IEEE COMPUTER SOCIETY REPRINT

A REAL-TIME ICONIC IMAGE PROCESSOR

Ronald Lumia
Michael O. Shneier
Ernest W. Kent

Reprinted from IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, ST. LOUIS, MISSOURI, March 25-28, 1985

IEEE COMPUTER SOCIETY
1730 Massachusetts Avenue, N.W
Washington, D.C. 20036-1903

IEEE
COMPUTER
SOCIETY
PRESS

A REAL-TIME ICONIC IMAGE PROCESSOR

Ronald Lumia, Michael O. Shneier, Ernest W. Kent
Sensory Interactive Robotics Group

Building 220, Room B-124
National Bureau of Standards
Gaithersburg, MD 20899

ABSTRACT

The Sensory-Interactive Robotics Group at the National Bureau of Standards is producing PIPE, a pipelined image-processing engine, for research in low-level machine vision. PIPE processes sequences of images at field rates through a series of point and neighborhood operations. It is divided into a variable number of identical stages, each of which performs an independent set of operations on the image data stored in the stage. A stage control unit determines the sequence of operations performed within a stage on each image. This sequence is easily modified by a host computer during the inter-field interval when all of the stage control units can be totally reconfigured.

Images flow through PIPE in several ways. In addition to the (standard pipeline) "forward" pathway, where an output image is sent to the next stage, an output image can also be sent to the same stage via a "recursive" pathway and to the previous stage via a "retrograde" pathway. As a result, PIPE can support relaxation operations, temporal neighborhood operations, and other local operations.

Several processing modes are available in PIPE in addition to the usual "SIMD" mode of pipelined processors. In an "MIMD" mode, one of several operations is performed on a region of interest which can be defined by the host device or by previous image operations. PIPE also supports variable resolution pyramids where an image is compressed or expanded as it passes between stages.

INTRODUCTION

PIPE was designed as a preprocessor for iconic (spatially indexed) images. It is intended to serve as a "front-end" for the vision portion of a multi-modal sensory processing system being developed for real-time robot guidance applications at NBS. Its role is to perform transformations on images to extract features similar to those in the primal sketch of Marr(1976). These features make intensity changes and local geometric relations explicit in images, while maintaining the spatial representation. In this, PIPE differs from many processors designed for image-processing. These other processors are usually designed to perform both local and global image-processing tasks, often in an interactive environment. Pipe, however, is intended to perform only local operations. Several goals influenced the design:

(1) Real-time processing of images at field rates.

(2) Provision for interactions between related images, such as those arising from dynamic image sequences or from stereoscopic views.

(3) Ability to apply different algorithms to different image regions in real time.

(4) Ability to guide processing by knowledge-based commands and "hypothesis images" supplied by the host.

This paper introduces the reader to PIPE and its capabilities. A more detailed discussion of PIPE can be obtained in (Kent *et. al.*, 1984).

OVERVIEW OF PIPE

PIPE was designed as a preprocessor for iconic images. Figure 1 shows how PIPE fits into the NBS image-processing system. It acquires images from a variety of sources, such as analog or digital television cameras, ranging devices, and conformal mapping arrays. It processes sequences of images in real time, through a series of local neighborhood and point operations and presents its output to such devices as monitors, robot vision systems, iconic-to-symbolic mapping devices and image-processing host computers. In essence, PIPE performs local operations and presents the results to other processors which can perform global operations.

PIPE features three, concurrent, interacting, image-flow pathways. These interconnect a variable number of identical modular image-processing stages. These stages are sandwiched between special purpose input and output boards which provide a clean interface with devices outside PIPE. The three pathways are: the "forward" pathway, which acts as a traditional pipelined image-processing path; the "retrograde" pathway, which carries images in the opposite direction, i.e., from the output of a stage to the input of its predecessor; and the "recursive" pathway, which carries an image from the output of a stage back into the input of the same stage. Figure 2 shows the connections of the processing stages, the three image-flow pathways, and the stage control units which store the sequence of operations for each stage.

Processing in PIPE involves point operations and neighborhood operations on the image data stored in a stage. The point operations are performed in look-up tables and in ALUs at various points in PIPE. Each stage can perform two simultaneous and independent arithmetic or Boolean neighborhood operations on the data stored within the stage. The results from these operations can be sent on any of the image-processing pathways previously described. In an alternative mode, neighborhood operations may be modified on a pixel-by-pixel basis using information stored in another buffer to choose between several different algorithms.

