The C Raster Image Manipulation Package

Andreas Kupries ActiveState Software Inc. 409 Granville Vancouver, BC CA
andreask@ActiveState.com

ABSTRACT

This paper provides an overview of a new image manipulation and processing package for Tcl, the C Raster Image Manipulation Package, or `CRIMP` for short. We will have a quick look into its history, internal organization and implementation, plus current and possible future features and directions to work in.

1. OVERVIEW

`CRIMP`, the C Raster Image Manipulation Package, is a new package for image manipulation and processing in Tcl.

Note that no mention was made of Tk. Its need for an (interactive) X11 display makes the scripting of image processing on head-less servers quite awkward to set up. Not impossible, but definitely not trivial either.

Thus one of `CRIMP`'s goals is to be independent of Tk and avoid that bit of unpleasantness. Another goal was easy extensibility, i.e. the ability to add more functionality quickly and without hassle.

With regard to the available functionality it doesn't contain much yet due to its very young age, and most of what is present is very basic, however I hope to rectify that in the time to come.

The remainder of this paper is structured as follows: In the next chapter an anecdotal overview of the history of this package is provided, such as it is. After that, in chapter 3, a general overview of its design, implementation, and features is given. This is followed by chapter 4 talking about build issues, and lastly chapter 5 discussing possible applications and future directions for the package.

2. HISTORY

`CRIMP` started with my desire to gain more understanding of the algorithms going into image processing in such diverse areas as the post-processing of photos (red-eye reduction, contrast enhancement, pyramid blending, ...) and document analysis (page recognition, binarization, rectification and dewarping, OCR, ...).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Tcl '2010 Oakbrook Terrace/Chicago, IL, USA

I was aware of pixane[4], which was not free, and tcl.gd[5] which was focused on drawing charts and not the kind of things I was interested in. Then there is tkImg [6], which is focused on I/O, and bound to Tk [13] as well.

In the end I based it on Andrew M. Goth's "Critcl Image Processing" work found on the Tcler's Wiki [1], with heavy modification of the internals for better caching of values.

Searching around while writing this paper I found a few more, none of which fit exactly my wants either. All are shown in table 1 on the next page.

It still feels a bit like how everybody writes their own OO system for Tcl. Water under the bridge.

Despite the stated goal of Tk independence the current sources are tied to Tk, simply to allow me to see what the algorithms do while I am developing, without having to leave Tcl/Tk.

The connection is however quite thin, and easily severed. It is enough to remove the files matching the glob patterns `read-tk.crimp` and `write-*-tk.crimp,` and to possibly rebuild the binary part, if `critcl` [3] is not used in interactive mode.

In the future one of the things I have planned to do is splitting these Tk dependent parts into a separate package. Right now this will however require either a complete overhaul of the build system, or work on `critcl` [3] to enable the proper handling of package-specific stub tables. One of my side-experiments in this area is the conversion of the `genStubs.tcl` application found in the Tcl [14] sources into a set of packages `critcl` [3] could use to generate the necessary C code just from `.decl` files.

Beyond that nothing much can be said about the history of the package, given that it is only a few months old[1], and very much a work in progress.

3. DESIGN & IMPLEMENTATION

First, images are values. This design decision is inherited from the "Critcl Image Processing" code [1] `CRIMP` is based on.

While this is a disadvantage memory-wise, with the intermediate results of operations piling up even for things were in-place processing would be possible, it makes for much easier semantics too, with processing pipelines constructed easily. And going back to memory, most intermediate results are likely to be reclaimed quickly, i.e have only a very short life with a refcount > 0.

