VIPS — a highly tuned image processing software architecture

Kirk Martinez
Electronics and Computer Science
The University of Southampton
Southampton, UK
km@ecs.soton.ac.uk

John Cupitt
Scientific Department
The National Gallery
London, UK
John.cupitt@ng-london.org.uk

***Abstract*—This paper describes the VIPS image processing library and user-interface. VIPS is used in many museums and galleries in Europe, America and Australia for image capture, analysis and output. VIPS is popular because it is free, cross-platform, fast, and can manage images of unlimited size. It also has good support for color, an important feature in this sector. Its architecture will be illustrated through examples of its use in a range of museum-driven applications. VIPS is free software distributed under the LGPL license.**

Keyrords—Image-processing architecture, large images.

I. Introduction

When we began work on the high-resolution multispectral VASARI scanner [1] in 1989 we expected to have problems handling the 1 GByte images on the systems available to us back then. Rather than adapting an existing image processing package, we designed a new one that exploited the then-novel idea of memory-mapped file IO. VIPS originally processed 1 GByte images on Sun workstations with only 32 MBytes of RAM; today it is used to process multi-GByte images and is an Open Source project running on most Unix/Linux flavors, Mac OS and Windows.

VIPS continued to be developed in later projects, including MARC[2] and CRISATEL[3], and now supports a number of interesting features. It is fully demand-driven, that is, all calculation occurs as a result of a need to produce output. As a result, VIPS only calculates pels that have to be calculated, saving computation time. It has an efficient and automatic system for dividing images into subregions, so images can be processed in many small sections rather than as single large objects. This feature reduces the amount of memory required for calculation. VIPS will automatically split computations over many CPUs in SMP systems, producing an almost linear speed-up (though of course this does depend strongly on the mix of operations). Finally, VIPS is easy to extend, having a stable and well-documented plugin system. This paper outlines a few of its typical applications in order to illustrate the software's properties.

II. Software Architecture

One of the limiting factors in many image-processing systems is the fact that whole images are usually loaded into RAM from a file and that intermediates must be held in RAM too. This reduces the total amount of image data that can be processed. Line-based or tile-based systems overcome these limits by reading one or more lines at a time from a file, processing them and then streaming the results to an output file. This style makes random or non-raster image access difficult. Systems such as GEGL [4] avoid this problem by using variants of Shantniz's ideas [5] to analyze an operation graph and predict in advance which pixels will need to be calculated. This is rather complicated, however, and does not remove all image size limits.

Unix has a file memory mapping system call (equivalent functionality is also available in Microsoft Windows) called mmap, which allocates virtual memory for a file but only reads disc sectors on demand. On a 32-bit operating system, files up to about 2 GBytes can be handled easily in this manner with only small quantities of RAM required. This is the basis of VIPS file input: you open an image for reading (which uses mmap) and use a C pointer to read pels. This maintains one of the aims of VIPS, which is to allow simpler programming for beginners. It is also possible to create whole intermediate images in RAM.

A region-based image IO system is also possible, called partial IO. This allows automatic parallelization on Symmetric Multiprocessor (SMP) machines, as well as intermediate minimization. In this mode, image functions are asked to produce a small section of their output, typically a 64 by 64 tile or image width by a few lines, and ask in turn for the necessary sections from their inputs. The VIPS kernel manages the data driving so the whole image can flow through a pipeline of functions, each with their own preference for image input. Additionally, because the VIPS kernel knows which parts of the input images are being used at any time, it can avoid mapping the entire image and instead just map a roving window into the large image. VIPS uses 64-bit arithmetic to calculate the mmap window position and size, making it possible to work on images much larger than 2 GBytes on a 32-bit machine. We routinely process 10 GByte multispectral images of paintings.