PIPE allows the construction of multiresolution, "pyramid", sequences of images. Pyramids have been found useful in a large number of image-processing applications (Shneier, 1983, Tanimoto, 1984). They have an added utility in a strictly local processor like PIPE because they allow information from spatially distant regions

---

Commercial equipment is identified in this paper in order to adequately describe the systems that were developed. In no case does such identification imply recommendation by the National Bureau of Standards, nor does it imply that this equipment was necessarily the best available for the purpose. This paper was prepared in conjunction with the official duties of United States Government employees, and is not subject to United States copyright.

to be made local. The basic operations available in PIPE for constructing image pyramids are sampling and pixel doubling. Sampling is used to reduce the resolution of an image, while doubling is used to increase the size of an image. Multi-resolution processing in pipe is discussed in greater detail in (Kent *et. al.*, 1984).

The problem of programming PIPE involves assigning the resources of each stage at each field time. The internal architecture of a stage in PIPE is shown in Figure 3. While accurate, this diagram does not reflect the way in which PIPE is programmed. An alternative representation is shown in Figure 4. It is this schematic representation of a stage which will be used as a programming aid. It has four distinct sections, which are connected by switching networks. At the top of the figure there is an input section, and below this are two image buffers. Next come the neighborhood operations, followed by the output processing. The final switching network routes the outputs to the wildcard busses and/or to the three output paths. Each section is discussed below.

The three boxes across the top, from left to right, represent functions of one argument (lookup tables) to be applied to the "forward", "recursive", and "backward" input pathways, respectively. Using a lookup table in the input path permits both arithmetic functions (e.g. square root, tangent, etc.) and Boolean operations (e.g. shift, nand, etc.) to be performed on the input data before it is combined. *Whatever functional transformation* is employed during a given cycle will be shown in the appropriate box in the example presented below. These functional transformations, which occur simultaneously, lead to a single box representing a combining function to be applied during the cycle to the three input paths. This function is performed by an ALU and can be any arithmetic or logical combination of the three inputs.

In section 2 there is a crosspoint matrix for switching the inputs between two image buffers labeled X and Y, each of which stores a 256 x 256 pixel image with 8 bits of resolution. There are three possible inputs and two possible outputs. Two of the inputs arise from the wildcard busses (marked VBUSA and VBUSB), while the third is the output from the combining function. Any of the inputs can be stored in buffer X at the same time that the same or another input is stored in buffer Y. The image buffers are represented by the boxes immediately below the switch.

Below the image buffers is another switch that selects which of the buffers will serve as the input to the neighborhood operators of section 3. Only one image buffer can be used as the source for the neighborhood operators. Notice that the output from either image buffer can be routed both to the neighborhood operator and to points further down in the stage (bypassing the neighborhood operator). The terminals in the switching networks in these cases are marked appropriately, although the lines connecting them to their sources are not shown. Thus, for example, the output of the X image buffer may be used unchanged at all points marked X.

The buffer selected for neighborhood processing passes first through a lookup table (in the center of the figure) and then through both the neighborhood operators (NOP1 and NOP2). The outputs from these operators are marked as 1 and 2, respectively. The neighborhood operators may be arithmetic or Boolean functions and are completely independent. At present, the neighborhood size is 3 x 3 pixels and the result of each neighborhood operation is completed within 200 ns in order to keep up with the field rate of the input device.

A "region of interest" operator allows each stage to switch between the normal (NOP1, NOP2) operation set and alternative (NOP1', NOP2') operation sets on a pixel-by-pixel basis. In this mode, the other image buffer of the stage contains a map of the operations to be performed on homologous pixels of the image buffer undergoing operations. Potentially, up to 256 different alternative operation sets could be specified by the eight bit contents of each pixel in the map. In practice, the number of alternative operation sets selectable during a field processing time is limited by the amount of memory available within the stage to store them, which may be enlarged at will. The operation sets stored in the available memory may be changed arbitrarily between fields. This allows earlier image operations, such as edge detection, to guide later processing.

