Proc. 2001 Australian Conference on Robotics and Automation
Sydney, 14-15 November 2001

Image Processing Libraries

Bing-Chang Lai & Phillip John McKerrow
School of Information Technology &
Computer Science
University of Wollongong, 2522
{bl12, phillip}@uow.edu.au

Abstract

There are a wide variety of image processing library implementations. Three implementations are discussed in this paper, in the hope of showing the diverse nature of image processing libraries. Datacube provides a hardware and vendor-specific image processing library, known as ImageFlow, developed to support their pipeling image processing hardware card. Vector, Signal and Image Processing Library (VSIPL) is a hardware-neutral approach with a focus on portability. Finally Vision with Generic Algorithms (VIGRA) is built on the principles of generic programming and is therefore flexible without incurring large speed penalties.

1 Introduction

There are many different image processing libraries available. These libraries provide image processing functions in vastly different ways. Some are totally hardware-specific, while others aim to provide a hardware-neutral solution. Some of these libraries have been around for years, while others have just been created. These libraries are vastly different due to their different aims.

This paper describes three of these libraries - Datacube (ImageFlow), Vector, Signal, and Image Proessing Library (VSIPL) and Vision with Generic Algorithms (VIGRA). These three were chosen because of each of them approaches image processing differently. Datacube is interesting because of it's hardware specific roots which allows it to perform at high speeds. VSIPL is discussed because it is a standard that is supported by many institutions. Finally VIGRA was picked because its approach to image processing using generic programming, allows it to be extremely flexible without incurring enormous performance penalties.

2 Datacube

Datacube [Datacube, 2001] is a commercial company that provides high-speed image processing solutions through the use of hardware boards. Their flagship image product is the MaxVideo series plug-in card which provides image processing capabilities in hardware, the newest model being the MaxVideo 2000, which comes with an embedded CPU (eCPU), a very large field-programmable gate array (FPGA) and high-speed memory. The eCPU provides real-time hardware control and communications, and is also available for use in image processing operations like blob labeling, correlation and alignment. The FPGA provides hardware acceleration for neighbourhood operations like morphology, convolution and correlation. The MaxVideo 2000 is capable of achieving very high image processing speeds.

Datacube's image processing products' high speed comes from the use of pipeline processing, as opposed to the sequential processing model of the Von Neumann computer. Pipeline processing allows more parallelism between processes. Coupled with the use of hardware to perform image processing routines, Datacube's products can be used for real-time applications without problems.

This pipeline processing model is programmed using the ImageFlow [Datacube, 1994] software. This is a library of C-callable functions that configures and manages data transfers between the different processing elements in the pipeline processing model. This library has been around for eight years.

This next section describes how image processing routines are performed using the DataCube.

2.1 Programming Datacube's products

Since Datacube uses pipeline processing to perform image processing functions, programming Datacube's products is vastly different from programming the ordinary computer. With Datacube, the programmer defines a flow of data from source to destination, passing the data through different devices which consists

of processing elements which transform the data. The programmer selects different devices, configures them, and connects them programmatically. Deciding how to connect the devices is usually done diagrammatically using Element Flow Diagrams. From a diagram, the programmer uses the appropriate ImageFlow C-functions to configure the card to process the data in the fashion specified by the diagram.

Programming ImageFlow basically becomes

1. Set up and initialise the IP System

   ImageFlow needs a system configuration file in order to work. This system configuration file contains hardware-specific information about the Datacube IP devices available, such as the base addresses and IRQ ports.

2. Construct a Data pipeline

   Designing a data pipeline requires the use of Element Flow Diagrams. This diagram shows details of the device with highlighted path(s). This highlighted path is the data pipeline that the program will be constructing.

3. Writing the program

   After obtaining the diagram, a C program can be written to read the system configuration file, and to connect the devices together in the manner specified by the Element Flow Diagram. Once this is done, the pipeline is ready and the program can then send data through the pipe.