A semi-issue with the representation as values is that they have a string representation, naturally, and thus are acces-

[1]The work started June 25, 2010

Name		Author	License	Notes
pixane	[4]	Evolane	QPL	non-free binaries
tclgd	[5]	Karl Lehenbauer	BSD	chart drawing, → libgd
tkimg	[6]	Jan Nijtmans	BSD	Tk bound, focus on file I/O
megaimage	[7]	George Peter Staplin	BSD	basic ops + chart drawing, abandoned
(tcl)image	[8]	Jim Garrison	GPL	I/O, scanner & OCR bridge; No repository/files
tclmagick	[9]	Rolf Schroedter & David Welton	BSD	→ ImageMagick / GraphicsMagick
imgop	[10]	Emmanuel Frecon	BSD	→tkimg, pure Tcl, `exec` ImageMagick
tkpng	[11]	Michael Kirkham	BSD	Tk bound, PNG I/O only
LRIPhoto	[12]	David Zolli	OLL	

Table 1: Other image handling packages

sible to all Tcl commands. Doing this is not recommended however as with images shimmering is quickly expensive in both time for the (re)construction of representations and memory used. It is best to use the accessor commands provided by the package instead, to obtain such information. They know the internal structure and can pull the various pieces of data out of it without requiring shimmering.

Here the original internals were modified and refactored, with the original list representation replaced by new Tcl_ObjType's specifically geared towards storing image types and images.

For a deeper look into the internals of the representation see figure 1 below. I am not going into the image types shown there here, save for table 2 on the next page providing a short overview of what we have and may get in the future.

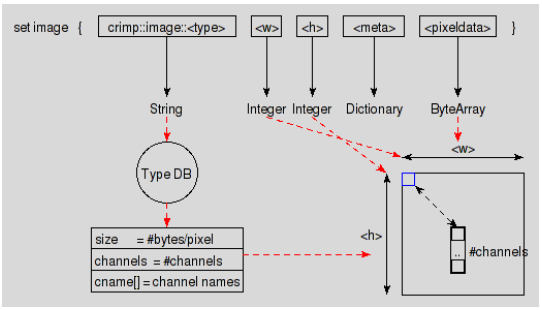

Figure 1: Image representation

Continuing on to the functionality the package is organized into two layers and five main categories. At the bottom the primitives implemented in C, and then a Tcl layer on top of that which implements the policies and heuristics for their easy use [15]. See figure 2 below for a visualization of this structure.

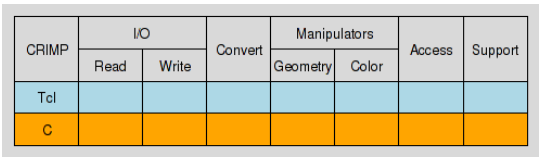

Figure 2: Package structure

Of the five categories both Support and Conversion are relatively small and bounded in size, i.e. with little to no opportunity for extension. C support is the Tcl_ObjType's for the images and image types, the Tcl layer contains commands to handle convolution kernels and function tables, see table table/support on page. Further, the conversion between image types is bounded by the set of types supported and the feasability and/or sensibility of conversion. See table 3 below for an overview.

→	rgba	rgb	hsv	grey8
rgba		X	X	X, split
rgb			X	X, split
hsv	X	X		split
grey8	join	join	join	

Table 3: Image type conversions

For accessors we have the basic ones for extracting the data found directly in an image, see again figure 1, and then higher level ones, image histograms and statistics. This area is thus quite small as well right now, but should have more potential for extension than the two discussed before.

The last two categories, I/O and image manipulation will be the big ones, also with the most potential for extension, with manipulation rating higher on both than I/O. The formats currently supported by the latter are shown in table 4 below. A few other relatively simple formats, like Windows BMP, should be implementable in Tcl as well, however for the more important formats, like PNG [17] and JPEG [18], I expect to not only require C, but also the use of external libraries. Going there might force the use of Tcl 8.6 as well, where the function `TclLoadFile()` is public [19]. A related issue still not clear at this point in time with respect to this, i.e. external libraries, is how to handle the differences between regular use as package, and use within starkits and -packs.

Format		Language	Read	Write
Tk	[13]	C	X	X
PPM	[20]	Tcl	X	X
PGM	[20]	Tcl	X	X
strimj	[16]	Tcl	X	

Table 4: Supported external image formats

A very condensed overview of the currently existing manipulation and support commands can be found in the tables 5 and 6 on the next page.

While this may look like a lot it should be noted that these are all still very basic. The only place where we are already going to higher levels are the commands for the creation of image pyramids.

