The openIP Open Source Image Processing Library

György Kovács
University of Debrecen
Egyetem ter 1.
4032 Debrecen, Hungary
gykovacs@inf.unideb.hu

János István Iván
University of Debrecen
Egyetem ter 1.
4032 Debrecen, Hungary
janos.istvan.ivan@gmail.com

Árpád Pányik
University of Debrecen
Egyetem ter 1.
4032 Debrecen, Hungary
asd0025@gmail.com

Attila Fazekas
University of Debrecen
Egyetem ter 1.
4032 Debrecen, Hungary
fazekas.attila@inf.unideb.hu

ABSTRACT
The openIP open source image processing library is a set of c++ libraries providing tools for education, research and industrial purposes. The aim of the development is to fill in the gap between the academic and commercial utilization of image processing. The openIP libraries are interoperable, open source and easy to install. To provide fast codes, assembler optimization, OpenMP parallelization and OpenCL is used to provide interface to the GPU processor.

Categories and Subject Descriptors
I.4.0 [Image Processing and Computer Vision]: General—Image processing software; I.4.3 [Image Processing and Computer Vision]: Enhancement—Filtering, Grayscale manipulation, Smoothing; I.4.6 [Image Processing and Computer Vision]: Segmentation—Edge and feature detection, Pixel classification, Region growing; I.4.7 [Image Processing and Computer Vision]: Feature measurement—Feature representation, Invariants, Moments, Size and shape

General Terms
Algorithms

Keywords
image processing, computer vision

1. INTRODUCTION
Education, research and industry are the three main areas, where image processing and computer vision is applied and used. These areas have considerably different requirements:

• Software tools used for educational purposes must have clear and simple abstractions, well documented sources and provide an easy to install environment for beginner and unexperienced programmers.

• Researchers need almost all the published methods implemented of the scientific area they work in making it easy to test and compare new approaches to the previous ones.

• Industry requires the available fastest implementations, using as many capabilities of the given hardware devices as possible.

Although there are several libraries, packages and tools for image processing, none of them fulfills all these requirements, implying that the algorithms developed by researchers must be reimplemented for commercial purposes, on the other hand it is hard to extend the highly optimized libraries. Based on experiencing this phenomena through several years working in image processing the openIP open source image processing library has been started to develop by the members of the Image Processing Group of Debrecen (http://ipgd.inf.unideb.hu), Debrecen, Hungary. In Section 2 the main concepts and ideas of openIP (http://code.google.com/p/openip) library are presented, developed for image processing and computer vision, providing some technical details of the library with a coarse enumeration of implemented algorithms, in Section 3 a sample application is discussed presenting the way of using our library and in Section 4 we conclude with the further plans.

2. THE MAIN CONCEPTS
The main concept of the openIP package is to provide a set of interoperable, open source libraries, satisfying the demands of education, research and industry as well.

2.1 The openIP in education
The interfaces and sources of the openIP libraries are totally documented in doxygen style, providing an easy to read set of code for developers wanting to extend it.

Each of the implemented image processing and computer vision algorithms has a reference for the paper, based on which it had been implemented. Using this reference, arbitrary user of the libraries can examine, how the methods are
implemented, students can learn c++ coding techniques and also check out the implementation of the complex numerical methods, which is usually the most difficult part when the algorithm to implement, has strong mathematical basis.

The libraries can be easily built from source in Windows and Linux environments, as well, since all the third party libraries are wrapped in own openIP libraries. In this sense unexperienced users do not have to spend time on collecting, configuring and compiling relatively small libraries and deal with complicated build and linker commands, since almost everything is integrated into the libraries.

2.2 The openIP in research

The literature of image processing and computer vision is enormous, e.g. there are more than 30 algorithms published for the adaptive thresholding [1] of images. When developing a new method for adaptive thresholding and performance tests are required, or trying to find the method best fitting the current problem, it is time consuming work to implement all the published algorithms. One of the most important concepts of the openIP libraries is to have several different algorithms implemented for the same purpose. The main areas of image processing and computer vision are systematically processed and more-or-less all the published methods are integrated. In the current version of the library the adaptive thresholding methods (29 pcs.) and the common contrast enhancement algorithms (8 pcs.) are implemented. The libraries are extended version-by-version, therefore in the next releases further set of algorithms are going to be integrated.

