JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 2, 50–78 (1985)

PIPE*

(Pipelined Image-Processing Engine)

ERNEST W. KENT, MICHAEL O. SHNEIER, AND RONALD LUMIA

Industrial Systems Division, National Bureau of Standards, Washington, D.C. 20234

The Sensory-Interactive Robotics Group of the National Bureau of Standards' Industrial Systems Division is designing and constructing an experimental multistage pipelined image-processing device for research in machine vision. The device can acquire images from a variety of sources, such as analog or digital television cameras, ranging devices, and conformal mapping arrays. It can process sequences of images in real time, through a serial pipeline of local operations, under the control of an external device. Its output can be presented to such devices as monitors, robot vision systems, iconic-to-symbolic mapping devices, and image-processing computers. In addition to a forward flow of images through successive stages of operations in the pipeline, other paths between the stages of the device permit recursive operations within a single stage, and feedback of the results of operations from a stage to the preceding stage. This architecture facilitates a variety of relaxation operations, interactions of images over time, and other interesting functions. Numerous operations are supported, including arithmetic and Boolean neighborhood operations on images within each stage, and between-stage operations on each pixel such as thresholding, Boolean and arithmetic operations, functional mappings, and a variety of functions for combining pixel data converging via the multiple image paths. The device can also be used to implement several alternative processing modes. Some operate within each stage, for example, to control edge effects or to implement "MIMD" operations specific to regions of interest defined by the host device. Others operate between stages, for example, to support variable-resolution pyramids.

1. INTRODUCTION

An image to be processed may be represented in a variety of ways. The most "natural" of these is to represent one or more intrinsic properties of the image (such as surface brightness, color, or range) in an ordered array whose cells correspond to the spatial locations of the image points. Such a map is often called an "iconic" image. An example of an iconic image is the representation of images on a television screen. An alternative image representa-

*The U.S. Government's right to retain a nonexclusive royalty-free license in and to the copyright covering this paper, for governmental purposes, is acknowledged.

0743-7315/85 $3.00

tion, often called a "symbolic" image, results when features of the image are represented by symbols which are stored in a linked list or similar data structure. A chain-coded representation of edges is an example of a symbolic representation.

Since an iconic image is spatially indexed, the whole image, or whole subregions of the image, must usually be processed during each operation, and massively parallel processing is required for real-time operation. Serial computer image-processing techniques typically attempt to reduce the image to a symbolic representation as rapidly as possible, to enhance the efficiency of serial processing. While this data compression brings many image-processing operations within the capabilities of ordinary serial computers, it makes many other operations more difficult, such as subtracting one image from another.

Parallel processors are ideally suited to the early stages of image processing, where local spatial and temporal features have not yet been discovered. They lend themselves to processing strategies based on multiresolution (pyramid) representations, and facilitate relaxation techniques for which a spatially ordered representation is most natural and efficient. Unfortunately, true multistage parallel processing is prohibitively expensive for images of useful size. The device described in this paper represents an attempt to achieve most of the advantages of a true multistage, fully parallel, iconic image processor through the use of very fast special-purpose hardware. The modular design of the hardware allows flexibility not only in the algorithms, but also in the organization of the device itself.

The Pipelined Image-Processing Engine (PIPE) is designed for use as a preprocessor to perform local neighborhood operations on iconic images. The processed images are intended for a host machine which will perform global operations and/or image understanding procedures, such as labeling, on the result. Thus, PIPE itself accepts iconic data images, and typically produces iconic images whose pixel values are Boolean vectors describing local properties of the pixel neighborhood. PIPE relieves the host of costly low-level local processing which must be performed over the entire image space. An auxiliary device (ISMAP) is also being constructed which will interface PIPE to a host by mapping the Boolean vector image from PIPE into histograms and other types of ordered list structures in the address space of the host.

PIPE consists of a sequence of identical processors (Fig. 1), sandwiched between a special input processor and a special output processor. The input processor accepts an image from any device that encodes two-dimensional images, such as a gray-scale camera, a range sensor, or other special hardware. It serves as a buffer between the rest of the processors and the outside world. Each successive processing stage receives image data in an identical format, operates on them, and passes them on to the next stage for further processing. This sequence is repeated every television field-time. When an image emerges at the far end of the sequence, it is processed by the special

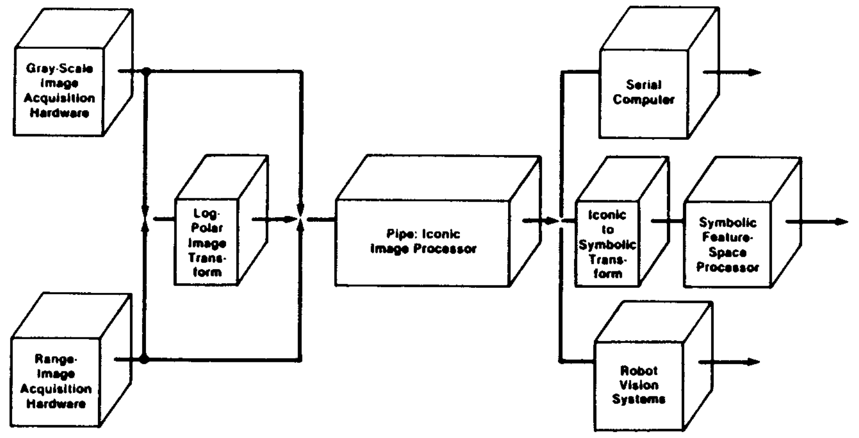
Gray-Scale Image Acquisition Hardware
Range-Image Acquisition Hardware
Log-Polar Image Transform
Pipe: Iconic Image Processor
Serial Computer
Iconic to Symbolic Transform
Symbolic Feature-Space Processor
Robot Vision Systems

FIG. 1. The PIPE processor and its relations to other elements of the NBS image-processing configuration.

output processor, and presented to some other device, such as a robot vision system or a serial computer.

The processors between the input and output stages are all identical and interchangeable, but can each perform different operations on the image sequences that they encounter. Usually, each processor receives three input images and transmits three output images. The input images arrive from the processing stage immediately behind each stage, from the processing stage immediately ahead, and from a result of the preceding operation just performed by the processor itself. Similarly, the results of processing its current image are transmitted by each stage to the next processing stage in the sequence, to the immediately preceding processing stage, and recursively back into the processor itself. These three outputs are not necessarily identical, and each may furnish part or all of the inputs to the other processors for the subsequent step in processing. The three inputs may be weighted and combined in each processor, in any fashion, before they are processed. In addition to the usual input and output paths, four "wildcard" paths are provided for both input and output. These paths are common to all stages, so that only one stage can write to a particular wildcard path at a time. The wildcard paths allow image frames to be moved arbitrarily between stages, instead of having to step through from stage to stage. There are no restrictions on the number of destinations for a frame output to a wildcard path.

Two main kinds of processing may be carried out in each processing stage. The first involves simple pointwise arithmetic or Boolean operations, on a pixel-by-pixel basis. The second is a neighborhood operation, which is performed in a pipelined fashion. (Notice that PIPE contains two kinds of pipelines. The first is a pipeline of images that proceed from processor to

processor, while the second is a pipeline of neighborhoods within each processor.)

There are numerous reasons for requiring the three input and output paths from each processor. It is clear that the forward path allows a chain of operations to be performed, giving rise in real time to a transformed image (with a constant delay). Similarly, the recursive path allows a pipeline of arbitrary length to be simulated by each stage, and also facilitates the use of algorithms that perform many iterations before converging to a desired result (e.g., relaxation algorithms, or the simulation of large neighborhood operators by successive applications of smaller neighborhood operators). The path to the preceding processor allows operations using temporal as well as spatial neighborhoods to be performed. It also allows information inserted at the output stage by the host to participate in the processing directly. This, for example, allows expectations or image models to be used to guide the processing at all levels, on a pixel-by-pixel basis.