Devices themselves are decomposable further to processing elements. These IP elements fall into the following categories:

1. Data Surface Store elements
   These elements store 2-dimensional arrays of data called data surfaces. These data surface stores often provide a mechanism for the host computer to directly access the data in the data surface.
2. Data Surface Generator elements
   These elements produce a data stream of pixel values typically used for an IP operation or calculation: a constant element generates a stream of pixels all set to the same constant value; a column generator generates pixel values that are column numbers in a surface and so on.
3. Data Gateway elements
   These elements control the flow of data at both the beginning (head) and end (tail) of a data pipeline.
4. Data Port elements
   Data Port elements carry digital data from one image processing (IP) device to another.
5. Intra-Pipeline elements
   These elements are digital processing elements that are used to contruct a data processing pipeline.
6. Multiple Operation State Controller (MOSC) elements
   MOSC elements is sued to control the operation state of other IP elements subordinate to it.
7. Processor elements
   These elements perform operations on surfaces. The nature of the operation is element-specific and usually non-programmable - the operation depends on the associated surface. eg. Attaching a histogramming surface to a processor, the processor does a histogram.
8. Analog elements
   These manipulate or carry ananlog signals.
9. Sensor elements
   Sensors determine the value of a signal.
10. Pixel Clock elements
    These provide pixel timing operations.
11. Timing Port elements
    These provide timing signals.
12. Address Generator elements
    These provide an (x,y) coordinate system that defines a warping operation through the gateway. They provide address values to an IMAC element.
13. Video Disk elements
    These are used to store large amounts of non-volatile video data.

Datacube is difficult to program. This has lead to the development of libraries like VEIL [dat, 1996] which sits between Datacube and the User. Other developments in libraries which seek to simplify Datacube include MERLIN [Olson *et al.*, 1996] which is actually implemented on top of VEIL.

2.2 Applications of Datacube

Since Datacube's products forte is high-speed, any application that requires high throughput should consider using Datacube. Examples of such applications include real-time vision in robots as used in [dat, 2001]. However the programming model locks the user to only Datacube's products, since ImageFlow is a vendor-specific language. This vendor-specific programming environment means that if the image processing board being used is likely to change manufacturers during the lifetime of the project, then using ImageFlow is probably not the best.

3 VSIPL

The aim of the Vector, Signal and Image Processing Library (VSIPL) [vsi, 2001; Geo, 2001] is to provide an API for vector, signal and image processing that is not dependent on any particular hardware platform. The project was started by the Defense Advanced Research Projects Agency (DARPA) for the development of the Tactical Advanced Signal Processor Common Environment (TASP COE), which required standards for some

of its key Applications Programming Interfaces. The problem was that the military wished to use commercial products in developing military weapons systems. However commerical product configurations were not stable over the life cycle of a typical military weapons system. So the idea is to create a standard API which commercial companies follow, and thereby reduce the amount of re-coding needed by the military when a new commercial product is available.

VSIPL is currently controlled by a forum of commercial companies and unversities, including Intel, Silicon Graphics, MIT Lincoln Laboratory and Georgia Tech/GTRI among others. These member organisations meet a few times a year to discuss directions for the VSIPL API. The VSIPL API is currently in version 1.01, and there is a proposal pushing VSIPL towards a real object-oriented design.

Specific VSIPL libraries are referred to as "Core" or "Core Lite" profiles. The "Core" profile includes most of the signal processing and matrix algebra functions, while "Core Lite" includes a smaller subset, suitable for vector-based signal processing applications. Apart from these profiles, VSIPL also defines two versions of libraries referred to as development and performance. Development libraries run slower, but contain extra code for error reporting. VSIPL compliant library suppliers may provide either one, or both versions.

This next section discusses VSIPL.

3.1 Programming VSIPL

VSIPL consists of a large number of C functions, with different versions for different types and different precision. It is designed with an "object-oriented" view, but the API itself does not use an object-oriented language like C++, Small-Talk or Java. Instead it uses structs to represent objects. Implementations of VSIPL are allowed to add their own data to these objects, though they should not be exposed[1].

Functions are provided for initialising and closing the library, for block allocation and deallocation, basic scalar operations, basic vector operations, random number generation, signal processing and linear algerbra.

Data Management

VSIPL functions work with *blocks* and *views*. Blocks consists of *data arrays*, which is the actual memory used to store the data, and a *block object*, which stores information that allows VSIPL to access the data array. Views of data can be created, and these views consists of a block and a *view object*, which stores information that allows VSIPL to access the data of interest stored in the block.

Blocks and views are VSIPL data structures and are opaque to the user. The user is not allowed to directly access any information inside blocks and views. In addition, creation and deletion of blocks and views are handled by VSIPL. Data arrays on the other hand, exist in one of two logical data spaces - the user data space, and the VSIPL data space. Basically, only the user is allowed to changed the data array when it is in the user data space, and vice versa. Moving data arrays between these two logical data spaces may occur penalties; these penalties, if any, are up to the implementation. For example, for most vector processors, the data must be aligned to certain boundaries; this would be an ideal place to ensure the user data is aligned for the vector processor.

