directory.
- *Rescale* – rescales the image to the given dimension. Enables applying this operation in batch to all images in actual directory.

- *Crop* – crops the image. The cropping area is defined via rectangular selection. If there is some rectangular selection in the image, the cropping area will be initialised with dimension of this selection. If there is no rectangular selection in the image, new rectangular selection will be created.

  If the selections is unlocked (see the menu item Selections – *Lock in selections / Unlock selections*), it is possible to change cropping area by manipulating with the selection.

  The cropping area can be set also by adjusting values of the spinners. The selection is automatically changed to correspond to entered values. Width and height spinners can be linked to keep the same ratio while changing one of the spinner. X offset and Y offset spinners can be also linked. Linking offset spinners arranges adding the same value to both spinners.

Enhance operations menu:

- *Brightness up* – heightens the brightness of the image.
- *Brightness down* – lowers the brightness of the image.
- *Brightness and contrast* – adjusts the brightness and contrast of the image.
- *Histogram* – displays the histogram of brightness of the image. Interpreting histogram is described in Section 6.6.
- *Curves* – runs the curves tool. Working with the curves tool is described in Section 6.7.
- *Noise suppression* – provides suppressing the noise in the image.
  - *Median filter* – starts median noise suppression filter. The level of noise suppression can be set by adjusting mask size. Setting higher values will cause greater noise suppression, but also greater image distortion.

6.5 Using Batch Processing

Batch processing allows applying given operation to all images in actual directory. Processed images will be saved to specified directory. Images in this directory with the same name will be rewritten.

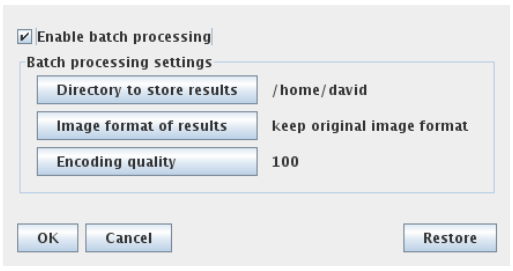
Batch processing settings

Batch processing panel offers following settings:

Directory to store results The directory in which processed images will be stored.

Image format of results The file format of processed images. If the file format is not specified, the tool assigns each processed image the file format of processing image.

Encoding quality The quality used while encoding images to given file format. This setting has great importance when choosing lossy file formats like JPEG. However, it can have the sense even for non lossy formats such as PNG.



✓ Enable batch processing
Batch processing settings
Directory to store results /home/david
Image format of results keep original image format
Encoding quality 100

Figure 6.1: Batch processing settings.

After confirming the batch processing operation, the batch processing will be started. The progress of batch processing is showed on the progress bar.

Running batch processing in the browsing or processing mode

Batch processing is currently supported by operation of Rotation, Rescaling and Median filter. This operations are accessible both from the browsing mode and from the viewing mode.

The behaviour of dialogs that enables settings of these operations can vary when they are run from the browsing mode or from the viewing mode. For example, when setting parameters of Rotation operation, there is preview available in the viewing mode. Next, batch processing can be activated or deactivated, when running these operations from the viewing mode.

6.6 Interpreting Image Histogram

The example of image histogram is showed in Figure 6.2. In the horizontal axis of the histogram are displayed the values of the brightness. In the vertical axis is displayed the proportion of number of pixels of given brightness value on the total number of pixels in the image.

Dark images have a majority of values on the left side, bright images have a majority of values on the right side.

It can be guessed, whether the image is underexposed or overexposed from the histogram. Big areas of white colour caused by overexposing image are indicated by big number of pixels with in the last bin of the histogram, big areas of black

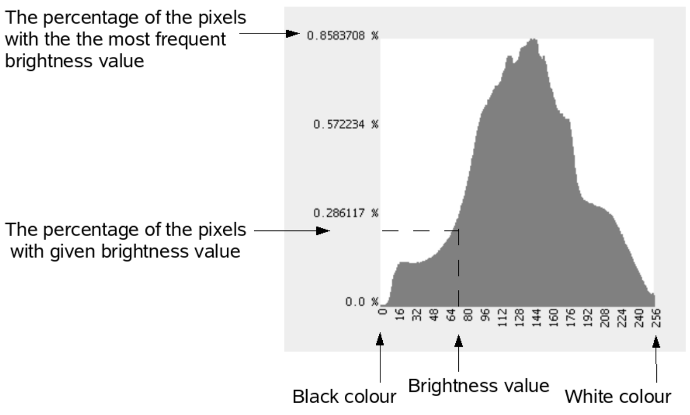


Figure 6.2: The histogram of the image.

colour caused by underexposing image are indicated by big number of pixels in the first bin of histogram. This effect is visible on both images in Figure 6.3. Note that these areas do not contain any further information and cannot be restored by histogram modification. One mean of indicating distorted image should by percentage of pixels with brightness value of 0 and 256.

Image with low contrast is indicated by narrow histogram. If there are no or very little values on the left or right side of the histogram, the contrast of the image can be easily enhanced by stretching the contrast. Stretching the contrast is described in the following section.

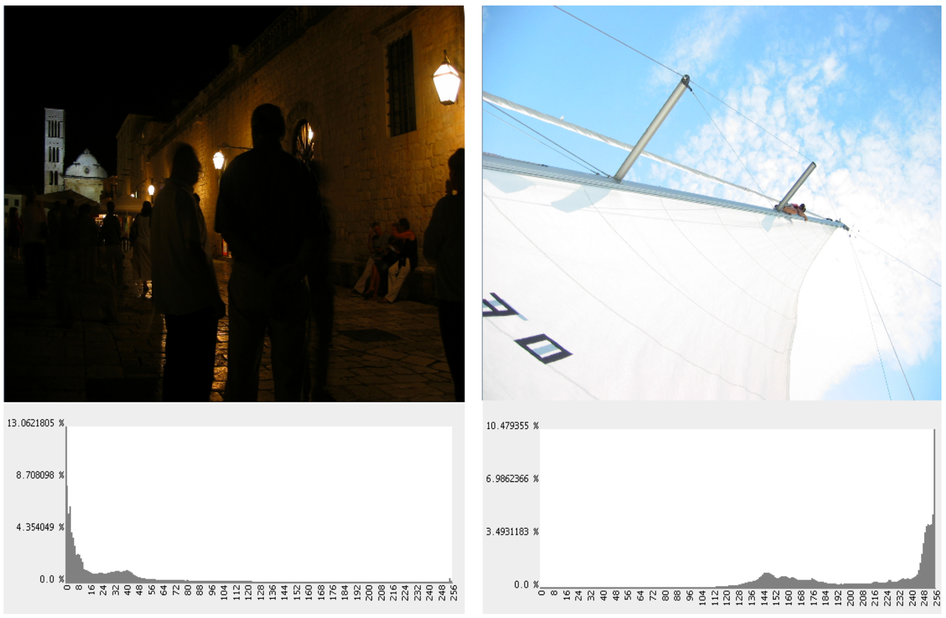


Figure 6.3: Histogram of a dark image (left) and the histogram of a bright image (right).

6.7 Manipulation with Brightness Levels

The mapping of brightness levels in the input image to the brightness levels in the output image can be specified using the curves tool.

Such mapping is realised by the transfer function. In the x coordinate are brightness values of the input image. On the left side is white colour, on the right side is black colour. The value of transfer function specifies the output brightness value of pixels with input brightness value given by x coordinate.

The transfer function can be specified by placing points that the transfer function must intersect. The curves tool is showed in Figure 6.4.

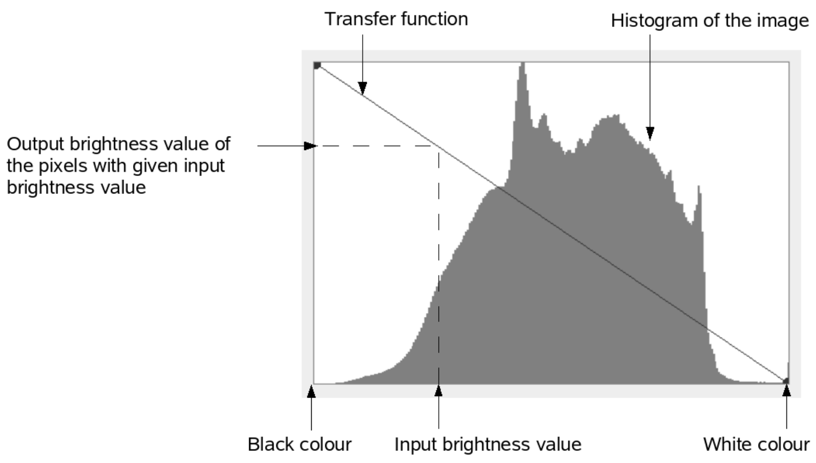


Figure 6.4: The curves tool.

Inverting image

In figure 6.5 is showed the transfer function that does not affect the brightness values of pixel and the transfer function that inverts the brightness values of the image. The lowest brightness value in the x coordinate – white colour – is mapped to the highest value in the y coordinate – black colour.

Simple contrast enhancement

If the image has no pixels with high or low brightness, the contrast of such image can be easily enhanced without the loss of information. Such transfer function is shown in Figure 6.6. This transfer function has its left point moved right and right point moved left. It maps all pixels less than 100 to black colour and all pixels with brightness value greater than 245 to white colour. Other brightness values are linearly stretched within whole dynamic range. The histogram of enhanced image is wider, which indicates higher contrast.

There is observable from the histogram, that the output image has unoccupied brightness levels within its range, and some of the brightness transitions are

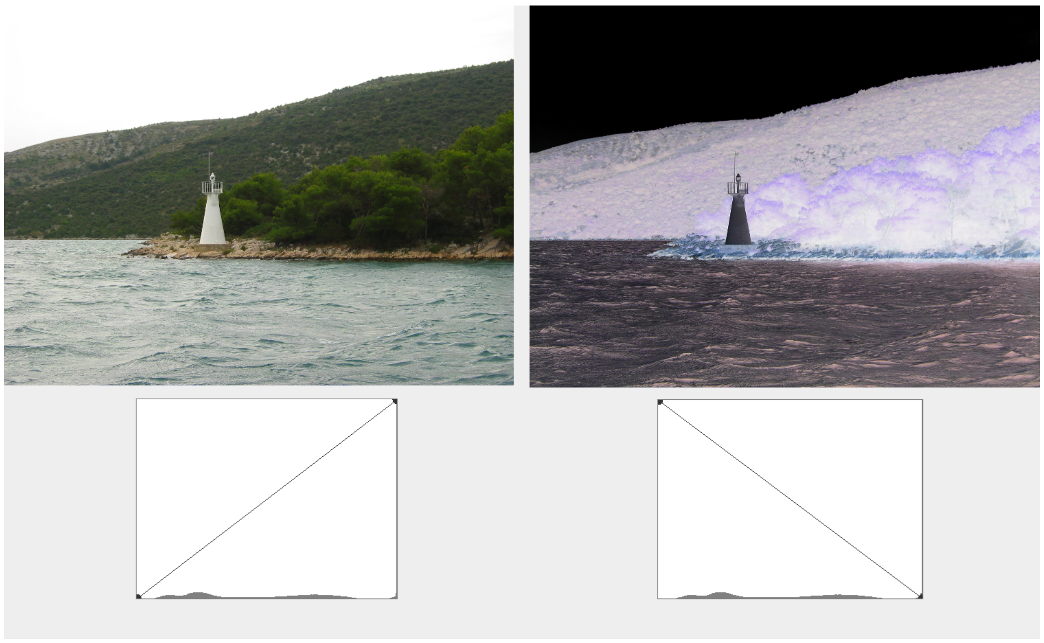


Figure 6.5: The transfer functions of original image (left) and the transfer function of the inverted image (right).

