

Perl Graphic Applications with Image Processing

Mustafa ŞİMŞEK*

Abstract

The Perl language is a powerful tool for creation of portable programs on a wide range of platforms. Support of the graphic user interface is not included in Perl, and is provided by several stand-alone graphical toolkits. The most popular modules are Perl-Tk and Wx, based on multi-platform libraries, correspondingly Tcl-Tk and wxWindows, both written in C. The toolkit Prima was written to provide features non-existent in the other toolkits, and it was used as a base for development of research applications with the need for both graphic interface and image processing.

The target biological research comprises a study of the effects of applying various peptide-based preparations on neuron culture, with subsequent quantification of morphological changes in cells. The software provides both manual and automatic quantification, where the former depends on graphic interface functionality, and both on numerical processing. The toolkits Prima and PDL are used to demonstrate Perl capabilities in the intersection area between image processing and graphic user interface.

Prima is a platform-independent Perl graphic toolkit with an object-oriented interface. Its features include an extensive set of Perl-coded widgets, a wide range of image types and conversion routines, and a visual builder. PDL is a popular numerical toolkit featuring efficient storage of numerical arrays. It contains a wide spectrum of calculating functionalities, including a set of image processing operators designed to work with two and more dimensional data. The work illustrates the usage of Prima and PDL libraries for image conversion and display.

Keywords: Image Processing, Perl, graphic applications.

Introduction

In the last years, significant amount of efforts has been put into porting the Perl language to a widening range of platforms. Mac OS, VMS, OS/2, Win32, just to name a few from the list of the supported non-unix systems are used to run scripts and modules, written for the other platforms (Wall L, 2000). Perl evolves, it faces new areas of programming, and one of these is the graphic user interface. Traditionally, GUIs are written using toolkits, which range from simple, single-platform wrappers found in abundance, to vast suites with long-formed traditions and aged community. In the Perl area, Perl-Tk (Walsh N, 1999) is probably the most popular library for the GUI programming. It has a large advantage over its rivals that it is a platform-independent tool, and has a relatively long history. Its closest competitor, WxWindows, shares most of Perl-Tk functionality, but its distribution is somewhat lesser. These toolkits offer a more or less orthogonal set of functions that cover window management, input, 2D graphic output, standard dialogs, and the event loop.

As a base for small scripts and programs with infrequent graphic needs, these libraries can be seen as an ideal solution. When it comes to larger projects though, C toolkits are usually preferred. The key in understanding why this are happening lies in the nature of Perl connection to the user C code, namely the XS interface. Although

this area is very well documented, it often turns into an unexpected obstacle to one who needs accessing C code from Perl, but have not been aware of existence of the XS layer. On the other hand, it is not possible to abolish the need of using C code for any particular Perl toolkit, nor this is desirable. Moreover, the balance between the functionality provided at Perl level and the expected functionality is the decision factor between Perl and C tools. Both Perl-Tk and Wx are based on C-written libraries, correspondingly Tcl-Tk and Wx-Windows. Their integration in the Perl domain introduced the decision line, which delimits features accessible and not accessible from Perl. The new toolkit Prima is designed so its functionality lies mostly in Perl, and in particular requires no C code to extend its widget set, - which is not so for Tk and Wx. As a demonstration of a successful combination of the both C and Perl, the PDL toolkit can be used as an example. PDL stands for Perl Data Language and is devised for manipulation of multi-dimensional numeric tables. One of its modules, PDL::PP, provides an efficient translation of pseudo-code into XS code, which converted into C code. With the calculation speed of C, and coding convenience of Perl, the toolkit presents an easily expandable framework.

The PDL toolkit (Glazebrook K., 2001) is a popular solution for Perl-based implementation of numeric algorithms. It is also an option when the delivery of a program is troublesome due to the licensing or other restrictions of

* A Ph.D. at the International Black Sea University at the Faculty of Computer Technologies and Engineering,

E-mail: msimsek2003@hotmail.com

research tools. As to the author's knowledge, there are no viable results of attempts to combine a mathematical and a GUI toolkit within the reach of the Perl language. Here PDL is demonstrated to be successfully used together with Prima.

