VIPS - A HIGHLY TUNED IMAGE PROCESSING SOFTWARE ARCHITECTURE

Kirk Martinez and John Cupitt
The University of Southampton and The National Gallery

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 colour, 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.

1. INTRODUCTION

When we began work on the high resolution multispectral VASARI scanner [1] in 1989 we expected problems in handling the 1 GByte images on the systems of the time. Rather than adapt an existing image processing package we designed one which exploited the features of Unix. Although initially design so that huge images could be processed on SUNs which only had 32Mbytes of RAM today it is used to process multi-GByte images and is an Open Source project running on most Unix/Linux flavours, MacOS and Windows.

VIPS has 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.

2. SOFTWARE ARCHITECTURE

One of the limiting factors in many 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 which can be processed. Line-based or tile-based systems overcome memory 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 analyse an operation graph and predict in advance which pixels 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) mmap, which allocates virtual memory for a file but only reads disc sectors on demand. On a 32 bit operating system files up to 2GB can be handled easily with far smaller quantities of RAM. This is the basis of VIPS file input and the simplest way of accessing an image is to open it for reading (which uses mmap) and using 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 to ease programming.

A sophisticated tiled-access technique is also possible, called partial I/O. This allows automatic parallelization on Symmetric Multiprocessors (SMP) as well as intermediate minimization. In this mode image functions demand an area of input image to work on, typically a 128x128 tile or image width by a few lines. The VIPS kernel then manages the data driving so the whole image can flow through a pipeline of functions each demanding 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 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 larger than 2GB 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, which the system 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 use *partial* functions. Figure 1 is a simplified visualization of a two-function process (eg preprocess and filter) running on a dual-processor system.
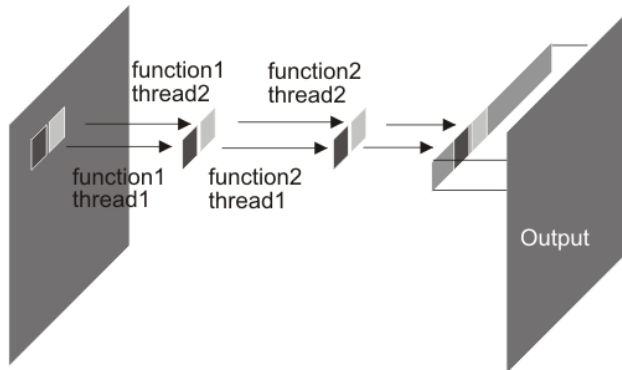

Figure 1 parallel processing using threads

VIPS has around 250 library functions and has strengths in colour processing as all the common models are supported, as well as multi-band and high precision processing. It lacks a data structure for extracted features which are created by functions such as vision or feature vector algorithms. However these have been handled as separate file types in the content-based retrieval projects described later.

3. EXAMPLE CODE

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

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

    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];
        im_writeline( y, out, lbuf );
        }
    return(0);
}
```

The above example code might be written by a beginner. This example has most error checking and casts removed for brevity. 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 which looks like:

```
int invert_generate( REGION *or, REGION *ir )
{
    int x, y;

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

    for( y = 0; y < or->valid->height; y++ ) {
        unsigned char *p = VIPS_REGION_ADDR( ir,
or->valid->left, y + or->valid->top );
        unsigned char *q = VIPS_REGION_ADDR( or,
or->valid->left, y + or->valid->top );

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

The key feature being that REGIONs are passed in and the macro VIPS_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).