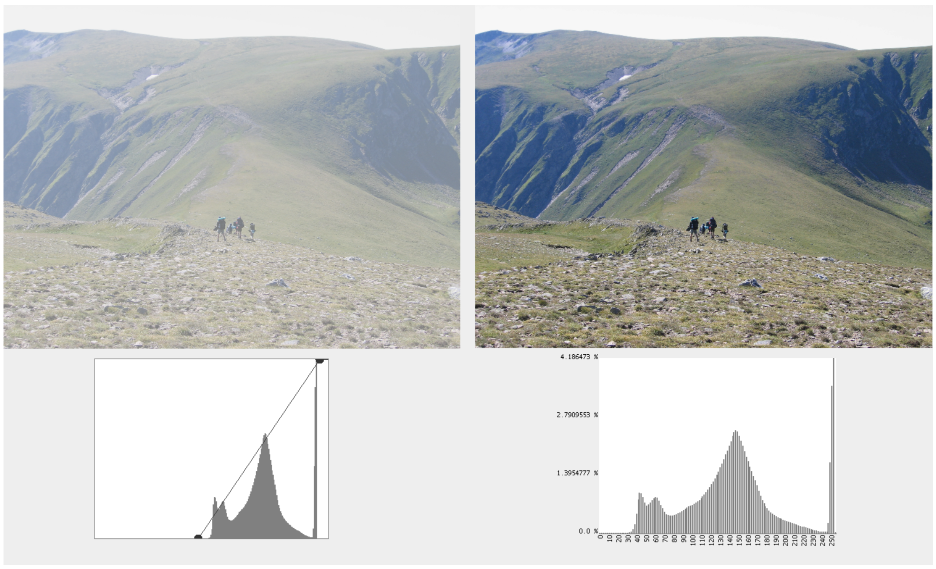


Figure 6.6: Simple contrast enhancement: Low contrast image and the enhancement transfer function (left); Enhanced image and its histogram (right).

larger than in the original image. This effect may result in noticeable grey scale contouring when the stretching is big.

The contrast was possible to enhance without the loss of information, because there were no brightness values less than 100 and higher than 245. Hence, similar transfer function can be useful for partial correction of highly over- or under- exposed images or for enhancing contrast of photographs taken from mobile phones which sensors often produce low contrast images with only few pixels with high or low brightness.

Contrast stretching

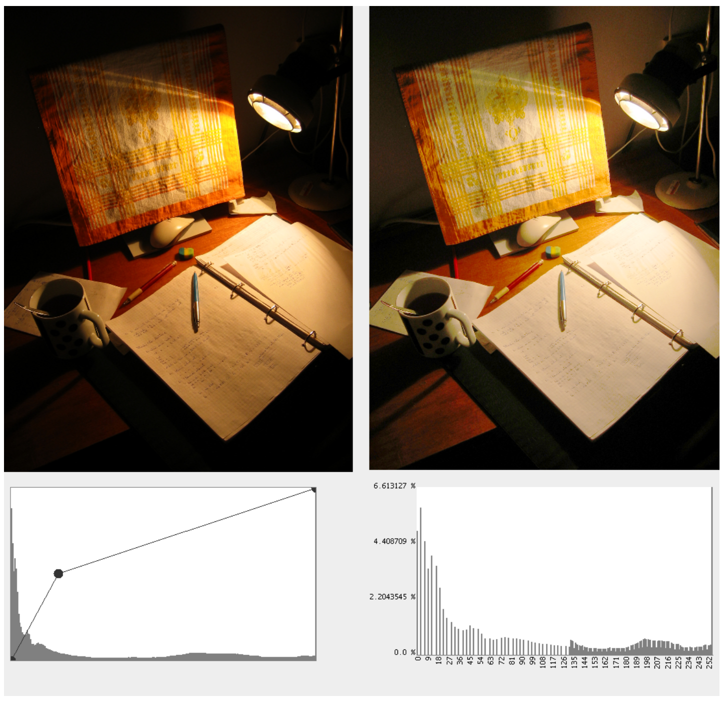


Figure 6.7: Contrast stretching: Original image and the enhancement transfer function (left); Image with stretched contrast in dark areas and its histogram (right).

The contrast of given brightness interval can be stretched by specifying the transfer function highly-pitched than average. Naturally, there must be parts where the transfer function is less high-pitched than average and where the contrast is contracted. The example of such contrast stretching is shown in Figure 6.7.

In fact, simple stretching of contrast was showed also in Figure 6.6. The transfer function is horizontal in the first interval, than highly-pitched than average in the second interval at than again horizontal.

Unfortunately, all pixels in the horizontal interval are mapped to one brightness value and the information about its difference and consequently about the details in such areas is lost. It is therefore good to set the transfer function horizontal only in the intervals with insignificant number of pixels.

Enhancing image

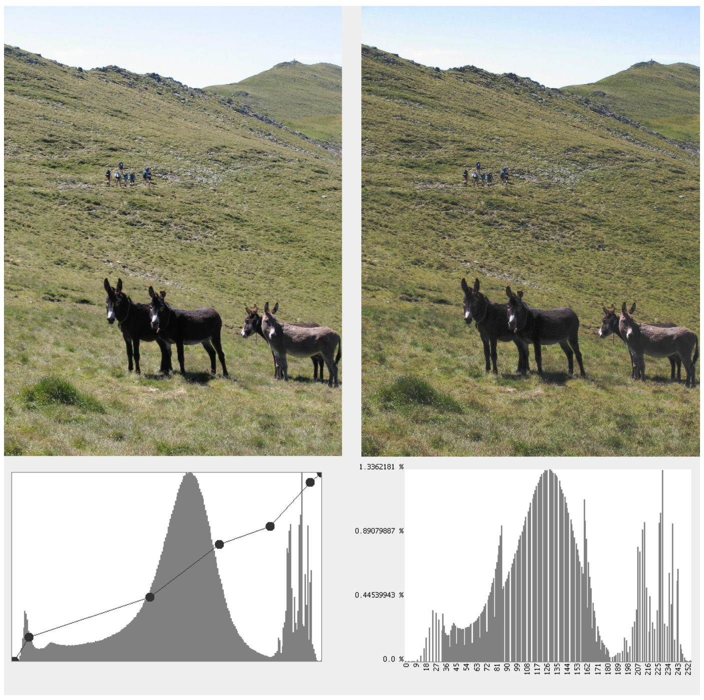


Figure 6.8: More complex contrast enhancement transfer function: Original image and the enhancement transfer function (left); Enhanced image and its histogram (right).

In [9] is pointed:

> The luminance histogram of a typical natural scene is usually highly skewed toward the darker levels; a majority of the pixel possess a luminance less than average. In such images, details in the darker regions is often not perceptible.

These details become better perceptible by stretching the contrast there. Generally, the brightness intervals, where there are a lot of pixels should be stretched and the brightness intervals with little pixels should be contracted. This means that the transfer function should be highly pitched than average in the intervals where the bins are high, lower pitched than average where the bins are low and horizontal in the intervals where there are no pixels.

The histogram of resultant image should be therefore more equalised. To reach such aim, the preview of the histogram of resultant image can be displayed by pressing Preview histogram button.

The example of such enhancement is showed in Figure 6.8.

Generating contrast enhancement transfer functions

It exist the techniques for automatic generating of transfer functions that enhance the contrast of the image. The term that is coined to denote such techniques is Histogram modification.

Generally, these techniques are based on the principle mentioned earlier. The contrast of brightness intervals where is high number of pixels is stretched and the contrast of brightness intervals where is few pixels is contracted.

The example of such technique is histogram equalisation. Histogram equalisation seeks to create such transfer function which transforms the brightness values of the pixels in that way, that all brightness values in the resultant image has the same number of pixels. Resultant images are usually not visually appealing, but have more details visible. This technique is therefore exploitable when extracting some information from the image – this is important for example in medical research. More information about histogram equalisation can be found in [9].

However, there are also similar techniques designed to improve the visual appearance of images. Thus, such techniques are exploitable for automatic enhancing of digital photographs.

6.8 Exposure Bending

Exposure bending serves to manual combining of photographs of the same object taken with different exposures into a single photograph with higher dynamic range.

Dynamic range of the scene

Dynamic range of the scene is the range of the brightness values of the scene. The scene with high dynamic range has high spectrum of brightness values. Typical scene with high dynamic range is the scene which one part is in the shadow and one part on the sunshine. The examples of scenes with extremely high dynamic range are a sunset or a sunrise. The sun is very bright and the other parts of the scene are relatively dark.

Photographing the scenes with high dynamic range

The problem is, that the dynamic range of the camera's sensor is limited. Only relatively narrow spectrum of dynamic range of the scene can be caught. If the scene has higher dynamic range than the sensor of camera and the exposition is adjusted according to the area on the shadow on the scene, the areas on the sunshine will be overexposed. The bright areas will appear like white in the resulting image and details in such areas will be lost. This is indicated by high number of pixels with the highest brightness value on the histogram. On the other hand, if the exposition is adjusted according to the areas on the sunshine, the areas on the shadow will be underexposed.

The example of such scene is in Figure 6.9. On the right side of the figure is the photography well-exposed in the bright areas, but underexposed in the areas in the shadow. On the left side of the figure is the photography well-exposed in the area in the shadow, but overexposed on the bright area.

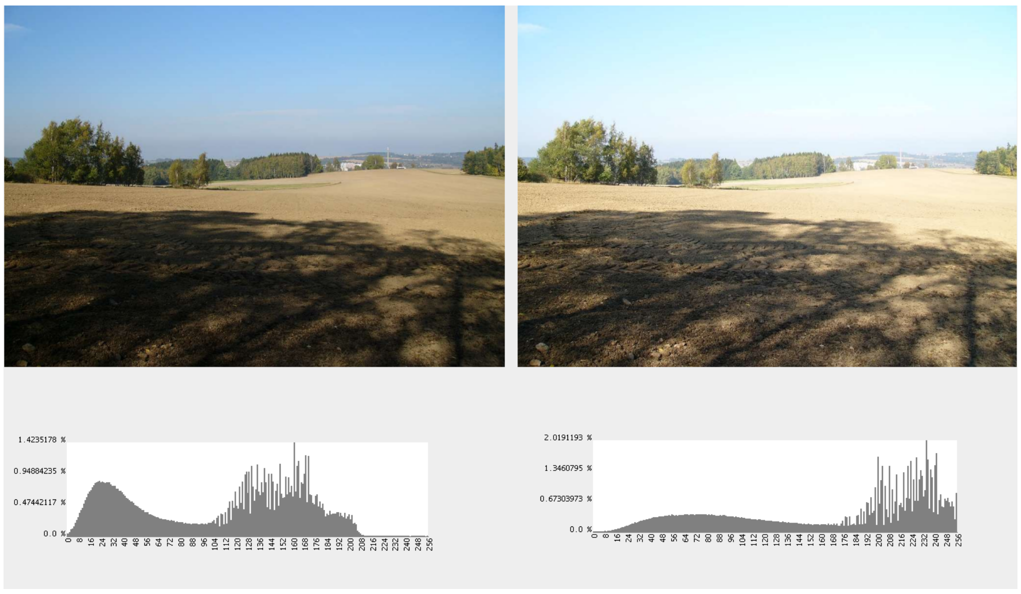


Figure 6.9: High dynamic range scene photographed with the exposure adjusted to the bright areas of the scene (left) and to the dark areas of the scene (right).

Obtaining photographs fitting for exposure bending

Obtaining photographs fitting for exposure bending consist only of getting two photographs of given scene. The first one well-exposed in the dark areas and the second one well-exposed in the bright areas.

To achieve best results, the photographs should be, with exception of different exposure, the as same as possible. The most important things that should be taken care of are:

- The scene should be static. All objects should be in the same positions in both photographs.

  Even bending photographs taken while strong wind was blowing and the trees was moving can cause undesirable artifacts in resulting image.

- All objects in the scene must have the same size in both photographs. Consequently, the photographs should be taken with the same zoom factor and from the same distance.

- Even though the tool enables to fit two photographs that are mutually translated and rotated, it is good to try not to move or rotate the camera between taking the photographs which will be bended. More important is not to rotate the camera, because only translating the scene is better corrigible. Tripod can help remarkably to archive aim.

Using exposure bracketing

Exposure bracketing is the function of camera that allows taking series of photographs which exposure vary by specified value. The photographs are taken at nearly the same time and therefore are are with exception of the exposure nearly the same.

Exposure bracketing is originally designed just to take more differently exposed photographs and to choose the best one later. By contrast, when using exposure bending, two badly exposed photographs are needed to obtain. One photography overexposed in the bright areas and well-exposed on the dark areas. Consequently, higher values of exposure adding in exposure bracketing settings should be set.

Exposure bracketing usually allows taking big quantity of differently exposed photographs. While using exposure bracketing in this tool, only two photographs are needed.

The disadvantage of using exposure bracketing is that the correct exposition only at for example dark areas can be adjusted and photographer must rely on that one of taken photography will be well-exposed in the bright areas. However, the exposure difference between well-exposed photography in bright areas and well-exposed photography in the dark areas can be measured and set the exposure bracketing difference to this value.

Exposure bending method used in the tool

The exposure bending method used in the tool aims to add the information missing in the underexposed or overexposed areas in the image and to keep the image naturally looking.