On SMP systems VIPS can automatically duplicate image pipelines and run them in different threads that the system then allocates automatically to different CPUs. Six CPUs have successfully been used in this way providing speed-ups of around five times. With the advent of multi-core CPUs this feature will be used even more. The programmer does not have to do anything special to obtain parallelization, except write or

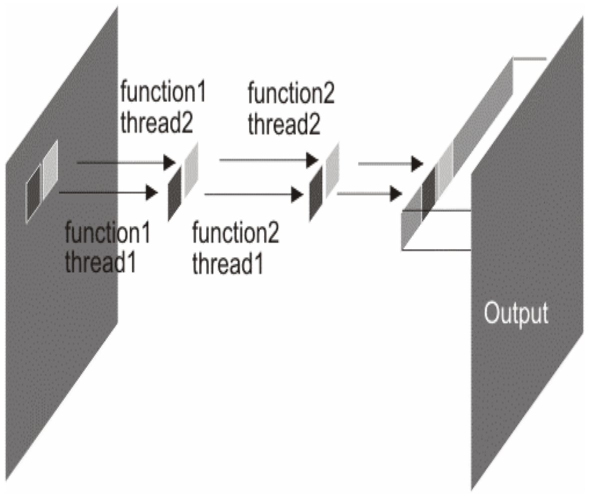


Figure 1. Parallel processing using threads

use partial functions. Figure 1 is a simplified visualization of a two-function process (eg. preprocess and filter) running on a dual-processor system. This system has been described in detail previously [6].

VIPS has around 250 library functions and has strengths in color processing (it supports all the commonly used color models), as well as multi-band and high precision processing. It lacks a data structure for extracted features (beyond a simple matrix type) created by functions such as vision or feature vector algorithms. These have been handled as separate file types in the content-based retrieval projects described later.

III. Example Code

A beginner might write simple code like this:

```
int
invert( IMAGE *in, IMAGE *out )
{
    int x, y;
    unsigned char *lbuf, *p;

    if( im_iocheck( in, out ) ||
        im_cp_desc( out, in ) ||
        im_setupout( out ) ||
        !(lbuf = malloc( IM_IMAGE_LSIZE( out ) )
        return( -1 );

    for( y = 0; y < in->Ysize; y++ ) {
        p = in->data + y * in->Xsize;
        for( x = 0; x < in->Xsize; x++ )
            lbuf[x] = 255 - p[x];
        if( im_writeline( y, out, lbuf ) )
            return( -1 );
    }

    return( 0 );
}
```

The key lines in the for loops set up the pointer p to the start of each line and inverts each pel value into the line buffer lbuf. A partial version has a processing function that looks like this:

```
int
invert_generate( REGION *or, REGION *ir )
{
    const int left = or->valid.left;
    const int top = or->valid.top;
    int x, y;
    unsigned char *p, *q;

    if( im_prepare( ir, &or->valid ) )
            return( -1 );

    for( y = 0; y < or->valid.height; y++ ) {
        p = IM_REGION_ADDR( ir, left, y + top );
        q = IM_REGION_ADDR( or, left, y + top );

        for( x = 0; x < or->valid.width; x++ )
           q[x] = 255 - p[x];
    }

   return( 0 );
}
```