Following the neighborhood processing, another switching matrix selects the inputs to two functions of two arguments. Any of the outputs from the neighborhood operators or the image buffers can serve as inputs to the functions. The box on the left represents an arithmetic operation, performed on a pixel-by-pixel basis using an ALU. The box on the right represents a twelve-bit lookup table. The input is twelve bits selected from any two eight-bit inputs in a manner chosen by the programmer. By choosing eight bits from one argument and setting the remaining four bits to "don't care" values in the table, a function of one input can be implemented.

The final output of the stage is selected from the contents of the X and Y buffers, from the outputs of both neighborhood operations, and from the results of both functions of two arguments. The outputs can be routed to the two wildcard buffers, shown on the right of the figure, and to the "backward", "recursive", and "forward" data paths from the stage, shown left to right at the bottom of the figure. The only restriction is that two different data streams cannot be routed to the same output path.

Throughout the programming example presented below, a blank box will represent an inactive operation. A simple pass-through operation will be denoted by the unity function, U. In practice many functions shown in these examples as unitary or elementary functions will be modified to provide rounding or scaling operations as required for optimal computation accuracy. Since the functions are derived from table lookup, these incidental computations may be inserted automatically from a compiler library, and remain transparent to the user.

A backward-chaining production system is being used to program PIPE at the high level. This program, written in PROLOG, is given a description of the desired algorithm as an AND/OR graph. It then ascertains which operations must occur in the specific sections of a stage in order to satisfy the constraints imposed by the algorithm. In classic production system form, it tries a solution and if it fails, it backtracks and tries another way. Preliminary results indicate that a Sobel edge operator solution can be generated on a VAX 11/780 at the rate of one per second. (Clearly, there arolutions.) Once this high level assignment is made, a simulator, also under construction, will be useful in debugging the program. Since the assignments of stage resources has a one-to-one correspondence with the microcode that is stored in the stage control units, the PROLOG program can be easily assembled into the code required to run PIPE.

PROGRAMMING PIPE -- THE SOBEL OPERATOR

PIPE can be used for any image-processing task which can be performed using a 3 x 3 neighborhood convolution window. An example of such an algorithm is the Sobel edge operator which calculates an approximation of the gradient G(r,c) and its associated direction $\phi(r,c)$ for each row(r) and column(c) pixel in the image. Formally,

$$I_1(r,c)=I(r,c)*C_1$$

$$I_2(r,c)=I(r,c)*C_2$$

$$G(r,c)=\sqrt{I_1^2+I_2^2}$$

$$\phi(r,c)=\tan^{-1}(I_2,I_1)$$

where $C_1=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$ and $C_2=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}$ and * denotes the convolution operator.

The program is shown in Figure 5 using the stage schematic described above. In order to illustrate how new images are inserted into PIPE, the above notation needs to be modified slightly. The first input image will be labeled I_p and the next input image will be denoted as I_q. All of the intermediate images required for the algorithm will carry an extra subscript indicating the source image. This is important because the second input image can be inserted into PIPE before the first image has been fully processed.

In the first row, the first input image I_p enters the Y image buffer of stage 1. The neighborhood operator C_1 is performed for every neighborhood in I_p and the result, I_{p1}, is passed to stage 2 through the "forward" pathway where it is stored in the X buffer. Since there is no specification for the Y buffer of stage 1, the input image remains unchanged in that buffer.

The second row calculates the convolution of the original image with the C_2 operator. The result of this operator, I_{p2}, is passed to stage 2 through the "forward" pathway where it is stored in the Y buffer. It is also sent through the "recursive" pathway of stage 1 and is stored in the Y buffer. This overwrites I_p but this is no loss because the input image is no longer needed.

In the third row, the angle of the gradient is calculated using the function of two variables lookup table set up for $\tan^{-1}(I_2/I_1)$. This angle is sent out the "forward" path. Simultaneously, the I_p buffer is sent out the "recursive" pathway.

The fourth row uses the input lookup tables to square the inputs from the "forward" and "recursive" inputs. These inputs are then added in the ALU and stored in the Y buffer. After the square root of this quantity is taken in the final lookup table, the estimate of the gradient G_p is available. Simultaneously, the next input image I_q overwrites the Y buffer of stage 1 and the pipelined process continues on the next image.

For the first image, four cycles are needed. However, for all subsequent images, only three cycles are required because the first cycle of the current image is processed simultaneously with the fourth cycle of the previous image.