The problem is, that not only the dynamic range of the camera, but also even the dynamic range of the resulting image and common monitors and printers is limited. Mapping very high dynamic range into limited dynamic range of such devices would usually result to the distortion of details in the image and mostly to unnaturally looking images. Because the information extracted from two images has usually big enough dynamic range to use all dynamic range of 8 bit image, more than bending more than two images is not supported.

However, the exposure bending process can be iterated to get desiderative result. Note that bending more images with exposure bending method currently used in the tool could cause the loss of contrast particularly in the middle tones, dark areas and border areas between the dark and bright areas. Moreover, it is more difficult to obtain more images suitable for exposure bending and correctly fit them.

6.9 Using Exposure Bending in the Tool

Go to the tools menu in the browsing mode and choose exposure bending option. The dialog will be displayed.

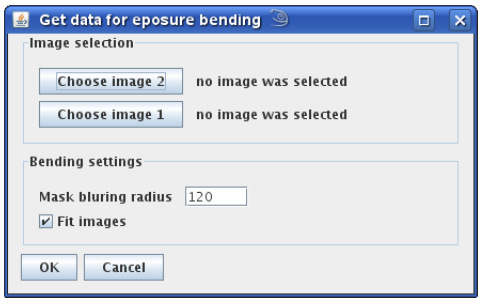


Figure 6.10: Exposure bending dialog.

Get data for exposure bending dialog

Following settings are available:

Image selection Choose the images that will be bended. One should be well-exposed in the dark areas and the other one well-exposed in the bright areas. It does not matter whether the darker image is chosen as the image 1 or the brighter image as the image 1. The tool will auto detect which image is the brighter one and which the darker one.

Mask blurring radius higher values of mask blurring radius suppresses the undesirable artifacts on the borders between dark and bright areas in the scene and suppresses blurring in such areas. These artifacts and blurring are caused by incorrectly fitting of images in such borders. Unfortunately, higher values of the mask blurring radius also lower the precision of exposure bending. Very high values cause that used exposure bending method became nearly simple image averaging. The blurring radius of 120 gives usually good results in most cases.

Fit images Choose whether the images might be fitted. Unselect this option only when the images are already fitted. Images can be fitted separately by using fitting images function. This is beneficial for example if you want to try out the settings of the mask blurring radius.

Fitting images dialog

Fitting images dialog is showed if the fit images option was chosen or if the image fitting tool was run.

The dialog provides following settings:

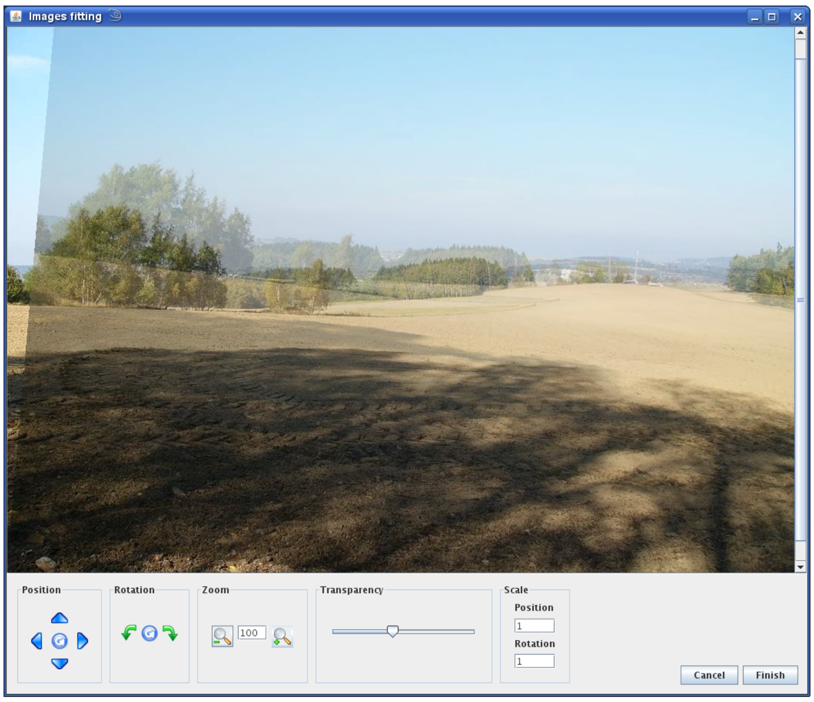


Figure 6.11: Fitting images dialog.

Position Position of the upper image can be adjusted pressing buttons in the position panel or dragging mouse over the image. The image can be moved to the original position by pressing the button in the middle of the position panel.

Rotation Rotation panel enables adjusting the rotation of upper image. The rotation can be canceled by pressing the middle button in rotation panel.

Zoom Zoom panel enables to set the zoom of images.

Transparency Transparency panel enables to set the transparency of upper image. Only bottom image will be visible if the transparency slider is on the left and only upper image will be visible if the transparency slider is on the right.

Scale Scale panel enables to set the scale of position and rotation. Set small values for fine adjusting the fitting.

Upper image should be translated or rotated to fit the bottom image as precisely as it is possible. Correctly fitted image is indicated by not blurring result image when the transparency slider is approximately in the middle. If the images are not correctly fitted, resulting image can be blurred and it can be impacted by undesirable artifacts.

These undesirable artifacts are usually most intrusive in the middle of the picture. Consequently, the images should be correctly fitted primarily in the middle. Note that bending method used in the tool creates these artifacts mostly in the boarders between dark and bright areas. Special attention should be therefore paid also to correctly fitting the images on these borders.

The result of exposure bending

The exposure bending operation will be started after confirming *Get data for exposure bending dialog* respectively *Fitting images dialog* if the option fit images was selected. The progress of the operation will be shown on the progress bar.

After the exposure bending finishes, the result of exposure bending will be displayed in the viewing window. The result can be saved by choosing File – Save or File – Save as.

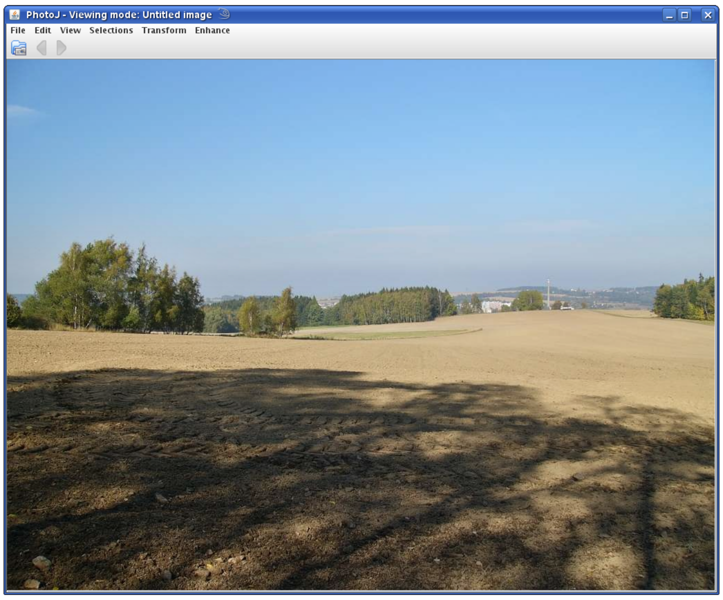


Figure 6.12: The result of exposure bending.

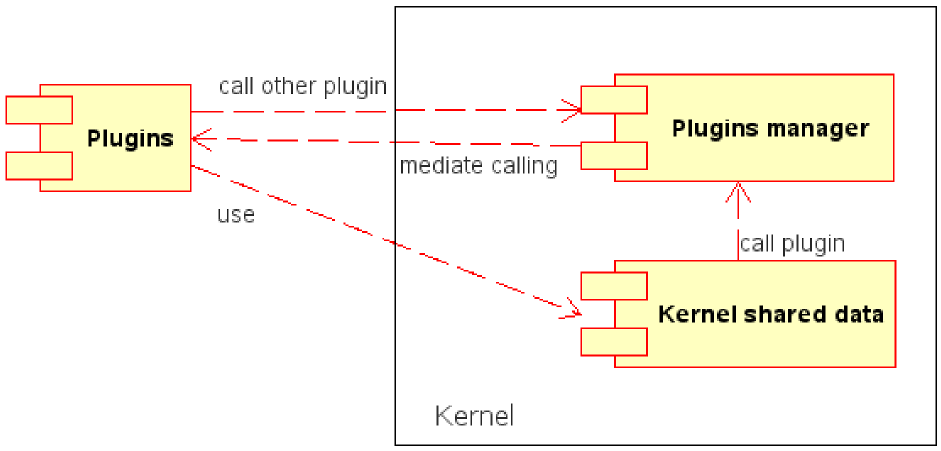
Chapter 7

Architecture of the Tool

7.1 Architecture Overview

The architecture of the tool is based on plugins – objects detected on the start-up of the tool. The plugins implement all the functionality of the tool.

The kernel of the tool loads the plugins, mediates the calling of the plugins, holds shared data used by the plugins, provides various helper classes and interfaces, and provides further functions such as managing a menu.



[htp]

Figure 7.1: The basic concept of the architecture.

7.2 Plugins Concept

Characteristics of the plugin

During the running of the tool, each plugin is represented by one instance of the main class of the plugin. This class must implement the interface given by the type of the plugin. The plugin is identified by this plugin type and by unqualified name of this class. The object representing plugin is created only once during the running of the tool and lives on between callings of the plugin.

It is possible to call plugins via interfaces in the kernel of the tool. These interfaces are accessible from plugins so it is possible to call any plugin from any other plugin.

Contract of the plugin

Each plugin provides some service. This service must be exactly described in the documentation of the plugin.

Plugins can be updated by replacing the class of the plugin by another class with the same name. Then, given service will be provided by the object of this new class. New class must fulfill the contract of the plugin. This means, that it must offer the same service and its interface must be the same.

The interface of the plugin is defined by further specification of types of arguments and types of return values of plugin's methods. This specification must be pointed in the documentation of the plugin. The interface of the plugin can be enriched by additional public methods not defined by the interface which must plugins of given type implement. The specification of such methods must be pointed in the documentation of the plugin too.

Then, plugin can specify inner classes which it provides and which `Class` object the callers of the plugin can obtain.

Plugins communication

There are two main ways how plugins can communicate. The first one is synchronous calling of the plugins. It is possible to call the plugin via specified static class in the kernel or even to get the object of the plugin and call an arbitrary method of the plugin's interface or invoke other method provided by the plugin.

The second concept is asynchronous communication based on events. It is possible to register a plugin to receive a message when some event occurs. A plugin can be also observable – it can notify registered plugins when some event occurs.

Asynchronous communication and shared data

Shared data are hold by objects or static classes accessible for all plugins. Updating of such data often causes an event. Plugins can register to receive asynchronous message when such event occurs.

The example of shared data is the image which is user currently viewing. When this image is updated, the event occurs. Plugin, which displays the image, receives the message and displays the image. Consequently, another plugins simply updates the image and does not need to care about the process of displaying the image to a user. Furthermore, more plugins can be registered to receive such message and can do work that is even more complex.

This is strong mechanism how to automate many technical processes and reduce the amount of complexity the programmer must deal with. Owing to the

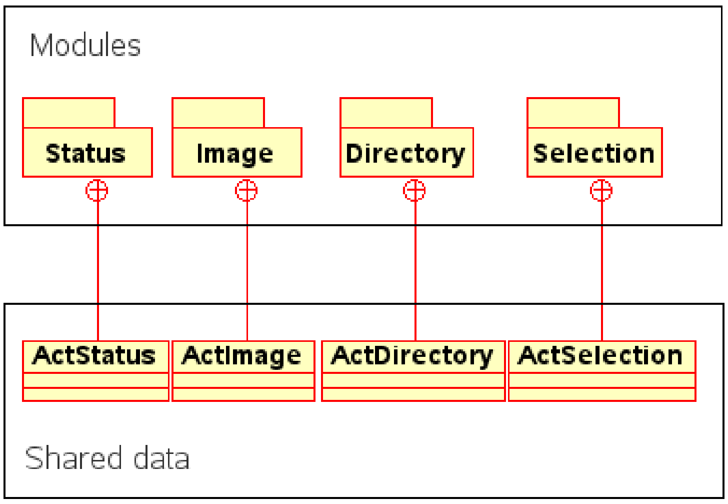


Figure 7.2: Shared data structures in the kernel of the tool.

fact, that plugins can be observable, the concept of shared data is not limited to the data structures included in the kernel.

Plugin types and interfaces

Plugins play different roles in the tool. Consequently, there are more types of plugins. Plugins of given type are called via given kernel module. This module also provides the object of class `PluginManager` which enables to get the object of given plugin. Plugin types and corresponding modules are shown in List 7.3.

