USE OF IMAGE ALGEBRA IN RACINE IP WP6, AND THE ABSTRACT IMAGE-PROCESSING MACHINE

PAUL COCKSHOTT

Abstract. We present an outline of the concept of an image algebra and the operations and types that would provide a suitable basis for hardware acceleration. We go on to define an outline abstract instructionset from which hardware or software components can be composed to form image processing operations.

This is a short introduction to Image Algebra. The proposal is that the Image Algebra may provide a suitable basis for defining the primitive hardware operations to be provided by Pandora. Development of image algebra at the University of Florida began in 1984 under the sponsorship of Eglin Air Force Base and DARPA. It was influenced by a history of use of special purpose image processing hardware going back to CLIP[2] in the 70s. Since that time, numerous results have been derived. The salient properties of image algebra, however, can be summarized briefly as follows:

- Image algebra is a translucent notation in which all image processing and computer vision image transformations may be described.
- Image algebra is based upon well-defined, well-understood mathematical systems, thus algorithms expressed in image algebra are amenable to formal analysis.
- Numerous Image Processing and Computer Vision algorithms have been expressed in image algebra [1] describes many of the algorithms and their variants in image algebra.
- Image algebra is amenable to computer implementation if appropriate restrictions on pointset topology and value sets are made.

A C++ object library implementing a rich image algebra subset has been developed and is available for distribution. Previous implementations have involved preprocessors for image algebra in FORTRAN, Pascal[5] and Ada[4], and interpretive implementations of image algebra in *lisp (for the Connection Machine CM2 [3]) and C.

More recently research has been done by researchers in Northern Ireland on translating image algebra into FPGA configurations[6, 7], particularly Xilinix based systems.

Key concepts

The image algebra is organised around the concepts of *point-sets*, *value-sets* and images. An image is thought of as a pair $(x, a(x))$ where $x$is a set of points and a is a function mapping from points into a value set. The value set would typically be something like the real numbers, the booleans or the integers, but it could be some more complex data-type. The point-set would typically be a set of coordinates in 2D space, but the co-ordinate system can extend to higher dimensional spaces for volumetric images or for time sequences like film.

Date: Version 2, 1st September 2005.

TABLE 1. Operations on points. Notation : **X,Y** denote point sets, **x**$\in$ **X, y**$\in$ **Y** denote points, k denotes a scalar, γdenotes an infix binary operator.

Operation	formula	return type
$\gamma \in \{+,-,*,/,\max,\min\}$	$\mathbf{x}\gamma\mathbf{y} = (x_1\gamma y_1, x_2\gamma y_2, ...)$	point
dot product	$\mathbf{x} \bullet \mathbf{y} = (x_1y_1 + x_2y_2 + ...)$	number
cross product	$\mathbf{x} \times \mathbf{y} = (x_2y_3 - x_3y_2, x_3y_1 - x_1y_3, ...)$	point
concatenate	$\mathbf{x} + +\mathbf{y} = (x_1, x_2, .., x_n, y_1, y_2, ...)$	point
γ reduction	$\backslash\gamma\ \mathbf{x} = (x_1\gamma x_2\gamma x_3...\gamma x_n)$	number
$\gamma \in \{+,-,*,/,\max,\min\}$		
Euclidean norm	$L^2(\mathbf{x}) = \sqrt{\sum x_i^2}$	real
Minkowski norm	$L^1(\mathbf{x}) = \sum \lvert x_i \rvert$	number
floor	$\lfloor\mathbf{x}\rfloor = (\lfloor x_1\rfloor, \lfloor x_2\rfloor, ...)$	point
ceiling	$\lceil\mathbf{x}\rceil = (\lceil x_1\rceil, \lceil x_2\rceil, ...)$	point
round	$[\mathbf{x}]$ =$([x_1], [x_2], ...)$	point
projection	$\rho(\mathbf{x}, i) = x_i$	number
characteristic fn	$\chi_{\mathbf{X}}(\mathbf{z}) = \begin{cases} 1 & if \mathbf{z} \in \mathbf{X} \\ 0 & otherwise \end{cases}$	number