2.3 The openIP in industry

Although the openIP libraries have quite clear and simple abstractions, the core (usually a couple of functions) of these abstractions is highly optimized to provide the fastest implementations. At the lowest level, currently in some filters SSE2 assembler based optimization is implemented. To use the hardware of newest desktop computers, the highly parallelizable methods like filtering, morphological operations, algebraic image operations use OpenMP[3] parallelization, furthermore OpenCL [4] support is integrated for the really time consuming tasks, like filtering in image space using large (> 7) kernels. Since the newest technologies are used in the library, the algorithms developed by the image processing researchers using the tools of the openIP, can get linked directly into the commercial application, without the need to reimplement and optimize it for the common environments and hardware devices.

3. THE STRUCTURE OF THE LIBRARIES

The current release of the openIP package contains 6 libraries, the structure can be seen in Figure 3. A short summary of the packages and the implemented features are presented in the following subsections:

3.1 openipDS

The openipDS library is the core of the openIP libraries, containing the basic data structures for signal and image processing. These data structures are highly optimized and provide lots of useful methods and abstractions to make the implementation of high level image processing algorithms easy and fast. Without the demand of completeness the implemented and abstractions are Vector, Image, Volume, Pix-

![Figure 1: The structure of the openIP libraries.](image)

Image processing tasks are considerably memory intensive, and many of the image processing algorithms requires temporary images to contain scalar fields or vector fields, however the allocation of memory areas for images any time an algorithm is called is not acceptable, and the passing of pre-allocated work spaces is usually confusing, since the functions require many more parameters for temporary data structures. To overcome this well known issue, a special caching mechanism has been developed: Whenever an algorithm is developed, it is known, how many and what kind of temporary images does it need to operate. Since all these requirements are known in the beginning of the program, the factory design pattern could be applied, that is taking the union of the requirements of sequentially operating algorithms, all the temporary images should be allocated once, in the beginning of the program and the temporary images are passed through the Cache object to the algorithms. In the level of implementation, there is a constant of type int declared to each of the algorithms, the first byte of the constant declares the number (0,255) of unsigned char temporary images required by the algorithm, and the next bytes in order the requirements of int, float and double images. (e.g. reading RGB images this constant is CACHE_READ_RGB_IMAGE and the first byte declares the need for 3 unsigned char images.) In the beginning of the program, the Cache object gets these descriptor constants and when the size of the image to work with is determined, the Cache object allocates all the required memory. Using this mechanism the instantiation of temporary images are avoided, since any number of temporary images are required by an algorithm, if it is declared in the descriptor constant, the algorithm must get only the Cache object, pull out the temporary images, and
pushing back them into the Cache for the further methods called by the program after the operation is finished.

Image processing algorithms like filtering, computing image statistics, etc., are integrated into the appropriate data structures. The core of these algorithms is optimized using OpenMP parallelization, but our concept is to support the OpenCL technology, as well. In the current 0.0.3 version of the openIP library the filter operations are equipped with OpenCL support: filter objects can generate the OpenCL kernel code as a string which can be compiled by the OpenCL environment in runtime and used directly with GPU processors.

3.2 openipIO

The openipIO library contains and wraps the most common image input/output libraries. Wrapping is needed to prevent the users from setting up complicated dependencies and environment for the openIP package. Due to this solution, the same set of code can be used in both Windows and Linux operating systems. In the current 0.0.3 release of the package the supported images formats are jpg, png, pgm, bmp, gif, tiff.

3.3 openipSC

The openipSC library contains and wraps the GNU Scientific Library [2]. The reason for containing the source code is similar as in the case of openipIO: the openIP package is a standalone set of libraries, no dependencies are required to be installed nor in Windows neither in Linux systems. The main features currently wrapped are numerical methods for linear algebra (e.g. eigenvalue/eigenvector computations, matrix inverse and determinant computing) and the Fourier transform.

3.4 openipLL

The name openipLL stands for low level image processing operations. Using the data structures of the openipDS library and the numerical methods of the openipSC library, the following features are implemented in the current release: color space conversions (RGB, HSV, HSL, XYZ, YCbCr, CIELuv), Fourier filters, image enhancement methods, interpolation techniques, binary/grayscale morphological operations, binary thinning methods, adaptive thresholding methods.