There may be some question concerning the host computer's ability to accept output images at the rate shown in this example (roughly 200 ns per pixel). For this example, there is no problem because the Sobel operator would probably be used as a preprocessing step for further processing in PIPE. If this were not the case, then the result could be sent to the host through the DMA channel at a suitably lower rate.

It is interesting to compare the processing capabilities of PIPE with the sequential techniques used in von Neumann computer architectures. To perform the Sobel operator for an n x m pixel image requires 19nm additions, 18nm multiplications, and 3nm lookups. Assuming that each operation takes the same amount of time and that each image has the RS-170 standard of 256 x 240 pixels, the total number of operations is 40nm = 2.46 Mops. For PIPE, the amount of time required (after the initial image) is three field times, i.e. .05 seconds. Consequently, a von Neumann type computer must operate at 49.2 Mops/second to keep up with PIPE in this application.

RELATED WORK

PUMPS (Briggs *et al.*, 1982) is an example of a multi-user system in which various task processing units are allocated from a pool. Each processor is specialized for a particular purpose, and images are transformed by passing them through a sequence of different processors. PIPE, on the other hand, consists of a sequence of identical stages, each of which has the power to perform several different operations on images. The programmer has the responsibility of specifying the task of each stage to ensure that the desired goal is attained. PIPE is also dedicated to a single user, although pipelines are easily constructed from a set of identical components, allowing each user to have a specially tailored PIPE system. In fact, a set of PIPE processors could be added to the pool of available processors in PUMPS, and used as a resource in the same way as the other processors.

Several other systems have components that perform some of the functions of PIPE. Usually, however, they operate on a single image at a time. For instance, the PICAP II system (Antonsson, *et al.*, 1982) has a filter processor, FIP, that performs some of the operations of a stage in PIPE. It also has other processors that are specialized for operations such as image segmentation. FLIP (Luetjen *et al.*, 1980) is similar to PIPE in that it has a number of identical processors, but it usually uses these processors in parallel on subimages of the same image instead of on successive versions of complete images. FLIP also allows greater flexibility in the connections between its processors. In PIPE, processors are normally connected only to their immediate predecessors and successors, although the wildcard busses allow selective but limited connections between arbitrary stages. FLIP, on the other hand, provides connections between all processors, allowing the processors to be arranged to suit each particular task.

Other special processors for image-processing include the massively parallel processor, MPP (Potter, 1983), and ZMOB (Kushner *et al.*, 1982), which is a more general parallel processor but has been studied extensively with regard to its abilities to perform image-processing tasks. MPP has 16K processors, and is a true parallel processor. Experience with the processor is limited, but a major difficulty appears to be the problem of transferring the data to each individual processor, and getting the results out of the machine. MPP does not have a true neighborhood operator, although each processor can be connected to four of its neighbors and use the pixel values there to compute its result. It is not clear that MPP has any advantage over pipelined systems, because images are usually obtained from an imaging system or storage medium in a stream, and sent to successive processors in the same fashion.

ZMOB consists of 256 processors connected by a ring-shaped high speed communications system. The communications link operates fast enough to make each processor appear to be connected to all others. Each processor is a general-purpose eight bit microcomputer, with 64K bytes of memory. Thus, many different computations can be performed at the same time, either on the same or different data. For image-processing applications, images are usually broken into parts, each of which is sent to a different processor. Many operations require interactions between the parts, especially when neighborhood operations are performed. This gives rise to the need for communications between processors. Given that the communications link is much faster than the processors' cycle time, there is very little overhead involved. But upgrading the processors might cause data transmissions to become significant. While PIPE is clearly less powerful than ZMOB, it is better suited to its role of low-level image processing.

A recent survey by Reeves (1984) divides image-processing tasks into two classes. Low level image-processing usually modifies parts of images, but maintains the image array. Higher level processing, however, works on symbolic representations of the contents of images. Low level processing has usually given rise to architectures based on single instruction stream, multiple data stream (SIMD) structures. The higher level functions are usually carried out using processors based on multiple instruction stream, multiple data stream (MIMD) structures. The design of PIPE allows it to act as a SIMD pipeline, or as a (restricted) MIMD pipeline. The MIMD mode is entered whenever the region-of-interest operators are used.

Cellular architectures have been used in several systems: CLIP4 (Duff, 1976), systolic arrays (Kung, 1982), and the cytocomputer (Sternberg, 1979). These architectures operate on images using simple, logical operations. As a result of this simplicity, many processors can be integrated onto a single chip.

