Architecture of the NBS Pipelined Image Processing Engine

Ernest W. Kent, Michael O. Shneier, and Ronald Lumia
National Bureau of Standards *

James M. Herriman, Randall L. Luck, and Gerald S. Henrici
Digital/Analog Design Associates, Inc.

Abstract

The Sensory-Interactive Robotics Group of the National Bureau of Standards is producing PIPE (Pipelined Image Processing Engine), an experimental, multi-stage, multi-pipelined image processing device for research in low-level 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 series of local neighborhood and point operations, under the control of a host device. Its output can be presented to such devices as monitors, robot vision systems, iconic to symbolic mapping devices, and image processing computers (Figure 1.)

In addition to a forward flow of images through successive stages of operations as in a traditional pipeline, other paths between the stages of the device permit concurrent, interacting pipelining of image flow in other directions. In particular, recursive operations returning images into each stage, and

*This work is a product of United States Government personnel, and is not subject to U.S. Copyright.

feedback of the results of operations from each stage to the preceding stage are supported in this manner. This architecture facilitates a variety of relaxation operations, interactions of images over time, and other interesting functions. Numerous operations are supported; within each stage these include arithmetic and Boolean neighborhood operations on images. Between-stage operations on each pixel include thresholding, Boolean and arithmetic operations, functional mappings, and a variety of functions for combining pixel data converging via the multiple pipelined image paths.

The device also implements several alternative processing modes. Some operate within each stage, for example to implement "MIMD" operations specific to regions of interest defined by the host device or by previous operations on the image. Others operate between stages, for example to support variable resolution pyramids.

Introduction

PIPE was designed as a pre-processor 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. It is appropriate, however, for any sort of iconic image, such as those produced by tactile sensor arrays or range-image systems. 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. PIPE is currently under construction, with prototypes expected in the fall of 1984. After a period of experimentation and evaluation, PIPE will be refined and, ultimately, reduced to a VLSI implementation.

We have also designed an associated device, ISMAP, which will map processed iconic images into symbolic (property-indexed) structures. ISMAP describes the processed images produced by PIPE by mapping the iconic image of Boolean vectors produced by PIPE into histograms and other types of ordered list structures in the address space of the host device, in real time. A major function of ISMAP is to map PIPE's image addresses into a space ordered by symbolic picture feature descriptors. This sves the host device the necessity of scanning the processed image to find locations of items of interest.

PIPE's design, discussed in this report, was developed with several principal goals in mind. These were: 1) real-time processing of images at field-rate, 2) provision for interactions between related images, such as those arising from dynamic image sequences or from stereoscopic views, 3) provision of the ability to apply different algorithms to different regions of the image in real time, and 4) provision for the ability to guide processing by knowledge-based commands and "hypothesis images" supplied from the host.

PIPE actually consists of three, concurrent, interacting, image-flow pathways. These interconnect a variable number of identical modular image-processing stages. 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.