[1]This is done in C using incomplete type definitions

Functions

VSIPL provides functions in the following categories:

1. Support Functions
2. Scalar Functions
3. Random Number Generators
4. Vector and Elementwise Operations
5. Signal Processing Functions
6. Linear Algebra Functions

Since VSIPL tries to provide a hardware neutral API, functions are defined for virtually every scalar mathematics operation, excluding basic operations like addition and subtraction. Vector operations are even more complete than scalar functions, with operations from basic operation like addition and subtraction to cosine and tangent to selection operations.

Image processing functions available make no indication of whether they use vector or scalar implementations. Therefore, it is up to the library supplier to process them as is seen fit.

Like Datacube, VSIPL is not the easiest API to use in the world. A fully object-oriented version of VSIPL [Campbell, 2000] has been suggested. However it is incomplete, and whether it will be endorsed or not by the VSIPL forum is another matter.

Implementing a typical image processing routine in VSIPL typically involves initialising the VSIPL library via "vsip_init". The image is created by binding blocks to arrays in memory representing the image. Views can then created from these blocks. The blocks are then admitted to VSIPL (ownership passed to VSIPL) after which the views associated with the blocks can be passed through a series of VSIPL functions to produce the desired output. After the VSIPL processing has finished, the blocks are released (ownership returns back to the user), after which the user can interrogate the result and manipulate the image at will. On program termination, blocks and views created are to be freed and VSIPL should be deinitialised via "vsip_finalize".

3.2 Applications of VSIPL

VSIPL is used extensively currently by the US military. It is used because it provides a consistent API across

products. Coupled with hardware, VSIPL can undertake real-time image processing. Examples of applications of VSIPL [Köethe, 2001] include U.S. Navy Tactical Advanced Signal Processor Common Operating Environment (TASP COE), Lockheed-Martin Government Electronics Systems' Naval Electronics & Surveillance Systems-Surface Systems, and MITRE's real-time Space Time Adaptive Processing (STAP).

4 VIGRA

Vision with Generic Algorithms (VIGRA) [Köethe, 2001; 2000b; 1998; 2000a; 1999] is a computer vision library created by Ullrich Kthe, as part of his PhD thesis "Generische Programmierung fr die Bildverarbeitung", primarily focused on flexible algorithms and generic programming. Built using template techniques similar to those used in the Standard Template Library (STL), VIGRA allows the user to easily adapt VIGRA components to their needs. This flexibility comes almost for free, since the design uses compile-time polymorphism (templates) - no virtual functions.

VIGRA provides support for reading/writing several popular image formats (via use of other libraries) liks BMP and GIF. In addition, it provides a range of image processing routines, filters, segmentation and image analysis routines. Examples of filters supported include convolution and non-linear diffusion. Segmentation routines include edge detectors and corner detectors. Detection of minima/maxima and region statistics make up part of the image analysis routines.

4.1 Programming VIGRA

VIGRA brings the world of generic programming to image processing. In general VIGRA is a much higher level image processing library than either Datacube or VSIPL. As dictated by generic programming, data is separated from algorithms that work on it. Access to data is done via iterators and accessors only.

Iterators allow access to the data. A variety of iterators are available, some of which provide read-only access. In general every iterator in VIGRA consists of two separate directions - X and Y.

While STL itself does not use accessor, these accessors allow VIGRA to support certain data structures, like multi-band RGB images, easily. Accessors provide another level of indirection. Data is set and read using accessor objects, which can change data representations between the iterator and user as needed.

Several general algorithms are provided by VIGRA, such as "transformImage" and "inspectImage". These algorithms require Functors to be of any use. The algorithms accept a functor, which they use to read, or change the data provided through iterators and accessors.

These algorithms however require a large number of arguments. To reduce the number of argument required to make each call, VIGRA uses tuples such as "srcImageRange" and "destImage". These tuples usually take an image as a parameter and provides default iterators and accessors to the algorithm functions.

Another difference from STL is in the use of allocators. STL allows allocators to be changed at will. VIGRA however does not allow this, though its code uses specialized allocate functions instead of the normal new and delete.

