ANIMAL: AN IMage ALgebra

An Image Processing Environment based on XLISP

R. Brunelli and C. M. Modena
Istituto per la Ricerca Scientifica e Tecnologica
I-38050 Povo, Trento, ITALY

Abstract— **In this paper we present ANIMAL, an interactive environment for image processing that is oriented toward the rapid prototyping, testing, and modifications of algorithms. In the first part the theoretical foundations of the environment are presented. The concept of image algebra is introduced through the definition of multisort algebra. In the second part a description of the implementation is given, together with some examples to illustrate the flexibility and open-ended nature of the system.**

Keywords: Image algebra, Image processing, Image analysis, Parallel Processing.

1. Introduction

In this paper we present ANIMAL, an interactive environment for image processing oriented toward the rapid prototyping, testing, and modifications of algorithms. In the first part the theoretical foundations of the environment are presented. The concept of image algebra is introduced through the definition of multisort algebra. ANIMAL is both a theoretical framework and an implementation of the working environment described in the second part. The whole environment can be seen as an extension of an existing LISP interpreter (XLISP[1]). We added to the original LISP kernel three other functional blocks: a software SIMD (Single Instruction Multiple Data) machine [1], a set of standard image processing operators (filters, statistical and slicing operators,...) and an object-oriented graphic interface. The LISP interpreter and the self-documenting capability of the operators working on images make ANIMAL a very interactive, flexible environment. One of the main goal of the project was the portability of the environment and it has been obtained through a C-language coding and use of the X11-Windows System[2] the *de facto* standard of windowing for UNIX[3] workstations.

2. WHAT IS AN ALGEBRA?

The goal of this paragraph is to present some definitions of modern and universal algebra [2][3] that will be useful for a rigorous definition of ANIMAL.

Definition 1: An *algebraic system* (or *algebra*) is a pair $(A; F)$, where A is a non-empty set and F is a (possibly empty or infinite) collection of operations: each operation $f_i \in F$ is a function

$$f_i : A^{n(i)} \to A \tag{1}$$

where $n(i)$ is a natural number that depends on f_i, called *arity*. An operation like f_i is called $n(i)$-ary operation on A. Generally, we refer to any n-ary operation as *finitary operation*.

Definition 2: A *partial algebra* is an algebra where some of the operations may be defined only partially, i.e. an n-ary operation is not defined on the entire domain A^n.

[1] XLISP: An Experimental Object-oriented Language by David Michael Betz (1986).
[2] X Windows System is a trademark of MIT.
[3] UNIX is a trademark of AT&T Bell Laboratories.

Example *1:* $(N; -)$ the positive integers under binary subtraction. This is a partial algebra because $m - n$ is undefined if $m \leq n$.

Definition 3: A *heterogeneous algebra* (or *multisort algebra*) is a family of sets $\{A_i\}_{i \in I}$ together with a collection F of (heterogeneous) functions: for each n-ary operation $f \in F$ there is an $n+1$-tuple $(i_1, \ldots, i_n, i_{n+1}) \in I^{n+1}$ such that

$$f : A_{i_1} \times \cdots \times A_{i_n} \to A_{i_{n+1}} \tag{2}$$

Example *2:* A vector space $(A, K; +, -, \oplus, \ominus, \times, \bullet)$ is a heterogeneous algebra with two carriers: vectors A and scalars K. $(A; +, -)$ is an additive group with a binary and a unary operation:

$$+ \ : \ A \times A \to A \text{and} - : A \to A; \tag{3}$$

$(K; \oplus, \ominus, \times)$ is a field; and $\bullet : K \times A \to A$ is a scalar multiplication.

Remark If F is a set (of operators) we can define an injective map ω from F to Σ, where Σ is the set of all names (strings). Sometimes we will write $\omega(f) =$ 'f'; 'f' is independent of the definition of f because it is only a name.

3. ANIMAL AS ALGEBRA

In this paragraph we give an abstract definition of ANIMAL as a particular heterogeneous algebra.