The architecture of the device is described in the next section. Later sections describe ways of using PIPE for various image-processing operations, and expand on the reasons for the various image pathways and the processing capabilities built into each processing stage. Interactions with the outside world are also discussed.

2. ARCHITECTURE

The organization of the image-processing section of PIPE, excluding the input and output processors, is shown in Fig. 2. PIPE is composed of a variable number of identical, modular, image-processing stages. Every stage

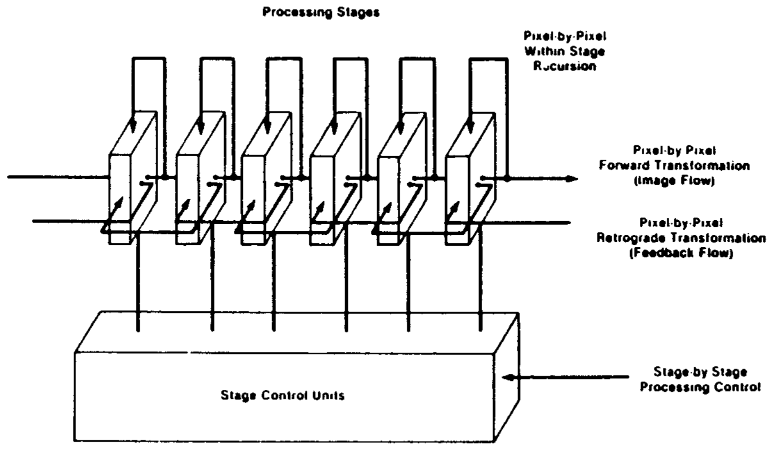
Processing Stages
Pixel-by-Pixel Within Stage Recursion
Pixel-by Pixel Forward Transformation (Image Flow)
Pixel-by-Pixel Retrograde Transformation (Feedback Flow)
Stage Control Units
Stage-by Stage Processing Control

FIG. 2. Major connections between processing stages in PIPE.

contains two field buffers, each of which holds a processed version of the image from a single field of data. During each field-time, each stage operates on one member of a set of consecutive image fields. The stages contain fast special-purpose logic that processes the contents of the buffers and carries out interstage interactions in a single field-time (16.67 msec). Each stage has an associated stage-control unit that can redefine the process to be performed by the stage and change its parameters during the interfield interval. The stages are connected by three distinct data paths, which are shown in Fig. 2.

PIPE's main processing tool is a neighborhood operator. This may comprise either an arithmetic convolution operator or a set of arbitrary Boolean operators. Additional operations are also possible; they are discussed in more detail below. The neighborhood operations (either arithmetic or Boolean) are performed on a neighborhood of each pixel in a stage buffer. Two such operations may be performed independently in a single field-time, on the neighborhood of every pixel in the field. The output of these operators can be sent to any of the paths out of the stage.

The image resulting from applying one of these neighborhood operators is carried forward into the subsequent stage (perhaps after undergoing other associated transformations). Similar processing occurs in all stages simultaneously, so that the system forms an image pipeline into which new images are accepted at field rate. This image-flow processing follows the path indicated in Fig. 2 as "pixel-by-pixel forward transformation." At every processing stage, different processes may be applied to the image. Interactions between stages, detailed below, extend the processing to the "temporal neighborhood" of the pixel, permitting time-domain operations on the scene. These are useful, for example, in the analysis of motion.

A second "backward" data flow path is supported by each stage. This path, indicated in Fig. 2 as "pixel-by-pixel retrograde transformation feedback flow," brings the output of one of the neighborhood operators of each stage back to combine with the image currently entering the preceding stage. The source of the data for this second, "retrograde" pathway may be the same as for the forward transformation, or may be the second neighborhood operator applied to each pixel neighborhood at the same time that the forward transformation operator is applied. Thus, the retrograde transformation may be independent of the forward transformation, or be identical with it. The results of the retrograde operation from one stage are carried into the preceding stage, permitting interaction forward in time (i.e., interaction with subsequent images; notice that the forward direction with respect to the pipeline stages corresponds to images that arrived earlier in time). This permits feedback loops to be formed in the image-flow processing.

Neighborhood operators can be used for a wide variety of image-processing tasks (e.g., averaging and noise reduction operations, edge, line, and point labeling operators, region growing, region shrinking, and finding (non-) minima and (non-) maxima). Some of these functions permit or require

repetitive recursive operations. That is, they require that the image resulting from one application of the operator be the input for a subsequent application of the same operator. This implies that the stage's field buffer must be able to be loaded from the output of its own forward or backward transformation operators. The alternative would be to accomplish recursive operations by cascading the image through multiple identical operations in sequential stages, which could require an arbitrary number of stages. In Fig. 2 the recursive path provided by PIPE is labeled "pixel-by-pixel within-stage recursion." It is shown here originating from the forward pathway, but it may optionally arise from the backward pathway.

The data actually stored in each stage are generated by logic that operates on, and performs various combinations of, the inputs from all three incoming pathways. Feedback values and recursion values may be combined with the ascending image value in any proportion, summed or differenced with it (with or without constant offsets), or combined by any Boolean operation. A schematic representation of the relationships between the forward and retrograde transformation operators and the spatial neighborhood of a single pixel is shown in Fig. 3. In this and other figures, the recursive pathway is shown originating from the forward transform unless otherwise specified, but origin from the retrograde transform is an option in all cases.

If the preceding and succeeding fields are considered to contain future and past instances of a field, respectively (as is true in a dynamic image), then

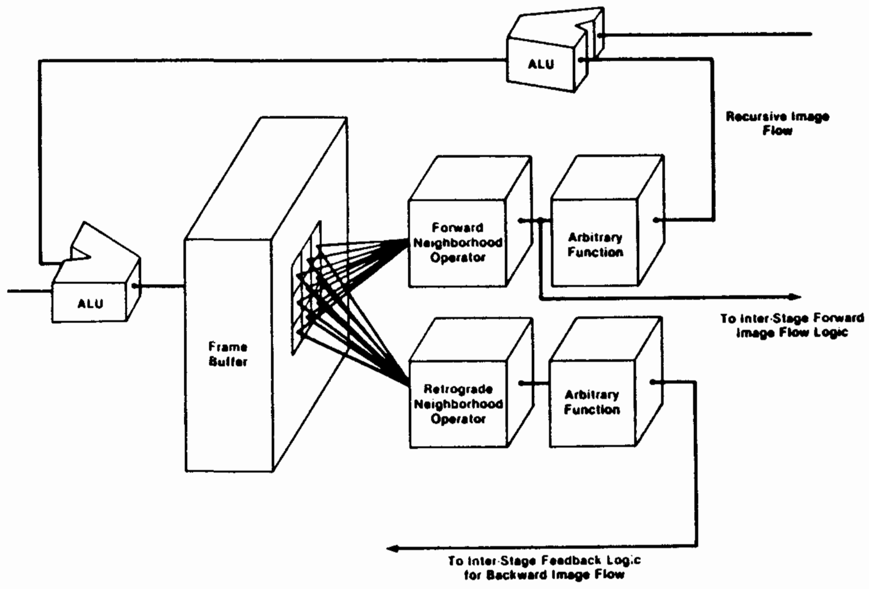
ALU
Recursive Image Flow
Forward Neighborhood Operator
Arbitrary Function
To Inter-Stage Forward Image Flow Logic
ALU
Frame Buffer
Retrograde Neighborhood Operator
Arbitrary Function
To Inter-Stage Feedback Logic for Backward Image Flow

FIG. 3. Generation of image-flow paths from simultaneous application of independent neighborhood operators.