[htp]

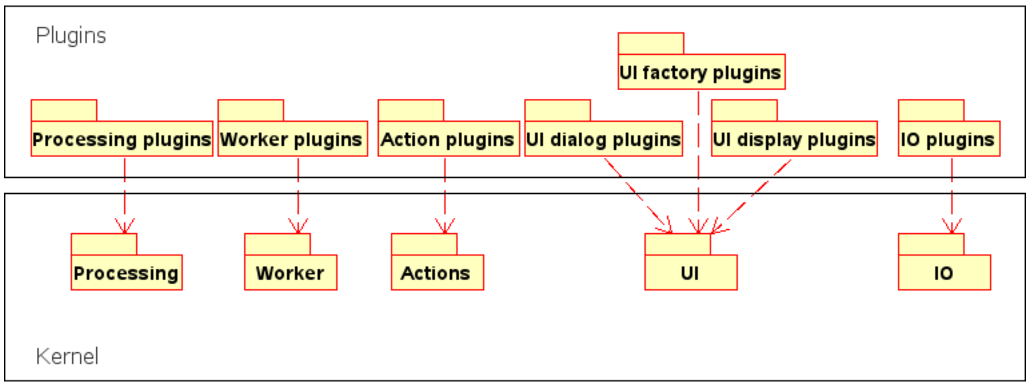


Figure 7.3: Plugin types and modules of the kernel.

Each plugin type is represented by the enum constant `PluginTypes`. This enum constant provides general information about plugins of given type such as the directory where plugins of given type are located and the package of the main class of plugins of given type.

Plugins of given type must implement given interface. The hierarchy of plugins interfaces is showed in List 7.4. The following List provides the description of plugins interfaces:

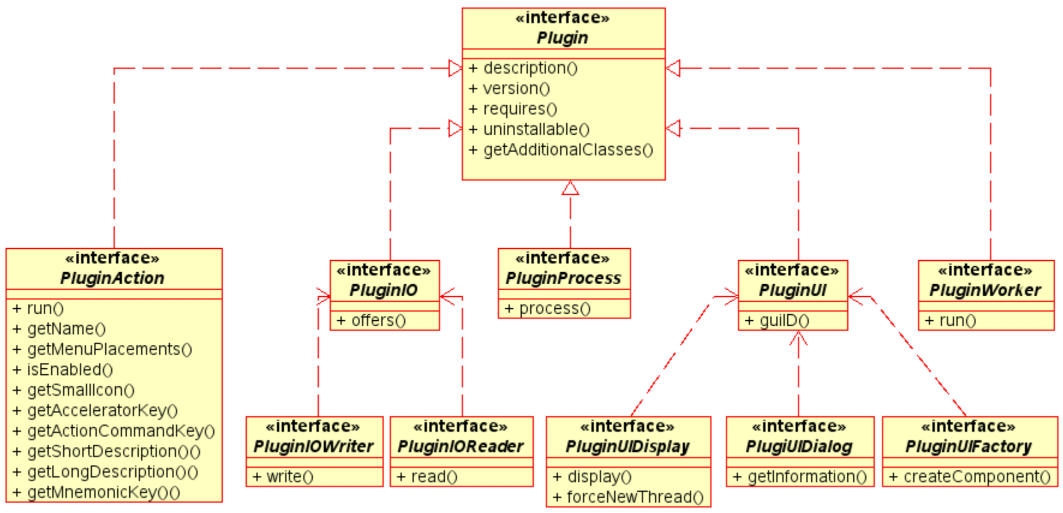


Figure 7.4: The hierarchy of the plugins interfaces.

Plugin Defines administrative methods needed for installing, uninstalling and updating plugins and methods that allow to specify additional classes that the plugin provides.

PluginProcess Is implemented by plugins that process an image. Such plugins are of type `PluginType.PROCESSING`.

PluginAction Implement plugins that will be started when user clicks to menu item or button. Such plugins are of type `PluginType.ACTION`.

PluginIO Defines a method that specifies which image format the plugin read or write.

PluginIOWriter Implement plugins that write some image format. Such plugins are of type `PluginType.IO`.

PluginIOReader Is implemented by plugins that read some image format. Such plugins are of type `PluginType.IO`.

PluginUI Implement user interface elements. Plugins programmer must specify in which GUI toolkit the plugin is implemented.

In the start-up of the tool, only plugins of GUI toolkit specified by the return value of the method `UI.getActGUI()` will be loaded.

PluginUIDialog Is implemented by plugins used to get the user input. Such plugins are of type `PluginType.UI_*_DIALOG`. Where * is identifier of GUI toolkit in which given plugin is implemented.

PluginUIDisplay Implement plugins that display something. Such plugins are of type `PluginType.UI_*_DISPLAY`.

PluginUIFactory Is implemented by plugins that usually create some user interface component. Such plugins are of type `PluginType.UI_*_FACTORY`.

PluginUIWorker Implement plugins that can be used for arbitrary not predefined purposes. Such plugins are of type `PluginType.WORKER`.

7.3 Image Data Structure

The class that stores image data in the tool is called `ImagePlus`. In this class are stored also additional information related to processing of the image such as information about image regions.

ImagePlus and JAI

JAI [13] is a library for processing the images in Java. It offers rich functionality and it supports native acceleration of common operations on various platforms.

It is therefore recommended to use JAI when it is possible. A `ImagePlus` object provides an object of `PlanarImage` class – the class used for storing image data in JAI. `PlanarImage` implements `RenderedImage` interface, which is used for accessing image data in AWT. Consequently, using AWT imaging instead of JAI is also possible.

Image regions

An image region – the object of the `ImageRegion` class – represents selected part of given image. The most common usage of image regions is to represent part of an image that should be processed or analysed. However, the usage of image regions can be wider. It can be used also for example to define the cropping area while getting the cropping bounds from the user.

`ImageRegion` objects belonging to given image are accessible via an object of the `ImageRegions` class stored in the image. This class contains methods for iterating all image regions, for applying geometrical transformation to all image regions, and for observing image regions.

Selections

Selections are used to create image regions and manipulate with image regions. It is possible to create new method of creating the selections and to create new method of manipulating with the selections. This is described section A.12.

7.4 Kernel Modules

The kernel is divided into a modules. The modules of the kernel are showed in Figure 7.5.

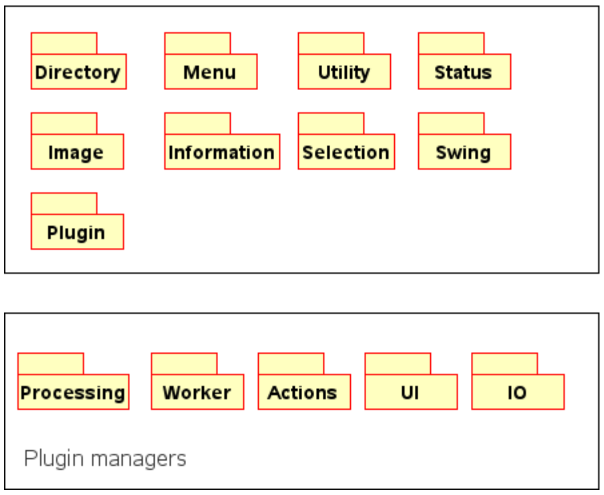
[ht] 

Figure 7.5: Kernel modules.

Processing module Arranges loading and calling of plugins that can process the image.

Actions module Manages loading and calling of plugins that can be run when user presses a button or a menu item. Action system also maintains correct enabled state of such buttons and menu items and offers functions for changing the labels on such elements.

Worker module Mediates calling of plugins that do some arbitrary work. It can be calculation of some value but also installing plugins or batch processing.

User interface module Arranges loading of plugins of correct GUI toolkit. It guarantees, that there are loaded only plugins from one specified GUI system. Then, it mediates the calling of user interface plugins.

Menu module Stores the information about menus in the tool and loads the data necessary for creating menus from XML files.

It does not handle the creation of component of used GUI toolkit. This is done by user interface factory plugin `MenuFactory`.

I/O module Provides calling of plugins for reading and writing of images. Offers the list of supported formats for reading or writing.

Image module Provides classes for working with an image. `ImagePlus` class was described in Section 7.3. Shared data structure `ActImage` holds information about the image which is the user currently viewing.

Directory module Includes the class `CachedImages` that provides the access to the images in given directory. It accelerates reading of the image by caching and notifies observers when some image is changed.

Shared data structure `ActDirectory` holds information about the directory with which user is currently working.

Selection module Manages selections in the tool. It provides interfaces for creating selections and manipulating with selections. Shared data structure `ActSelection` holds the information about actual selection method in the tool.

Status module Provides shared data structure `ActStatus` that holds information about actual status of the tool.

Plugin module Stores information about plugin types and provides the class `PluginManager` that serves for loading and calling of plugins of all types. Then, it provides the interface Plugin and corresponding adapter class AbstractPlugin.

Information module Stores system properties such as file separator, preferences of the tool, and stores information about viewing and browsing window.

Utility module Provides various functions like handling error messages, handling debug messages, utility functions for working with files, functions for loading of plugins, or functions for localisation.

Messages module Provides classes useful for sending messages in the tool via arguments of the methods.

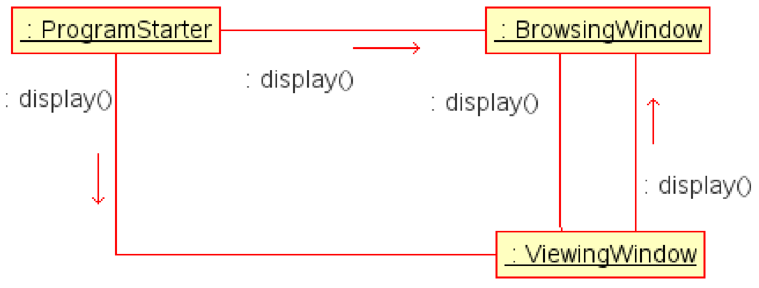
Swing module Provides the support of the work with Swing GUI toolkit. It offers several utility classes and classes of several swing components. However, majority of swing components are provided as an object returned by user interface factory plugins. In the Swing module are primarily such swing components that provides more complicated interface and components that could be used for inheritance.

7.5 Process Flow

There are three main plugins that are called during the running of the tool. The worker plugin `ProgramStarter` is called at the start-up of the tool. It handles parsing the command line arguments and initialising of the tool. According to passed command line arguments, the plugin `ProgramStarter` calls either the user interface plugin `BrowsingWindow` or user interface plugin `ViewingWindow`.

Browsing window

The browsing window is realised by the user interface plugin `BrowsingWindow`. The primary function of the browsing window is to allow a user to browse filesystem and to choose an image he wants to view or process. If the user chooses the image, the viewing window is called and it displays the image.



[ht]

Figure 7.6: The basic process flow in the tool.

The browsing window contains amenu that allows running action plugins placed there. Such menu is represented by enum constant `MenuInfo.MAIN_MENU_BROWSING` and it is defined in the XML file `MenuInfo.MAIN_MENU_BROWSING.getMenuPath()`.

Then, browsing window can contain component that displays information about running operations. This information is stored in shared data structure `ActState`.

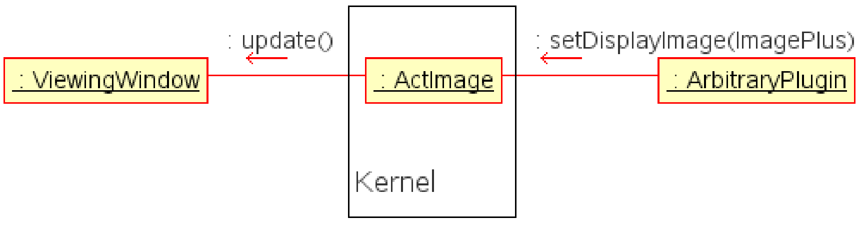
When browsing the filesystem, browsing window uses shared data structure `ActDirectory`. Browsing window must guarantee that this data structure really contains the information about directory which is the user currently browsing.

The object that physically displays the browsing window is stored in the class `Windows`. It is of class given by conventions of actually used GUI toolkit. For Swing, it is `JWindow` or `JFrame`. Such object can be obtained by other plugins and used for example for specifying the owner argument when calling display or dialog plugins via the class `UI`.

Viewing window

Viewing window is realised by user interface plugin `ViewingWindow`. The primary function of the viewing window is displaying the image contained in `ActImage` shared data structure.

Viewing window contains a component that is automatically updated when actual image is changed. Consequently, if some entity in the tool wants to display particular image, it only sets given image to the `ActImage` data structure.