Notations Let us now consider the set $\mathbf{M}$ of all infinite $\mathbf{Z}$-matrices with entries in $\mathbf{Z}$ (it can also be viewed as the set of all the functions on $\mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$). We call $\mathbf{P}$ (parameters) the set of all the n-tuple with integer or real values for each $n \in \mathbf{N}$. With $\mathbf{T}$ we denote the set of all (ordered) trees whose nodes belong to Σ or $\mathbf{P}$.

Definition 4: On the family of all the unions spanned by the sets

$$\mathbf{M} \times \mathbf{T}, \{\mathbf{Z}^h\}_{h \in N'}, \{\mathbf{R}^k\}_{k \in N} \tag{4}$$

we can define a set F of partial heterogeneous operators, so that we obtain a partial multisort algebra. We call *images* the elements of the set $\mathbf{M} \times \mathbf{T}$ and ANIMAL the corresponding algebra (from AN IMage ALgebra). The second entry of an image is called *history*.

Remark Often we call *image* only the first element m of the pair (m, h), when there is no danger of confusion.

Example *3:* Let

$$\texttt{LEFT} \ : \ \mathbf{M} \times \mathbf{T} \times \mathbf{Z} \to \mathbf{M} \times \mathbf{T} \tag{5}$$

be an operator of F defined as follows:

$$\texttt{LEFT}((m, h), z) = (\texttt{LEFT}^*(m, \ z), \texttt{concat}(\omega(\texttt{LEFT}), h, z)) \tag{6}$$

where $\texttt{LEFT}^*(m, z) : \mathbf{M} \times \mathbf{Z} \to \mathbf{M}$ is such that $(\texttt{LEFT}^*(m, z))_{ij} = m_{i,j+z}$ for each i, j in $\mathbf{Z}$, and concat yields a tree with root the name ω(LEFT), first subtree h and second subtree the leave z.

Definition 5: An image (m, h) is called *coherent* if

$$\text{eval}(h) \ = \ (m, \ h) \tag{7}$$

where eval is a function that substitutes each name $\omega(f)$ present in the history h with the operator f, using the injectivity of ω.

4. IMAGES IMPLEMENTATION

We cannot implement the general model of image, because the first entry of an image is a boundless matrix. However, for our purposes we can restrict our interest to a limited region of the image, bounded by a rectangular region of the discrete plane; outside this region the elements of the matrix have a constant value that can be changed, on an image by image basis, at every computational steps. This implementation requirement is not restrictive, as real images are bounded anyway. Such an infinite matrix is consequently equivalent to a structure consisting of a starting point of the interesting region (called *focus*, the data of the region and the constant value outside it. There are several advantages in representing images as boundless arrays:

- all points, including those on the boundary of the focus, have the same number of neighbours (this is important in convolutions, filtering, template matching, and so on);
- the *shift* functions are performed without loss of information and are simply realized changing focus coordinates;
- the operations acting on two images defined through binary commutative operations on integers (e.g. `ADD, AND` are commutative as they are independent of foci size.

The history is useful to trace back the elaborations performed on a given image: it records all the operations which acted on the image, together with their parameters. In this way one can use the history (by evaluation as `LISP` list) later on, either to reproduce the output image or, after some editing, to realize a new library operator. The possibility to enable/disable the history tracing allows the user to create new operators that output coherent images.

A (real) image can be represented as follows:

```
x-init, y-init,
data-array,
outside-value,
h-flag, history;
```

and this corresponds to our implementation.