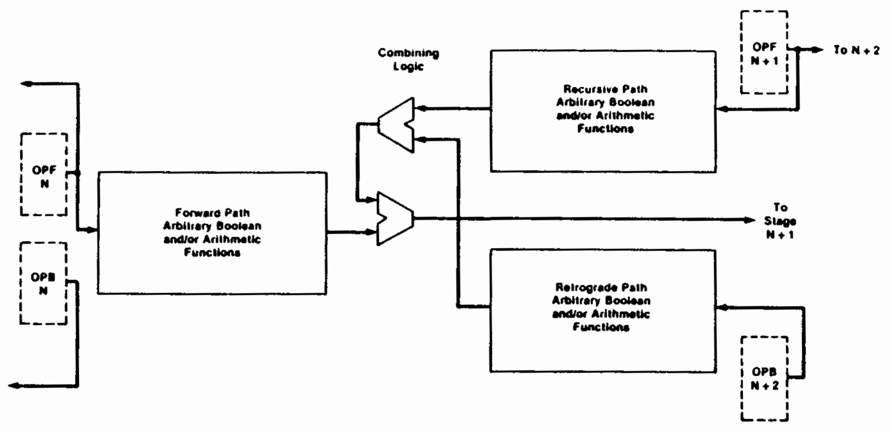
Combining Logic
OPF N
OPB N
Forward Path Arbitrary Boolean and/or Arithmetic Functions
Recursive Path Arbitrary Boolean and/or Arithmetic Functions
Retrograde Path Arbitrary Boolean and/or Arithmetic Functions
OPF N + 1
To N + 2
To Stage N + 1
OPB N + 2

FIG. 4. The interstage combining logic.

forward transformation corresponds to a path from the future, recursion to a path from the present, and retrograde transformation to one from the past. The weighted sum of the three paths may be set up as a convolution operation on the temporal neighborhood of a pixel. This may occur at the same time as the convolution operation is being performed on its contemporary spatial neighborhood (i.e., eight spatial neighbors and two temporal neighbors). Combined uses of the retrograde pathway to implement both feedback loops and temporal convolutions can also be envisioned; their utility is a matter for exploration.

Figure 4 shows some details of the interstage combining logic to clarify the interaction of the pathways in the temporal domain. The outputs of the forward neighborhood operations (OPF) and the retrograde operators (OPB) are shown for stages $N, N + 1$, and $N + 2$, together with the combining logic linking them. At every PIPE stage, data from any of these pathways may be subjected to a comparison operation (e.g., thresholding) to transform arithmetic data to Boolean data. The arbitrary Boolean and/or arithmetic functions shown in this figure represent a versatile set of possible operations (including arithmetic-to-Boolean conversion) that may be applied to each data stream prior to combination by table lookup.

It is helpful in understanding the functions of these processing pathways to consider each in isolation first. If only the forward transformation path is operative (i.e., the weights for the retrograde and recursion operators are set to zero), we have a simple image pipeline processor which can sequentially apply a variety of neighborhood operators to the series of images flowing through it. It can perform either arithmetic or Boolean neighborhood operations and, by thresholding, convert an arithmetic image into a Boolean image. For example, it might be used to smooth an arithmetic gray-scale image, apply edge detection operators to it, threshold the "edginess" value to

form a binary edge image, and then apply Boolean neighborhood operations to find features in the edges. The operation types and parametric values for these operations would be set individually for each stage by the stage control units, which in turn would be instructed (for example, from the host) via the input marked "stage-by-stage processing control" in Fig. 2.

A second single-path case results if both the forward and retrograde paths' combining functions are zero. Assume that images had previously been loaded into the processing stages. The recursion path would then cause the image field in each stage to pass through the forward or backward transformation operation recursively, while the images "marched in place." A variety of relaxation operations can be implemented in this way.

For the final single-path case, consider what happens when the weights assigned to the forward and recursive paths are zero, leaving only the retrograde pathway active. When the set of such paths is considered in isolation, it becomes clear that it forms a processing chain that is a retrograde counterpart of the forward pipeline. It would, in fact be possible to select appropriate retrograde transformations, insert fields of data at the back of the device, process them through to the front, and get the same result as running the system in the normal direction. The purpose of this is not to provide a bidirectional image processor, but to permit input (at the "output" end of the device) of synthesized images. Such images influence the processing of the normally flowing images by direct interaction, and correspond to "expectancies," "models," "hypotheses," or "attention functions."

The retrograde images are not only able to affect processing of the forward images, but are affected themselves by interaction with them. (The effects that the two image sequences exert on each other may be different because the neighborhood operators on the forward and backward paths are independent.) Retrograde images will usually be generated by knowledge-based processes in higher-level components of the host system. They may initially appear in Boolean form, but, as shown in Fig. 5, provision is made for all four possible combinations of arithmetic and Boolean inputs and outputs in the combining logic between stages. This permits a descending Boolean image to be instantiated into arithmetic image values by interaction with the ascending arithmetic image. This occurs in the same stage in which the ascending arithmetic image representation is thresholded to become a Boolean image. Both the ascending data image and the descending "hypothesis" image can pass across this interface. A major function of PIPE will be to explore the effectiveness of various approaches to hypothesis-guided iconic image processing.

Boolean information can be processed in an interesting way by combining the outputs of the forward operator from the previous stage and the recursive input from the current stage. Consider the case of a single stage treated in this fashion for eight field-times, using "SHIFT then OR" as the combining operation. If the incoming images from the previous stage have Boolean

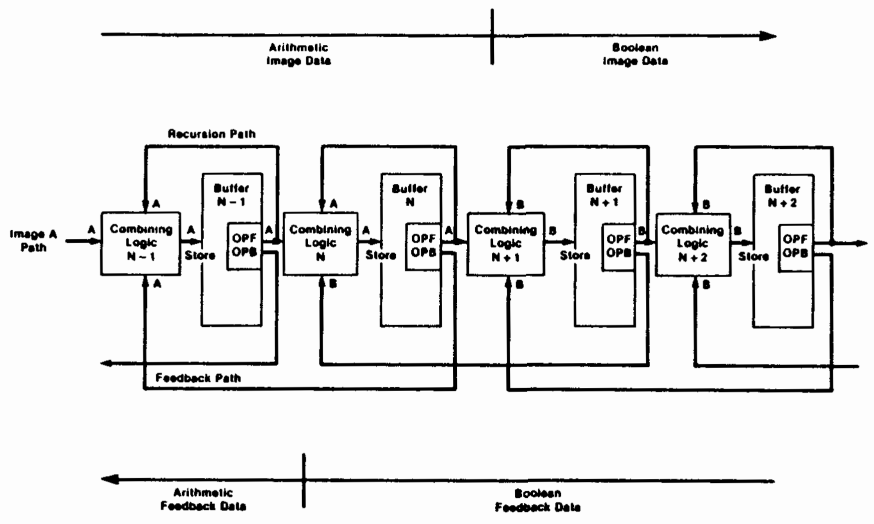

FIG. 5. Interactions between arithmetic and Boolean images.

values resulting from successive independent operations and comparisons, such a stage will accumulate images from the eight preceding Boolean operations into an image composed of 8-bit Boolean vectors. Subsequent Boolean neighborhood operations may apply independent operators to each bit plane of a neighborhood of such vectors.