Each stage can perform two simultaneous and independent

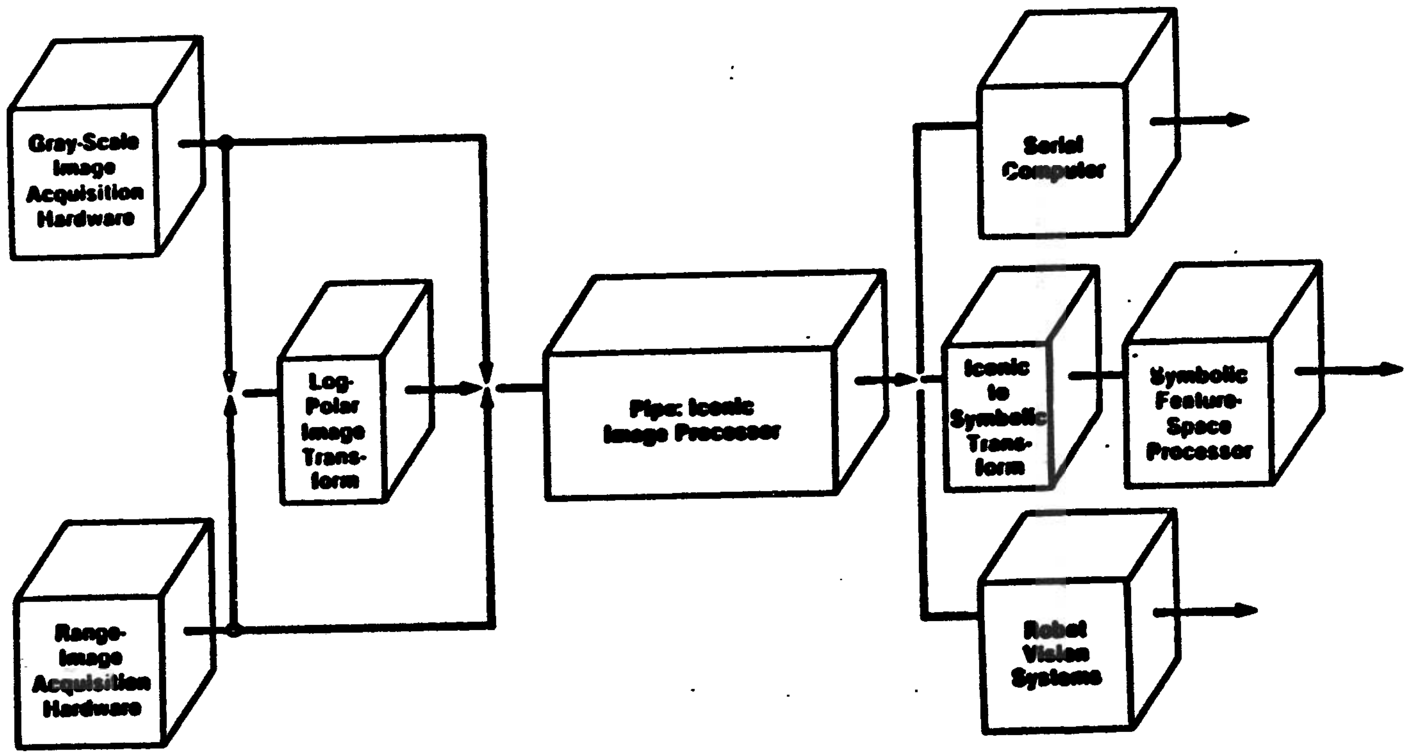
Gray-Scale Image Acquisition Hardware
Range-Image Acquisition Hardware
Log-Polar Image Trans-form
Pipe: Iconic Image Processor
Serial Computer
Iconic to Symbolic Trans-form
Symbolic Feature-Space Processor
Robot Vision Systems

FIGURE 1: The PIPE Processor and its relations to other elements of the NBS Image-Processing Configuration.

arithmetic or Boolean neighborhood operations. The forward, recursive, and retrograde pathways, in any combination, may accept images from the result of either of these operations. At the input to each stage, the images carried by the three pathways may be combined by any arithmetic or Boolean operation. Any arithmetic or Boolean operation may then be performed on the resulting image, prior to its storage in one of two buffers within the stage.

Within each stage, the two neighborhood operators may act on an image in either of the two buffers. Following the neighborhood operations, and prior to output from the stage, the resulting images may undergo a further transformation by an arbitrary function of one or two arguments. If the function is of two arguments, the second argument may be drawn from a variety of sources within the stage, including the contents of the homologous pixel in the buffer which did not supply the image for the neighborhood operations.

In an alternative mode, one of the two image buffers in each stage may serve as a map for selecting the processing algorithms being applied to the contents of the other buffer. In this mode, PIPE functions as an MIMD machine, with one buffer defining regions of interest over which each set of algorithms shall be applied, on a pixel-by-pixel basis, as the image is processed. In the prototype version, sixteen such alternative processing algorithms may be selected for different regions of the image within a single field time. However, this is easily increased (up to 256 processing algorithms) simply by adding additional storage to each stage to hold the appropriate tables and parameters.

A third mode permits PIPE to function as a multi-resolution pyramid machine. In this mode, the images carried by the forward pathway are reduced in size by one half at each stage, while sizes of the images carried by the retrograde pathway are doubled at each stage. The images carried by the recursive pathway remain unchanged in resolution. Any combination of stages may operate in this mode, under program control.

In any mode, the operations of every stage are completely independent, and can be completely reconfigured in the inter-field time by an associated stage control unit, which in turn may select stage configurations from a stored sequence, or on command from the host. All operations run in real time, so that images pass between stages at field rate (16 msec.).