TABLE 2. Point-set arithmetic, with $\mathbf{X}, \mathbf{Y} \subset \mathbf{Z}$

operator	formula	return type
$\gamma \in \{+,-\}$	$\mathbf{X}\gamma\mathbf{Y} = \{\mathbf{x}\gamma\ \mathbf{y} : \mathbf{x} \in \mathbf{X}\ and\ \mathbf{y} \in \mathbf{Y}\}$	$\mathbf{P}\subset \mathbf{Z}$
$\omega \in\{\leq, \geq, <, >, =\}$	$\mathbf{X}\omega\mathbf{b} = \{\mathbf{x} : \mathbf{x}\omega\ \mathbf{b}, \mathbf{x} \in \mathbf{X}\}$	$\mathbf{P}\subseteq \mathbf{X}$
set to point	$\mathbf{X}\ \gamma\ \mathbf{y} = \mathbf{X}\ \gamma\ \{\mathbf{y}\}$	$\mathbf{P}\subset \mathbf{Z}$

It is important to recognise that the set x need not totally cover the range of possible co-ordinates. When dealing with pictures that are 512x512, one might have an image for which the point-set was some rectangular sub-range of the image, or more generally some arbitrary set of disconected points. For instance one might have an image derived from some prior image such that it includes all points that were not a particular shade of blue.

On top of these basic types the image algebra specifies a collection of operators that operate on pointsets, valuesets and images.

OPERATIONS ON SETS

The standard set theoretic operations are defined on sets : $(\cap, \cup, \backslash)$ intersection, union and subtraction. In addition the functions

```
Card(set):integer,
Min(set of t):t
Max(set of t):t
Choose(set of t):t
```

are available.

Operations on points. Point-sets have the further possibility of being re-mapped. A point set will be composed of elements that are tuples of numbers. As such the points can be treated as vectors and are subject to the normal operations on vectors including adding, subtracting, matrix multiplication etc. These allow the coordinate space of images to be remapped for image warps. We summarise these in table 1.

Operations on point-sets. As mentioned above point-sets inherit the operations $(\cap, \cup, \backslash)$which apply to sets in general in addtion they support addition and subtraction as shown in table 2.

TABLE 3. Unary point set operations

operation	formula	return type
negations	$-\mathbf{X} = \{-\mathbf{x} : \mathbf{x} \in \mathbf{X}\}$	point set
complement	$\overline{\mathbf{X}}=\{\mathbf{z} : \mathbf{z} \in \mathbf{Z}$ *and* $\mathbf{z} \notin \mathbf{X}\}$	$\mathbf{P}\subset \mathbf{Z}$
reduction	$\backslash\gamma\mathbf{X}, \gamma \in \{+, \min, \max\}$	point

The notion of arithmetic on point sets takes a little thought consider the following example:

Suppose we want to add the sets {(1,2),(7,9)} and {(1,1),(2,2),(7,8)} then the result would be the set:

{(2,3),(8,10),(3,4),(9,11),(9,10),(14,17)}, i.e., the set of the sums of all possible pairs from the original sets. One can see that if one had a small mask **M** defined as s point set. If one had another point set **N** being points of interest in image **i** then **M**+**N** would be the points of interest under the mask. The unary operations are negation, complement and reduction[1].

Comparison between a point and a pointset allows restriction of the range of an image. Thus given a pointset **P** defined on the plane 1..10,1..20, the operation $\mathbf{P} <(11, 11)$ would give us the pointset for the plane 1..10,1..10.

OPERATIONS ON VALUES

Values of pixels are assumed in image algebra to be the integers $\mathbb{Z}$, the reals $\mathbb{R}$, or the complex numbers . In practice we can restrict $\mathbb{Z}$ to $\mathbb{Z}_{\not\models\neg}$, the binary integers of length k. The normal arithmetic operations are expected on these types. It is important to also provide saturating operations. Thus the real representation should include representations of $\pm\infty$and should ensure that $a + \infty = \infty$,$a - \infty = -\infty$. For integer representations there should exist distinct top and bottom values which have these algebraic properties.

Suggested primitive types are:1bit boolean values, 8bit unsigned integer, 8 bit signed fixed point with `FFH` representing -1.0 and `7FH` representing +1.0, 32 bit floating point. The 32 bit floating point type is suitable for holding most higher precision integer types.

It should also be possible for an image to have pixels whose values are vectors of primitive values.

OPERATIONS ON IMAGES

An image $\mathbf{F}^{\mathbf{X}}$of some value type **F** and over a point-set **X** is concieved of as a set of pairs pairs

$$\{(\mathbf{x}\ ,a(\mathbf{x})) : \mathbf{x} \in\ \mathbf{X},a(\mathbf{x}) \in \mathbf{F}\}$$

where a is a mapping function. Typically, but not necessarily this mapping function can be implemented as a multidimensional array. Typically, but not necessrily, **X** can be implemented as a multidimensional bitmap, storing the characteristic function of the set.