PIPE is not a simple parallel image pipeline. Each stage in the pipeline of images contains its own, internal, pipeline which is used to perform the neighborhood operations. That is, the operations are not applied to every pixel neighborhood of each stage simultaneously, but are performed sequentially, raster scan fashion, over the stage within one field-time. Of course, the stages all operate in parallel, so that the whole pipeline of images is processed in a single field-time. The sequential nature of the within-stage processing, together with the existence of the recursive data path, could pose problems in performing the neighborhood computations. If each neighborhood operation were to be computed using values taken directly from the image, then those points above and to the left of the central pixel in the neighborhood would already have been processed, and perhaps altered, by previous pipelined operations. To avoid such problems, the pixel neighborhoods being processed in each field are read/write shifted so that the incoming pixels from the pipeline, which are continuously updating a field belonging to a later image epoch, do not appear in the neighborhoods of pixels being processed in the current image epoch. Between-field read/write address differences simulate time delays to compensate for this staggering and thus ensure that homologous pixels from each field are received by the combining logic. The manner in which this staggering is related to the interactions of the

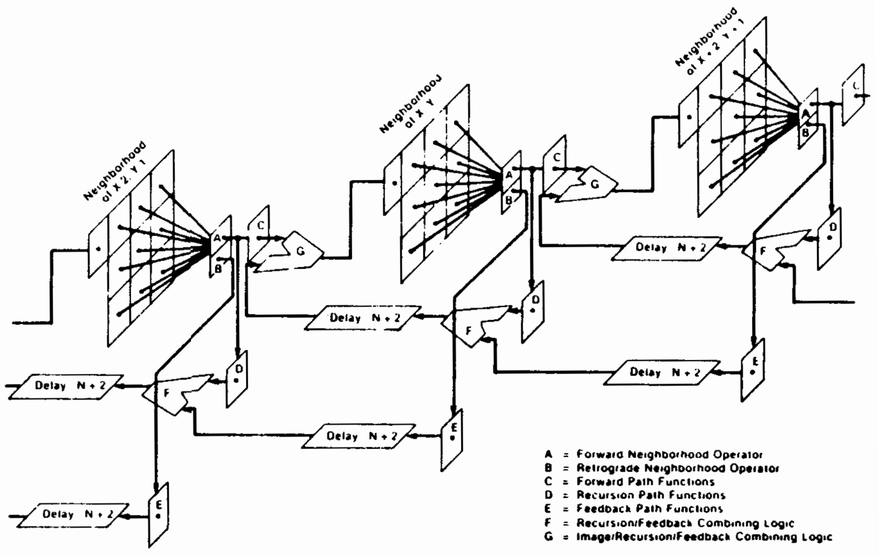
Neighborhood of X-2, Y-1
Neighborhood of X, Y
Neighborhood of X+2, Y+1
Delay N + 2
A = Forward Neighborhood Operator
B = Retrograde Neighborhood Operator
C = Forward Path Functions
D = Recursion Path Functions
E = Feedback Path Functions
F = Recursion/Feedback Combining Logic
G = Image/Recursion/Feedback Combining Logic

FIG. 6. Staggering of read/write operations between stages.

various pathways is shown in Fig. 6. The required parameters of the simulated delays are illustrated in the figure as actual delay lines for clarity, although there are no such physical delay elements in the machine.

PIPE has a variety of features and operating modes in addition to the neighborhood operations discussed above. Its input stage has the ability to fill one field buffer with the difference between the contents of either buffer and the incoming image. In this fashion, it can force PIPE to process only those portions of the image which change from field to field.

Another option allows PIPE to accept definitions of "regions of interest" in an image, and to cause any stage to apply complete alternative operation sets within each of its regions of interest. Regions of interest for an image buffer are specified via a bit map resident in the other image buffer of a stage. In this fashion PIPE can act as an MIMD machine, in that different operations can be performed over different portions of the data stream on a pixel-by-pixel basis. This feature may be used to generate globally nonlinear image-processing operations, as, for example, in preserving edges while smoothing non-edge portions of an image. Up to 256 alternative operation sets are specifiable in principle by the bit-mapping procedure. The actual number available will depend on the amount of memory attached to each stage. In the prototype device, sixteen will be available. The nature of the available alternative operation sets, like all other aspects of stage operations, may be changed between fields.

A further mode of operation allows multiresolution image processing. For example, PIPE can use its forward image-processing path to reduce an image into successive half-resolution representations, and its retrograde path to construct successive double-resolution representations, thus implementing a form of multiresolution pyramid. Processing within any level of such a pyramid can be accomplished through use of the recursive pathway, while interactions between levels of the pyramid are accomplished through the forward and retrograde image paths. In interlevel interactions, the mapping of pixels into higher- or lower-resolution images occurs automatically. Multiresolution image processing using PIPE is discussed further below. Another useful feature of PIPE is the provision of two "wildcard" buses, which allow images to be transferred from a stage to any or all other stages in a single field-time. These buses allow quick access to, and dissemination of, intermediate results. They also make it possible to connect the stages into a ringlike structure, or to bring synthetic images from the host to any stage.

3. Functional Details of PIPE Stages

3.1. *Input Stage*

A special input stage is used to capture images from input devices. This allows PIPE to accept digital or analog signals from any device using standard RS-170 television signals and timing. Analog signals are digitized by an 8-bit real-time digitizer. The input stage is capable of acquiring a digitized image of 256 × 240 pixels while remaining synchronized with RS-170 signals. Alternatively, it can capture 256 × 256-pixel images from non-RS-170 signals while internally employing nonstandard pixel rates. It can continually capture such images at standard television field rates, and place them in either of the two field buffers contained in the input stage. While storing an image into one of these buffers the input stage can also simultaneously store an image, such as a difference image, formed by an ALU operation between the incoming image and a previously captured image, into either buffer. The contents of either of the buffers in the input stage can be sent to the first of the processing stages, while the next image is being acquired.

PIPE accepts 8-bit input data, and this precision is maintained throughout the machine. Intermediate arithmetic operations within subsequent stages are carried to sufficient precision to ensure no loss of accuracy when the result is rounded to 8 bits for transmission to subsequent stages. The data may be treated as either unsigned 8-bit numbers, or as signed numbers with the high bit indicating the sign. Unsigned input data may be processed as such, until an operation which generates negative values occurs, and treated as signed data thereafter.

3.2. *Processing Stages*

The first processing stage is one of a series of modular intermediate processing stages (MPS). The MPSs are the "stages" described in the preceding

sections, and are the elements which perform most of PIPE's processing. All MPSs are of identical modular construction, and are physically interchangeable simply by switching card edge plugs and circuit boards. Thus, any MPS can operate at any position in the processing chain, and the processing chain can have a variable length. Eight MPSs are planned for the present development phase of PIPE.

The *N*th MPS accepts three 8-bit 256 × 256-pixel images as input. These come from the forward output of the $(N - 1)$st MPS, from the recursive output of the operation performed on the previous contents of the *N*th MPS, and from the retrograde output of the $(N + 1)$st MPS. Each data stream may consist, independently of the other two, of arithmetic or Boolean (8-bit Boolean vector) data, but a given data stream entering a MPS must be entirely Boolean or arithmetic within any single image field.

Before generating a final 8-bit image from the three data streams, each MPS performs comparison, Boolean, and/or arithmetic operations on each of them independently and simultaneously, according to the type of data present. If an input stream contains arithmetic data, either comparison or arithmetic operations are possible by table lookup. The comparison (conversion) operation may thus be a multiple-window comparison, which converts an arithmetic pixel to a Boolean vector, with the bits of the Boolean vector independently specifiable. The arithmetic operation can consist of any function of a single argument. If an input stream contains Boolean data, the Boolean operation can perform functions such as 0- to 7-place barrel shift, and apply AND (NAND), OR (NOR), and EXOR operations to the result.

The resulting three Boolean and/or arithmetic data streams are then combined through independently-programmable full-function ALUs into a single arithmetic or Boolean data stream. This data stream (or when enabled, a DMA data stream provided by an external device) is then used to load either of the two selectable field buffers within the MPS. Alternatively, either or both buffers can be filled using the wildcard buses. The contents of both of these field buffers are then available to subsequent operations of the MPS. External device access to these buffers is also available; an external device may read from or write into either buffer in a random access manner at 400,000 pixels/sec, with autoindexed addressing supported on command. The wildcard buses provide streaming access to external devices (including monitors) at pixel rates.