In addition to the pathways mentioned, PIPE contains four "wild card" busses which may be used to transport images from the host, or from any buffer in the machine, into any other buffer in the machine. It also has two special stages, for input and output, which communicate with the sensors and the host, or with special auxiliary devices. DMA channels allow video display or streaming input and output from any buffer in the machine.

Rationale and Overview

PIPE is an attempt to seek a compromise between a fixed function, hardware image processor and a fully general purpose, parallel computer. Through its design, it facilitates a variety of common and important image-processing techniques, as well as several experimental approaches. Within the broad limits of the processes it supports, it is an extremely fast and flexible device. On the other hand, it is not a general purpose computer and it is not possible to program arbitrary algorithms on it, or at least not in an efficient manner. In most cases we have found that processes which are well suited to PIPE may be substituted for others which are not, while accomplishing the same image-processing goal. PIPE is intended as a processor for local operations on images; it is not designed to perform efficiently operations requiring global knowledge of the image. It is intended that PIPE operate in conjunction with a host which will perform global image operations, relieved of the processing burden of large scale repetitive local operations. (However, extensions to PIPE are under consideration to perform common global functions such as region labeling.)

Other extensions to PIPE are also planned into the initial design. The 3 x 3 neighborhood operators in each stage of PIPE reside on a separate board from the rest of the stage. 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 x 5 neighborhood would be easily attainable at slightly greater cost per stage. For our application, in which the camera is mounted on a mobile arm, the 256 x 256 image size is adequate, but this is not necessarily true in all applications. There is no fundamental feature of PIPE's design, however, which limits image size. A version of PIPE which can handle 512 x 512 images is already under design, and extensions to 2048 x 2048 are contemplated. It is also planned to enhance the capabilities of PIPE by adding pre-processors. A processor, already being designed, will be inserted before the input stage to perform conformal mappings (Figure 1). 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.

The prototype version of PIPE consists of a sequence of identical processors (Figure 2.), 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. 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 it, and passes it on to the next stage for further processing. This sequence is repeated every television field-time. When an image emerges at