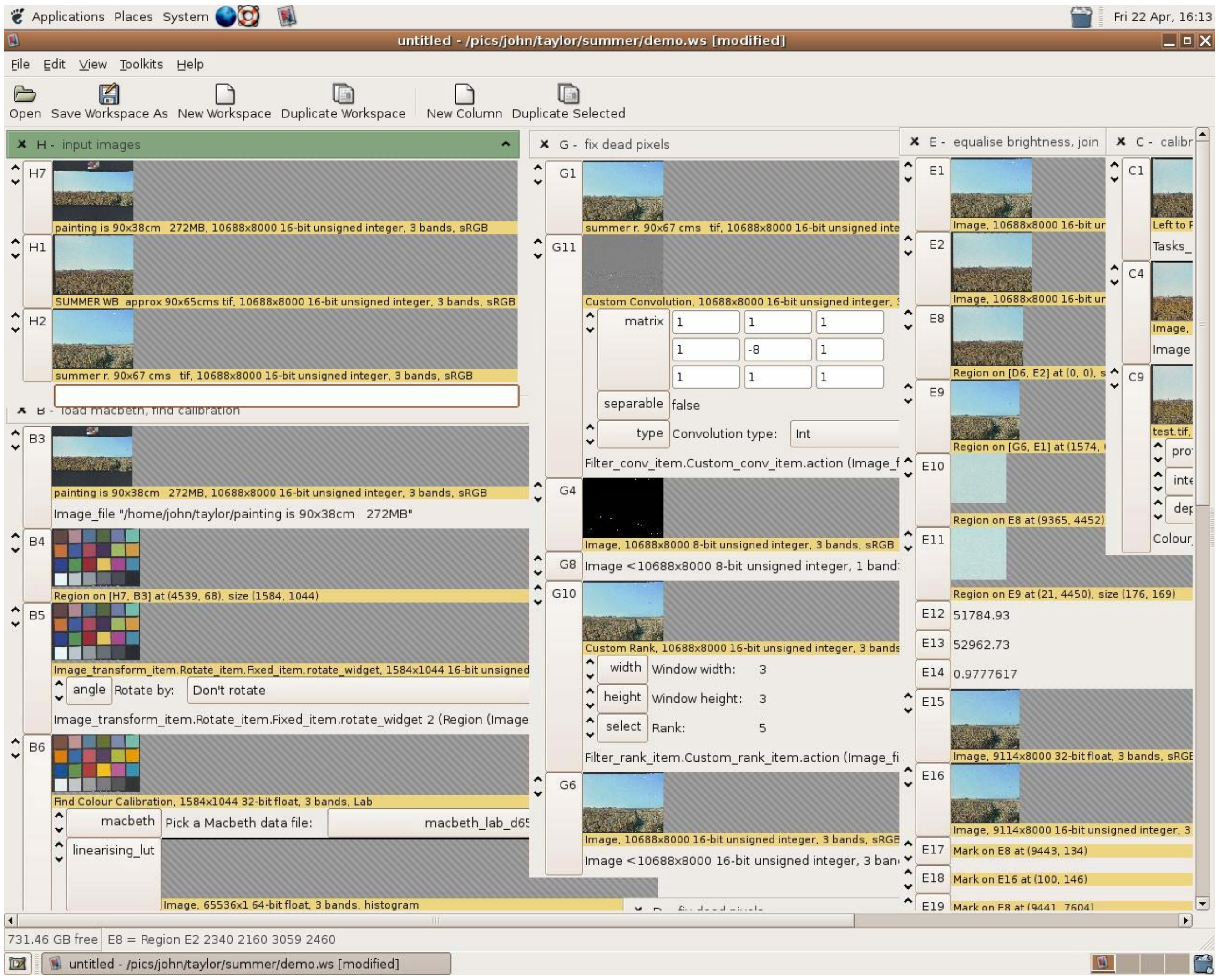
Now invert_generate is required to fill the output region or with pels calculated from the input region ir. The macro IM_REGION_ADDR is used to get valid pointers to both input and output pels. Together with a standard wrapper function this transforms the code into partial mode. It can be seen here that the general philosophy is to use pointers to access pels. Thus code can be prototyped simply before converting to partial mode for library inclusion. Facilities are available to handle pels in any C format: char, short, int up to as large as double complex (used for complex images).

IV. NIP – The User Interface

We have built an application on top of the VIPS library that exploits its features to produce a fully demand-driven image-processing environment. It allows experimentation with small regions of images, whole image spreadsheet-style processing and interaction. Pels are only calculated when absolutely required, either to update part of the screen or so that another calculation can proceed. As a result, operations appear to occur almost immediately, even on extremely large images. The full calculation is only done when the user selects “save” on the final image in a workspace.

As well as a lazy image-processing library, nip also has a lazily evaluated functional language for scripting, resembling Miranda [7,8] and Haskell [9]. All of the menu items are implemented in this extension language. The interpreter implements a number of useful optimizations, including common-subexpression removal, replacement of arithmetic operations by lookup tables where possible, and operation memoisation. It remembers the last few hundred VIPS operations and if it sees a repeat, it reuses the previous result rather than starting a new computation.