3.5 openipML

The openipML library implements some basic machine learning algorithms, like k-means clustering, fuzzy means clustering, Gaussian mixture model. Simulated Annealing is implemented to solve the Hidden Markov Models also present in openipML library.

3.6 openipVS

The name openipVS stands for visualization. This is the only library depending on a 3rd party package, namely on the Qt4, containing visualization tools for both images and 3D data sets. On Figure ?? one screenshot of the 3D visualization software can be seen. There are several image processing algorithms in which user input is required, however building Graphical User Interfaces (GUIs) in the phase of research and the development of algorithms is time consuming and in many cases unnecessary, since the final software can have much different and complicated GUI. To overcome this problem, we have worked out the concept of image processing dialog primitives: The user interactions required by image processing algorithms are usually similar. Besides opening and saving images, the most common interactions are

- the selection of set of position and working with the coordinates (e.g. initialization of snakes)
- the selection of region of interest (ROI), when some operation should be performed only in a special region of the image.

Currently these special dialog primitives are implemented in the openipVS library as standalone classes, therefore once the developer needs some interaction on images, no GUI building is required, only the appropriate dialog primitive must be instantiated and parametrized with the images on which interaction is needed, and the coordinates of selected points, the region of interest are pushed back as output parameters of the dialog primitive. Besides the interactions in the phase of developments, the chain of dialog primitives can be used even as final GUI for simple applications, similar to the graphical interface of Windows installer applications, but satisfactory for educational, research or even for some commercial purposes.

3.7 The build system

The build system is a crucial part of any software development, specially in interoperable environments. To insist upon the open source approach we are using the open source CMake [5] as general build system and the GNU Compiler Collection in both Linux and Windows environments.

4. SAMPLE CODE

Below a short tutorial is provided to reflect the way of coding in openIP. The code block realizes the application of the locally adaptive Niblack thresholding method to the
grayscale version of the RGB input image passed by command line argument. First of all a Cache2 object is instantiated, storing enough image to read an RGB image and one more for the result. The constant CACHE_READ_RGB_IMAGE represents the need for three unsigned char typed image objects, one more image from the same type is required to store the grayscale image and some further temporary images are required by the Yasuda thresholding method, declared in the CACHE_THRESHOLDING_YASUDA variable. Note that at this point of the application no memory gets allocated.

```
Cache2 c(CACHE_READ_RGB_IMAGE + CACHE_UCHAR_1 + CACHE_THRESHOLDING_YASUDA);
```

Pointers are declared for the channels of the image and the resulting image.

```
Image<unsigned char> *r, *g, *b, *gray;
```

The image is read in and when the size of the image gets known, all the images get allocated and three unsigned char typed images are grabbed out from the cache and put into the pointers.

```
readImage(argv[1], &c, &r, &g, &b);
```

The forth unsigned char typed image is grabbed out from the cache to contain the grayscale image.

```
c.get(&gray);
```

Now the RGB channels are converted into one grayscale image.

```
rgb2gray(*r, *g, *b, *gray);
```

The adaptive local thresholding method of Yasuda is applied, and since it requires temporary images, the cache is passed, as well. Since the red channel of the original image is not required any more, the thresholded image gets into the images referenced by the pointer r.

```
thresholdingLocalYasuda(*gray, *r, &c);
```

The result is written out using the name of the second command line argument:

```
writeImage(argv[2], r, r, r);
```

The input and output images of this sample application can be seen in Figure 4.

5. CONCLUSIONS AND FURTHER PLANS

The purpose of the openIP is to provide an easy to use, but highly optimized set of open source libraries dedicated to image processing and computer vision. Since the covering of all fields in image processing exceeds our means, we would like to build an international community working and developing through openIP, resulting a continuously extending tool for anybody dealing with images processing. Although the library is less than a half year old, there are several features implemented, and the concepts seems to provide good bases for further developments in image processing. Currently we are extending the library in several directions:

- integrating OpenCL support for further data structures,
- the implementation of the more-or-less common filter and feature extraction algorithms,
- the integration of further machine learning algorithms like Support Vector Machines and Neural Networks,
- development of further dialog primitives,
- integrating the support for real time video processing,
- implementation of 3D stereoscopic reconstruction algorithms,
- fine tuning of the build system,
- integrating support for further image formats.

6. REFERENCES