3. NIP – THE HUMAN INTERFACE

We have built an application on top of the VIPS library which 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 [6,7] and Haskell[8]. All of the menu items are implemented in this extension language. The interpreter implements a number of useful optimisations, including common-subexpression removal, replacement of arithmetic operations by lookup tables where possible, and operation memorisation. It remembers the last few hundred VIPS operations and if it sees a repeat, it reuses
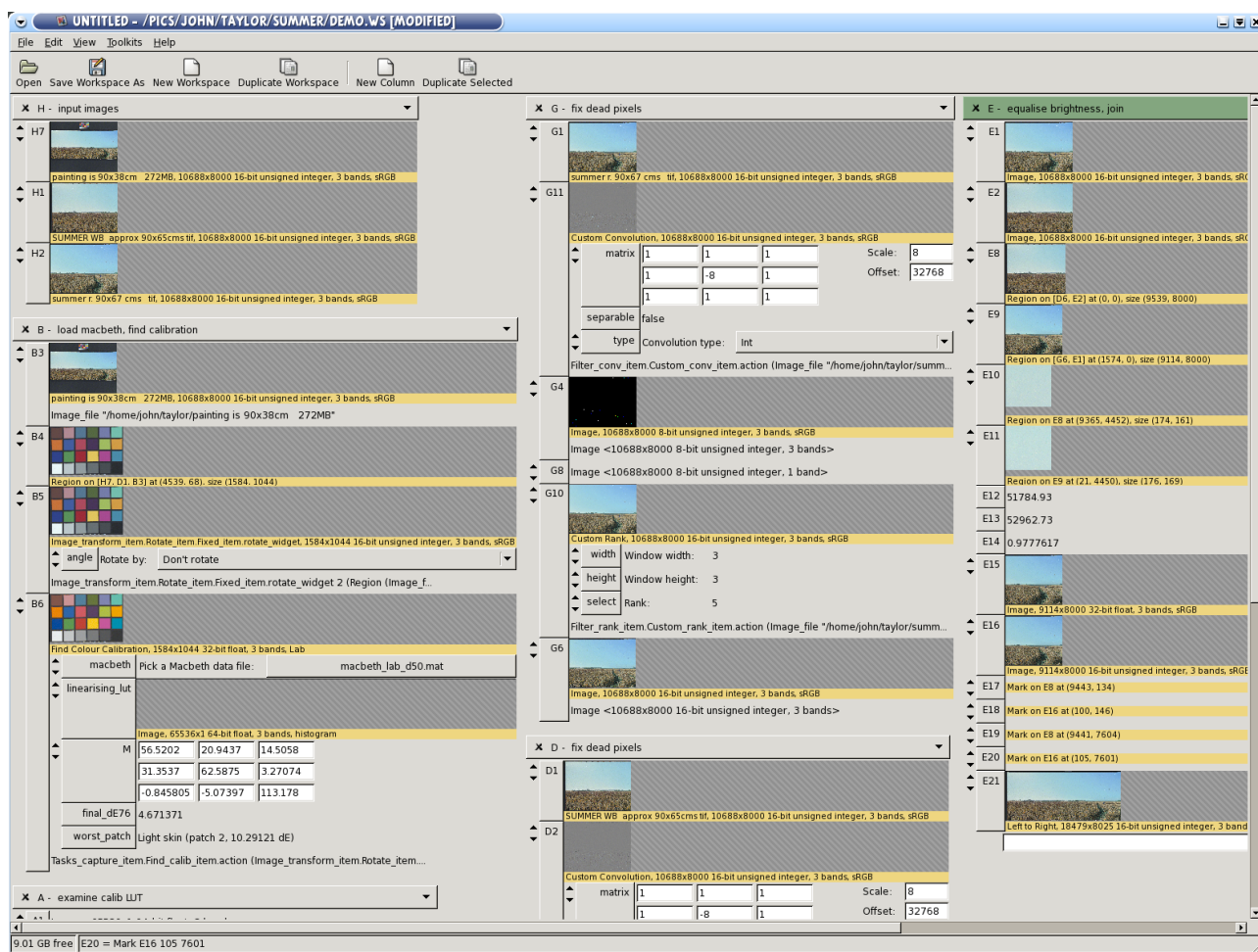
Figure 2. Example of a complex nip worksheet

the previous result rather than starting a new computation. In use, nip works rather like a spreadsheet

The workspace is split into rows and columns, and each row can contain a value (image, matrix, number, string and so on) and the formula that made that value. If you make a change anywhere, then it is able to recalculate only rows affected by that change. Again, because calculation is demand-driven, changes are rapid, even for large workspaces manipulating many-gigabyte images.

Figure 2 shows nip being used to repair, colour-correct and assemble an 800 MB image of "West Bergholt Summer" by Stephen Taylor. The original painting is 1910mm by 810mm, the camera used takes 10,000 by 10,000 pixel images, so to be able to print a 1:1 reproduction at 300dpi 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

which 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 mode). And finally, the camera colour was not very good, so the Macbeth Color Checker Chart included in one of the images needed to be used to correct the colour. The workspace is reasonably responsive during use: for example, if you change one of the convolution kernels, nip updates in about 1.5s. The final save as a 16-bit Adobe98 RGB TIFF takes about 5 minutes on a dual 2.5GHz Xeon machine.

On startup, nip uses about 30 Mbytes of RAM (as reported by the RES column in "top"). 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 colour-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.

4.OTHER APPLICATIONS

Initially one of the most used features of VIPS in museums was mosaicing: for putting together many images from infra-red 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 [9] and SCULPTEUR [10] projects content-based image retrieval was based around VIPS functions. These systems produced small *feature vector files* including histograms, CCVs, PWTs etc which were later compared to give match scores. The VIPS functions were wrapped as mySQL database 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 possible with careful debugging.

4.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 Gigabytes and RAM sizes are only just moving past the 2GByte barriers of 32 bit computing. So VIPS has been able to provide a scalable solution over a considerable time with the benefit that debugged 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 which is used in production. The software is freely available from www.vips.soton.ac.uk.

11. 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] S. Thomson, Miranda: The Craft of Functional Programming, Addison-Wesley, 1995.
[7] S. Thompson, "Laws in Miranda." ACM Communications, Vol. 2, No. 3, 1986.
[8] J. Hughes , "Why Functional Programming Matters" The Computer Journal, Vol. 32, No. 2, 1989, pp. 98–107, 1989.
[9] 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.
[10] 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.