There is a system of induced arithmetic on images such that if γ is some binary operator on **F**, and $\mathbf{a} \in\ \mathbf{F}^{\mathbf{X}}, \mathbf{b} \in\ \mathbf{F}^{\mathbf{Y}}$then

$$\mathbf{a}\gamma\ \mathbf{b} =\{(\mathbf{z}, \mathbf{c}(\mathbf{z})) : \mathbf{c}(\mathbf{z}) = \mathbf{a}(\mathbf{z})\gamma\ \mathbf{b}(\mathbf{z}), \mathbf{z} \in\ \mathbf{X} \cap \mathbf{Y}\}$$

Thus we have addition, subtraction etc of images under the and of their respective masks. This generalises to reduction operations over images thus $\backslash+\mathbf{a}$ would be the

[1]For reduction see definition of point reduction, given that order of members in a set is illdefined only commutative ops allowed.

sum of all pixels in the image $\backslash \max \mathbf{a}$ would return the highest valued pixel in the image etc.

It is also possible to have comparison operators which yield pointsets from images. Thus the expression $\mathbf{a} < \mathbf{b}$ for $\mathbf{a}, \mathbf{b} : \mathbf{F}^{\mathbf{X}}$or $\mathbf{a} : \mathbf{F}^{\mathbf{X}}, \mathbf{b} : \mathbf{F}$ or $\mathbf{a} : \mathbf{F}, \mathbf{b} : \mathbf{F}^{\mathbf{X}}$ will yield a pointset $\mathbf{Y} \subseteq \mathbf{X}$. These pointsets can be used in restriction operations.

Restriction. Given an image $\mathbf{a} : \mathbf{F}^{\mathbf{X}}$ and a pointset $\mathbf{Y}$ we can create a restricted image $\mathbf{a}|_{\mathbf{Y}}$ composed of only those pixels within the set $\mathbf{Y} \cap \mathbf{X}$.

Extension. The converse - extension takes two images and combines their pixels $\mathbf{a}|^{\mathbf{b}}$gives us a picture such that if $\mathbf{a} : \mathbf{F}^{\mathbf{X}}, \mathbf{b} : \mathbf{F}^{\mathbf{Y}}$ then
$\mathbf{a}|^{\mathbf{b}} = \{(\mathbf{z}, \mathbf{c}(\mathbf{z})) : \mathbf{c}(\mathbf{z}) = \mathbf{a}(\mathbf{z})\text{if } \mathbf{z} \in \mathbf{X}, \mathbf{c}(\mathbf{z}) = \mathbf{b}(\mathbf{z})\text{if } \mathbf{z} \notin \mathbf{X},\ z \in \mathbf{X} \cup \mathbf{Y}\}$

Functional compostion.

Left composition. We also have the implicit map of unary operators or functions over images, such that if $f : \mathbf{F} \to \mathbf{G}$, $\mathbf{a} : \mathbf{F}^{\mathbf{X}}$, then $f \circ \mathbf{a} : \mathbf{F}^{\mathbf{X}} \to \mathbf{G}^{\mathbf{X}}$ and

$$f(\mathbf{a}) = \{(\mathbf{x},\ \mathbf{c}(\mathbf{x})\ ) : \mathbf{c}(\mathbf{x}) = f(\mathbf{a}(\mathbf{x})), \mathbf{x} \in\ \mathbf{X}\}$$

Right Composition. If we right compose a function over points with an image we get spatial remappings of the image, thus given $f : \mathbf{Y} \to \mathbf{X}$ and $\mathbf{a} : \mathbf{F}^{\mathbf{X}}$ then $\mathbf{a} \circ f$ is defined by

$$\mathbf{a} \circ \mathbf{f} = \{(\mathbf{z}, \mathbf{a}(f(\mathbf{z}))) \colon \mathbf{z} \in\ \mathbf{Y}, f(\mathbf{z}) \in \mathbf{X}\}$$

Thus f could be some mapping function on coordinate points which allow for the construction of a virtual image over space $\mathbf{Y}$ that is obtained by sampling image $\mathbf{a}$ with coordinate space $\mathbf{X}$. For the purposes of hardware implementation, it is probably adviseable to restrict f to linear operators over $\mathbf{Y}$ that can be represented by real valued matrices.

Templates

A template is an image whose pixel values are themselves images.