5. ANIMAL SIMD MACHINE

One of the main applications of `ANIMAL` is in the rapid prototyping on algorithms to be implemented on a `VLSI` chip whose structure is that of a `SIMD` machine [4]. Let us briefly recall what a `SIMD` machine is. The acronym `SIMD` stands for *Single Instruction Multiple Data*[1]. This is one of the possible parallel organizations for a set of processing elements (`PE`) and corresponds essentially to array processors. An array processor is a synchronous parallel computer with multiple arithmetic logical units (`PE`s) that can operate in parallel in lock step fashion. A set of masking schemes can be used to enable/disable the processing elements during the execution of vector instructions. The topological structure of a `SIMD` array processor is mainly characterized by the data routing network used to interconnect the processing elements. Formally, such an inter-`PE` communication network can be specified by a set of data-routing functions [5]. If we identify the addresses of all `PE`s in a `SIMD` machine by the set S, each routing function is a bijection from S to S. When a routing function f is executed via the interconnection network the `PE`(i) copies its data into the data location of `PE`($f(I)$). A `SIMD` computer $\mathbf{C}$ is then characterized by the following 4-tuple:

$$\mathbf{C} =< N, F, I, M >$$

where:

N is the number of `PE` in the system;
F is a set of data-routing functions provided by the interconnection network;
I is the set of machine instructions for scalar, vector, data routing and network manipulation operations;
M is the set of masking schemes.

The `SIMD` organization is a natural one for fine-grained parallel machines and for image processing tasks. Many early vision problems map naturally onto an array of processors, each one performing the same instruction using the data available in the local neighbourhood. A software simulation of a `SIMD` machine is available in `ANIMAL`.

We can associate a processing element to every point of the plane $\mathbf{Z} \times \mathbf{Z}$. To realize the local memory of the `PE`s (`PEM`) we can imagine to overlay an image on the plane $\mathbf{Z} \times \mathbf{Z}$. Multiple local memories can be realized using list of images, each pixel corresponding to a single local memory of the aligned `PE` (a multi-layer memory structure). The interconnection network is realized through the shift operation: in this way we can align each `PEM` with every other `PEM`, in a rigid way for the whole image structure. This allows us to use the near-neighbor-meshes (4-connected, 6-connected, 8-connected) frequently used in image processing and many more. Each `PE` can then operate on its data and on the data of virtually every other `PE`. The instructions a single `PE` can execute are the following:

arithmetic : `add sub mult div`
relational : $> < \geq \leq$; max min
logical : `or and xor not`.

All of these function (with exclusion of `not`) have the same structure:

(function arg_1 arg_2 &optional arg_3)

where:

arg_1 must be an image;
arg_2 can be an image or an integer;
arg_3 can be a bit-image or a 4-tuple of points identifying a rectangular region in the plane.

The first argument corresponds to the data-output of the corresponding `PE`; the second argument corresponds to the data obtained via the interconnection network and stored in the `PEM`. If the second argument is an integer this corresponds to the broadcasting of data by the `CU`. The modification of a single pixel via the standard `setf/aref LISP` function corresponds to the distribution of data via the system bus. The third argument reflects the possibility of defining a mask to inhibit `PE`s' execution. The default activation mask is on in the rectangular region bounding the image arguments. An alternative activation mask can be specified using a single bit image (1 = active, 0 = disabled) or via a 4-tuple identifying a rectangular region.

6. THE ANIMAL ENVIRONMENT

The image processing community needs flexible and powerful tools to develop and implement algorithms ([6a][6b][6c], [7]). One of the major difficulties stems from the huge variety of specialized hardware used for image processing tasks and the related types of software tools. Our work stems from the belief that there is no sharp division between concept of hardware and software. They are *dual* concepts: the functions of one can be implemented by the other. We can then effectively consider a language as a software machine [8]. .br `ANIMAL` is a natural out growth of this philosophy. We chose a very flexible language, `LISP`, and we extended it with a software implementation of an `SIMD` machine allowing the manipulation of images as a whole and not as a collection of pixels. This results in two different architectures within the same environment:

serial: the original `LISP` kernel;
parallel: the implemented `SIMD` machine.

