IAPR Workshop on CV - Special Hardware and Industrial Applications OCT.12-14, 1988, Tokyo

An Image Processing Language for Step-by-step Development of Image Processing Systems Consisting of Special-purpose Hardwares and/or Micro-programmed Processor

Koichiro DEGUCHI

Faculty of Engineering, University of Tokyo
Tokyo 113 Japan

ABSTRACT

An image processing language was developed to define several types of image processing and execute them on high-speed image processing systems used in research laboratories. It is common to apply a combined series of simple basic operations (called image processing primitives) on image data step by step. And it is also common to construct high-speed image processing system by combining several special purpose hardwares and/or micro-program modules for high-speed processor for those basic primitives. Because those special hardwares and micro-programs are usually not easy to be implemented nor flexible, they must be improved well prior to the implementation. This language supports the efficient development of programs on high-speed image processor by making primitives into macro-primitives stepwisely with confirming their performances, even at the same time of the system construction.

1. Introduction

For image processing, it is common to apply combined simple basic operations on image data step by step. And it is also common to construct high-speed image processing system by combining several special purpose hardware and/or micro-program modules which have good or weak capabilities for each basic operations, respectively. Then it is important to distribute each basic operations to the most suitable processors for such a complex system. But, because the configuration of those processors might be changed according to hardware developments, and each basic operation itself must be improved also according to system reconfigurations and algorithm refinements, it is important to have efficient language support for development of programs on high-speed image processor as well as for the system construction at the same time.

In this paper, a prototype of the language system is presented for this purpose. In this language, each of the operations is expressed using functions, and the sequence of the operations is described with combinations of the functional expressions. The user can define complex image processings interactively by combining the simple functions of image processing primitives or their macros. At that time, the system improvements can be took place independently of the user program developments.

2. High-Speed Image Processing and Program Development

2.1 Target Image Processing System

The configuration of an image processing system which is the target of the installation of the language developed is shown in **Fig.1**, for example. Of course, this language is not only for this system. The system has three groups of processors, (1) a host mini-computer (HOST), (2) micro-program driven high-speed image processor (IP), and (3) special hardwares for rather simple basic processings with video memory (VM). The characteristics of these groups are listed on Table 1.

The host computer controls the total system. The microprograms on IP are loaded and started by the host computer every time needed. Each special hardware on VM is driven by commands from the host. It is intended that rather complex image processings are took place

Table 1. Characteristics of processors.

Processor	Performance	Programming
HOST	Not high-speed (General purpose)	High-level language
IP	High-speed (Special-made LSI)	Micro-program
VM	High-speed (Special hardwares)	Fixed

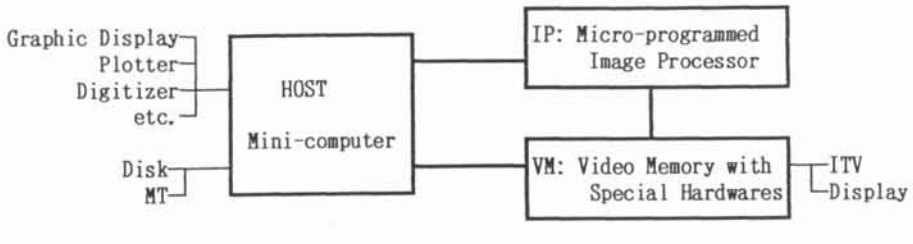

Fig.1 Configuration of the high-speed image processing system the prototype of the language is installed.

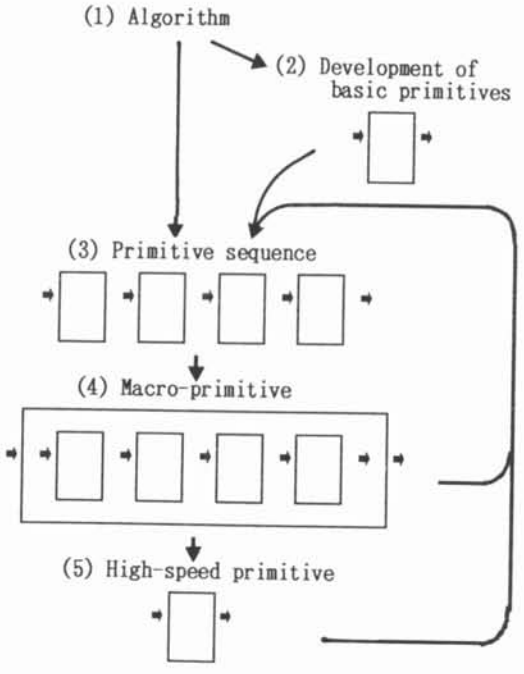

Fig.2. The step-wise improving developments of image processing programs the language supporting.

with high-speed on IP and simple and regular processings mainly accompanied with input or output of images are on VM.

