Language for Scientific Imaging: Design, Implementation and Applications

Vladimir A Smirnov, Evgeniy V Korolev and Olga I Poddaeva

Moscow State University of Civil Engineering

Abstract**:** *Currently, there is a lot of software designed for digital image processing. At the same time, high availability of such software often leads to inappropriate selection of implementations which are not intended for scientific imaging. In the present paper the necessary and desirable requirements for the scientific imaging software is summarized. We have offered simple and easy to use language which can be used for description of many digital imaging tasks, notably in material science. The implemented interpreter for the proposed language is highly portable and can be compiled on many POSIX compliant systems and also on Microsoft Windows. Target area of applications includes scanning probe imaging, frequency-domain analysis and pattern recognition.*

Keywords: *Digital signal processing, Image processing, Problem-oriented language*

1. Introduction

In computer science, image processing is an enormous topic. Broad area of applications includes digital artwork processing, video games, geographic information systems and many others. Because of numerous applications there are also various requirements for methods, algorithms and software implementations.

For example, in digital artwork processing, eligibility of software is usually defined with criteria based on visual acceptance of resulting image and, to the less extent, on execution time. For video games, two mentioned criteria are swapped: execution time becomes more important than visual quality. In either case, the details of implementation (precision of calculations, etc.) are not taken into account.

On the contrary, in scientific applications it is of high importance to use exact algorithms and strictly defined translation methods (precise control of resulting machine code). There is no criterion similar to "visual quality", though visualization is still important. If the execution time is unacceptably high, then the common (for industrial applications) way to remedy this can be used, namely increasing processing power of hardware (albeit effectiveness of algorithms is still substantial).

The presence of these two distinctive subdomains of image processing shows that there can be many groups of requirements, possibly intersecting each other. But one of the requirements is common for all groups (in fact, it is relevant to vast majority of objects other than software), and, at the same time, is the most complex. It is denoted by simple term "easy of use". Though the exact meaning of this term can be revealed in systems engineering [1,2] and ergonomics, for our purposes it is enough to formally represent "easy of use" criterion as:

$$\sum_{i=1}^{N} t_i \to \min, \tag{1}$$

where t_i is the time of i-th stage of life cycle, N is the number of stages in life cycle.

The life cycle can be composed of:

- searching among available open-source and commercial software;
- developing the software;
- training to use the software;
- using the software for successful solution of specific tasks.

It is obvious that outcome of some stage can affect the presence and duration of some other stages. This is the primary source of complexity for "easy of use" criterion.

Considering image processing, the first of aforesaid stages can be quite long. There are a lot of implementations, both open-source and commercial. The common mistake consists in using the digital artwork processing software (general-purpose raster graphics editors) for scientific applications. As a rule, this mistake is due to two interlinked reasons: high availability of artwork processing software and natural desire to exclude all aforementioned life cycle stages except the last one. But, after all, why this should be considered as a mistake? To justify such a statement it is enough to summarize basic requirements for scientific image processing software and examine several commonly used packages starting from raster graphics editors.

2. Scientific Image Processing

We can define "scientific digital image processing" as digital image processing (which, in turn, is a form of digital signal processing) by algorithms and programs satisfying following requirements.

- Comprehensive information about implementation details. All implemented transformation must be either based on widely accepted and extensively verified library routines, or documented in detail.
- Control of precision. For all integer algorithms, it should be possible to use "bit perfect", or "bit identical" implementations. For such implementations, in particular, if $F(\mathbf{A})$ is reversible transform and $\Phi(\mathbf{B})$ is the inverse of $F(\mathbf{A})$, than every element of the $\Phi(F(\mathbf{A}))$ should be exactly equal to a_{ij}. For the floating point arithmetic the equality is relaxed, but the error should be of same order of magnitude as floating point precision.

There are also several desired features which are not strictly necessary.

- Suitability for packet jobs. All functionality of the software must be exposed either via command line switches, or via embedded scripting language; this does not mean that graphical user interface (GUI) can not be used.
- Suitability for high-performance computing environments. As many as possible levels of parallelization should be implemented: coarse grained process-level parallelization (by means of fork/CreateProcess and inter-process communication), thread-level parallelization (POSIX or Windows threads) and fine-grained parallelization appropriate for computations on geometry processing units (GPU), preferable with OpenCL interface, if it is proven that such a choice does not lead to significant loss of performance in comparison with proprietary GPU application programming interfaces.
- Portability of software across different environments.
- Minimized use of the third-party libraries. For command line software, it is preferable to use only system runtime libraries.