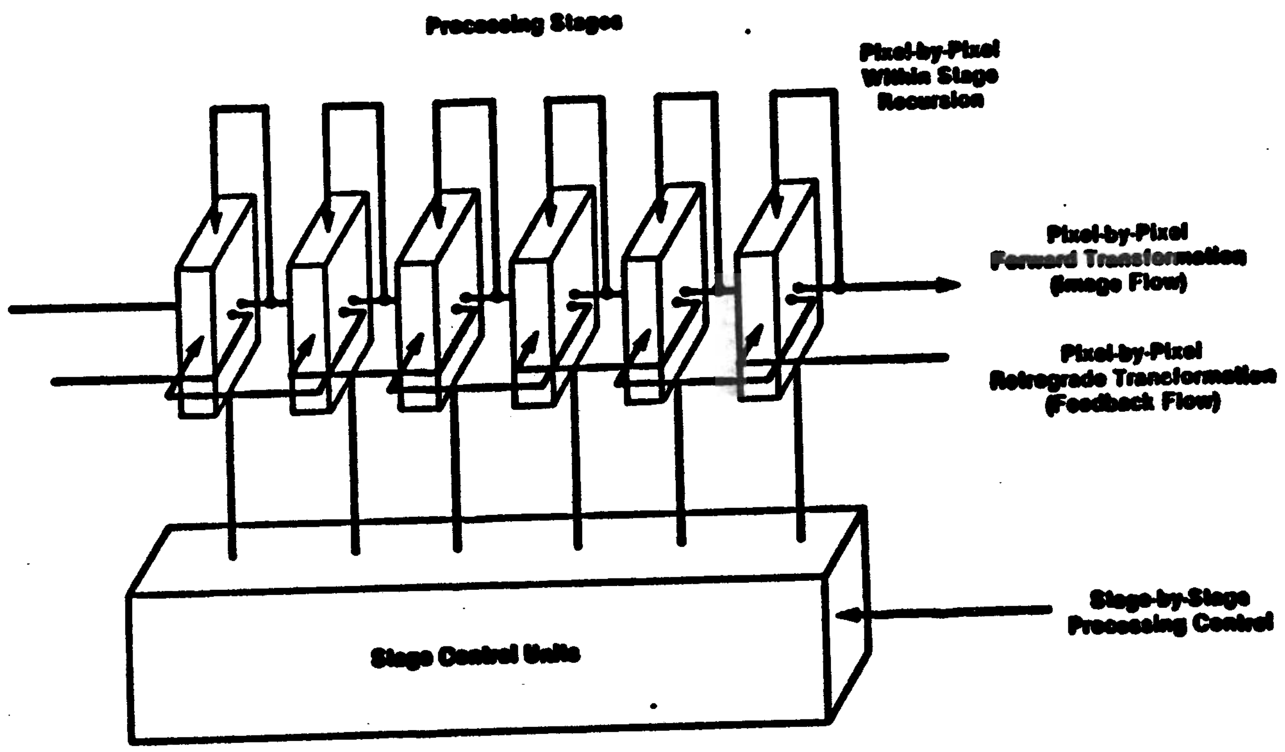

FIGURE 2: Major connections between processing stages in PIPE

the far end of the sequence, it is processed by the special output stage and presented to a host 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 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 these 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 images 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 an image output to a wildcard path.

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 to be performed using temporal as well as spatial neighborhoods. 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.

Although there are physically only three pathways between stages (excluding the "wildcard" busses), every pixel neighborhood in an image is processed and sent over these paths in every field time. The result is that PIPE simulates a fully parallel data flow machine; each pixel appears to have a real-time, private line to an homologous pixel processor in target stages.

Details of Architecture

The organization of the image-processing section of PIPE, excluding the input and output processors, is shown in Figure 2. PIPE is composed of a variable number of identical, modular, image-processing stages. Every stage 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 inter-stage 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 inter-field interval. The stages are connected by three distinct data paths, which are shown in Figure 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 Figure 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 Figure 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.,

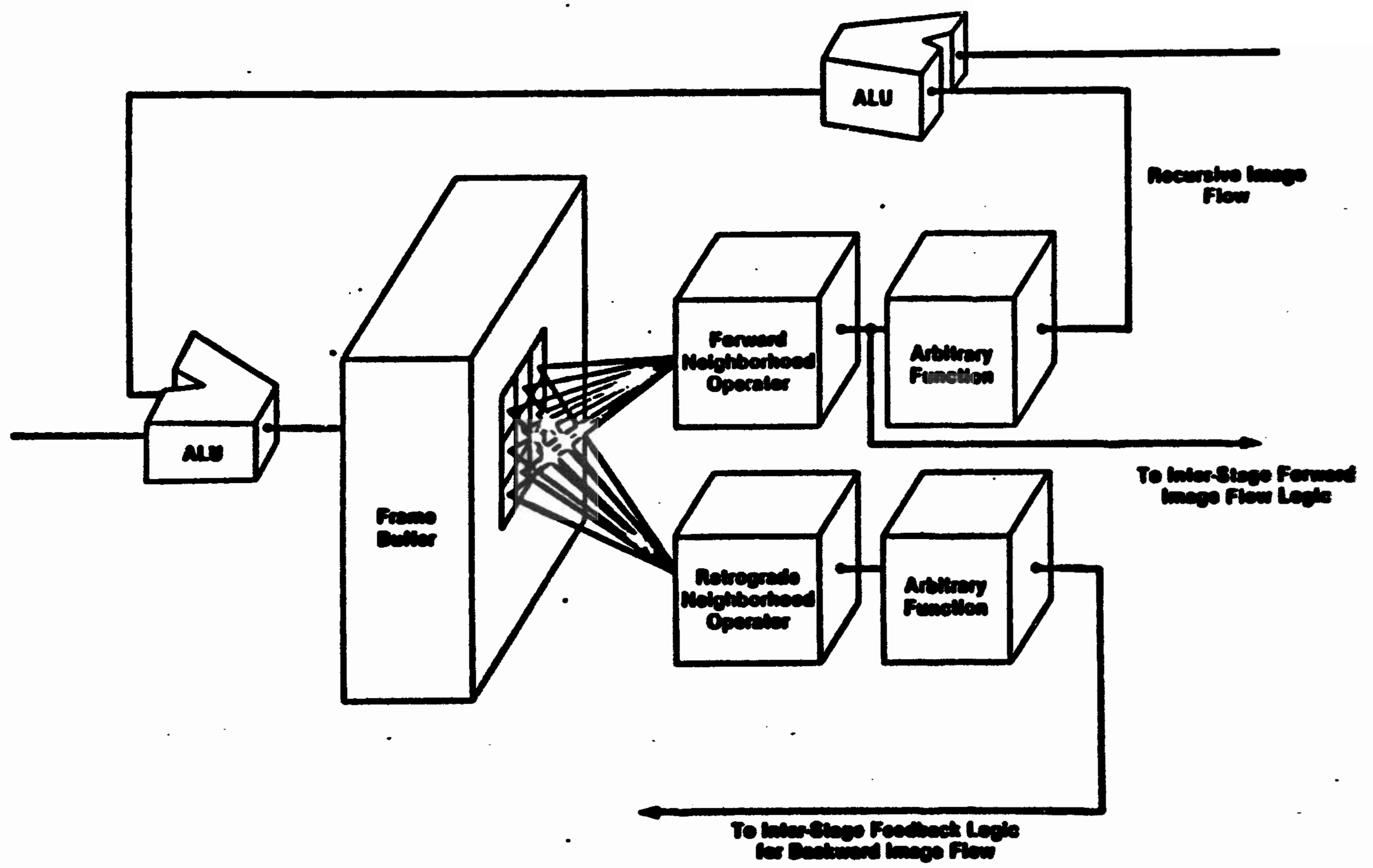
ALU
Recursive Image Flow
Forward Neighborhood Operator
Arbitrary Function
ALU
Frame Buffer
To Inter-Stage Forward Image Flow Logic
Retrograde Neighborhood Operator
Arbitrary Function
To Inter-Stage Feedback Logic for Backward Image Flow