In use, nip feels rather like a spreadsheet. The workspace is split into rows and columns and each row contains a value (image, matrix, number, string and so on) and the formula that made that value. If you make a change anywhere, nip is able to recalculate only rows affected by that change. Again, because


Applications Places System
Fri 22 Apr, 16:13
untitled - /pics/john/taylor/summer/demo.ws [modified]
File Edit View Toolkits Help
Open Save Workspace As New Workspace Duplicate Workspace New Column Duplicate Selected

H - input images

H7 painting is 90x38cm 272MB, 10688x8000 16-bit unsigned integer, 3 bands, sRGB

H1 SUMMER WB approx 90x65cms tif, 10688x8000 16-bit unsigned integer, 3 bands, sRGB

H2 summer r. 90x67 cms tif, 10688x8000 16-bit unsigned integer, 3 bands, sRGB

B - load macbeth, find calibration

B3 painting is 90x38cm 272MB, 10688x8000 16-bit unsigned integer, 3 bands, sRGB
Image_file "/home/john/taylor/painting is 90x38cm 272MB"

B4 Region on [H7, B3] at (4539, 68), size (1584, 1044)

B5 Image_transform_item.Rotate_item.Fixed_item.rotate_widget, 1584x1044 16-bit unsigned
angle Rotate by: Don't rotate
Image_transform_item.Rotate_item.Fixed_item.rotate_widget 2 (Region (Image

B6 Find Colour Calibration, 1584x1044 32-bit float, 3 bands, Lab
macbeth Pick a Macbeth data file: macbeth_lab_d65
linearising_lut
Image, 65536x1 64-bit float, 3 bands, histogram

G - fix dead pixels

G1 summer r. 90x67 cms tif, 10688x8000 16-bit unsigned inte

G11 Custom Convolution, 10688x8000 16-bit unsigned integer, 3

matrix			
	1	1	1
	1	-8	1
	1	1	1

separable false
type Convolution type: Int
Filter_conv_item.Custom_conv_item.action (Image_f

G4 Image, 10688x8000 8-bit unsigned integer, 3 bands, sRGB

G8 Image <10688x8000 8-bit unsigned integer, 1 band:

G10 Custom Rank, 10688x8000 16-bit unsigned integer, 3 bands
width Window width: 3
height Window height: 3
select Rank: 5
Filter_rank_item.Custom_rank_item.action (Image_fi

G6 Image, 10688x8000 16-bit unsigned integer, 3 bands, sRGB
Image <10688x8000 16-bit unsigned integer, 3 ban

E - equalise brightness, join

E1 Image, 10688x8000 16-bit ur

E2 Image, 10688x8000 16-bit ur

E8 Region on [D6, E2] at (0, 0), s

E9 Region on [G6, E1] at (1574,

E10 Region on E8 at (9365, 4452)

E11 Region on E9 at (21, 4450), size (176, 169)

E12 51784.93

E13 52962.73

E14 0.9777617

E15 Image, 9114x8000 32-bit float, 3 bands, sRGB

E16 Image, 9114x8000 16-bit unsigned integer, 3

E17 Mark on E8 at (9443, 134)

E18 Mark on E16 at (100, 146)

E19 Mark on E8 at (9441, 7604)

C - calibr

C1 Left to F
Tasks_

C4 Image,
Image

C9 test.tif,
pro
inte
dep
Colour

731.46 GB free E8 = Region E2 2340 2160 3059 2460

untitled - /pics/john/taylor/summer/demo.ws [modified]

Figure 2. Example of a complex nip workspace

calculation is demand-driven, changes are rapid, even for large workspaces manipulating multi-GByte images.

Figure 2 shows nip being used to repair, color-correct and assemble an 800 MByte image of “West Bergholt Summer” by Stephen Taylor. The original painting is 1,910 by 810 mm, the camera used takes 10,000 by 10,000 pixel images, so to be able to print a 1:1 reproduction at 300 dpi the painting had to be shot in two halves and then assembled. Specular highlights on the surface of the painting caused burnout in the image that had to be fixed (a simple convolution with a threshold was used to detect the burnout, then the dead pixels were replaced by the local median). And finally, the camera color was not very good, so the Macbeth Color Checker Chart included in one of the images was used to correct the color. The workspace is reasonably responsive during use: for example, if you change one of the convolution kernels, nip updates the screen in about 1.5s. The final save as a 16-bit Adobe98 RGB TIFF takes about 4 minutes on a dual 2.5 GHz Xeon machine.

On startup, nip uses about 30 MBytes of RAM as reported by the RES column in “top” output. Most of this is the extension language interpreter and the support libraries. After loading the workspace shown in Figure 2, it was using 45 MBytes of RAM. During save as TIFF, RAM use rose to 120 MBytes. The largest intermediate image produced during computation was 1.7 GBytes.