If $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{Y}}$then $\mathbf{t}$ is a template defined over $\mathbf{Y}$ yielding images of type $\mathbf{F}^{\mathbf{X}}$ when indexed by a point $\mathbf{y} \in \mathbf{Y}$. The pixels of $\mathbf{t}(\mathbf{y})$ are the *weights* of the template $\mathbf{t}$ at point $\mathbf{y}$. As such a template can be though of as a generalisation of the idea of filter kernel, except that it potentially allows for a different set of weights at each point. Note that the images indexed by $\mathbf{y}$ each have as

A hardware implementation should preferably allow for templates which have only a single set of weights i.e
$\mathbf{t}(\mathbf{y})(\mathbf{x}) = \mathbf{t}(\mathbf{p})(\mathbf{q}) : \mathbf{p}, \mathbf{y} \in \mathbf{Y}, \mathbf{x}, \mathbf{p} \in \mathbf{X}, \mathbf{y} - \mathbf{x} = \mathbf{p} - \mathbf{q}$

Generalised matrix product

This is a generalised form of filtering parameterised by operators. A standard linear filter is built around the operations of multiplication and addition. This can be generalised to any operators $\bigcirc, \ddagger$ where $\bigcirc$distributes over $\ddagger$ and $\ddagger$ is associative and commutative. We can write this as

$$\mathbf{b} = \mathbf{a} \bigcirc . \ddagger \mathbf{t}$$

and $\mathbf{b} \in \mathbf{F}^{\mathbf{Y}}$
and

$$\mathbf{b}(\mathbf{y}) = \backslash \ddagger (\mathbf{a} \bigcirc \mathbf{t}(\mathbf{y}))$$

For example for linear filters $\bigcirc$ would be multiply and $\ddagger$ would be $+$.

Hardware Interface

In this section we look at a possible way of presenting image algebra algorithms to the drivers that will control the Pandora hardware. The key concept here is that of an Abstract Image-processing Machine, or AIM.

Image algebra is a notation for writing down image processing algorithms as a concise sequence of symbols. The symbols broadly divide into variables, brackets and operators. Formulae written in this way can be parsed using well known techniques to derive tree structures in computer memory. Techniques are also well known to translate such trees into *static single assignment form* which represents it as a sequence of register transfer operations in which each "register" is assigned to only once.

When compiling for an ordinary computer the registers store simple scalars, the technique has been extended to handle registers that store vectors of numbers, and there is no reason in principle why the registers could not represent the sorts of values handled by the image algebra : points, sets of points, images, templates. I do not imply that the "registers" need actually store entire images. The "registers" could represent streams of data that are processed sequentially. To allow for this one could call them channels rather than registers.

Whatever they are called, the use of registers or channels means that one can define an abstract RISC style register machine to perform the operations.

One can then see a process by which image operations are translated using a software pipeline as follows:

step	Format
1	Image algebra expression
2	Tree structure
3	AIM channel to channel instructions
4	Implementation as interconnected hardware modules

Alternatively one could have other pipelines

step	Format
1	Image algebra expression
2	Tree structure
3	software interpreter

step	Format
1	Image algebra expression
2	Tree structure
3	AIM channel to channel instructions
4	Compile to SIMD code

One has only to recall the process by which the microprocessor was invented by Intel to see the advantage of defining instructionsets to handle tasks that had previously been seen as the domain of specialised hardware. The I4004, the first microprocessor was designed in response to a request from a Japanese calculator company for a chipset for a desktop calculator. In the early 70s such machines used networks of specialised chips configured to each possible model of calculator - multiplier chips, tailored keyboard scanners, display drivers, adders, dividers etc. The initial design for the calculator presented by the Japanese company involved over 20 specialised chips. By introducing a single chip capable of running an instructionset, Intel were able both to dramatically reduce the chip count of the calculator and also to allow the same chip to be used in multiple different calculators. In due course this led to the growth of the huge microprocessor market which substituted instruction sequences for dedicated hardware. In doing so it economised on a very scarce skill - that of chip designers - and substituted a cheaper skill - that of software design.

In the case of Pandora, the skill of the logic designers is a very limited resource. It is not practicable for this to be invoked for each particular algorithm that users of their hardware may wish to implement. It is however feasible for Pandora to define a libary of operators implemented in hardware that are dynamically configured by the device driver when the latter is given a sequence of AIM instructions.

What might an abstract machine for image algebra look like?

We suggest an instructionset that is superficially similar to that of RISC computers, with a 3 register/channel format.