FIGURE 3: Generation of Image-Flow paths from simultaneous application of independent neighborhood operators

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 Figure 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 Figure 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 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 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 inter-stage combining logic to clarify the interaction of the pathways in the temporal domain. The outputs of the forward neighborhood operators (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

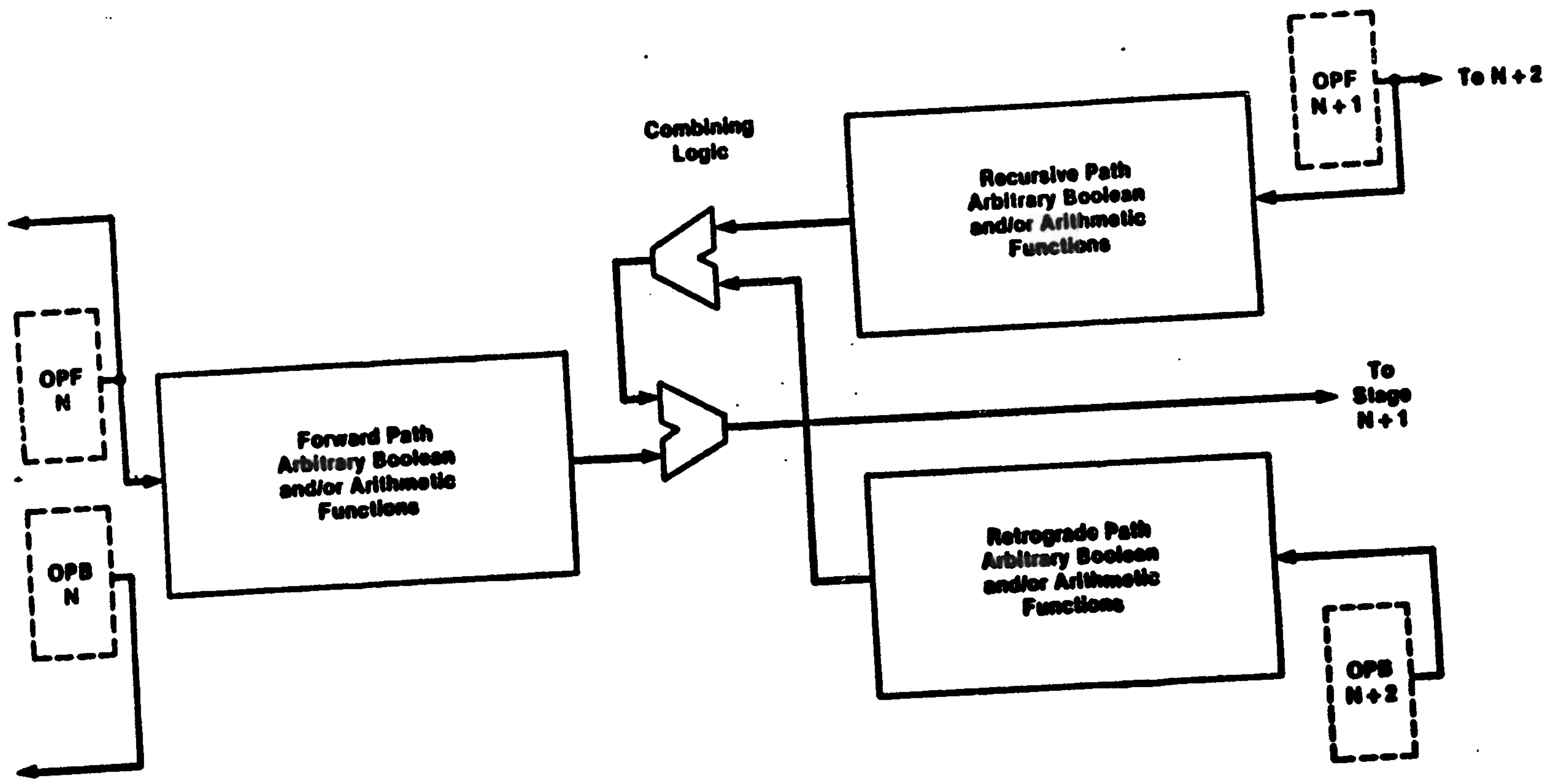
Combining Logic
Recursive Path Arbitrary Boolean and/or Arithmetic Functions
OPF N+1
To N+2
Forward Path Arbitrary Boolean and/or Arithmetic Functions
OPF N
OPB N
To Stage N+1
Retrograde Path Arbitrary Boolean and/or Arithmetic Functions
OPB N+2