[ht]

Figure 7.7: Displaying the image in the viewing window via the mechanism of backward calling.

Viewing window contains menu that allows running action plugins placed there. Such menu is represented by enum constant `Menus.MenuInfo.MAIN_ME-`

NU_VIEWING and it is defined in the XML file Menus.MenuInfo.MAIN_MENU_VIEWING.getMenuPath(). Entries of this menu should enable to go to the next and preview image in the actual directory, save the image, do undo and redo, zoom the image, or it should enable user to work with selections.

Viewing window contains also a pop-up menu. This menu is represented by enum constant Menus.MenuInfo.POPUP_MENU_VIEWING and it is defined in the file Menus.MenuInfo.POPUP_MENU_VIEWING.getMenuPath().

Like the browsing window, viewing window contains the component that displays information about running operations stored in shared data structure ActState.

The object that physically displays the viewing window is stored in the class information.Windows.

Viewing window must keep actualised information class ViewingWindowProperties. It contains for example the information about actual state of the viewing window or the dimensions of the area usable for displaying the image.

Chapter 8

Evaluation – Comparing with Similar Tools

8.1 Compared Tools

Following tools were compared:

XnView [**18**] Is well known multimedia viewer, browser, and converter available for several platforms.

Gwenview [**19**] Is an image viewer and manipulator for KDE.

Almara [**20**] Is an image viewing and manipulating tool for UNIX written by students of MFF UK.

Picasa [**23**] Is an image viewer, browser, and organiser from Google.

Among other interesting tools belongs Ekspos Image Viewer [21], an image viewer written in Java, Fotox, a tool which supports exposure bending function, or IrfanView [24], popular image viewer for Windows.

8.2 Methodology

Tools were compared according to platforms on which they are available, provided extension capabilities, and provided functions supporting the work with photographs.

Support of platforms

It was checked out, whether the tool supports Windows, Unix, and Mac OS X.

Extension support

- **Open source** – it was checked up, whether the tool is open source.
- **SDK/API available** – it was tested, whether SDK or API is available. SDK / API enables to extend the tool without modification of source codes of the tool.
- **Localisation** – it was checked up, whether the tool supports adding localisation strings by users.

Functions that support the work with photographs

- **Rotation, Cropping and Rescaling** – it was tested, whether the tool includes function or image rotation, cropping, and rescaling.
- **JPEG lossless** – it was tested, whether the tool enables lossless operations with JPEG images. Lossless operations enable to transform directly the data of the JPEG file without decoding it to the image data and encoding back to the JPEG file.

  The support of JPEG lossless operations usually includes the support of rotating the image with angles that are multiples of 90 degrees and vertical or horizontal flipping of the image pixels.
- **Batch processing** – it was tested, whether the tool supports applying image operations in the batch.
- **Brightness, Contrast and Saturation** – it was tested, whether the tool supports adjusting the brightness, contrast, and saturation of an image.
- **Image histogram and the Curves tool** – it was tested, whether the tool supports displaying image histogram and whether it includes the curves tool.
- **Noise reduction, Sharpening and Red eye reduction** – it was tested, whether the tool supports the function of noise reduction, image sharpening, and red eye reduction.
- **Auto enhancement** – it was tested, whether the tool includes automatic enhancement functions such as automatic tone balancing.
- **EXIF** – it was tested, whether the tool reading and writing EXIF information.

  EXIF [8] – Exchangeable Image File Format, enables to store additional information about photography in image file. It can be for example the date when the photography was taken, information about exposure settings, or description of the photography.

- **Video playback** – it was tested, whether the tool supports the video playback.
- **Organising images** – it was checked out, whether the tool includes functions for organising images such as sorting images to the albums, searching images, or creating web photo albums.
- **Panorama tool** – it was tested, whether the tool supports assembling a mosaic of photographs into a panorama image.
- **Exposure bending** – it was checked out, whether the tool includes the function of exposure bending.

8.3 Comparison Results

The results of comparison are shown in Tables 8.1, 8.2, and 8.3.

System	**PhotoJ**	**Gwenview**	**Almara**	**Picasa**	**XnView**
Windows	YES	YES	*NO* [a]	YES	YES
UNIX	YES	YES [b]	YES	YES [c]	YES [d]
MAC OS X	YES	*NO*	*NO*	*NO*	YES [e]

[a]It can run under windows with help of Cygiwin [25].
[b]It depends on KDE libraries.
[c]Only the support of Linux, uses wine emulator [26].
[d]Only version 1.7.
[e]Only version 1.7.

Table 8.1: Comparison of platform support.

Function	**PhotoJ**	**Gwenview**	**Almara**	**Picasa**	**XnView**
Open source	YES	YES	YES	*NO*	*NO*
SDK/API available	YES	YES [a]	*NO*	*NO*	YES [b]
Localisation	YES	YES	*NO*	*NO*	YES

[a]Only the integration of external tools.
[b]SDK only for I/O available.

Table 8.2: Comparison of the extension capabilities.

Function	**PhotoJ**	**Gwenview**	**Almara**	**Picasa**	**XnView**
Rotation	YES	YES [a]	YES	YES	YES
Cropping	YES	*NO*	YES	YES	YES
Rescaling	YES	*NO*	YES	YES	YES
JPEG lossless	*NO*	*NO*	YES	*NO*	YES
Batch processing	YES	YES [b]	*NO*	YES	YES
Brightness	YES	YES	YES	YES	YES
Contrast	YES	YES	YES	*NO*	YES
Saturation	*NO*	*NO*	YES	YES	YES
Image histogram	YES	*NO*	*NO*	*NO*	YES [c]
Curves tool	YES	*NO*	*NO*	*NO*	*NO*
Noise reduction	YES	*NO*	YES	NO	YES
Sharpening	*NO*	*NO*	*NO*	*NO*	YES
Red eye reduction	*NO*	*NO*	*NO*	YES	YES [d]
Auto enhancement	*NO*	*NO*	YES	YES	YES
EXIF	*NO*	YES	YES	YES	YES
Video playback	*NO*	YES	YES	YES	YES
Organising images	*NO*	*NO*	YES	YES	YES
Panorama tool	*NO*	*NO*	*NO*	*NO*	YES
Exposure bending	YES	*NO*	*NO*	*NO*	*NO*

[a]Only multiples of 90 degrees.
[b]With the usage of extern tools.
[c]Currently supports only the version of the tool for Windows.
[d]Currently supports only the version of the tool for Windows.

Table 8.3: Comparison of functions that support the work with photographs.

8.4 Comparison Conclusion

PhotoJ offers the widest support of platforms. Such a support of platforms is unique not only among tested tools, but also among image viewing and manipulation tools in general. The only other tested tool that supports all tested platforms is XnView. However, the latest version of XnView, currently available only for Windows and Linux versions of XnView, provides only X11/Motif user interface. Next, PhotoJ offers the broadest support of extending. No other tested tools provides a straightforward integration of more complex image processing techniques.

Even though the support of image processing functions is not as broad as in the other tools, PhotoJ offers basic processing functions and it can be used for viewing and manipulating photographs.

The most limiting is probably the lack of support of reading and writing EXIF information. Another important missing function is a red eye reduction and auto enhancement functions. For some users, the lack of support of video playback, is also limiting.

On the other hand, PhotoJ contains the function of exposure bending, which does not provide any other compared tool. Moreover, with the exception of EXIF support, it is possible to implement all other appointed missing functions in plugins.

Chapter 9

Conclusion

PhotoJ, a portable tool for image viewing and manipulation, fulfills the requirements set out in the specification. It can be used for image viewing and manipulation in various platforms and it includes the function of exposure bending. The functionality of the tool can be widely extended via plugins. Almost one hundred of plugins currently available demonstrates the efficiency of the extension mechanism.

The level of the tool extensionability enables the integration of the arbitrary image processing function. Consequently, the tool could be profitable for programmers who have already implemented a processing technique and would like to integrate it into the image viewing tool.

Features that are more complex would be usually implemented in more plugins of various types. When such a feature is not distributed with the program, it is necessary to install all plugins forming this feature independently. Hence, one of the next possible improvements is to provide the function of automatic installation and the uninstallation of more plugins coining one feature.

Another improvement of plugin mechanism would be to offer the class loader that would extend the class path of a plugin, so that it would include the classpaths of plugins, on which the plugin depends. This will enable to call public methods of plugins, on which a given plugin depends directly without the use of Reflection API.

Another important point is adding the support of the reading and writing an EXIF [8] image file format. Integration of this feature requires a small modification of `IO` module of the tool kernel.

Other improvements would include adding functions for automatic contrast stretching or adding support for suppressing the red eye effect. Last but not least, usability improvements such as a support of quick navigation in browsing mode would be also integrated.

Experience with the Development of the Tool

The design and the development of the tool was a new experience for me. Due to the fact that, I had almost no knowledge of software engineering, I was unable to design the tool correctly. Owing to many mistakes in the design of the tool, I had to spend a lot of time refactoring it later on.

Then, I found the tool too broad to implement by one programmer. Because of the big quantity and because of extensive refactoring, it was hardly manageable to sustain the quality of the code acceptable.

However, the experience with the development of the tool has been invaluable for me. I had the possibility of familiarising myself with the problems of software engineering and such techniques as modular programming, design patterns, reflection, or event handling. Next, the possibility of discussing problems with my project manager was very precious. I can say, that the work has encouraged me to think about software development in wider context than before.

Appendix A

Plugin Developers Manual

Further material describes how to write plugins for the tool and how to use functions both of the kernel of the tool and of some, important built in plugins.

However, there is not systematically described the architecture of the tool. The architecture of the tool was described in Chapter 7.

In Sections A.1, A.2 and A.3 is described basic usage of tool APIs that should be sufficient for most cases. Selected APIs are described in more detail in Sections following Section A.9.

In Sections A.4, A.5, A.6, A.7, A.8, and A.9 is described how to write given type of plugins. Even though it is possible to start with these sections, at least basic knowledge of tool's APIs is needed when writing more complex plugins.

A.1 Working with Images in the Program

Getting and updating image which is user currently viewing

The image which is user currently viewing and the information about such image are accessible via static methods of the class `ActImage`. The list of such methods is in Table A.1.

The image which is user currently viewing is stored in this static class as an object of `UndoableImage` class. It is possible to get and update the image data. If the image is updated and viewing window is visible, the image is automatically displayed to the user. The list of selected methods of the class `UndoableImage` is in Table A.2.

Processing image

The image itself is stored in object of class `ImagePlus`. This class provides the method `getPlanarImage()` that returns the object of the `PlanarImage` class.

Method	Description
`UndoableImage getActImage()`	Gets the object which stores the image which is user currently viewing
`File getActImageFile()`	Gets the file of actual image
`String getActImageName()`	Gets the filename of actual image
`boolean unsavedChanges()`	Gets true if there are unsaved changes in the actual image

Table A.1: Static methods of the class ActImage.

`PlanarImage` is class of JAI [13] image library. Consequently, it can be used directly as the input of the JAI operators. It implements also `Renderable` interface so AWT imaging can be used as well. Naturally, also the array with the values of the pixels can be obtained and the image can be processed without the usage of any library. However, it is recommended to use JAI because of performance optimalisation.

In the code in Listing A.2 is showed obtaining the `PlanarImage` object and performing simple operation with the image. In the code in Listing A.1 is showed the same operation with direct accessing the pixels.

Using of `PlanarImage` is described in more detail in [15] and [14].

Listing A.1: Obtaining actual image and performing simple JAI operation

```
// Get the image
ImagePlus image = ...
// Process the image
image.setPlanarImage = JAI.create(
  "invert", image.getPlanarImage());
```

Listing A.2: Direct access to the image data

```
// Get the image
ImagePlus image = ...

// Get the data of the image
PlanarImage pi = image.getPlanarImage();
int width = pi.getWidth();
int height = pi.getHeight();
SampleModel sm = pi.getSampleModel();
int nbands = sm.getNumBands();
Raster inputRaster = pi.getData();
int[] pixels = new int[nbands*width*height];
inputRaster.getPixels(0,0,width,height,pixels);



// Process obtained data
int offset;
for(int h=0;h<height;h++)
```

Method	Description
`ImagePlus getProcessingImage()`	Gets the image which is user currently viewing.
`ImagePlus updateProcessingImageDisplay(ImagePlus)`	Updates the image which is user currently viewing and displays it to the user. The old version will be accessible via `Redo()`.
`ImagePlus updateProcessingImageDisplayFit(ImagePlus)`	Like previous method but the image will be rescaled to fit viewing window while displaying.
`ImagePlus setProcessingImageUpdateDisplay(ImagePlus)`	Sets new image which is user currently viewing. The undo and redo will be reseted.
`ImagePlus undoImage()`	Does undo operation.
`ImagePlus isUndoAvailable()`	Indicates whether the undo operation is available.
`ImagePlus redoImage()`	Does redo operation.
`ImagePlus isUndoAvailable()`	Indicates whether the redo operation is available.
`void resetUndo()`	Resets undo.