Image processing in VIGRA typically involves the use of an algorithm and a functor. The algorithm applies the functor to the image (access through iterators and accessors). For example, to invert an image, the "transformImage" algorithm would be used. A "linearIntensityTransform" functor would be applied to each pixel in the source image using the "transformImage" algorithm. The pixels to be transformed using the "linearIntensityTransform" is accessed through two iterators to the source image, pointing at the beginning and the end of the region, while the destination is marked by a single iterator to the destination image. Finally accessors for both the source and destination image. This however produces a large number of arguments, and VIGRA provides functions to shorten the argument list by providing the most common configurations - for example each iterator has a default accessor.

4.2 Applications of VIGRA

VIGRA is free. It is flexible. However VIGRA is only available in C++ because it uses templates. It is possible to integrate different languages together if required, but the code that uses VIGRA directly must be written in C++.

The lack of hardware image processing solutions which support VIGRA means that it cannot be used for real-time applications. Despite this, VIGRA's ease of use, extensive feature set and flexibility makes this a good choice for use in software image processing solutions.

5 Summary

Many different image processing libraries exist. One of the most popular commercial libraries is Datacube's offering. While it provides extremely high-speed image processing because of the use of hardware image processing cards, the manner in which Datacube is programmed is vastly different from traditional programming, and can be regarded as difficult.

An emerging standard for image processing is VSIPL. This on the other hard takes a totally software approach, choosing not to have any hardware-specific dependencies. This allows VSIPL to be used on and ported to a wide variety of systems. There is some hardware support for VSIPL so real-time image processing should be possible with VSIPL. However while VSIPL is easier to use than Datacube, it is not as easy as VIGRA. Furthermore, VSIPL defines a huge range of functions, all with different names, which can make it tiresome for library vendors to implement.

VIGRA is the result of a recent Ph.D. thesis by Ulrich Kethe. While it is lacking in support, it is the easiest to use and to extend. It is easy to add more functionality to VIGRA, and to use VIGRA with user data structures. And best of all this flexibility does not come at a severe cost to performance.

Acknowledgement

This work was funded by an Apple University Development Fund Grant.

References

[Campbell, 2000] Daniel P. Campbell. Prototype extension of vsipl into c++ and object-oriented design. *Georgia Tech Research Institute*, September 2000.

[dat, 1996] Virginia's extensible imageing library (veil), 1996. see http://www.cs.virginia.edu/~vision/projects/veil.

[dat, 2001] Minnesota robotic visual tracker, 2001. see http://www.cs.umn.edu/Research/airvl/facilities/mrvt.html.

[Datacube, 1994] Datacube. *Datacube Image Processing Manual - ImageFlow System Manual*, 1994.

[Datacube, 2001] Datacube. Datacube website, 2001. see http://www.datacube.com.

[Geo, 2001] Georgia Tech Research Corporation. *VSIPL 1.01 API*, March 2001.

[Köethe, 1998] Ullrich Köethe. On data access via iterators (working draft). February 1998. see http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/doc/documents/DataAccessors.ps.

[Köethe, 1999] Ullrich Köethe. Reusable software in computer vision. *B. Jähne, H. Haussecker, P. Geissler: "Handbook on Computer Vision and Applications"*, 3, 1999. see http://kogs-www.informatik.uni-hamburg.de/~koethe/papers/handbook.ps.gz.

[Köethe, 2000a] Ullrich Köethe. *Generische Programmierung für die Bildverarbeitung.* PhD thesis, 2000. see http://kogs-www.informatik.uni-hamburg.de/~koethe/papers/DissPrint.ps.gz.

[Köethe, 2000b] Ullrich Köethe. Stl-style generic programming with images. *C++ Report Magazine*, pages 24–30, January 2000. see http://kogs-www.informatik.uni-hamburg.de/~koethe/papers/GenericProg2DC++Report.ps.gz.

[Köethe, 2001] Ullrich Köethe. *VIGRA Reference Manual for 1.1.2*, April 2001. see http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/doc/index.html.

[Olson *et al.*, 1996] Thomas J. Olson, John R. Taylor, and Robert J. Lockwood. Programming a pipelined image processor. *Computer Vision and Image Understanding: CVIU*, 64(3):351–367, 1996.

[vsi, 2001] Vsipl website, 2001. Maintained by Dr. Mark Richards of Georgia Tech Research Institue for DARPA and U.S. Navy.