FIGURE 4: The interstage combining logic

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 Figure 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 Figure 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 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 eight-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

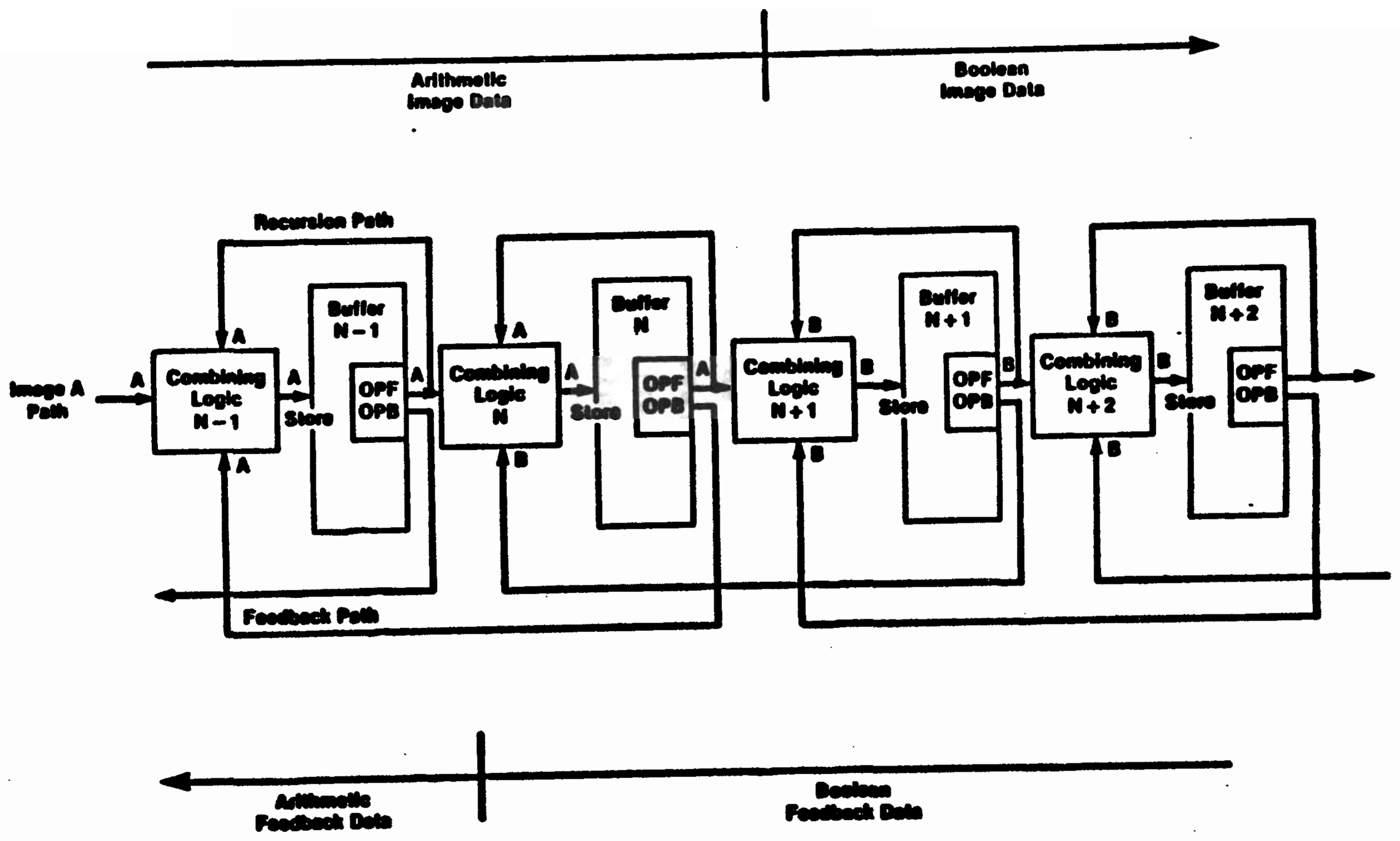

FIGURE 5: Interactions between arithmetic and Boolean images

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 insure 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 