In CLIP4, a processor is placed at each pixel location of the image. Currently, a 96 x 96 array of processors has been implemented. It performs neighborhood operations with the 8 neighbors of a pixel as well as point operations on the pixel itself. The output of each

processor can be stored in the processor, when it is a partial result, or it can be propagated to any of its neighbors. While CLIP4 can process 96 x 96 pixels in parallel, processing larger images would appear to incur large I/O delays. PIPE, with its pipelined structure, can perform the functions of CLIP4 as well as many others because its richer interconnection network permits both spatial and temporal pixel combinations. Furthermore, since the image is usually obtained from a camera, it is difficult to take advantage of the parallel nature of CLIP4.

Systolic arrays (Kung, 1982) balance computation speed with I/O limitations by interconnecting a set of cells, each capable of some simple operations, in a regular pattern. The salient characteristic of systolic arrays is that input data as well as (partial) result data flow through the system. In pipelined systems, such as PIPE, only results flow. In spite of this difference, the architectures have a great deal in common. Systolic arrays are built from generic building blocks for specific applications. In a similar fashion, the number of stages in PIPE is application dependent. Also, systolic arrays and PIPE share the concept of having very regular interconnection structures. In PIPE, however, the "wildcard" buffers offer greater interconnection flexibility.

A processor that has many features in common with PIPE is the cytocomputer (Sternberg, 1979). This machine performs neighborhood and table-lookup operations, but lacks most of the other features of PIPE. It does not have the "retrograde" or "recursive" data paths, has no region-of-interest operators, and cannot perform multi-resolution image-processing. Neither can it combine more than one image in an operation. Even without these features, however, the cytocomputer has shown itself to be extremely useful for low level image processing.

CONCLUSIONS

This paper has described a new image preprocessor, consisting of a sequence of identical stages, each of which can perform a number of point and neighborhood operations independently. An important feature of the processor is the provision of forward, "recursive", and "backward" paths to allow image data to participate in temporal as well as spatial neighborhood operations. The "backward" pathway also allows expectations or image models to be inserted into the system by the host. These expectations can then participate in the processing in the same way as images acquired from the input device. The region-of-interest operator is also a powerful, and unique, feature of PIPE, allowing the results of feature-extraction processes to guide further image analysis. PIPE also provides a multi-resolution capability, enabling global events to be made local. This is important in a machine that has only local operators. Much research needs to be done to explore the capabilities of the system, but early experiments indicate that the system will have a wide range of applications in low-level real-time image-processing.

REFERENCES

D. Antonsson, B. Gudmundsson, T. Hedblom, B. Kruse, A Linge, P. Lord, and T. Ohlsson, PICAP - A system approach to image processing. *IEEE Trans. Computers C-31 10,* October 1982, 997-1000.

F. A. Briggs, K. S. Fu, K. Hwang, and B. W. Wah, PUMPS architecture for pattern analysis and image database management. *IEEE Trans. Computers C-31 10,* October 1982, 969-983.

M. J. B. Duff, CLIP4: a large scale integrated circuit array parallel processor *Joint Conference on Pattern Recognition,* 1976, 728-732.

E. W. Kent, M. O. Shneier, and R. Lumia, PIPE - Pipelined image-processing Engine. *J. Parallel and Distributed Computing,* 1984 (in press).

H. T. Kung, Why Systolic Architectures?, *Computer 15, 1,* 1982, 37-46.

T. Kushner, A. Y. Wu, and A Rosenfeld. image-processing on ZMOB. *IEEE Trans. Computers C-31 10,* October 1982, 943-951.

K. Luetjen, P. Gemmar, and H. Ischen, FLIP: A flexible multiprocessor system for image-processing. *Proc. Fifth International Conference on Pattern Recognition,* Miami, Florida, 1980, 326-328.

D. Marr, Early processing of visual information. *Phil. Trans. Royal Society* B.275, 1976.

J. L. Potter, image-processing on the Massively Parallel Processor. *IEEE Computer Magazine 16 1,* January 1983, 62-67.

A. P. Reeves, Parallel computer architectures for image-processing. *Computer Vision, Graphics, and image-processing 25,* 1984, 68-88.