Prima

Prima (<http://www.prima.eu.org/>) is a platform-independent, Perl-based GUI toolkit, working on UNIX, Linux, Win32, and OS/2 platforms. In addition to the functionality of the reviewed GUI libraries, it provides an extensible set of widgets, coded purely in Perl. Having been designed originally for image processing needs, the toolkit developed into a library with several features never being implemented or well-developed in the Perl GUI world before (Berezin A., 1999). As proof, the statement below exemplified the toolkit concept of callback functions. The most widespread paradigm in object-oriented programming is to define a set of virtual subroutines, where overloading is performed via sub-classing. While this approach is also implemented in the toolkit, the other, more favorable method was used, when one or more anonymous subroutines is attached to the object instance, contrary to the object class. This technique yields a convenient coding style when a widget or a window needs custom functionality.

For example, on Paint callback type is called whenever the re-painting is needed, very much alike in many GUI designs, but the difference is that the function can be either anonymous or a classic overloaded subroutine. Moreover, there can be more than one anonymous callback function for each callback type. The concept aims the same goals as the subroutine overloading approach does, but achieves these with greater flexibility. In the example above, on Destroy, callback function is not the only code executed when the Destroy notification is triggered; the class `Prima::Window` in capsule intrinsic code for the notification, plus the object instance can contain other on Destroy callback functions. The example 'Hello World' is a fully functional script that shows a window with string "Hello world" and terminates when the window is closed by the user. The concepts of object options (size, centered, etc), and the event loop is similar to the ones in Tk, except the painting routine, which allows wider set of functions. This aspect is important to the applications, dealing with image drawing and data plotting. While the Tk approach allows easier management of the on-screen objects, it is badly scalable, and requires C coding when the more complex drawing routines are required, for example, with custom clipping regions or raster logical operations. Another feature of the Prima toolkit is a large set of supported pixel formats, and the means to convert between these. In addition to 1, 4, 8, and 24 bits per pixel formats, also byte, short, long, float, double, and complex pixel types are sup-

ported.

If an image conversion involves data down-sampling, one of four error distribution algorithms can be selected. List of the other features of the toolkit includes a RAD3-style visual builder, Pod4 viewer classes, and range of supported image file formats, Unicode, and PostScript(tm) printer interface.

```
use Prima
qw(Application);
my $code = scalar('cat $0');
Prima::Window-> create(
size => [ 200, 200],
text => "Hello world",
centered => 1,
onPaint => sub {
my ( $self, $canvas) = @_;
```

```
Scanvas-> clear;
Scanvas-> draw_text($code, 0,
0, $canvas-> size); },
onDestroy => sub {
$::application-> close;});
run Prima;
```

PDL

The Perl Data Language is a numerical framework that in particular is suitable for implementing image-processing algorithms. Some algorithms that apply to more than 2D image processing, for example, Fourier transform, are present in PDL already. Instead of re-implementing the routines to be accessible from under PDL, a connector package PDL-Prima Image was written.

As the conversion between PDL scalars and Prima/IPA images is made easy, the platform for using the calculational powers of the both can be established with lesser efforts. Although PDL does provide very good visualization aids, and has intrinsic plotting functions, these are badly portable. In particular, PDL::FAQ complains about lack of graphics support on win32 platform, what is probably an extra reason to befriend PDL and Prima.

PDL itself contains a relatively large amount of numeric routines, and therefore its thorough description is not the goal of the paper; other sources (Glazebrook K., 2001) and books (Orwant J., 1999) cover the topic more compre-

hensively. PDL modules, useful for the image processing, are listed below:

- PDL::Graphics::LUT - look-up tables
- PDL::Image2D - set of 2-D image processing functions
- PDL::ImageND - set of N-D image processing functions
- PDL::ImageRGB - utility functions for RGB images

Applications

As a part of my research study, a set of programs has been implemented to facilitate the quantification of morphology of biological structures. The software development is performed primarily on FreeBSD, whereas the end users run the software on MS Windows. The treatment response is reflected in the morphology of the cell, in particular, in the development of neural (Ronn L., 2000). The quantification of changes in morphology is performed manually, and is both tedious and laborious, since it consists primarily of the point-and-click registration routine. The application of set of algorithms on the images, recorded using different kinds of microscopy, produces reliable results that closely correlate to the data, acquired by human observers.