Name	Colorspace	#Channels	Data/Channel	Data/Pixel	Notes
rgba	RGB	4	1 Byte	4 Byte	
rgb	RGB	3	1 Byte	3 Byte	
hsv	HSV	3	1 Byte	3 Byte	
grey8	Greyscale	1	1 Byte	1 Byte	
grey16	Greyscale	1	2 Byte	4 Byte	Rudimentary
float	Float.Point	1	4 Byte	4 Byte	Rudimentary
grey32	Greyscale	1	4 Byte	4 Byte	Future
bw	Black/White	1	1 Bit	1 Bit	Future

Table 2: Image Types

rotate	90 ° (counter)clockwise, half turn
flip	horizontal, vertical, diagonals
matrix	Arbitrary projective transforms
resize	crop, cut, expand, up- & downsample
	decimate, interpolate
alpha	blend, over, set, opaque
binary	add, subtract, difference
	multiply, screen, max, min
blank	
montage	horizontal, vertical
morph	erode, dilate, open, close
	gradient, igradient, egradient
	tophatw, tophatb
filter	convolve, rank
remap	general, invert, solarize, (de)gamma
	thresholds
pyramid	gauss, laplace, generic
wavy	effect, inherited
psychedelia	effect, inherited

Table 5: Manipulation commands

kernel	make, transpose
table	identity, invert, solarize, (de)gamma
	thresholds (above, below, in-, outside)
	gauss, linear, stretch, log, sqrt, eval
map	s.a.

Table 6: Manipulation support commands

Image processing is a large field and we should be able to add to this package for years to come. Some of the possibilities will be addressed in chapter 5.

4. BUILDING

Going forward I have to address a few issues with building this package. While the inherited use of `critcl` [3] has made this relatively easy, i.e. just run `critcl -pkg crimp.tcl`, it is not all roses.

First, it is necessary to hack the package underlying it a bit. It comes with the C files for the Tcl/Tk 8.4 stub declarations to avoid the need for an explicit stub library to link against. `CRIMP` however is based on Tcl 8.5, so we need its stub tables and C files. While a fairly easy replacement having to do it is a barrier to use. It would be, IMHO, much better if critcl could infer the correct version to use from the `package require Tcl ...` statement found in `CRIMP`'s sources.

Then comes the issue of the application not having an explicit `exit` command. Which means that when we use `critcl -pkg` to build `CRIMP`'s shared library for distribution the application will **not** exit, but enter the eventloop started by the `package require Tk`.

Beyond these bugs we have a number of smaller annoyances, namely the need to explicitly list the Tcl files used by the package (via `critcl::tsources`), instead of infering them from the executed `source` command, and, similarly, the need to explicitly use `critcl::config tk 1` to signal that this is a Tk package, instead of infering the same from the `package require Tk`.

5. FUTURE DIRECTIONS

Compared to the short history shown in chapter 2, I hope that the future of `CRIMP` is much longer and enduring. It is certainly filled with lots of work needing to be done.

Here I will mention only a few things I consider to be interesting, and otherwise refer to the ticket system reachable through `CRIMP`'s Wiki page [2], both for getting a list of existing ideas, and as the place where more can be entered, hopefully even with implementations.

1. The C level primitives currently have a great deal of redundancy in them, like replicating code for RGB and HSV, etc. stuff which is formally different, yet identical in code.

   Cleaning this up should be a medium-sized project. It is not trivial IMHO, because of the opposing tension to have the whole code of a primitive in a single function, to enable compilers to perform lots of optimizations, like loop unrolling and such. Which is something we really should preserve.

2. Most image processing tasks are what is called "embarassingly parallel". Thus an "easy" path to higher performance should be to slice an image and then dispatch the parts tiles, stripes) to many threads which can then make easy use of today's multi-core processors.

   This requires more refactorization of the C layer, moving the actual functionality into functions which are separate from the API functions. With that in place we can then add the slice'n'dice code and the thread-management.

3. Better algorithms, either faster, or conserving memory. For example, the current convolution filter primitives are implemented in the spatial domain. An FFT based implementation should be faster, especially for large kernels.

That said, FFT is a useful primitive in its own right, and should be exposed, not just used for convolution.