Third and fourth features are quite obvious, as they lead to reduction of t_N in (1) directly. Fifth and sixth features in long term perspective can also reduce several summands in (1), though during development of new software they cause extra time consumption. Furthermore, it should be noted that fifth and sixth features are somewhat opposite to each other because the best way to achieve portability is to use third-party abstraction layers (so-called "middleware") between software and underlying platform.

3. Prior Work

Upon definition of mandatory and optional features of the scientific image processing software, we can proceed to the first term in (1): analysis of available software for processing 2D arrays of chromatic coordinates.

GNU Image Manipulation Program (GIMP) [3, 4] is the most widely used open-source raster graphics editor. It contains advanced tools for the transition between different color spaces. Statistical analysis of the information in any color channel is possible; however, this analysis is performed by means of an interpreted language and therefore quite demanding for computing resources. Convolutions in spatial domain can be calculated (Main menu, "Filters/Generic/Convolution Matrix") with kernels up to 5x5 in size. GIMP is extensible and can operate in batch mode. The input language is a superset of a LISP-like language. The syntax and semantics of this language are very different from the syntax and semantics of the most widespread universal programming

languages. Because of this, we can expect extra time consumptions reflected by term in (1) which corresponds to "training" stage. The core functionality of GIMP is sequential, while several plugins can utilize thread level parallelization.

As many other general-purpose raster editors, GIMP lacks many features required for scientific image processing (Fourier transforms, convolution and deconvolution with arbitrary kernels, etc.). At the time of this writing, the primary drawback of the GIMP is an inability to work with high-precision (high dynamic range, HDR) images. This feature is planned for stable 2.10 release, and is now only in unstable 2.9 branch. Inability to process floating point data makes several operations impossible, including arbitrary convolutions (if the sum over kernel are not unit value, then convolution leads to data loss due to clamping).

Darktable [5] is another open source image processing tool directed to interactive operation via GUI. Its primary aim is postprocessing of digital camera images, thus there are only minimal pixel-level editing features (this is similar to many dedicated scientific imaging software). Implemented in Darktable transforms of chromatic coordinates are of very limited use in scientific image processing.

The ImageMagick [6] and GraphicsMagick [7] software suites are originally designed for batch jobs, with "UNIX Way" in mind (the latter suite was forked from former one in 2003). In these suites several commands (either external shell commands [6] or arguments of "gm" command [7]) are implemented for typical image processing tasks, including image input/output, resampling, linear operations on chromatic coordinates, convolution with predefined and arbitrary kernels, etc. HDR imaging is supported, though in many cases support for appropriate floating point file formats (such as OpenEXR) is not enabled at compilation stage. There is also no way for explicit precision control. The consistent and stable API for image processing is exposed and can be utilized from almost any compiled language. Thread-level parallelization using OpenMP application programming interface is implemented in GraphicsMagick [7]. The software is extensible through "Magick scripting language" (it is in "alpha" state for several years) with XML syntax, which is not well suited for writing programs by hand.

To summarize three foregoing cases of general purpose raster editors, it is enough to say that for any of them first requirement is met only implicitly (due to ability to analyze source code) and second one is met only partially. In fact, it means that neither one of the requirements is met; this justifies aforesaid statement about common mistake. As for commercial raster editors, inability to analyze source code may lead to further complications.

Of cause, image processing functionality can be either implemented or utilized within numerical mathematics (computer algebra) software systems such as MATLAB [8], Octave [9], Scilab [10] or R [11]. Image processing packages for computer algebra systems usually implements most useful functionality, including integral transforms and convolution in reciprocal space. All computations are internally carried out in floating point format. In principle, all necessary input/output procedures can be implemented in native language of used computer algebra system. However, this approach is not free from some drawbacks. If the required image processing functionality is not already implemented, than it is necessary for end user to familiarize himself with new programming language and object model (making batch script for ImageMagick is less time consuming). Utilization of parallel computation for image processing is impossible in case of sequential basic routines. Though many open source computer algebra systems are cross-platform, there is usually a heavy burden of numerous third-party libraries and extraneous environments such as Java Runtime. In fact, the latter, while proposed as a remedy for portability, long time serves as an opposite (notably, this is especially true for FreeBSD).