The hardware that implements those MPS functions subsequent to the field buffer storage step is physically contained on a separate circuit card to allow it to be replaced with other special-functional modules, should this be desirable. This circuitry is represented by the area labeled "Section 2" in Fig. 7. In operation, an 8-bit image is first selected by reading the contents of one of the two field buffers in the MPS. The image is transformed by an arbitrary, programmable, single-valued mapping function, and the pixels of the resulting image are subjected to two neighborhood operators, of which there are

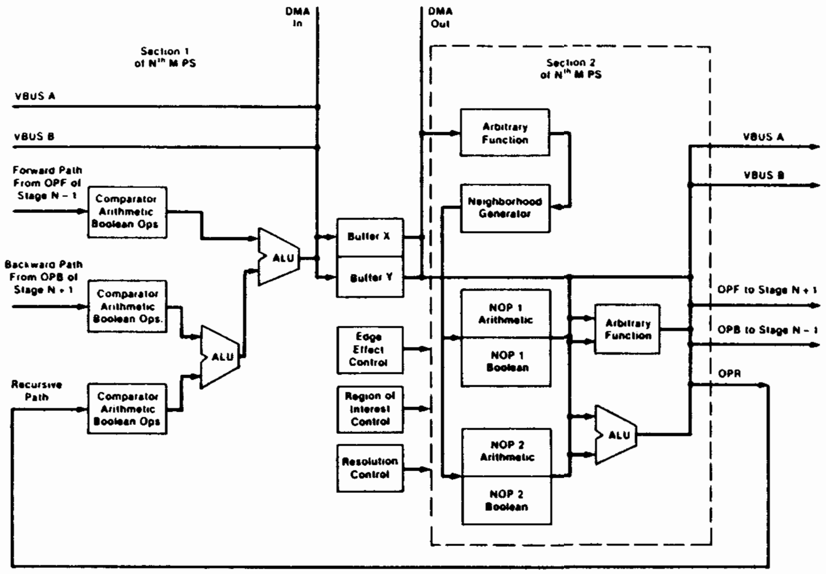

FIG. 7. The internal architecture of a processing stage.

two kinds. The first type of neighborhood operator is an arithmetic convolution operation, while the second is a Boolean operation. For either operator, the neighborhood of operation is (at present) 3 × 3 pixels square, and the operation is accomplished in 200 nsec. Pixel neighborhoods are generated by passing the data stream through a three-line buffer.

In the arithmetic case, the convolution operation uses arbitrary positive or negative 8-bit neighborhood weights, and maintains 12-bit accuracy in its intermediate results. The final 8-bit arithmetic result entering the buses is produced by nonbiased rounding, from a 20-bit sum. This ensures that no loss of precision occurs within a stage due to arithmetic underflow or overflow. The full 8-bit precision of the input is thus maintained between stages throughout the machine. In the Boolean case, the neighborhood operation consists of arbitrary Boolean operations (a sum-of-products AND–OR array equivalent) between the set of all the pixels of the data neighborhood, and the set of all corresponding pixels of an arbitrarily specified comparison neighborhood. Any bit of either neighborhood may be independently defined as true, false (complemented), or "don't care." Each of the eight bit-planes forms an independent set of inputs, subject to independent neighborhood operations. As a result, eight independent 1-bit results are obtained from a single pass of the data through the pipeline, yielding an orthogonal 8-bit Boolean vector as output.

Two neighborhood operators (NOP1 and NOP2) are applied independently and simultaneously. They operate on the same data stream, using neigh-

borhood operations which may be different. Their outputs may be independently subjected to a second transformation by either of two programmable functions. The first of these is a look-up table mapping function. This transformation may be a function of two arguments. If two arguments are used, one is taken from the homologous pixel of the field buffer which is not the source of the image being subjected to the neighborhood operators. The number of bits used from the two arguments must total 12; for example, the 6 most significant bits of each. If only one argument is used, all 8 bits are available, the remaining bits being interpreted as "don't care." The other function is an ALU with two 8-bit inputs operating on the same sources. The data stream arising from either of these operations applied to one of NOP1 or NOP2 (a result denoted by OP1 and OP2, respectively) is selected to become the forward output (OPF) of the MPS, and the data stream arising from the operation applied to either the same or the other of NOP1 and NOP2 becomes the backward output (OPB) of the MPS.

A "region-of-interest" operator allows each MPS to switch between the normal (OPF, OPB) operation set and alternative (OPF′, OPB′) 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 8-bit contents of each pixel in the map. In practice, the number of alternative operation sets selectable during a field processing time will be 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.

PIPE allows the construction of multiresolution, "pyramid," sequences of images. Pyramids have been found useful in a large number of image-processing applications. They have an added utility in a strictly local processor like PIPE because they allow information from spatially distant regions 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. A further option allows sampling to occur in staggered pixels in alternate rows, to generate samples with uniform (square root of two) neighbor distances. An example is presented below to illustrate the utility of pyramid operations in conjuction with the other operations in PIPE.

Both the sampling and doubling operations are performed by manipulating address lines within a stage. The places in the stage at which the operations are performed are different because of timing considerations. Sampling is achieved by incrementing the source image addresses twice as fast as the destination addresses. That is, on each row, the first pixel in the source image is written to the first pixel in the destination image. The second source pixel is also written to the first destination pixel, overwriting the previous value. The address of the destination pixel is then incremented, and the procedure is repeated. The same process is used to overwrite every other row in the destination image. The result is that the destination image is one-quarter the size of the source image, and occupies the upper left quadrant of the image buffer. Each pixel in the destination image is written out four times, to result in the reduced-size image. This is not wasteful because the source image is being read at field rates and the new image is created in a single field-time.

Doubling is accomplished by the inverse of the sampling process. That is, the addresses in the source image are now incremented at half the rate of those in the destination image. For each row in the source (reduced-resolution) image, two identical rows are output in the enlarged image. For each pixel in each row of the input image, two identical pixels are stored in the output image. This results in an enlarged image that has four times as many pixels as the input image.

The simple operations of image sampling and pixel doubling are not of themselves very useful except for a narrow range of applications. Combined with the other operations in PIPE, however, a much broader class of operations becomes possible. The sampling process occurs as the image enters a stage buffer. This means that a number of operations can be performed on the source image prior to constructing the reduced-resolution version. Of these, perhaps the most useful is the neighborhood operator, which can be used, for example, to smooth the image before sampling. By iterating the neighborhood operation prior to sampling, the effects of neighborhoods larger than 3×3 can be obtained, allowing, for example, the construction of "Gaussian" pyramids using the hierarchical discrete correlation procedure of Burt [3].

In the inverse situation, when the image is magnified, the doubling occurs as the image leaves a stage buffer. Once again, operations can be performed on the doubled image before it is stored into another stage buffer. In this situation, however, the 3×3 neighborhood is not as valuable as in the sampling case. This is because the "field of view" of the operator does not encompass all the neighbors of a pixel (the pixels have been enlarged to 2×2 blocks). To include all the neighbors, at least two iterations of a neighborhood operation must be applied to the image. This means that expanding a pyramid may take twice as long as compressing it. (By replacing the 3×3 operator with a larger one, this asymmetry can be overcome.)

An important issue in dealing with images of varying sizes is how to overcome edge effects that arise when the neighborhood operator is applied. This issue is dealt with in the same manner for all sizes of images. It is the programmer's responsibility to ensure that each stage knows the size of the images in each of its buffers. In principle, it would be possible to make all border pixels belong to a region of interest. Special neighborhood operators could then be applied there to overcome the edge effects. PIPE provides as

a default solution the replication of border pixels. If a neighborhood has a row or a column that lies outside the boundaries of the image (either beyond the image buffer itself or beyond the extent of a low-resolution image), the nonexistent pixels are replaced by the pixels in the border row or column. For a 3×3 neighborhood, this is equivalent both to reflecting the image and to repeating the border pixels. This is achieved in the same way as the varying resolution images are constructed, i.e., by manipulating the address lines of the buffer.