various pathways is shown in Figure 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

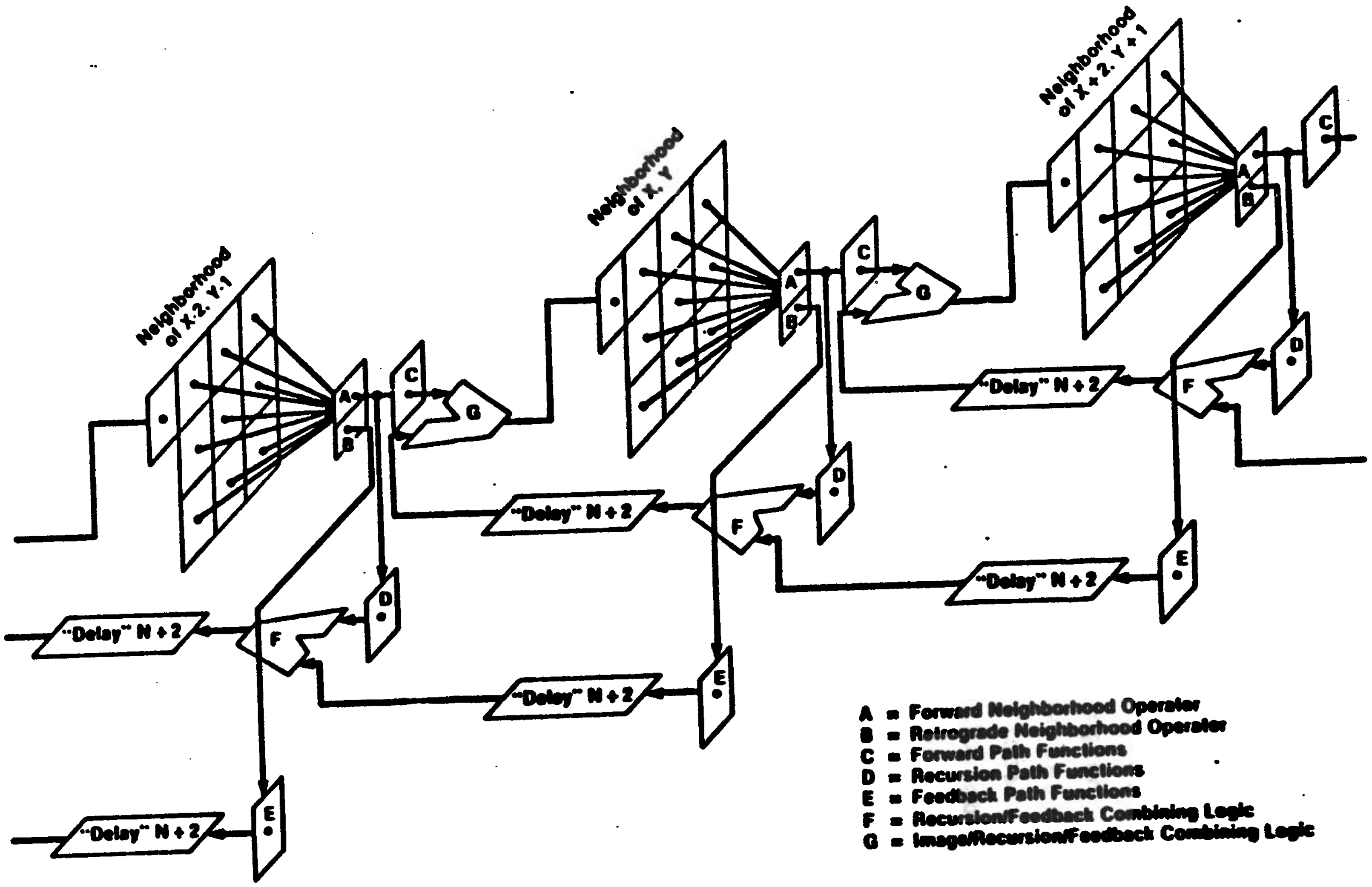
Neighborhood of X-2, Y-1
Neighborhood of X, Y
Neighborhood of X+2, Y+1
A
B
C
D
E
F
G
"Delay" N+2
A = Forward Neighborhood Operator
B = Retrograde Neighborhood Operator
C = Forward Path Functions
D = Recursion Path Functions
E = Feedback Path Functions
F = Recursion/Feedback Combining Logic
G = Image/Recursion/Feedback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