4. Our Design

During several research works in material science it had became obvious to us that, taking into account aforesaid mandatory and optional requirements, it is preferable to implement image processing software from

scratch. Aims of the new design were formulated [12] according to (1) and requirements which are summarized above.

- Either 32- of 64-bit floating point arithmetic is used for all internal computations (selection of floating point precision can be done during compilation stage).
- All computationally intensive routines (integral transforms, per-element matrix operations, etc.) are implemented in compiled language.
- There is no object model and there are only two base types (vector and matrix; another one was added later on). There is no flow control in base language. Thus, it is almost nothing to learn to start using the software.
- Every instance of base type is defined by series of sequential operations, starting from input operator. These series are position-independent.
- It is possible to extend the base language. In the extensions, there is only one flow control statement. Loops over elements are denoted by special keywords. Such design allows easy translation into code for geometry processing unit.
- Implementation is not based on third-party libraries. Required middleware is bundled with source code.

5. Implementation

The software and middleware were written in ANSI C programming language; we believe that object-oriented design can easily be implemented without object-oriented language like C++, Objective C or any current fashionable dialect such as C# (the latter only decreases portability). After more than ten years of maintenance it is proved that such decision was quite wise: the software can be compiled on modern 64-bit Linux and FreeBSD, with latest GNU compiler collection or LLVM/clang without changing any single line of inherited source code.

According to the design goals, image processing software was implemented as an interpreter for problem-oriented language. Service layer, platform-dependent calls and many library routines (Fourier transform, matrix operations, lexical analysis) are encapsulated in middleware. Because of this, the total number of lines in source code is only about 10^4. Such small amount is perfectly comparable with amount of code in third-party packages for computer algebra systems; but the performance of C implementation is significantly better.

The formal definition of syntax was given in previous works [12, 13].

6. Examples Of Application

6.1. Basic Example

The structure of input script for the designed software can be illustrated by the following example.

```
global {
    alias    width   256
    alias    height  128
}
matrix plasma {
    expand width height
    uniform 0 1
    fourier_forward
    band_pass 0 5
    fourier_inverse
```

```
    range_normalize 0 1
}
targets { output out.bmp RGB plasma plasma plasma }
```

As a result of execution, seamless grayscale "plasma" texture will be produced (Fig. 1). To make similar color texture with full value and intensity and "plasma" pattern in hue channel, it is enough to make following changes:

```
targets { output out.bmp HSV plasma unit unit }
matrix unit {
    state     pad_value 1
    expand  width height
}
```

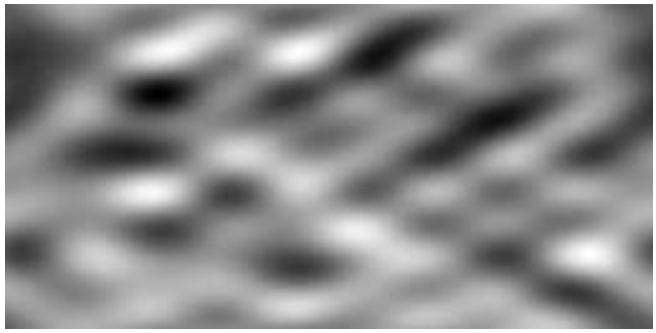

Fig. 1: Raster image produced by script in Section 6.1.

Parsing of the script starts from "global" section. This section is for definition of constants and control variables (e.g. initial value for random number generator). Then, lexical analyzer proceeds to the "target" section. Every line in this section denotes either raster image or object of the base type (matrix, vector or binary image decomposition). Definition of raster image includes color space ("RGB", "HSV", "RGBA", etc.) and matrices for each channel of the color space.

For every explicitly specified of implicitly used (during image definition) object of base type, search in the cache is performed. If the object is not yet evaluated, current position in input stream is saved on top of stack; lexical analyzer finds and evaluates required object and then returns to processing of "targets" section.