3.3. *Output Stage*

The output stage fulfills a role at the end of the processing chain similar to that of the input stage at its beginning. The final MPS delivers its forward image output (OPF) to either one of a pair of field buffers in the output stage, and can simultaneously read from the other buffer of the output stage. The data read from the output stage are used as the input to the retrograde (feedback) path of the final MPS. Without interrupting the image processing, either buffer of the output stage can be read from or written into by an external device, which is both the consumer of the processed forward data flow and the supplier of data for the retrograde path.

4. Using PIPE

The hardware of the pipeline may be set up for the task of interest by inserting the number of stages required by the application. A pipeline consists of a special input stage, for capturing images; a variable number of stages for processing the images; and an output stage, for delivering the processed data to a controlling device. Setting up the pipeline involves adding or removing standard cards. Except for the input and output stages, these cards are all interchangeable.

4.1. *Stage Control*

Once the pipeline has been set up, the individual operations to be carried out must be chosen, and the host device must load each stage with appropriate instructions for processing successive fields. Each stage has a stage controller that is loaded by the host device. Interfaces between the control units and host devices are 16-bit input and output ports. Each stage-control unit can completely reconfigure the operations and operating parameters of its associated stage on the basis of current or stored instructions from the host device. Changes are made in the interval between image fields.

The stage-control units store and select multiple alternative configurations for their stages. The sequencing orders of these configurations are provided by the host device. Programming PIPE thus consists of specifying to the control units the operations and operation sequences to be performed by each stage, and loading the corresponding operators, parameters, and tables into the stages. In operation, the host device may instruct the stage-control units to select a stage program, instruct it to branch in the specified sequence of operations, or permit it to follow the preset sequence of operations (which may contain branch points on repetition counts). For program development, the contents of any buffer and the output of any operator in the system can be displayed on a video monitor, with or without freezing the contents of the buffer, and the whole processor can be single stepped.

4.2. *Program Examples*

The following examples make use of a schematic MPS representation employed as labeled in Fig. 8. The diagram has four distinct sections, which

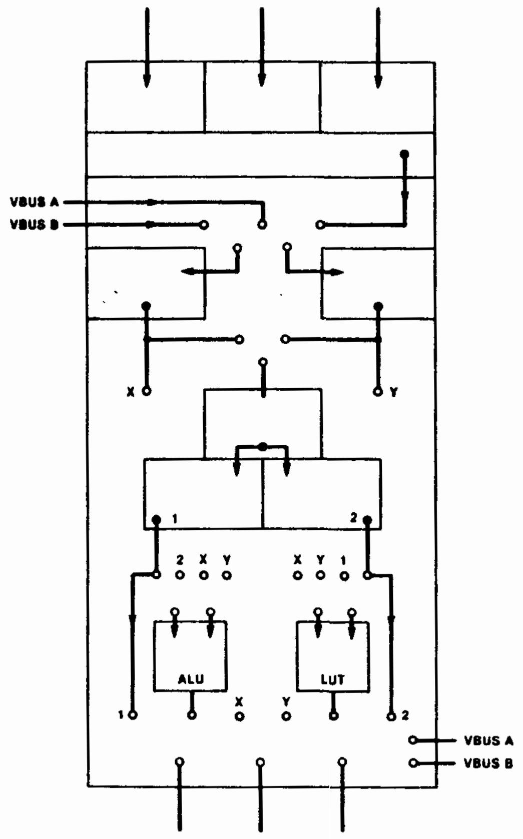

FIG. 8. A schematic MPS representation for use in program development.

are connected by switching networks. At the top of the figure, there is an input section, and below this are the two image buffers. Next come the neighborhood operations, followed by the output processing. The final switching network routes the outputs to the wildcard buses and to the three output paths.

The three boxes across the top represent the functions of one argument (lookup tables) to be applied to the forward, recursive, and backward input pathways, in that order, from left to right. Whatever functional transformation is employed during a given cycle will be shown in the appropriate box. Below these is a single box representing the combining function to be applied during the cycle to the results of applying the above three functions. The output functional variable here will be represented as A, or B, according to its destination (in image buffer X or image buffer Y of the stage). The input variables will be represented as *f*, *r*, or *b*, according to the pathway of origin.

Next there is a crosspoint matrix switching the inputs between the X and Y image buffers. There are three possible inputs and two possible outputs. Two of the inputs arise from the wildcard buses (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. The X buffer is on the left, and the Y buffer is on the right. As an image is stored into one of the image buffers, it may undergo a sampling operation, reducing its resolution by a factor of 2.

Below the image buffers is another switch that selects the buffer that will serve as the input to the neighborhood operators. Only one image buffer can be used as the source for the neighborhood operator. 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.

As an image is removed from a buffer, it may undergo a pixel-doubling operation, increasing the size of the image. Note that the output to the neighborhood operator is the same as that bypassing the operator. If one is to be expanded, then both must be expanded.

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.

Following the neighborhood processing, another switching matrix selects the inputs to the 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 the 12-bit lookup table. The input is 12 bits selected from any two 8-bit inputs in a manner chosen by the programmer. The selection will be marked on the line joining the selected terminals of the switch to the lookup table. By choosing 8 bits from one argument and setting the remaining 4 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 our programming examples, a blank box will represent an inactive operation. A simple passthrough 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.

Two examples are presented below using this schematization. In these examples, data-flow charts are constructed with stages proceeding from left to right in order of spatial data flow through the machine. That is, in any given row of the chart, the Nth stage is to the right of the $(N - 1)$st stage, and to the left of the $(N + 1)$st stage.

Data flow in time is represented in the vertical direction. Thus, each row represents a snapshot of the state of the machine during a given cycle. Each component of the row represents a single MPS. Each column represents the evolution of successive states of a single MPS stage over time. It follows that data must always enter a stage from the row above, and exit to the row below. Data which are stationary in space (i.e., follow the recursive path only) will proceed vertically down a column. Data moving forward through successive stages will flow through the chart diagonally down and to the right, while data moving backward through successive stages will move diagonally down and to the left. In general, there will be multiple ways of programming a given operation in PIPE. Some of these may be space intensive, while others may be time intensive. Trade-offs may be made between space- and time-intensive use of PIPE's resources according to the demands of the task.

4.3. *Example 1: The Sobel Edge Operator*

PIPE can be used for any image-processing task which can be performed using a 3 × 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 $\theta(r, c)$ for each row (r) and

column (c) pixel in the image. Formally,

$$\begin{aligned} I_1(r, c) &= I(r, c) * C_1 \\ I_2(r, c) &= I(r, c) * C_2 \\ G(r, c) &= \operatorname{sqrt}(I_1^2 + I_2^2) \\ \theta(r, c) &= \operatorname{atan}(I_2, I_1) \end{aligned}$$

where

$$C_1 = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \quad \text{and} \quad C_2 = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

and * denotes the convolution operator.