OP c1,c2,c3

meaning

c1←c2 OP c3

with c1, c2, c3 representing channels.

These operation instructions would be augmented with load store instructions that connect channels to buffers in the host computer memory.

The channels would be drawn from a suitable sub-range of the integers with the number of channels that can be used in a given algorithm being constrained by the number of FPGAs available.

Possible instruction-set. Note that in the following, when an operation like c1:=c2+c3 is specified, the type of the addition is generic - it could apply to scalars, points, images depending on the sort of data refered to by the channels. The channels have their data types specified by the load instructions which specify what type of data - scalars, images, points etc are being refered to.

instruction	comment
`load c,t,M`	c:channel, t the type being loaded, M host buffer address
`store c,,M`	c:channel, M host buffer address
`add c1,c2,c3`	c1:=c2+c3
`sub c1,c2,c3`	c1:=c2-c3
`mul c1,c2,c3`	c1:=c2×c3
`div c1,c2,c3`	c1:=c2÷ c3
`union c1,c2,c3`	c1:=c2∪c3, only on sets
`inter c1,c2,c3`	c1:=c2∩c3, only on sets
`res c1,c2,c3`	c1:=c2 $\|_{c3}$ restriction of an image by a set c3
`ext c1,c2,c3`	c1:=c2$\|^{c3}$ where c2,c3 both images
`lt c1,c2,c3`	c1:=c2 < c3 for images returns set that are less
`eq c1,c2,c3`	c1:=c2 = c3 returns set
`min c1,c2,c3`	c1:= c2∨c3
`max c1,c2,c3`	c1:=c2∧c3
`reduce c1,`γ`,c2`	c1:= $\backslash\gamma$c3, this is gamma reduction of images,sets,points
`comp c1,c2`	c1:= complement of c2, applies to sets
`neg c1,c2`	c1:=-c2
`floor c1,c2`	c1:=$\lfloor c2 \rfloor$
`ceil c1,c2`	c1:=$\lceil c2 \rceil$
`round c1,c2`	c1:=$[c2]$
`dot` α, β,`c1,c2,c3`	c1:=c2 $\alpha \bullet \beta$c3 generalised matrix product
`dom c1,c2`	c1:= index set of c2, applies to images
`norm n,c1,c2`	c1:=$(\sum c2^n)^{1/n}$ this gives euclidean and Minkowski norms
`near c1,c2`	c1:= pixels adjacent to set c2
`ftk c1,c2,c3`	form template from kernel, c2 pointset, c3 kernel

It will be apreciated that the above is a hurried first draft of an instructionset for AIM and that a number of things would have to be determined yet: the base precisions of arithmetic to be supported, the ways in which operands would be represented in memory etc.

It is intended merely as an initial discussion document.

References

[1] Ritter, G., Wilson, J., *Handbook of computer vision algorithms in Image Algebra*, Center for Computer Vision, University of Florida, 1999.

[2] Duff, M., Watson, D., Fountain, T., Shaw, G., “A cellular logic array for image processin”, Pattern Recognition, vol 5, pp 229-247, Sept 1973.

[3] Hillis, W, *The Connection Machine*, Cambridge MA, MIT Press, 1985.

[4] Wilson, J., “Use of Image Algebra for Portable Image Processing Algorithm Specification”, Proc SPIE Vol 1659, 1995.

[5] Dougherty, E.R.; Sehdev, P.; “A robust image processing language in the context of image algebra”, Computer Vision and Pattern Recognition, 1988. Proceedings CVPR ’88., Computer Society Conference on , 5-9 June 1988,Pages:748 - 753

[6] Crookes, D.; Alotaibi, K.; Bouridane, A.; Donachy, P.; Benkrid, A.; “An environment for generating FPGA architectures for image algebra-based algorithms Image Processing”, 1998. ICIP 98. Proceedings. 1998 International Conference on , 4-7 Oct. 1998 Pages:990 - 994 vol.3

[7] Crookes, D.; Benkrid, K.; Bouridane, A.; Alotaibi, K.; Benkrid, A.; “Design and implementation of a high level programming environment for FPGA-based image processing”, Vision, Image and Signal Processing, IEE Proceedings- , Volume: 147 , Issue: 4 , Aug. 2000 Pages:377 - 384

Department of computing science University of Glasgow, 17 Lilybank Gardens, Glasgow, G12 8RZ, Scotland
E-mail address: wpc@dcs.gla.ac.uk