So, we must distribute image processing operations to their suitable processing elements. However, this is not so straight-forward. First, the construction of micro-programs are usually not easy and must be improved well prior to the implementations. Secondly, the user want to test the performance of the algorithm of image processing by using familiar and flexible high-level language on the host computer at the first step of program development. And, some special hardwares will be added to the system in future and the system bill be reconfigured.

Therefore, an image processing language which supports program developments as well as hardware improvements is needed for such a high-speed processing system especially in research laboratories.

2.2 Step-by-Step Program development

We call simple basic unit image processing operations, such as filterings, binarizations or contour extractions, primitive operations. And image processings are performed by combining the primitives. Each the primitives are distributed and operated by each processor modules in the system. On the other hand, each modules must be improved well prior to the implementation for high-speed processings independently of the program developments. Then, the language for development of image processing program on such high-speed image processing system must have capability to support also the development of hardware. That is, it is important that the users need not mind the configuration of the system, or if those primitives are operated with special hardware, micro-programmed processor, high-level language programmed host computer or their combinations, and when the reconfiguration of these implementations are took place.

For this purpose, this language supports the developments of image processing programs through next stepwise refinement processes (**Fig. 2**);

(1) develop the prototype image processing program on host computer, if necessary, using even a high-level language to confirm its algorithm and performance,

(2) sub-divide and reduce the program into a sequence of basic image processing primitives, some of which may be already implemented as special hardwares or micro-programs on the system, and others may be in high-level languages,

(3) perform the image processing by sequentially operate the series of primitives and confirm the results,

(4) replace the useful primitive with high-speed versions by special hardwares or micro-programmed procedures, and the useful series of primitives with macro-primitive which can be operated by calling its given name in batch, and,

(5) implement the useful series of primitives into one basic primitive as a special hardware or micro-programmed procedure.

For the system having such several types of processing modules, it is efficient to develop the programs by this stepwise improvement using macro descriptions and revision into high-speed versions of processings. And it is also important that the system hardware or micro-program improvements can be took place independently of the user program development, espccially for the systems used in research laboratories.

3. Description of Image Processing Algorithms using Functions

3.1 Functional Expression of Image Processings

Because most image processings can be constructed with step-by-step sequential basic operations, it is useful to register the each basic operations as common image processing primitives not only for programmings but also for descriptions of the algorithms.

Using functions for these primitives, the above sequential operations are interpreted as a sequence of stepwisely applying next function on each output the previous function returns. We denote the output the function $func$ applied on image IN

$$func(\ IN\ ).$$

The $func$ returns the value having the "type" which is defined as the type of $func$. Therefore, when we want to apply "Image type" function $func1$ on an image IN, next $func2$ on its output, then $func3$ sequentially, and to obtain an image OUT as a final output, the operation is expressed as

$$OUT = func3(\ func2(\ func1(\ IN\ ))).$$

If a function requires a number of arguments $IN1$, $IN2$ and $IN3$, for example, the expression becomes

$$func(\ IN1, IN2, IN3\ ).$$

When some output data, which might be final or intermediate ones, must be saved for later uses, they must be assigned to a given name for the data. For the case where more than one output are required, this is not seldom in image processings, multiple outputs can be obtained by the way of so-called side effects using this assignments, for example,

$$OUT = func2(\ MID = func1(\ IN\ ))$$

These schematics are shown in **Fig. 3**. And examples of the description of image processing procedures using this language are shown in **Fig. 4**.

3.2 Data Types

In image processings, an application of each processing primitive

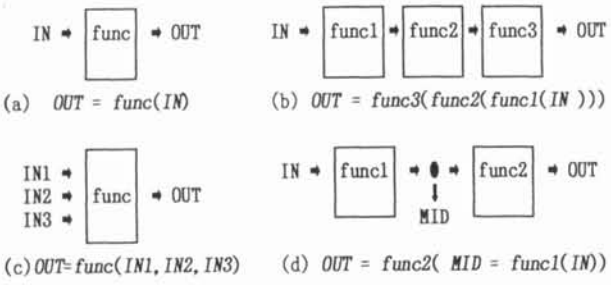

Fig. 3. Schematics of the functional expression of the sequence of image processing operations.

LADY range(LADY)

Fig. 5. Example of execution of the macro-function *range*.

has its own meaning only on data of specific type. So, by checking the match of types between the data and the function to apply, definitions of meaningless operation or mistakes can be avoided easily. In Fig.4(b), *histogram*(*IN*) is a function to calculate the histogram of gray levels of the image *IN*, and specified that it can accept only "image type" data. Then, it outputs an "array type" data which is assigned to also "array type" data *HIST*. The *valley* is specified to accept "array type" data and returns "value type" data.