The program is shown in Fig. 9 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_{p_1}, 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_{p_2}, 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 atan (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

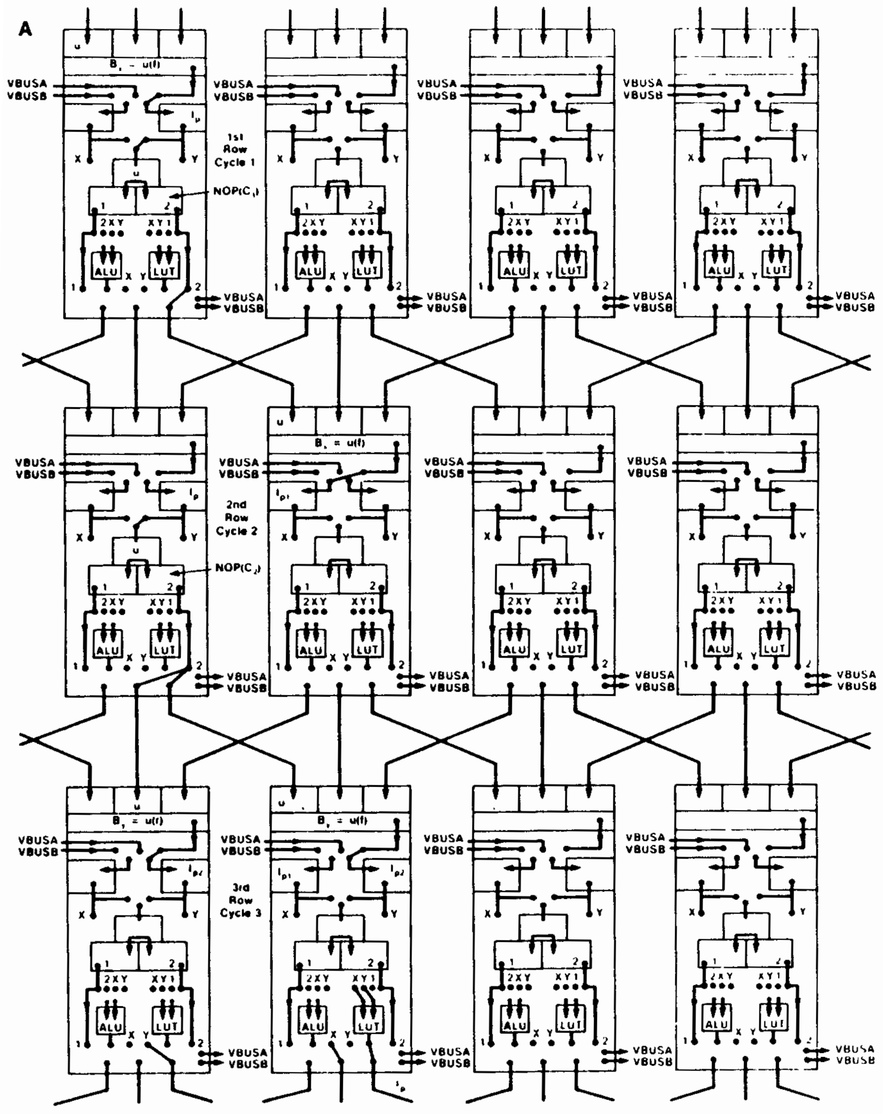

FIG. 9. Representation of PIPE operations for the example of a Sobel operator. (A) Input image I_p. (B) Input image I_q.

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 nsec 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

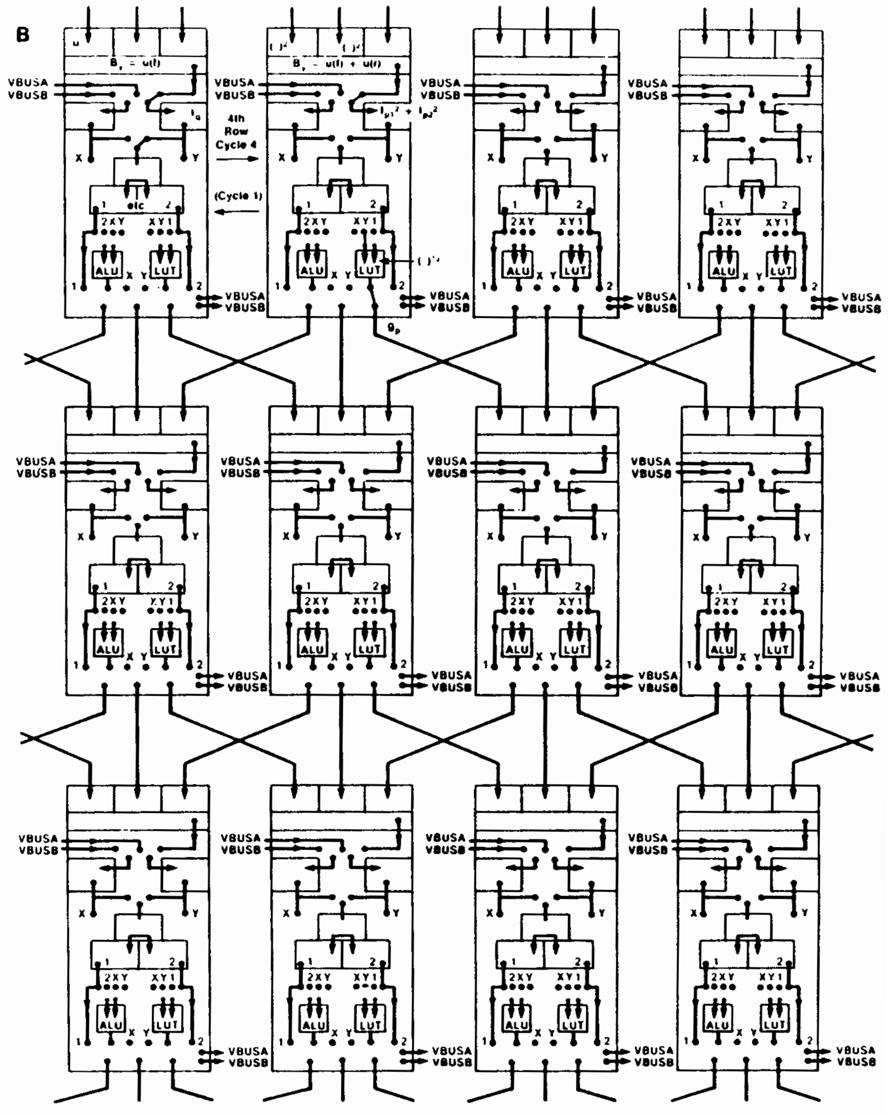

FIG. 9—*Continued*

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 $n \times m$-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

256×240 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., 0.05 sec. Consequently, a von Neumann type of computer must operate at 49.2 Mops/sec to keep up with PIPE in this application.

4.4. *Example 2: Using Image Pyramids*

As an example of pyramid-based processing using PIPE, a means will be described of computing and applying local thresholds for compact object detection. The method is a simplified version of the method developed in [10]. The problem is to find occurrences in an image of compact objects whose approximate sizes are known. The procedure uses a pyramid of images to locate and extract such objects. Objects are extracted using a spot detector applied at the level in the pyramid corresponding most closely to the sizes of the objects being sought. Thresholds are computed and projected down the pyramid. They are applied to the original image in the regions corresponding to the locations of the spots.

The method is to construct a pyramid of images $I_0, I_1, \ldots, I_k$. Here k is the level at which single pixels correspond to regions in the original image of about the right sizes (i.e., within the nearest power of 2). At level k, a spot detector is applied to the image to locate pixels that contrast strongly with their background. A local threshold is computed for extracting each object, and is projected down the pyramid to the bottom level. There it is applied to the original image, resulting in an image containing only objects of the right size.

The pyramid is constructed using a 3×3 averaging operation A, followed by sampling (via PIPE's pyramid mode operation into the forward pathway). Here, the neighborhood operator used is

$$A = \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Following the application of A, each pixel is divided by 8 (or left shifted three times) to form the average, before the pyramid sampling operation takes place. The number of successive pyramiding operations applied is a function of the sizes of the objects being extracted.