Discussion

The basic image processing operators have been re-implemented many times during the last 30 years. The most widely known software packages that include these are commercial Intel's IPL (Image Processing Library), Microsoft's Visual SDK, Matlab, Imagetek's REX, and freeware ImageMagick, SciLab, ipl98, etc. Many software and hardware vendors come with their own code also. This, plus the availability of books on image processing algorithms implementation (Gonzalez R, 1993) drives new libraries onto a highly competitive ground. Though none, except Matlab, of the listed libraries are not reported being connected to Perl, it is not an unfeasible task to find a suitable library and make it available for Perl.

As IPL has its own memory allocation policies, it can be connected to Perl via PDL, which has a specifically designed interface for such a linkage. Matlab is another alternative, but its connection to Perl is other way round - instead of providing access of its internal functions to Perl, it integrates Perl into itself. Although such a linkage does not leave any place for numerical Perl packages, still it realizes a way of interchanging data from Perl and Matlab, especially given the power of the latter. Matlab also contains a data visualization package, and in same regard it is superior to Prima, Perl-Tk, Wx, and is competitive to PDL's PGLOT as a research tool. It is hardly possible though to use Matlab as a standalone library, and also to

ship due to its licensing restrictions. Among the freeware tools the most promising is Scilab, the scientific software suite, which adopts certain similarity to Matlab. The Scilab also contains an image processing toolboxSIP. The suite also provides interface to Tcl/Tk and several contributed packages that use the Tcl scripting. Prima's closest competitor, Perl-Tk, includes larger base of code, and unifies both Perl and tcl communities by preserving original Tcl-Tk API paradigm. This Perl-Tk feature is possible due to the fact that large amount of code is written in C, which also makes its porting easier for under the other languages; the same is valid for Wx also. From the other hand, porting Prima to another platform would require much less effort, because it does not include access to the platform widgets and standard dialogs. This is especially important for X11-based environments, with the most popular X11/Athena, Motif, GTK, and Qt toolkits.

Wx supports several such front-ends, but it comes at the cost of supporting the common X11 code. The functionality of Prima and Perl-Tk overlap, but single program cannot use features of both, except of Prima image subsystem, which can be used separately. Perl-Tk unique features are widget pack and grid geometry managers; these are planned to be implemented in Prima also. List of Prima unique features include pure-perl implemented set of widgets, image conversion subsystem, and a visual builder.

The Perl implementation of sophisticated widgets, like a HTML browser, is not prohibitively expensive any longer, given the speed of the modern day computers. Such an implementation is (arguably) easier to develop and support than a C or C++ one, and in particular saves the expenses of revealing eventual memory corruptions and leaks, intrinsic to the low-level language implementations.

Conclusions

The aim of this paper is to demonstrate the capabilities of Perl language for creating portable programs on a wide range of platforms by using other stand-alone graphical toolkits such as Prima and PDL on image processing.

The toolkit Prima and PDL are used to demonstrate Perl capabilities in the intersection area between image processing and graphical user interface. Prima is a platform-independent Perl graphic toolkit with an object oriented interface. Its features include an extensive set of Perl coded widgets, a wide range of image types and conversion routines, and a visual builder while PDL is a popular numerical toolkit featuring efficient storage of numerical arrays. It contains a wide spectrum of functionalities for calculation, including a set of image processing operators designed to work with two and more dimensional data. The work illustrates the usage of Prima, and PDL libraries for image conversion and display.

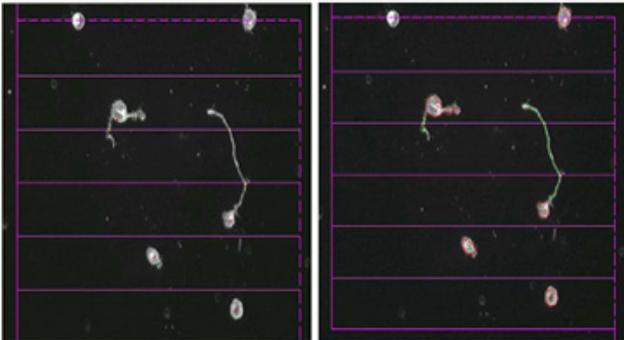


Figure 1: Manual and Automated Sampling of Lengths of Neuritis.