Table A.2: Selected methods of the class UndoableImage.

```
  for(int w=0;w<width;w++) {
    offset = h*width*nbands+w*nbands;
    for(int band=0;band<nbands;band++) {
      pixels[offset+band] = 255 -- pixels[offset+band];
    }
  }
}

// Create new PlanarImage from processed data
WritableRaster outputRaster =
  inputRaster.createCompatibleWritableRaster();
outputRaster.setPixels(0,0,width,height,pixels);
// TitledImage is writable descendant of PlanarImage
TiledImage ti = new TiledImage(pi,1,1);
ti.setData(outputRaster);

// Set the data into the image
image.setPlanarImage(ti);
```

Using image regions

Image region represents part of the image. The concept of image regions is described in Section 7.3.

It is possible to obtain image regions from object of class `ImagePlus` and iterate it.

It is also possible to add image region to the image. New image region can be added simply by calling method `add(Shape)` of given `ImageRegions` object. However, user would not have the possibility to manipulate with such image region. Enabling user to manipulate with image regions is described in Section A.12.

If the image is modified by geometrical modification, the image regions should be modified in the same way. It is possible to send the message to transform to all image regions by calling the method `transform(AffineTransform)` of the `ImageRegions` class. This mechanism is described in more detail in Section A.11.

The usage of image regions is showed in Listing A.3.

Listing A.3: Using image regions

```
// get the image
ImagePlus image = ...

// add image regions to the image
image.getImageRegions().add(new Rectangle(100, 100));
image.getImageRegions().add(new Rectangle(50, 150));

// process image regions
Iterator<ImageRegion> iterator = image.getImageRegions();
  iterator();
while (iterator.hasNext()) {
  ImageRegion imageRegion = iterator.next();
  // process image region
}

// rotate image regions
AffineTransform roiRotation = AffineTransform.
  getRotateInstance(angle, xRotCentre, yRotCentre);
image.getImageRegions().transform(roiRotation);
```

Using interpolation

The enum `Interpolation` holds information about interpolation. Interpolation constant is usually passed to processing plugins. Processing plugin should process the image with given interpolation. It is possible to get JAI interpolation instance from given interpolation constant by calling method `getJAIInstance()`.

The static methods of enum `Interpolation` enables to set and get default interpolation settings for given type of processing. The setting of interpolation constants is done in the start of the tool by worker plugin `InitializeProgram`.

Plugins should use these settings. If action plugin calls processing plugin to create the preview of some operation, it should pass it the interpolation constant given by `Interpolation.getPreviewInterpolation()`. If the processing plugin doesn't get interpolation constant in the argument, it should use interpolation constant given by `Interpolation.getDefaultInterpolation()`. The list of interpolation constants is in Table A.3, the list of the static methods used to get the default interpolation constants is in Table A.4.

The enum `InterpolationQuality` is used to map integer numbers representing the interpolation quality to the interpolation constants defined in enum `Interpolation`. The list of static methods of enum `InterpolationQuality` is in Table A.5.

Constant	**Description**
`NEAREST`	Neareast neighbour interpolation.
`BILINEAR`	Bilinear interpolation.
`BICUBIC`	Bicubic interpolation.

Table A.3: Interpolation constants defined in enum Interpolation.

Method	**Description**
`Interpolation getPreviewInterpolation()`	Gets default interpolation constant for processing preview images.
`Interpolation getDisplayInterpolation()`	Gets default interpolation constant for processing images which will be displayed to user but which do not affect further processing.
`Interpolation getPermanentInterpolation()`	Gets default interpolation constant for processing images which will affect further processing.
`Interpolation getDefaultInterpolation()`	Gets default interpolation constant.

Table A.4: The static methods used to get the default interpolation constants.

Method	Description
`Interpolation getBestInterpolation()`	Gest the interpolation which offers the results of the best quality.
`Interpolation getFastestInterpolation()`	Gets the fastest interpolation method.
`Interpolation getInterpolation(int)`	Gets the interpolation constant associated with the quality of given integer value.

Table A.5: The static methods of enum InterpolationQuality.

A.2 Using Plugins

Processing plugins

Processing plugins can be called via the class `Processing`. The arguments passed of the plugin are be specified in the documentation of plugin. In the table A.6 is the list of most frequently used static methods of this class.

Method	Description
`ImagePlus callPlugin(String, ImagePlus, Object...)`	Process given image. with given processing plugin.

Table A.6: Most frequently used static methods of the class Processing.

User interface plugins

User class `UI` allows calling user interface plugins. The arguments of the plugin should be specified in the documentation of plugin.

User interface display plugins usually serve to display something to user, user interface dialog plugins usually serve to get some input from user and factory plugins usually serve to create some user interface component. The static methods which mediates the calling of the user interface plugins are listed in Table A.7.

Display and dialog user interface plugins can use the argument owner of methods `UI.callDisplayPlugin()` and `UI.callDialogPlugin()` for example to set the owner of window which they display or to centre such window. Static class `Windows` enables to to get the objects can be used as argument owner when the owner should be browsing or viewing window.

Note that factory plugins often creates objects of classes defined in given plugin. Using specific methods of such classes is described in A.14

Method	Description
`Arguments callDisplayPlugin( String, Object, Object...)`	Calls the display. plugin with given identifier and given arguments.
`Arguments callDisplayPluginInNewThread( String, Object, Object...)`	Calls the display plugin with given identifier and given arguments. Run it in the new thread.
`Arguments callDialogPlugin( String, Object, Object...)`	Calls the dialog. plugin with given identifier and given arguments.
`Object callFactoryPlugin(String, Object...)`	Calls factory plugin with given identifier and given arguments.

Table A.7: Static methods of class UI providing the calling of the user interface plugins.

Worker plugins

Worker plugins can be used to handle arbitrary function. Calling worker plugins mediates the class `Worker`. The list of static methods used for calling worker plugins is in Table A.8.

Method	Description
`Arguments callPlugin(String, Object...)`	Calls given plugin.
`Arguments callPluginInNewThread(String, Object...)`	Calls given plugin. in new thread.

Table A.8: Static methods of the class Worker used for calling worker plugins.

A.3 Other API of the Program

Using the status of the tool

Actual status of the tool is stored in the class `ActStatus`. It is possible to set the status and get the status. If the status is set, components displaying the status will be automatically updated. The most frequently used static methods of the class `ActStatus` are in Table A.9

Method	Description
`void setStatus(String, int)`	Set the status text and progress value.
`void setStatus(String)`	Set the status text. The progress value will be not available.
`void finished()`	Send message to actual status that the operation is finished. It can be used to notify user, that the operation is finished.

Table A.9: Most frequently used static methods of the class ActStatus.

Writing error and debug messages

The class `ErrorMessages` serves for handling error messages. You should always use this class for writing error messages. The list of static methods of this class is in Table A.10.

Method	Description
`void error(String, String)`	Handles the error message form given source file.
`void error(String, String, Exception)`	Handles the error message from given source file caused by given exception.
`void error(String, String)`	Handles the fatal error message form given source file. Typically stops the tool.
`void error(String, String, Exception)`	Handles the error message from given source file caused by given exception. Typically stops the tool.

Table A.10: Static methods of the class ErrorMessages.

The class `DebugMessages` serves for handling debug messages. Debug messages are messages that are usually written to the command line and includes further information for the user.

Getting information viewing window

Classes `ViewingWindowProperties` offers static methods that allow getting information about viewing window. The list of such methods is in Tables A.11.

Note that the methods allowing to set this information should be used only by plugins representing browsing and viewing window.

Method	**Description**
`Object getViewingWindowContainer()`	Get the container which represents viewing window. It should be used for example to specify the owner in UI.callDialogPlugin() and UI.callDisplayPlugin().
`ViewingWindowState getState()`	Get the state of the viewing window.
`Dimension getImageDisplayAreaDimension()`	Get the dimension of area used to display image in the viewing window.

Table A.11: The most frequently used static methods of the class ViewingWindowProperties used for getting information about viewing window.

Getting and storing tool preferences

The class `Prefs` handles storing tool preferences to disk and loading it from the disk.

`java.util.prefs.Preferences` object can be obtained via method `getPreferencesObject()` and used to store the preferences.

Localisation plugins

Built in plugins are localised using class `Localisation`. This class enables getting localised strings according to current locale settings in the computer. Translations are stored in files in format `GeneralResources_locale`.

Note that `java.util.ResourceBundle` class is used in `Localisation` class internally. This class doesn't allow UNICODE files. Non-ASCI characters must be therefore escaped.

The programmers of plugins that are not distributed with the tool and are installed via plugin manager could not modify localisation files. Consequently, they must organise the localisation themselves. It is possible to use for example `java.util.ResourceBundle` class to handle localisation.

A.4 Writing Plugins

Common conventions

The unqualified name of the main class of the plugin is the identifier of the plugin used for calling plugins of given type. The package of class is given by `SystProp.PluginTypes.PLUGIN_TYPE.getPluginsPackage()`.

All plugins which should be loaded must be placed in the directory given by `SystProp.PluginTypes.PLUGIN_TYPE.getAbsPath()` in jar files. The plugin's jar file must have the same name as the main class of the plugin and must have directory structure given by the package of the plugin. The main class of the plugin must be in the package given by `SystProp.PluginTypes.PLUGIN_TYPE.getPluginsPackage`.

Plugin must exactly define its contract. The contract is defined in Section 7.2. All versions of plugin should respect this contract. Calling plugin that do not respect the contract can cause runtime error. It is recommended to copy the specification of contract of previous version of the plugin when writing the plugin.

It is discouraged to rely on the characteristics of plugin that are not described in the contract of given plugin, because updated version of plugin needn't to fulfill this characteristics.

Concurrent calling of plugins

In the section 7.2 was described that the object representing the plugin during the execution of the tool is created only once and the same object handles all callings of the given plugin. Consequently, all callings of given plugin share the instance variables of this object. Hence, it must be paid attention to provide correct synchronisation of the instance variables.

Example of not correctly synchronised plugin which concurrent calling can cause an error is showed in Listing A.4. If the plugin is called concurrently, the second calling will rewrite instance variable `startupTime`. This causes problem if the first calling will use this variable.

In Listing A.5 is showed fixing of this problem simply by not having any instance variables. In this approach, independent instance of class `WorkingClass` is created by each calling of the plugin.

Listing A.4: The plugin which concurrent running can cause the error

```
public class SamplePlugin extends AbstractPlugin
  implements PluginWorker {

  // This variable is shared for all calling of the plugin
  private Date startupTime;

  public Arguments run(Object[] args)
    throws PluginNotSucceededException {
    startupTime = new Date();
    // ...
    // some computation
    // ...

    /* Variable startupTime can be rewritten during
     * previous computation by other calling of run() method
     */
    return doSomethingWithStartupTime(startupTime);
  }
}
```

Listing A.5: The plugin which is runnable concurrently

```
public class SamplePlugin extends AbstractPlugin
  implements PluginWorker {

  /* Implements all functionality of the plugin
   */
  private static class WorkingClass {
    private Date startupTime;

    public ComputationClass(Date startupTime) {
      this.startupTime = startupTime;
    }
    public Arguments run(Object[] args) {
      // ...
      // some computation
      // ...
      return doSomethingWithStartupTime(startupTime);
    }
  }

  public Arguments run(Object[] args)
    throws PluginNotSucceededException {
    Date startupTime = new Date();
    /* The independent instance of the class WorkingClass
     * is created for each invocation of the plugin.
     */
    WorkingClass notShared = new WorkingClass(
      startupTime);
    return notShared.run(args);
  }
}
```

A.5 Writing Action Plugins

Action plugins are called by user by pressing given menu item or button.

Action plugins are usually not called from other plugins and parameters of `run(Objec[])` method are usually not used so action plugins have usually no exact contract specified.

Interface PluginAction

Action plugins must implement the interface `PluginAction`. It defines methods which defines a placement of the plugin in menus, the text which should appear at the component which runs the plugin, the accelerator key which should run such component, the conditions in which should be such component, and other properties of given button or menu item.

The adapter class `AbstractPluginAction` offers default implementations of some methods of this interface.