When the top level of the pyramid is reached, each pixel corresponds to a region of roughly the desired size. At this point a spot detector S is applied to the image. This is a 3×3 Laplacian operator looking for a central peak surrounded by a valley. The value of the central weight can be adjusted to increase the contrast required between the peak and the valley. For example,

$$S = \begin{matrix} -1 & -1 & -1 \\ -1 & 7 & -1 \\ -1 & -1 & -1 \end{matrix}$$

The output of this operator is thresholded, giving a set of peak spot valu (1's) and a background of nonspot values (0's). The thresholded spot imag is stored in the alternate buffer of MPS stage k.

By applying the spot detector to a low-resolution version of the imag responses to spots smaller than the desired size have been minimized, and global operation has been made local. The pyramid does not, howeve prevent the spot detector from responding to spots larger than the desired siz To ensure that this is accomplished, a second neighborhood operation, F, applied to the output of the spot detector S. F ensures that a spot has no mo than some maximum number, n, of neighbors that are also spots.

$$F = \begin{matrix} -1 & -1 & -1 \\ -1 & n & -1 \\ -1 & -1 & -1 \end{matrix}$$

Here a spot detector is being applied to the thresholded output of the origin spot detector. If desired, the same spot detector can be applied in both case

The output of the operator F_1 is stored in the alternate buffer of the sam stage via the recursive pathway. The original (reduced-resolution) image still in its buffer, unaltered by the processing. The next step is to compute threshold to be applied to the original, full-resolution image in the regio corresponding to each spot. This is achieved using yet another neighborhoo operation. This time, however, the operator is applied only at pixels desig nated by the region-of-interest operator. The operator, G, is applied to th gray-scale image, using the output of the spot detector as the template for th region of interest. The result of applying G is then divided by 16 to provid the desired threshold for each region.

$$G = \begin{matrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{matrix}$$

The thresholds are stored in the pixels corresponding to the regions to whic they are to be applied. The threshold resulting from applying the abov sequence of operations is the average of the gray value at the center of eac detected spot and the mean of its eight neighbors. This was found to give goo results [10].

Now that local thresholds have been found for each spot, it remains t project the values down the pyramid to cover the corresponding regions, an to apply the thresholds to the original image. Projecting the thresholds dow the pyramid involves simply duplicating each pixel of the low-resolutio image into a 2×2 block of identical pixels at the next-higher resolution. Thi is a function provided by PIPE's retrograde pathway in the pyramid mode

The final step of thresholding is done by comparing the original image with the projected image of thresholds on a pixel-by-pixel basis. This operation uses PIPE's logic for combining inputs to a stage, and results in an image that has 1's for points within regions of the desired size, and 0's elsewhere.

5. Discussion

PIPE is designed as a front-end processor for low-level iconic-to-iconic image processing. It is intended to perform transformations on images to extract features similar to those in the primal sketch of Marr [7]. 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.

PUMPS [2] is an example of a multiuser 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 [1] 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 [6] 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 buses 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 [8], and ZMOB [5], 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 8-bit microcomputer, with 64K bytes of memory. Thus, many different computations can be performed at the same time, either on the same or on 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 [9] divided 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. The limitations on these operators are that there are at most 256 different operators available per stage, and that using the region of interest generally precludes using some other operators, such as the functions of two arguments. Using the retrograde pathway to insert expectations into the image analysis process also blurs the distinction between high-level and low-level processes.

A processor that has many features in common with PIPE is the cytocomputer [11]. 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 multiresolution 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.

While several theoretical designs have been proposed for hierarchical (pyramid) processors (e.g., [4, 13]), there is apparently only one that has actually been constructed [12.]. This is a SIMD machine consisting of a pyramidal array of processing elements connected to a general-purpose computer. Each processing element connects to 13 other elements, comprising its eight neighbors at the same pyramid level, its four children at the level below, and its parent at the level above. Neighborhood operations can be performed on this set of elements, as well as pointwise operations and image input and output. Tanimoto presents a number of algorithms that take advantage of the pyramid structure to perform common image-processing operations. The pyramid processor has an advantage over PIPE in the explicit links to its children. To achieve the same result with PIPE requires complex manipulations using the region-of-interest operator and a bit map of four values (one for each child) in the alternate buffer. PIPE, however, can be reconfigured to produce overlapped pyramids, using larger neighborhoods, and is not restricted to pyramid-based operations.

6. Future Enhancements

The neighborhood operators in PIPE all reside on separate boards. This is intended to facilitate changes to the stages as faster neighborhood operator chips become available that can handle larger neighborhoods in the required time. With current technology, a 5 × 5 neighborhood would be easily attainable at slightly greater cost per stage.

Investigations will also be pursued into producing a VLSI chip to perform the operations of an individual MPS. Since most of these operations are useful for general-purpose preprocessing of images, such chips should provide the ability to construct pipelines using off-the-shelf components. This should simplify the construction of special-purpose real-time image processors.

For our application, in which the camera is mounted on a mobile arm, the 256 × 256 image size is adequate, but this is not necessarily true in all applications. There is no fundamental feature of PIPE's design which limits image size. A version of PIPE which can handle 512 × 512 images is already under design, and extensions to 2048 × 2048 are contemplated.

It is also planned to enhance the capabilities of PIPE by adding pre- and postprocessors. A processor will be inserted before the input stage to perform conformal mappings. This will, for example, allow rotations in the image plane and range changes (image scaling) to be converted to image translations. The image-differencing and motion-detection abilities of PIPE will then simplify analysis of the changes.

At the output end of PIPE, another processor will convert image data to symbolic data (the iconic-to-symbolic mapper, or ISMAP). This processor

can collect information about the locations of all features of a particular kind and present it to the host as a unit. It will also be able to histogram images and reduce the amount of data to be handled by the host processor to more manageable levels.

7. 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. 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, and to 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 multiresolution 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

1. Antonsson, D., Gudmundsson, B., Hedblom, T., Kruse, B., Linge, A., Lord, P., and Ohlsson, T. PICAP—A system aproach to image processing. *IEEE Trans. Comput.* **C-31,** 10 (Oct. 1982), 997–1000.
2. Briggs, F. A., Fu, K. S., Hwang, K., and Wah, B. W. PUMPS architecture for pattern analysis and image database management. *IEEE Trans. Comput.* **C-31,** 10 (Oct. 1982), 969–983.
3. Burt, P. J. Fast hierarchical correlations with Gaussian-like kernels. *Proc. Fifth International Joint Conference on Pattern Recognition,* Miami, Fla., 1980.
4. Dyer, C. R. A quadtree machine for parallel image processing. Knowledge Systems Laboratory Tech. Rep. KSL 51, University of Illinois at Chicago Circle, 1981.
5. Kushner, T., Wu, A. Y., and Rosenfeld, A. Image processing on ZMOB. *IEEE Trans. Comput.* **C-31,** 10 (Oct. 1982), 943–951.
6. Luetjen, K., Gemmar, P., and Ischen, H. FLIP: A flexible multi-processor system for image processing. *Proc. Fifth International Conference on Pattern Recognition,* Miami, Fla., 1980.
7. Marr, D. Early processing of visual information. *Philos. Trans. Roy. Soc. London Ser. B* **275** (1976)
8. Potter, J. L. Image processing on the Massively Parallel Processor. *IEEE Comput. Mag.* **16,** 1 (Jan. 1983), 62–67.
9. Reeves, A. P. Parallel computer architectures for image processing. *Computer Vision, Graphics, and Image Processing* **25** (1984), 68–88.

10. Shneier, M. Using pyramids to define local thresholds for blob detection. *IEEE Trans. Pattern Analysis and Machine Intelligence* **PAMI-5,** 3 (May 1983), 345–349.

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

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

13. Uhr, L., Thompson, M., and Lockey, J. A 2-layered SIMD/MIMD parallel pyramidal array/net. *IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management,* Hot Springs, Va., Nov. 1981, pp. 31–34.