In the first example the channel with "plasma" pattern is used for creation of grayscale image (RGB color space, all channels are set to identical patterns). In the second example "hue-saturation-value" (HSV) color space is selected for output image; saturation and value are set to maximum (which represented by unit value) and "plasma" pattern is used for hue channel.

Generation of "plasma" pattern is done by means of digital filtering. Size of the resulting matrix is set to 128x256, than "uniform" operation is used to set all elements to uniformly distributed in range from zero to one values (in fact, due to renormalization, latter values can be arbitrary). Next operation, "fourier_forward", replaces all values of the spatial domain pattern with values if its Fourier transform (for real-valued images the space required for such transform exactly equal to the space of original image). The "band_pass" operation is used for rejection of high spatial frequencies in reciprocal space; "fourier_inverse" return the image to spatial domain. The last step is to perform linear transformation of values in spatial domain in such a way that elements of new image will be in range from zero to one.

6.2. Scanning probe microscopy: roughness of the surface

During processing of digital images acquired with scanning probe microscopy (SPM) it is often necessary to compute several scalar characteristics describing the morphology of the surface. One of such characteristics is the roughness. For continuous fields it is defined as:

$$r = \sqrt{\frac{1}{D}\iint_D \left(z - \langle z \rangle\right)^2 ds}\,, \tag{2}$$

where D is the area and $\langle z \rangle$ is the average applicate of the surface:

$$\langle z \rangle = \frac{1}{D}\iint_D z ds\,. \tag{3}$$

Unbiased discrete analogues for (2) and (3) are:

$$r = \sqrt{\frac{1}{(M-1)(N-1)}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(z_{ij} - \langle z \rangle\right)^2}\,,\ \langle z \rangle = \frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M} z_{ij}\,. \tag{4}$$

In the designed software there are internal commands for evaluation of (4). But evaluation of roughness can also be done with help of embedded scripting language:

```
targets { matrix in }
matrix in {
    input spm.bmp value
    matrix_transform roughness
}
matrix_transform roughness() {
    step {
            var dz, avg, dev;
            avg = 0;
            dev = 0;
    }
    pass { avg = (avg,$CELL,+); }
    step {
            avg = (avg,$MAX_ROW,1,+,$MAX_COL,1,+,*,/);
    }
    pass {
            dz = ($CELL,avg,-);
            dev = (dev,dz,dz,*,+);
    }
    step {
            dev = (dev,$MAX_ROW,$MAX_COL,*,/,0.5,^);
    print dev;
    }
}
```

It is evident from the last example that expressions have to be represented in postfix notation. Such notation was initially selected for the sake of code simplification. Still, we observe no reasons to change the notation.

The roughness for image on Fig. 2 is equal to 0.18.

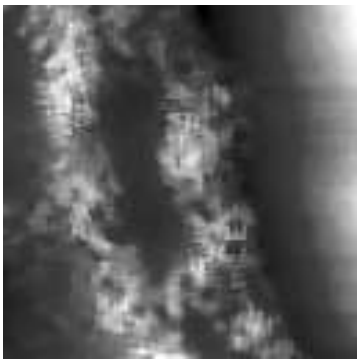

Fig. 2: Reference SPM image.

Aside from (4), other advanced methods for SPM image characterization are also implemented. These methods are based on hierarchical decomposition both in spatial and frequency (Fourier) domains.

6.3. Scanning probe microscopy: histogram of the image

Histogram of the image in Fig. 2 can be computed as follows:

```
matrix in { input spm.bmp value }
vector histogram {
    state custom_size 16
    range_histogram in 0 1
    save out.txt
}
targets { vector histogram }
```

The resulting histogram is presented on Fig. 3.