3.3 Macro Functions

A sequence of operations can be defined as a new function, called a macro function, having its own name. To define the macro function a system command described later is used. As a function of "range filtering", for example, can be defined as a subtraction operation of output of "minimum filtering" *min* from that of "maximum filtering" *max*, by the definition,

.def range(*$x*)
sub(*max*(*$x*), *min*(*$x*)) ,

(where *$* denotes an argument) the function of range filtering will be available later by

range(*IN*) .

Macro functions are registered as the text defining them, and expanded into a sequence of functions at the executions. They can be "nested" with arbitrary levels in their definition, that is, macro function can be defined by using also macro functions. As is mentioned, useful macro functions can be up-graded into unit high-speed primitives by the system manager at any time. It is important that the users need not mind what kind of primitive functions just they are using.

4. Outlines of the Language Specifications

4.1 Programs

The details of the syntax of the language developed and its implementation were described in [2], and here its outlines are described briefly. Programs for image processing using this language usually consist of the declaration part and the execution part.

The each sentence of the execution part has the form

[<output>=] function(<argument>[, <argument>] · · ·)

and are separated by ";". An intermediate outputs are discarded after evaluation by next function, that is, for example, the output of *func* 1 in

$$OUT = func\,2(\, func\,1(\, IN\,))$$

will disappear. But, by defining

$$OUT = func\,2(\, MID = func\,1(\, IN\,))\,,$$

the data will remain and can be referred by the name *MID*. This is because the assignments themselves return the value of type same as the assigned data.

In this language, control statements are also given as functions. It has functions *if*, *when*, *unless* and so on. They are used as

if(<condition>,<function1>,<function2>)
when(<condition>,<function>)
unless(<condition>,<function>)

and according to the conditions the argument functions will be executed or not.

The declaration part consists of name, expression and type definitions and declarations. User defined types are available as well as macro definitions. The basic types and their substantial values to

display (PARTICLE) HIST=histogram(PARTICLE) LEVEL=valley(HIST); display(thrshld(PARTICLE, LEVEL))

Calculate histogram (histogram) of the image PARTICLE, find the thrshld value LEVEL from the histogram (valley) and binarize the original image.

Fig. 4. Example of descriptions of image processings.

Table 2. Basic types and their values.

Type	Substantial return value
pic (image data)	pointer to the top address of the image data stored
value	pointer to the value stored
array	pointer to the top of the data array
bool	pointer to the boolian value
user-defined	pointer to the specified record

be returned are listed in **Table 2**.

4.2 Primitives

In this language, formally, image processing primitives are thought of not only simple basic processing elements but also macro-functions, values, image data, and so on, which are referred by their name in the programs. They are classified by their own attributes as listed in **Table 3**, but the programmer need not minds the attributes. For example, an actual image data and a function generating a image data need not be distinguished by users if both have image type and the function generates the same image data. This is the basic mechanism enabling the stepwise improvements of primitives.

In addition to the primitives listed as entries of library, users can register new primitives using system commands.

4.2 Operation modes

The language system has two operation modes, the direct and the batch modes. In the direct mode, a sentence is expanded into a sequence of primitives and executed immediately after input of the statment line. This mode is for interactive program developments by step-by-step checking of the operations.

In the batch mode, a total program is expanded into a sequence of image processing primitives, and linked to the tail of a list called "object list". The object list is a sequence of primitives to be executed, and executed in batch by using a system command described later.

Using these two modes properly, the programmer can check his operation step-by-step and retry a sentence at a half way of his program, so that the program on the high-speed image processing system can be developed effectively.

4.3 System commands

The system has commands to manage the total system, some of which are listed in **Table 4**, for example. These commands can be executed at any time to define names, macros or types, or execute image processings etc.

5. Conclusions

A language for image processing systems having multiple processing elements was presented, and its implementation on a prototype system was described. In this language, image processings are constructed as sequences of primitive operations. By describing the primitives as forms of functions, the process of the image processing can be represented clearly. And effective step-by-step program developments as well as step-by-step hardware improvements are supported.

The major features remained for the language to have higher capabilities are a support for block structure of programs and a versatility for definitions and references of data types.

Table 3. Attributes of some primitives.

Attribute	Substantial entity
entry	entry of the function of operation
micro-program	micro-program to be loaded on high-speed processor
macro function	text defining the sequence of operations
value	value
data	name pointing to some kind of data

Table 4. System commands.

.init	initialize the system.
.quit	terminate the system.
.{	initialize object list.
.}	execute the sequence of primitives on object list.
.exec	execute the specified primitive.
.ls, .lsa	list the name defined.
.def	define macro function.
.set	define values or etc..
.rm	remove defined name.
etc.	

References

[1] M.Duff edt., *Languages and Architectures for Image Processing*, Academic Press (1981).

[2] K.Deguchi, "An Image Processing Language for the System Having Multiple Image Processors," Technical Report, Information Processing Society of Japan, SE52-21, 161/168 (1987) (in Japanese).