The picture on the left is a screenshot of the program, running under MS Windows in manual mode. Centers of cells and crossings of neuritis are marked manually by the user. The picture on the right is a screenshot of the same program, running under X11 in automatic mode. Centers of the cells and neuritis are extracted and outlined in different colors.

Example:

```
use Prima;
my $i = Prima::Image-> load('lena.gif');
die "Cannot load:$@" unless $i;
my $result = Prima::Image-> create (
width => $i-> width * 2, height => $i-> height * 2, type => $i-> type,); my $k;
# copy normal, non-converted image into the upper left corner
$result-> put_image(0, $i-> height, $i);
# copy result of simple thresholding into the upper right corner
$k = $i-> dup;
$k-> set( conversion => ict::None, type => im::bpp1);
$result-> put_image($i-> width, $i-> height, $k);
# copy 8x8 ordered image into the lower left corner
$k = $i-> dup;
$k-> set( conversion => ict::Ordered, type => im::bpp1);
$result-> put_image(0, 0, $k);
# finally convert the original image using error diffusion algorithm
$i-> set( conversion => ict::ErrorDiffusion, type => im::bpp1);
$result-> put_image ($i-> width, 0, $i);
$result-> save('lena4.gif');
```



Figure 2: Image Conversion Facilities of Prima Toolkit.

Down-sampling of a grayscale image: with constant threshold, with the threshold matrix, with error distribution.

```
# demo of all toolkits
use PDL;
use PDL::Image2D;
use PDL::IO::Pic;
use Prima;
use PDL::PrimaImage;
my $img = PDL-> rpic('onions.pgm');
# read image as piddle scalar
my $mask = PDL::PrimaImage::image( $img);
# convert into Prima/IPA image
$mask = PDL::PrimaImage::piddle( $mask);
# convert back to piddle scalar
$mask /= -255; # normalize values to 0 and 1
$mask++; # average by PDL::Image2D::patch2d
my $result = patch2d $img, $mask;
```

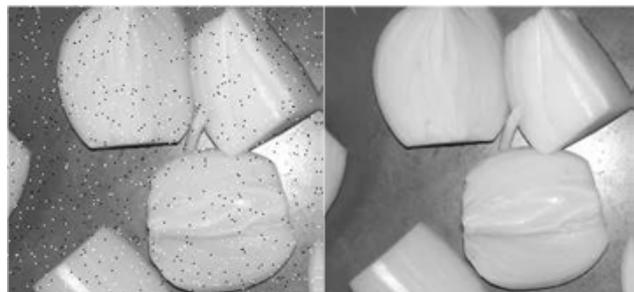


Figure 3: PDL: De-noising a Grayscale Image by the Averaging

Selected pixels Original image with "salt and pepper"
Missing pixels averaged noise

References

- Berezin A., Karasik D, Belman V, Berezin V, Bock E (1999)- PRIMA - Perl toolkit for X, win32 OS/2 PM. Proceedings on Third Perl Conference, O'Reilly 1999
- Gonzalez RC,Woods RE, (1993), Digital Image Processing, Addison-Wesley 1993
- Klette R,Zamperoni P(1996), Handbook of Image Processing Operators, Wiley 1996
- Orwant J,Hietaniemi J,Macdonald J,(1999) , Mastering Algorithms with Perl. O'Reilly 1999
- Ronn L, Raets I, Hartz BP, Bech M, Berezin A, Berezin V, Moller A, Bock E (2000), A simple procedure for quantification of neurite outgrowth based on stereological principles, J Neurosci. Methods, 100:25-32, 2000
- Glazebrook K., (2001), PDL Scientific Programming in Perl , Cambridge University Press
- Wall L,Christiansen T,Orwant J, (2000), Programming Perl, 3rd Edition, O'Reilly 2000
- Walsh N ,(1999), Learning Perl/Tk, O'Reilly 1999
- Agust, 2012, from <http://www.wxwindows.org/>