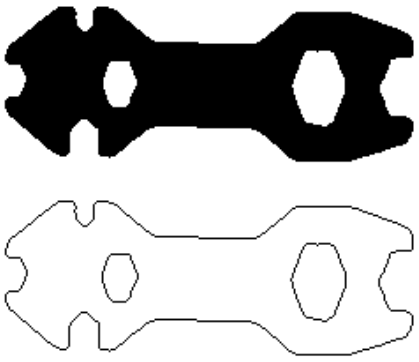

Fig. 1. Binary image representing the silhouette of a wrench and its edges computed using ANIMAL EDGE opeerator

Each problem can be solved using the structure best reflecting it. It is important to note that a parallel algorithm implemented on a serial machine, that simulates a parallel computer, does not run necessarily much slower than a corresponding serial solution on the same machine [8]. Some tests with ANIMAL show that the efficiency can be almost the same.

The environment we have implemented satisfies the following fundamental requests:

flexibility : image dimensions are unconstrained and images can be stored with variable precision (1-bit, 8-bit, 16-bit, 32-bit and float);

efficiency : the relational and arithmetic operators of the SIMD machine are carefully optimized for speed;

universality : image, numerical and structural processing can be performed in the same environment;

extensibility : the use of a LISP interpreter guarantees a good interactivity and an excellent extensibility of the system with the possibility to create new functions (the user expands the set F of ANIMAL operators).

We have implemented in the kernel of ANIMAL many operators like the logical, arithmetic and relational operations on images, filters, statistics (histogram, cooccurrence matrix, etc.), extraction of connected components, and many others: their detailed description is beyond the scope of this paper; the reader is referred to [9][10]. Using the SIMD machine it is possible to create many useful operators by functional composition of the old ones. A mathematical framework for the developing of image processing algorithms is *mathematical morphology* [11][12]. The available arithmetical and relational operators make the implementation of its basic operations (erosion, dilation, opening and closing) an easy task. Their combination into more powerful blocks can be realized through the LISP functional mechanism. This approach reflects the original idea of complete and self contained stages of image analysis algorithms.

The following EDGE algorithm is an example of a simple composition of basic image operators taken from a still growing library:

Example *4:*

```
(defun EDGE (img)
  (AND img
       (NOT (AND (UP img)
                 (AND (DOWN img)
                      (AND (LEFT img)
                           (RIGHT img)))))))
```

In Fig.1 we reported the output of the EDGE operator applied to a binary image.

As another example in which image functions and standard LISP functions are combined, we present the general convolution algorithm:

Example *5:*

```
(defun CONVOLVE (img knl)
  (let* ((deltax (/ (nth 1 (array-dimensions knl)) 2))
         (deltay (/ (nth 0 (array-dimensions knl)) 2))
         (mymask (make-mask (aref img 'X)
                            (aref img 'Y)
                            (aref img 'WIDTH)
                            (aref img 'HEIGHT)))

         ; my-tmp is a temporary image
         ; initialized to the null image

         (my-tmp (MAKE-IMAGE (aref img 'X)
                             (aref img 'Y)
                             (make-array
                               (array-dimensions
                                 (aref img 'DATA))
                               :element-type 'INT)))
    (do ((i (- deltay) (+ i 1)))
        ((¿ i deltay) my-tmp)
        (do ((j (- deltax) (+ j 1)))
            ((¿ j deltax))
            (setq my-tmp (ADD
                           (MULT
                             (SHIFT image (- j) (- i))
                             (aref knl
                                   (+ i deltay)
                                   (+ j deltax)))
                           my-tmp mymask)))))
```

Remark From the code of example 7.2 we have left out the history tracing mechanism.

Images are essentially visual objects and human beings found it more natural to see them represented on a screen using grey-levels or colors than to see their numerical print-outs. ANIMAL provides a graphic interface based on the X11-Windows System and is realized through an object system. It consists of several classes of windows specialized for image display, function and histogram plots. Each class allows the user to select interactively semantically meaningful regions (rectangular region for an image window and ranges for function or histogram plots) and supplies some other basic functions like zoom and scroll. .br The whole interface is written in XLISP using a set of functions that directly use the X11-Windows System: this structure allows the willing user to customize the environment. We should note that the graphic interface is completely independent of the implemented image processing operators, so that should the X11-Windows System not be available on a particular computer, one can build a new graphic interface without changing the kernel of ANIMAL. It is possible to process images even when no graphic interface is available: the processed images can be saved in files and can be viewed later on using other programs.