Conventions

Action plugin can do any arbitrary work. However, they should be independent on given GUI toolkit. User interface plugins should be called to handle actions dependent on given GUI toolkit such displaying something to the user or to get user input. If there is no suitable user interface plugin, new user interface plugin should be created.

It is also recommended not to do any processing of images in action plugins. Processing plugins should be used instead. Then, worker plugins should be used to implement other potentially reusable operations and called from action plugins.

Placing action plugin in the menu

It is possible to place Action plugins to the menu by editing the XML file with the definition of given menu. This is proper when the tool is distributed to the user with the plugin and the plugin is not uninstallable.

Furthermore, it is possible to define the placement in the menus via method `MenuPlacement[] getMenuPlacements()`. This enables automatic placement of the plugin to the menu during an installation of the plugin, automatic replacement of plugin from the menu during an uninstallation of the plugin, and automatic changing of the menu placent when updating the plugin.

Listing A.6: Example of the action plugin

```java
package plugins.action;
/** Rotates actual image left. */
public class RotateLeft extends AbstractPluginAction {
  /**
   * Rotates actual image.
   * @param args not used
   */
  @Override
  public void run(Object[] args)
  throws PluginNotSucceededException {
    // gets image which is user currently viewing
    ImagePlus image = ActImage.getActImage().
      getProcessingImage();

    // rotates the image and automatically translates it
    image = Processing.callPlugin("RotateAuto",
      image, 270.0f,
      Interpolation.getPermanentInterpolation());

    // updates the image which is user currently viewing
    actImg.updateProcessingImageDisplayFit(image);
  }

  public String getLabel() {
    /* Returned string will be displayed as the label
     * of the button or menu item
     */
    return Localisation.getString("rotateLeft");
  }

  @Override
  public KeyStroke getAcceleratorKey() {
    /* The accelerator key which should run the button
     * or menu item which runs the plugin
     */
    return KeyStroke.getKeyStroke('l');
  }

  @Override
  public MenuPlacement[] getMenuPlacements() {
    MenuPlacement[] placement = new MenuPlacement[2];

    /* The placement in the main menu of the viewing
     * window
     * The plugin will be placed in the sub-menu view
     */
    String[] pl0 = { "view" };
    placement[0] = new MenuPlacement(
      MenuInfo.MAIN_MENU_VIEWING, pl0);

    /* The placement in the pop-up menu of the viewing
     * window
     * The plugin will be placed in the sub-menu
     * image -> view
     */
    String[] pl1 = { "image", "view" };
    placement[1] = new MenuPlacement(
      MenuInfo.POPUP_MENU_VIEWING,

    return placement;
  }

}
```

A.6 Writing Processing Plugins

Processing plugins serves for processing the image. They must implement interface `PluginProcess`. This interface defines method `ImagePlus process(ImagePlus, Object[])`. Further specification of the argument of this method is part of plugin's contract.

Regions of interest can be used when processing an image. For example, only the part of the image that is in the regions of interest can be processed. Note that if geometrical transformation is done with the image, regions of interest should be transformed in the same way. Working with image regions is described in Section A.3.

Listing A.7: Example of the processing plugin

```java
/**
 * Rotates the image with given angle and rotation centre.
 */
public class Rotate extends AbstractPlugin
  implements PluginProcess {

  /**
   * Rotates the image.
   * @param args the message to the plugin
   * args[0] ... (float) rotation angle (in radians)
   * args[1] ... (float) x coordinate of the center
   *     of the rotation
   * args[2] ... (float) y coordinate of the center
   *     of the rotation
   * args[3] ... (optional) (pj.messages.Interpolation)
   *             interpolation constant
   */
  public ImagePlus process(ImagePlus image, Object[] args)
      throws PluginNotSucceededException {

    // Get arguments
    float angle = (Float) args[0];
    float xRotCentre = (Float) args[1];
    float yRotCentre = (Float) args[2];
    Interpolation interpolation;
    if (args.length > 3) {
      interpolation = ((Interpolation) args[3])
          .getJAIInstance();
    } else
      interpolation = Interpolation.
          getDefaultInterpolation().getJAIInstance();

    // Rotates image
    // ... implementation of image rotation

    // Rotates image regions
    AffineTransform roiRotation =
      AffineTransform.getRotateInstance(angle,
        xRotCentre, yRotCentre);
    image.getImageRegions().transform(roiRotation);

    return image;
  }

}
```

A.7 Writing User Interface Plugins

User interface plugin serves to get user input, to display something to user or to create user interface components used in other user interface plugins. All user interface plugins must implement the interface `PluginUI` which defines method `GuiIDs guiID()`. This method specifies which in which GUI system is given plugin implemented.

Dialog plugins

The interface `PluginUIDialog` defines method `Arguments getInformations(Object, Object[])`. This method typically displays some dialog to user and returns user input. Further specification of the argument of this method is part of plugin's contract.

The kernel of the tool provides several classes, which supports the using of dialog plugins. The example of such classes is the `DialogParametres` class that can be used as an argument passed to dialog plugin. The class `AbstractSwingDialog` automates work with swing dialogs. Classes `OkCancelPanel` and `OkCancelRestore` panel offers panels which serves for confirming actions.

Display plugins

The interface `PluginUIDisplay` defines method `Obliect display(Object, Object[])`. This method displays the user interface component to user. Further specification of the argument of this method is part of plugin's contract.

Factory plugins

The interface `PluginUIFactory` defines method `Object createComponent(Obect[])`. This method should create arbitrary user interface component. Further specification of the argument of this method is part of plugin's contract.

Factory plugin often returns object of class defined inside it. If it is necessary to call specific methods of such class, plugin must define this methods in its contract.

Listing A.8: Example of the user interface plugin

```
package plugins.ui.swing.dialog;

/**
 * Displays standard yes / no dialog
 *
 */
public class YesNoDialog extends AbstractPlugin
  implements PluginUIDialog {
  /**
   * Displays standard yes / no dialog
   * @param args message to the plugin
   * args[0] ... (String) message displayed in the dialog
   * args[1] ... (optional) (String) title of the dialog
   *
   * @return the object which method get() returns  (Boolean)
   *   true ... user selected yes;
   *   false ... user selected no
   */
  public Arguments getInformations(Object owner,
    Object[] args)
      throws PluginNotSucceededException {

    // Get arguments
    Component ownerC = (Component) owner;
    String message = (String) args[0];
    String title = null;
    if (args.length > 1)
      title = (String) args[1];

    // Display the dialog
    int o = JOptionPane.showConfirmDialog(ownerC, message,
      title, JOptionPane.YES_\textit{NO}_OPTION);
    Arguments retValue = new Arguments();
    if (o == JOptionPane.YES_OPTION) {
      retValue.set(true);
    }
    else {
      retValue.set(false);
    }

    return retValue;
  }

  public GuiIDs guiID() {
    return GuiIDs.SWING;
  }

}
```

A.8 Writing I/O Plugins

I/O plugins handles reading images from the disk and writing images to disk.

Each I/O plugin must implement interface `PluginIO`. This interface defines method `PluginImageFormat[] offers()`. Plugins specify there which image formats it reads or writes and the priority of given format. If there are two plugins that read or write the same format, the plugin with higher priority will be called.

I/O plugins are usually not called directly so they have no exactly specified contract.

The interface `PluginIOReader` implement plugins that read the image from disk and encode it from given format. The interface `PluginIOWriter` implement plugins, which decode image data and write it to disk.

A.9 Writing Worker Plugins

Worker plugins are general-purpose plugins that can handle arbitrary functionality.

Worker plugins implement interface `PluginWorker`. This interface defines the method `Arguments run(Object[])`. Further specification of the argument of this method is part of plugin's contract.

Listing A.9: Example of worker plugin

```
package plugins.worker;

/**
 * Delete plugin. Delete plugin from the disc.
 * Do not care to dependencies etc. This do
 * _UninstallPlugin.
 *
 * @see plugins.worker._UninstallPlugin
 */
public class DeletePlugin extends AbstractPlugin
  implements PluginWorker {

  /**
   * Delete plugin.
   *
   * @param args
   * (String) args[0] ... name of the plugin
   * (String) args[1] ... information about the plugin
   * (Plugin) args[2] ... plugin
   *
   * @return Arguments.NO_ARGUMENTS
   */
  public Arguments run(Object[] args) throws
   PluginNotSucceededException {
    // Get arguments
    String plName = (String) args[0];
    PluginTypes plInfo = (PluginTypes) args[1];
    Plugin plugin = (Plugin) args[2];

    // delete plugin from the disc
    File path = new File(
      plInfo.getAbsPath() + plName + ".class");
    if (!path.delete()) {
      path = new File(
        plInfo.getAbsPath() + plName + ".jar");
      if (!path.delete()) {
        String message = "Cannot delete the plugin file.";
        ErrorMessages.error(message, sourceFile);
        throw new PluginNotSucceededException(message);
      }
    }

    // replace action plugin from the menu
    if (plInfo == PluginTypes.ACTION) {
       Worker.callPlugin("ReplacePluginFromAllMenus",
        plugin, plName);
    }

    return Arguments.NO_ARGUMENTS;
  }
}
```

A.10 Using Action Plugins

The static class `Action` arranges automatic updating of enabled state of menu items or buttons which wraps action plugins according to the actual result of the method `isEnabled()` when some pre-defined event occurs. It is also possible to add new event which causes updating the state of the actions. Then, it is possible to update the enabled state of all action plugins or of selected action plugin manually.

The class `Action` also enables to set new label of action plugin.

While using action plugins, it is necessary to wrap the action plugin to component representing button or menu item. The class `SwingActions` enables to get object of `SwingAction` class that is descendant of class `javax.swing.AbstractAction.AbstractAction` which wraps given action plugin. The objects of class `AbstractAction` can be used to create both buttons and menu items.

Listing A.10: Using action plugins in the buttons

```
AbstractAction action =
  SwingActions.getSwingAction("ActionPlugin")

// Create a button which runs given action plugin
// The button will be created according to the settings
// provided by given action plugin.
// Enabled state of the button will be automatically updated
JButton button = new JButton(action);

// ...

// Update the buttons label
Actions.setLabel("ActionPlugin", "New label");

// Manually update the state of the button
Actions.updateEnabled("ActionPlugin");
```

Built-in user interface factory plugin `MenuFactory` offers the swing component that represents specified menu of the tool. This menu component includes all action plugins specified in XML file with definition of given menu.

Listing A.11: Creating swing menu component

```
// Creates new window
JFrame frame = new JFrame();

// the window will have the same menu as the BrowsingWindow
setJMenuBar((JMenuBar) UI.callFactoryPlugin("_MenuFactory",
  new MenuFactoryArguments(MenuTypes.MENU_BAR),
  MenuInfo.MAIN_MENU_BROWSING.getMenuData()));
```

A.11 Advanced Use of UndoableImage

The image that is user currently viewing is represented by the object of `UndoableImage` class. This object holds two images. The processing image stores the data of the image that is user currrently viewing. This data can be used for further processing or for writing image to disk. The second stored image is display image – the image that is displayed to the user. Typically, it is processing image rescaled to fit the viewing window or translated to be in the center of viewing window.

Fortunately, it is usually not necessary to care about this concept. Both images are updated automatically when using methods such as `updateProcessingImageDisplay(ImagePlus)` or `updateProcessingImageDisplay-`

`Fit(ImagePlus)`. However, setting the processing image and display image independentely is needful for example when the image is zoomed and it is beneficial for example when showing the preview of some operation to the user.

The relationship between the processing and the display image

To allow the proper function of the tool, the processing image and the display image must represent the same image. Moreover, the geometrical transformation between these two images needs to be known.

Thus, the transformation which converts the coordinates of processing image to the coordinates of display image must be passed when using method `setDisplayImage(ImagePlus, AffineTransform)`. Note that this transformation mustn't include the scale transformation. The scale transformation is computed automatically using the dimensions of processing and display image.

Actual transformation between the processing and display image can be obtained by calling method `getProcessingToDisplayCoordinatesConventer()` or `getDisplayToProcessingCoordinatesConverter()`.

When using method `setDisplayImageTransform(ImagePlus)`, the image passed in the argument will be transformed to correspond the processing image in that way that affine transformation between new processing image and new display image will be the same as affine transformation between old display image and new display image. The method `setDisplayImageFit(ImagePlus)` transforms given image to fit the display area of the viewing window. Correct transformation between processing image and display image will be automatically set.

Observing UndoableImage

Class `UndoableImage` is observable. Observers can be registered to be notified when `UndoableImage` is changed. The argument `arg` of of the method `update()` which receive such observers is type `MessagesToObserver`. It specifies the event that occurred.