M. Shneier, Using pyramids to define local thresholds for blob detection. *IEEE Trans. PAMI-5 3,* May 1983, 345-349.

S. R. Sternberg, Parallel architectures for image-processing. *Proc. 3rd International IEEE COMPSAC,* Chicago, 1979, 712-717.

S. L. Tanimoto, Sorting, histogramming, and other statistical operations on a pyramid machine. In *Multiresolution image-processing and Analysis* (A. Rosenfeld, ed.), Springer-Verlag, Berlin, 1984.

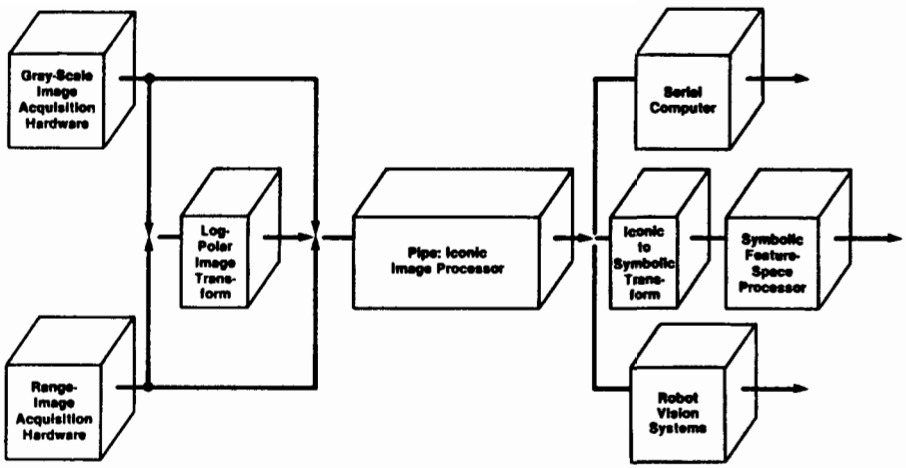

Fig. 1: PIPE and its relation to the other elements of the NBS Image Processing Configuration