Remark ANIMAL is highly portable, because the entire environment is written in C-language and uses the X11-Windows System, a windowing system that is rapidly becoming a *de facto* standard[4].

7. CONCLUSIONS

In this paper we described ANIMAL: AN IMage ALgebra, as a heterogeneous algebra whose family of operators can grow according to users' needs. The environment, implemented at

[4]Some technical information: ANIMAL consists of 1MB of C-language source code, 400KB of executable code and 200KB of LISP code. Image dimensions are limited by the maximum virtual memory available to the process. The current implementation runs on HP9000 S300, Sun3, Sun4, Sun386i. ANIMAL is available on a non disclosure agreement by sending a request directly to the authors.

I.R.S.T., gives to the user a set of fast image processing operators and the possibility to combine them under a LISP interpreter for specific applications. **ANIMAL** proved very useful in reducing the time lag between the formulation of an algorithm and its implementation, relieving the user from many time consuming programming details.

REFERENCES

[1] M. J. FLYNN, *Very High-Speed Computing Systems*, Proc. IEEE, Vol. 54, 1966; pp. 1901-1909;

[2] J. D. LIPSON, *Elements of Algebra and Algebraic Computing*, Addison-Wesley, Reading MA, 1981; pp. 57-64;

[3] G. BIRKHOFF, *The Role of Algebra in Computing*, in: *Computers in Algebra and Number Theory*, G. Birkhoff and M. Hall Eds., SIAM AMS Proc., Vol. IV, 1971;

[4] L. STRINGA and A. ZORAT, *A single Chip Sensor and Processor: A Strategic Project*, in: *Biological and Artificial Intelligence Systems*, E. Clementi and S. Chin Eds., ESCOM, 1988; pp. 499-508;

[5] K. HWANG and F. A. BRIGGS, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York NY, 1985; pp. 325-388;

[6a] Z. KULPA, *PICASSO, PICASSO-SHOW and PAL - A Development of a High-level Software System for Image Processing*, pp. 13-24;

[6b] T. RADHAKRISHNAN, R. BARBERA, A. GUZMAN and A. JINICH, *Design of a High-level Language (L) for Image Processing*, pp. 25-40;

[6c] S. LEVIALDI, A. MOGGIOLO-SCHETTINI, M. NAPOLI, G. TORTORA and G.UCCELLA, *On the Design and Implementation of PIXAL, a Language for Image Processing*, pp. 89-98; in: *Language and Architectures for Image Processing*, M.J.B. Duff and S. Levialdi Eds., Academic Press, Orlando FL, 1981;

[7] A. P. REEVES, *A systematically Designed Binary Array Processor*, IEEE Trans. on Computers, Vol. C-29, No. 4, 1980;

[8] A. WOOD, *The Interaction between Hardware, Software and Algorithms*, in: *Language and Architectures for Image Processing*, M. J. B. Duff and S. Levialdi Eds., Academic Press, Orlando FL, 1981; pp. 1-11;

[9] R. BRUNELLI and C. M. MODENA, *ANIMAL: AN IMage ALgebra; User's Guide*, I.R.S.T. Int. Rep., 1988;

[10] R. BRUNELLI and C. M. MODENA, *ANIMAL: AN IMage ALgebra; User's Manual*, I.R.S.T. Int. Rep., 1988;

[11] J. SERRA, *Introduction to Mathematical Morphology*, Comput. Vision, Graphics and Image Process., Vol. 35, No. 3, 1986; pp. 283-305;

[12] R. M. HARALICK, S. R. STERNBERG and X. ZHUANG, *Image Analysis Using Mathematical Morphology*, IEEE Trans. Pattern Anal. Machine Intell., Vol. PAMI-9, No. 4, 1987; pp. 532-550.