Nip also has a non-graphical mode where it can be used as a script interpreter. The National Gallery shop has a posters-on-demand kiosk where visitors can buy a color-facsimile A4, A3 or A2 print of any painting in the collection. The 8,000 PDF files are generated by a nip script in about 3 days of processing, though a lot of this time is spent fetching the high-resolution originals from a central file server in the IS department.

V. Other Applications

Initially one of the most-used features of VIPS in museums was for mosaicing: the process of putting together many small images from infrared or X-ray imaging. The VIPS algorithm was specifically devised for mosaicing images with a small overlap and little geometric distortion. It uses correlation of interesting points in the overlaps and is fairly robust. It has been used to stitch and balance large numbers of infra-red images because the cameras used typically only produce roughly 700x600 resolution images. In the case of X-ray mosaics each image is a flatbed scan of a chest film and the final mosaic can easily become larger than 1GByte. With nip it is possible to manually locate starter points for the mosaicing, preview the results then only process on saving the result.

In the Artiste[10] and SCULPTEUR[11] projects content-based image retrieval was based around VIPS functions. These produced small feature vector files including histograms, CCVs, PWTs etc., which were later, compared to give match scores. The functions were wrapped as MySQL modules and ran under the control of the database, frequently having to run on 50,000 images. One key issue was reliability as any crashes would affect the whole database and this was possibly with careful debugging.

VI. Conclusions

The initial design of VIPS centered around handling images which were much larger than available RAM. At the time it might have been predicted that RAM would increase in size eventually and negate this part of the design. However image sizes now have reached many Gbytes and RAM sizes are only just moving past the 2 GByte barriers of 32-bit computing. So VIPS has been able to provide a scalable solution over a considerable time with the benefit that debugged and tuned code can be relied on. It is becoming widely used due to its speed compared with common image manipulation packages, especially when used on multiprocessor systems. In terms of aiding research VIPS has allowed perfected image processing modules to be reused for over ten years because of the continuity of its API. It is often possible to prototype complex processes in nip because of the large number of primitive functions before putting together an integrated function that is used in production. The software is available from http://www.vips.soton.ac.uk.

References

[1] K. Martinez, J. Cupitt, D. Saunders, R. Pilay, “10 years of Art Imaging Research”, Proceedings of the IEEE . Vol. 90, No. 1, pp. 28-41, Jan 2002.

[2] J. Cupitt, K. Martinez, and D. Saunders, “A Methodology for Art Reproduction in Colour: the MARC project”, Computers and the History of Art, Vol. 6 No. 2, pp. 1 20 1996.

[3] A. Ribes, H. Brettel, F. Schmitt, H. Liang, J. Cupitt, and D. Saunders, “Color and Multispectral Imaging with the CRISATEL Multispectral System”, Proc. PICS, IS&T, pp. 215, 2003.

[4] http://www.gegl.org/

[5] M. A. Shantzis, “A model for efficient and flexible image computing”, SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques, pp. 147-154, 1994.

[6] J. Cupitt and K. Martinez, “VIPS: an image processing system for large images”, Proc. SPIE Vol. 2663, pp. 19-28, 1996.

[7] S. Thomson, “Miranda: The Craft of Functional Programming”, Addison-Wesley, 1995.

[8] S. Thompson, “Laws in Miranda.” ACM Communications, Vol. 2, No. 3, 1986.

[9] J. Hughes, “Why Functional Programming Matters”, The Computer Journal, Vol. 32, No. 2, 1989, pp. 98 107, 1989.

[10] Lewis, P. H., Martinez, K., Abas, F. S., Ahmad Fauzi, M. F., Addis, M., Lahanier, C., Stevenson, J., Chan, S. C. Y., Mike J., B. and Paul, G. “An Integrated Content and Metadata based Retrieval System for Art”. IEEE Transactions on Image Processing, Vol. 13, No. 3, pp. 302-313. 2004.

[11] Addis, M., Boniface, M., Goodall, S., Grimwood, P., Kim, S., Lewis, P., Martinez, K. and Stevenson, A., “SCULPTEUR: Towards a New Paradigm for Multimedia Museum Information Handling”, in Proceedings of Semantic Web ISWC 2870, pp 582 -596, 2003.