4. Another venue to explore is dynamic code generation, in various forms.

   Our use of `critcl` [3] already allows a bit of this, although it requires an external compiler, which is not present everythere. Even so, when we come to implementing the binary morphology operators we can already follow Leptonica's [27] route which is to generate C code for a number of important structuring elements (SEs) for full performance, and falling back to a general implementation for anything outside of that set.

   With a truly integrated compiler we would be able to do one better, generating C code for any SE, which then gets compiled to machine code. Then add caching. Another possibility, skip C and the compiler and use a general IR instead, like the input for LLVM, essentially a portable assembler.

5. Higher level operations for the processing and analysis of photos, like

   (a) Fully automatic contrast enhancement.

   (b) Fully automatic panorama stitching. This alone will require operations like SIFT[2] keypoint[24] extraction and matching, computing projective transforms, RANSAC [25], etc.

   (c) Construction of 3D information, lots of overlap with the previous point, plus bundle-adjustment, Levenberg-Marquardt [21], generally "Structure from Motion" algorithms.

   This then goes on into the wider field of photogrammetry and mapping.

   (d) Face detection and recognition.

   (e) General object detection and recognition.

   (f) Generally computer vision [22].

6. Higher level operations for the processing and analysis of documents, like

   (a) Page segmentation (text/image areas, fore/background).

   (b) Detection, analysis and removal of document warp.

   (c) OCR

   (d) Barcode detection, recognition, and extraction.

In conclusion, the Fossil [26] repository containing `CRIMP`'s sources can be reached via the package's Wiki page [2] and is open to all and sundry to provide ideas, code, bug reports and -fixes, etc.

APPENDIX

A. REFERENCES

[1] Andrew M. Goth, Critcl image processing. `http://wiki.tcl.tk/26052`

[2] Andreas Kupries, CRIMP. `http://wiki.tcl.tk/crimp`

[3] Jean-Claude Wippler, Steve Landers, CriTcl. `http://wiki.tcl.tk/critcl`

[4] Evolane, pixane. `http://www.evolane.com/software/pixane/`

[5] Karl Lehenbauer, tclgd. `http://code.google.com/p/flightaware-tcltools/`

[6] Jan Nijtmans, tkImg. `https://sourceforge.net/projects/tkimg/`

[7] George Peter Staplin, megaimage. `http://whim.linuxsys.net/files/megapkg_rev_2415.tar.bz2`

[8] Jim Garrison, tclimage. `http://sourceforge.net/projects/tclimage`

[9] Rolf Schroedter, David Welton, tclMagick. `http://tclmagick.sourceforge.net/`

[10] Emmanuel Frecon, imgop. `http://www.sics.se/emmanuel/?Code:imgop`

[11] Michael Kirkham, tkPNG. `http://www.muonics.com/FreeStuff/TkPNG/`, `https://sourceforge.net/projects/tkpng/`

[12] David Zolli, LRIPhoto. `http://wfr.tcl.tk/LRIPhoto` (French)

[13] Various, Tk. `https://tcl.sourceforge.net`

[14] Various, Tcl. `https://tcl.sourceforge.net`

[15] Jean-Claude Wippler, Poli-C. `http://wiki.tcl.tk/polic`

[16] Richard Suchenwirth. Strimj. `http://wiki.tcl.tk/strimj`

[17] PNG Homesite. `http://www.libpng.org/`

[18] JPEG Committee Homepage. `http://www.jpeg.org/`

[19] Kevin Kenny, TIP 357. `http://tip.tcl.tk/357`

[20] Jef Poskanzer, Portable any map. `http://en.wikipedia.org/wiki/Portable_anymap`

[21] Wikipedia, Levenberg-Marquardt. `http://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm`

[22] Wikipedia, Computer Vision. `http://en.wikipedia.org/wiki/Computer_vision`

[23] Wikipedia, Computer graphics. `http://en.wikipedia.org/wiki/Computer_graphics`

[24] Wikipedia, SIFT. `http://en.wikipedia.org/wiki/Scale-invariant_feature_transform`

[25] Wikipedia, RANSAC. `http://en.wikipedia.org/wiki/RANSAC`

[26] Richard Hipp, Fossil SCM. `http://www.fossil-scm.org`

[27] Dan Bloomberg. Leptonica. `http://leptonica.org/`

[2] Or other feature detection algorithm