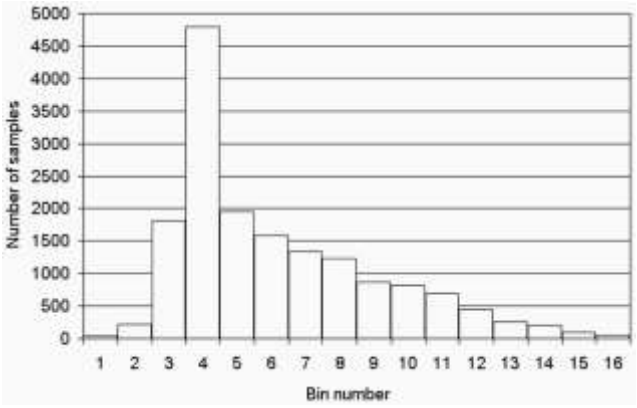

Fig. 3: Histogram of the SPM image on Fig. 2.

6.4. Convolution and deconvolution

During application of SPM techniques it is necessary to correct raw image from the distortion caused by non-zero dimensions of the SPM probe. The correct way to do this is in two step procedure. On the first step, shape of the probe should be determined with reference surfaces of predefined relievo. On the second step it

must be taken into account that SPM image is the convolution of real relievo with transfer function of the probe. Thus, the sough-for relievo can be computed as deconvolution; for such purpose, per-element division of two Fourier transforms ("mat_unscale" operation) can be used in the designed software. The script for the deconvolution can be composed from "target" section (with one target image) and two matrix definitions. One of the definitions should contain "fourier_forward", "mat_unscale" and "fourier_inverse" operators.

7. Summary and conclusion

In the present work we have formulated mandatory and optional requirements for the software which is to be used for scientific digital imaging. It is demonstrated that "easy of use" criterion commonly attributed to interactive raster editors has, in fact, almost nothing to do with GUI. Software should be considered as "easy to use" when total time required for development, learning and application is minimized. Because of this, it is sometimes desirable to design and implement software even in such areas as image processing. Formulated specific requirements were taken info account during design and implementation; the resulting software is an interpreter for problem-oriented language.

Syntax of the proposed language is mush simple than syntax of general-purpose languages, both compiled and interpreted. The software is of high portability and can be compiled on many POSIX platforms (including Linux and FreeBSD) as well as on Microsoft Windows. Several examples of practical applications for scanning probe imaging are presented. The examples cleanly demonstrate that for series of routine tasks total time consumption required for image processing can be reduced significantly if we use the designed software. This is especially true for the joint use with traditional POSIX tools, shell scripting and macro processors.

8. Acknowledgements

This work is supported by the Ministry of Science and Education of Russian Federation, Project #7.11.2014/K "Theoretical and experimental study of the dynamics of constructions".

9. References

[1] R. Stevens, *Systems Engineering: Coping with Complexity*, Pearson Education, 1998.

[2] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*, Addison-Wesley Professional, 2003.

[3] GIMP – The GNU Image Manipulation Program. Available: http://www.gimp.org

[4] O. Lecarme and K. Delvare, *The book of GIMP: A Complete Guide to Nearly Everything*, No Starch Press, 2013.

[5] S. Fornari, M. Latronico and N. Manea, *Digital Photo Development with Darktable*, Available: http://sourceforge.net/projects/darktable/files/darktable/book/1.1.1/darktable-1.1.1.pdf/download

[6] M. Still, *The Definitive Guide to ImageMagick*, Apress, 2005.

[7] GraphicsMagick Image Processing System. Available: http://www.graphicsmagick.org

[8] G. Blanchet and M. Charbit, *Digital Signal and Image Processing using MATLAB. Volume 1. Fundamentals*. Wiley, 2014.

[9] J.W. Eaton, D. Bateman and S. Hauberg, *GNU Octave Manual*, Network Theory Ltd, 2002.

[10] H. Galda, Image Processing Design Toolbox. Available: https://atoms.scilab.org/toolboxes/IPD

[11] T. Perciano and A.C. Frery, Package 'ripa'. Available: http://cran.r-project.org/web/packages/ripa/index.html

[12] V.I. Loganina, V.A. Smirnov, S.N. Kislytsyna, O.A. Zakharov and V.G. Christolybov, "Estimation of properties of decorative coatings," *Russian Coatings Journal*, no. 8, pp. 10-12, 2004.

[13] V.A. Smirnov, E.V. Korolev, A.M. Danilov and A.N. Kruglova, "Fractal Analysis of the Nanomodified Composite Microstructure," Nanotechnologies in Construction: *A Scientific Internet-Journal*, no. 5, pp. 78-86, 2011.