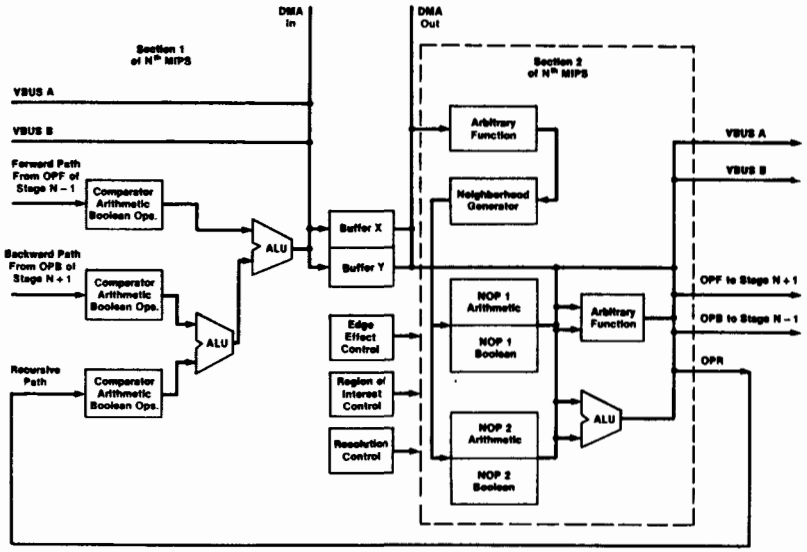

Fig. 3: Internal architecture of a processing stage

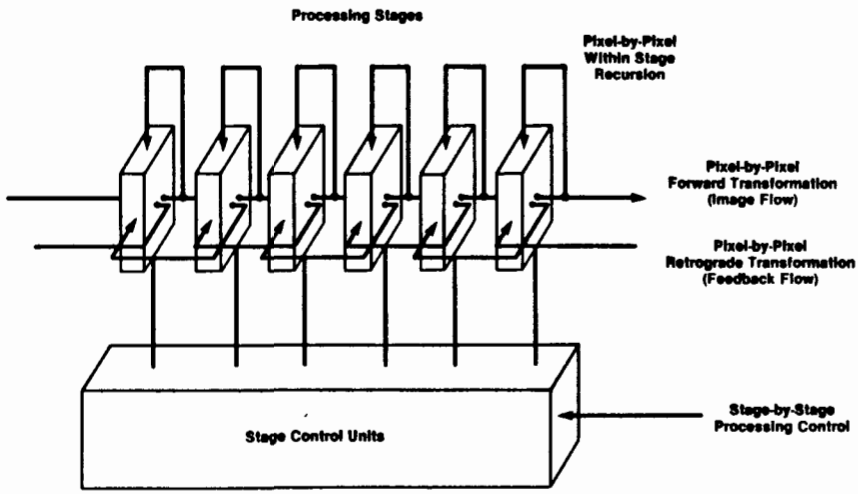

Fig. 2: Major connections between processing stages in PIPE

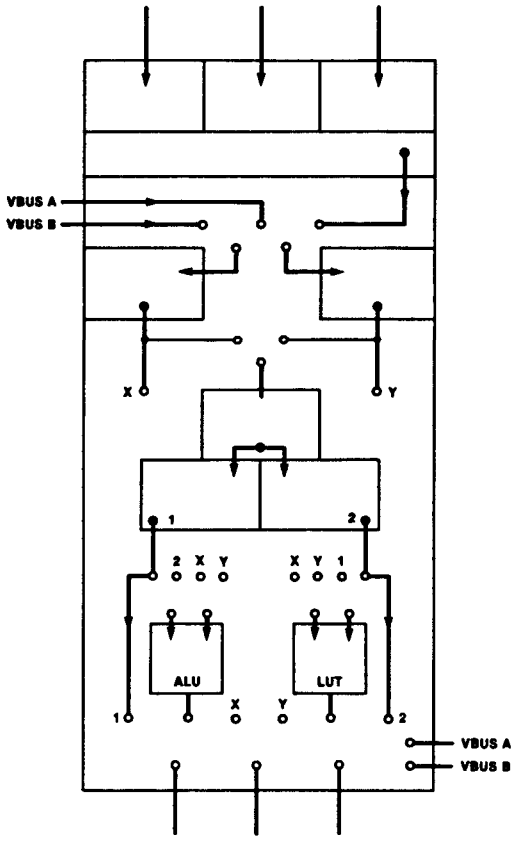

Fig. 4: Schematic representation of a processing stage

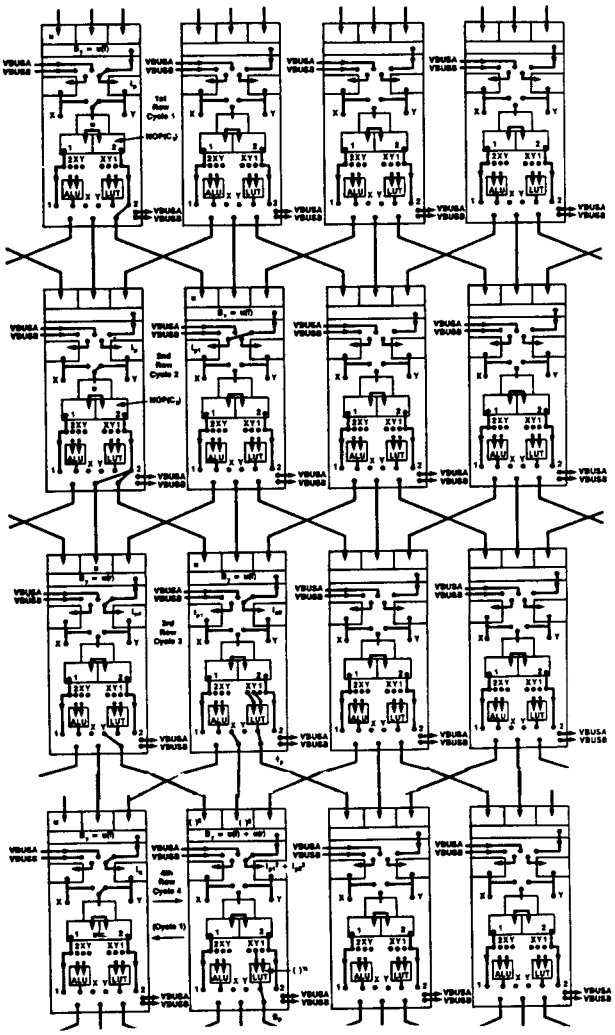

Fig. 5: Programming the Sobel edge operator in PIPE