Observing undoable image is used for example in the component which displays actual image. When the actual image is updated, the component displays it.

A.12 Working with Selections

Selections are used to create image regions and to manipulate with them.

Creating selections

Selections are typically created by the user. The process of creating a selection varies on the shape of the resultant selection. Such process is directed by the object of class implementing `SelectionBuilder` interface. When the selection is created, the object of the class `SelectionManipulator` can be obtained.

Setting the default method of creating selections

The object of `SelectionBuilder` class that is used to create new selections in viewing window is stored in `ActSelection` class. Plugins can change the process of creating new selections in viewing window by setting new object instead of this.

Manipulating with selections

Created selections are represented by objects implementing interface `Selection-Manipulator`. This objects handles the manipulation with the selection and offers method for obtaining image region selected by the selection. Either the manipulation can be driven by mouse events on the image, or it can be necessary to apply some geometrical transformation on the selection.

The object implementing `SelectionManipulator` interface can be created directly or by objects implementing `SelectionBuilder` interface.

Selections and image regions

In fact, image regions in `ImageRegions` class are represented by objects implementing `SelectionManipulator` interface. Consequently, it is possible to transform image regions in the image and manipulate with them.

In spite of that, new image region can be added to an image simply by calling method `add(Shape)` of the object of the class `ImageRegions` stored in the image. In this case, it will be created object of class `ArbitrarySelectionManipulator` from given shape and this object will represent the given image region. However, this object only handles geometrical transformations of image regions. It doesn't enable user manipulation with given image region.

To enable user to manipulate the image region, fitting image manipulator object must be added using method `add(ImageManipulator)`.

Observing selection manipulator

The class `SelectionManipulator` is observable. Observing selections is presently used for example in the implementation of the `Cropping` plugin to change the cropping parameters of the selection representing cropping area moves.

Observing ImageRegions

It is also possible to observe whole `ImageRegions` object. Observers can register to be notified either when some image selection manipulator was changed or when some selection manipulator was added or deleted.

A.13 Using Batch Processing

Batch processing enables to apply given processing plugin to all images in given directory.

This can do built in worker plugin `BatchProcessing`. This plugin accepts in the argument the identifier of processing plugin which will be applied in batch, parameters of this plugin and object of the class `BatchSettings` in which the settings of batch processing and the parameters of given processing plugin are stored.

Getting batch processing settings from standard dialogs

Dialog plugins that serve for obtaining general input from the user often offer getting the batch processing settings. The example of such dialog is `TwoLin-kedNumberSpinnersDialog`.

Getting batch processing settings in custom dialogs

Factory plugin `BatchSettingsPanel` provides a panel which can be added to any swing container and which enables user to enter the batch processing settings. This panel is also used in implementation of general-purpose dialogs mentioned earlier.

Calling plugin which enables batch processing from Browsing and Viewing window

Action plugins that enable batch processing are usually applicable also to the single image and are placed in a menu of a viewing window. It is also good to place such plugins to the menu of the browsing window to the sub-menu called batch processing.

The appearance of such plugin should little vary according to what window is actually active. This can be detected by using method `getState()` of a class `ViewingWindowProperties`.

If viewing window is active, user should choose whether he wants to do batch processing or not. If browsing window is active, the batch processing should be enabled and user shouldn't have the possibility to disable batch processing. Next, the batch processing must be disabled, when the `ActDirectory` is in `DirectoryStates.`*`NO`*`T_BROWSABLE` state.

The panel obtained by plugin `BatchSettingsPanel` do this two things automatically.

Next, if the plugin offers the preview of processing operation, it must be disabled when the viewing window is not active.

The example of action plugin runnable from both processing and viewing window is in Listing A.12

Listing A.12: Example of the the action plugin which enables batch processing

```
package plugins.action;

/**
 * Rotation of image with arbitrary angle.
 * Supports mass processing.
 */
public class Rotate extends AbstractPluginAction {

  @Override
  public void run(Object[] args) throws
    PluginNotSucceededException {

    // Prepare arguments of the dialog
    DialogParametres dialog = new DialogParametres("Rotation",
        "Rotation angle");
    // the dialog will enable user to enter batch settings
     dialog.setBatchSettingsState(BatchSettingsStates.ENABLE);
    SliderParameters slider = new SliderParameters(
        new NumberIntervalWithValue<Integer>(0, 360, 0));
     slider.setTextFieldState(TextFieldStates.DISPLAY);
    // enable preview only if the viewing window is visible
    if (ViewingWindowProperties.getState() ==
      ViewingWindowState.VISIBLE) {
      enablePreview(slider);
    }

    // Obtain user input
    Arguments rotSettings = UI.callDialogPlugin("_Slider",
         ViewingWindowProperties.getViewingWindowContainer(),
        dialog, new SliderParameters[] { slider });

    // Cancel button was pressed
    if (rotSettings == Arguments.NO_ARGUMENTS) {
      if (ViewingWindowProperties.getState() ==
        ViewingWindowState.VISIBLE) {
        // restore the changes caused by preview
         ActImage.getActImage().setDisplayImageFit(
             ActImage.getActImage().getProcessingImage());
      }
      return;
    }

    // Get the parameters of rotation
    int[] values = (int[]) rotSettings.get("values");
    float rotationAngle = ((Integer)values[0]).floatValue();
    // Get the parameters of batch processing
    BatchSettings batchGeneral = (BatchSettings) rotSettings
```

```
          .get("batchSettings");

    // Process the image
    if (batchGeneral == BatchSettings.NO_MASS_PROCESSING) {
      // without batch processing
      // ... process the image, update ActImage
    } else {
      // do batch processing
      Worker.callPlugin("_BatchProcessing", "RotateAuto",
          batchGeneral, rotationAngle,
           Interpolation.getPermanentInterpolation());

    }
  }

  private void enablePreview(SliderParameters sliderArg) {
      // changes display image when the slider moves
      ChangeListener sliderChangeListener =
        new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
          // ...
          }
        }
      };

      sliderArg.setSliderChangeListener(
        sliderChangeListener);
  }

  public String getName() {
    return Localisation.getString("rotate");
  }

}
```

A.14 Other Advanced Issues

Using actual directory

The images in the directory that is user currently browsing and the information about this directory are accessible via static methods of the class `ActDirectory`.
The list of static methods of the class `ActDirectory` is in Table A.12.

The images in the actual directory are stored as an object of `CachedImages` class. This class provides loading images of given files and caching it. This class

Method	**Description**
`void changeActDirectory(File)`	Changes actual directory and initialise the image cache.
`File getDirectoryFile()`	Gets the file of the actual directory.
`void recache()`	Gets images from actual directory and creates new caches from this images.
`CachedImages getCachedImages()`	Gets the object which stores images from actual directory.

Table A.12: Selected static methods of the class ActDirectory.

is `Observable`. Observers can register to be notified when the cache is changed. The list of most frequently used methods of `CachedImages` is in Table A.13.

Method	**Description**
`ImagePlus getImage(int)`	Loads image with given index to the cache and returns it.
`ImagePlus getImage(File)`	Loads image with given File to the cache and returns it.
`ImagePlus geImageStartCaching(int)`	Gets image with given index and starts new thread which will be caching images around this image.
`ImagePlus geImageStartCaching(File)`	Gets image with given file and starts new thread which will be caching images around this image.
`ImagePlus getNextImageCacheAlong()`	Gets next image from the cache and cache image along this image.
`boolean isNextImage()`	Indicates whether there is stored next image in the cache list.
`ImagePlus getPrevImageCacheAlong()`	Gets next image from the cache and cache image before this image.
`boolean isPrevImage()`	Indicates whether there is stored previous image in the cache list.

Table A.13: Most frequently used methods of the class CachedImages

Using I/O

I/O module serves to reading and writing images.

In the table A.14 is the list of most frequently used static methods of the `IO` class.

Method	**Description**
`void saveActualImage()`	Saves actual image to the disk. Updates `ActImage` and `ActDirectory` to be in proper state.
`void write(PlanarImage, String, Object...)`	Writes the image to the disk.
`PlanarImage read(String filename)`	Reads the image from disk.
`boolean isReadSupported(String)`	Returns true if reading of the format format is supported.
`boolean isWriteSupported(String)`	Returns true if writing of the format format is supported.

Table A.14: Most frequently used static methods of the class IO.

Advanced use of the status

The class `Status` class is observable. Hence, observers can register to be noticed when the status was changed.

Consequently, custom component that automatically displays the status of the tool can be created. Such component offers factory plugin `AutoStatusPanel`.

Getting plugins objects via the plugin manager

Typically, plugins are not used directly. One pre-defined method of the plugin's class given by the type of the plugin is called via static classes in the kernel. This makes using and writing of plugins unified and simple.

However, sometimes is the possibility of calling only one pre-defined method of the plugin limiting. It is therefore possible to get the object of given plugin and call some its method via Reflection API.

Note that it is not guaranteed, that all versions of the plugin have the methods that are not specified in the contract of the plugin. Consequently, only the methods given by the interface of given plugin type and methods specified in the contract of the plugin should be called.

Getting the object of the plugin provides the object of `PluginManager` class. Classes `processing.Processing`, `worker.Worker`, `actions.Actions`, `UI`, `io.IO` enables to get such object.

Using classes defined in the plugin

Plugin can return object of class defined inside it. If it is necessary to use specific methods of this class, plugin must specify such methods in its contract and other callers can use this methods via Reflection API.

Next, plugin can provide `Class` objects of classes defined inside the plugin via method `getAdditionalInnerClasses()`. Via Reflection API it is possible to create instances of such classes and call methods of such classes.

A.15 Licence

PhotoJ is distributed under GNU Lesser General Public Licence (LGPL) [16].

Runnable scripts for Windows and Linux (`photoj.vbs` and `photoj.sh`) are distributed under GNU General Public License [17].

Bibliography

[1] HubbleSite: The Telescope - Nuts & Bolts - COSTAR,
http://hubblesite.org/the_telescope/nuts_.and._bolts/optics/costar/

[2] Molecular Expressions Microscopy Primer: Digital Image Processing - Focus Limitations,
http://micro.magnet.fsu.edu/primer/digitalimaging/russ/focuslimitations.html

[3] Wikipedia, the free encyclopedia: Deconvolution,
http://en.wikipedia.org/wiki/Deconvolution

[4] Quarktet SeDDaRA: SeDDaRA blind deconvolution,
http://www.quarktet.com/

[5] ImageJ, Image Processing and Analysis in Java,
http://rsb.info.nih.gov/ij/

[6] JPlugin: Java plugin framework,
https://jplugin.dev.java.net/

[7] JPF: Java Plugin Framework,
http://jpf.sourceforge.net/

[8] EXIF.org: unofficial site dedicated to Exchangeable Image File Format (EXIF)
http://www.exif.org

[9] William K. Pratt: Digital image processing, John Wiley & Sons, Inc., 2001

[10] HDRsoft SARL: FAQ – HDR images for Photography,
http://www.hdrsoft.com/resources/dri.html

[11] Sun Microsystems, Inc: Java Runtime Environment: download,
http://www.java.com/en/download/manual.jsp

[12] Apache Ant: Java-based build tool,
http://ant.apache.org/

[13] Sun Microsystems, Inc: The Java Advanced Imaging API (JAI),
http://java.sun.com/javase/technologies/desktop/media/jai/

[14] Sun Microsystems, Inc: JAI manual,
http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/

[15] Java Advanced Imaging Stuff (jaistuff): JAI tutorial and many examples,
https://jaistuff.dev.java.net/

[16] GNU Lesser General Public Licence (LGPL)
http://www.gnu.org/licenses/lgpl.html

[17] GNU General Public License (GPL)
http://www.gnu.org/licenses/gpl.html

[18] XnView: multimedia viewer, browser, and converter,
http://perso.orange.fr/pierre.g/

[19] Gwenview: image viewer,
http://gwenview.sourceforge.net/

[20] Almara: photo album viewer,
http://almara.sourceforge.net/www/

[21] Ekspos: platform independent Java image viewer program,
http://www.kiyut.com/products/ekspos/index.html

[22] Fotox: Linux program for improving digital photographs,
http://kornelix.squarespace.com/fotox/

[23] Google Picasa: image viewer, browser, and organiser,
http://picasa.google.co.uk/

[24] IrfanView: graphic viewer,
http://www.irfanview.com/

[25] Cygwin: Linux-like envinronment for Windows,
http://www.cygwin.com/

[26] Wine HQ: an Open Source implementation of the Windows API
http://www.winehq.org/