FIGURE 6: Staggering of read/write operations between stages

stream on a pixel-by-pixel basis. This feature may be used to generate globally non-linear 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 protoype device, sixteen will be available. The nature of the available alternative operation sets, like all other apects of stage operations, may be changed between fields.

A further mode of operation allows multi-resolution 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 multi-resolution 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. Multi-resolution image processing using PIPE is discussed further below. Another useful feature of PIPE is the provision of four 'wildcard' busses, which allow images to be transferred from a stage to any or all other stages in a single field time. These busses allow quick access to, and dissemination of, intermediate results. They also make it possible to connect the stages into a ring-like structure, or to bring synthetic images from the host to any stage.

Functional Details of PIPE Stages:

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 eight-bit real-time digitizer. The input stage is capable of acquiring a digitized image of 256 x 240 pixels while remaining synchronized with RS-170 signals. Alternatively, it can capture 256 x 256 pixel images from non-RS-170 signals while internally employing non-standard 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 eight-bit input data, and this precision is maintained throughout the machine. Intermediate arithmetic operations within subsequent stages are carried to sufficient precision to insure no loss of accuracy when the result is rounded to eight bits for transmission to subsequent stages. The data may be treated as either unsigned eight-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.

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 Nth MPS accepts three eight-bit 256 x 256 pixel images as input. These come from the forward output of the N-1st MPS, from the recursive output of the operation performed on the previous contents of the Nth MPS, and from the retrograde output of the N+1st MPS. Each data stream may consist, independently of the other two, of arithmetic or Boolean (eight-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 eight-bit image from the three data streams, each MPS performs comparison, Boolean, and/or arithmetic operations on each 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 a 0 - 7 place barrel shift, and apply AND (NAND), OR (NOR), and EXOR operations to the result (See Figure 7).

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 busses. 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 auto-indexed addressing supported on command. The wildcard busses 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 Figure 7. In operation, an eight-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 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 x 3 pixels square, and the operation is accomplished in 200 nsec. Pixel neighborhoods are generated by passing the data stream through a 3-line buffer.

In the arithmetic case, the convolution operation uses arbitrary positive or negative eight-bit neighborhood weights, and maintains twelve-bit accuracy in its intermediate results. The final eight-bit arithmetic result entering the busses is produced by non-biased rounding, from a 20-bit sum. This insures that no loss of precision occurs within a stage due to arithmetic underflow or overflow. The full eight 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

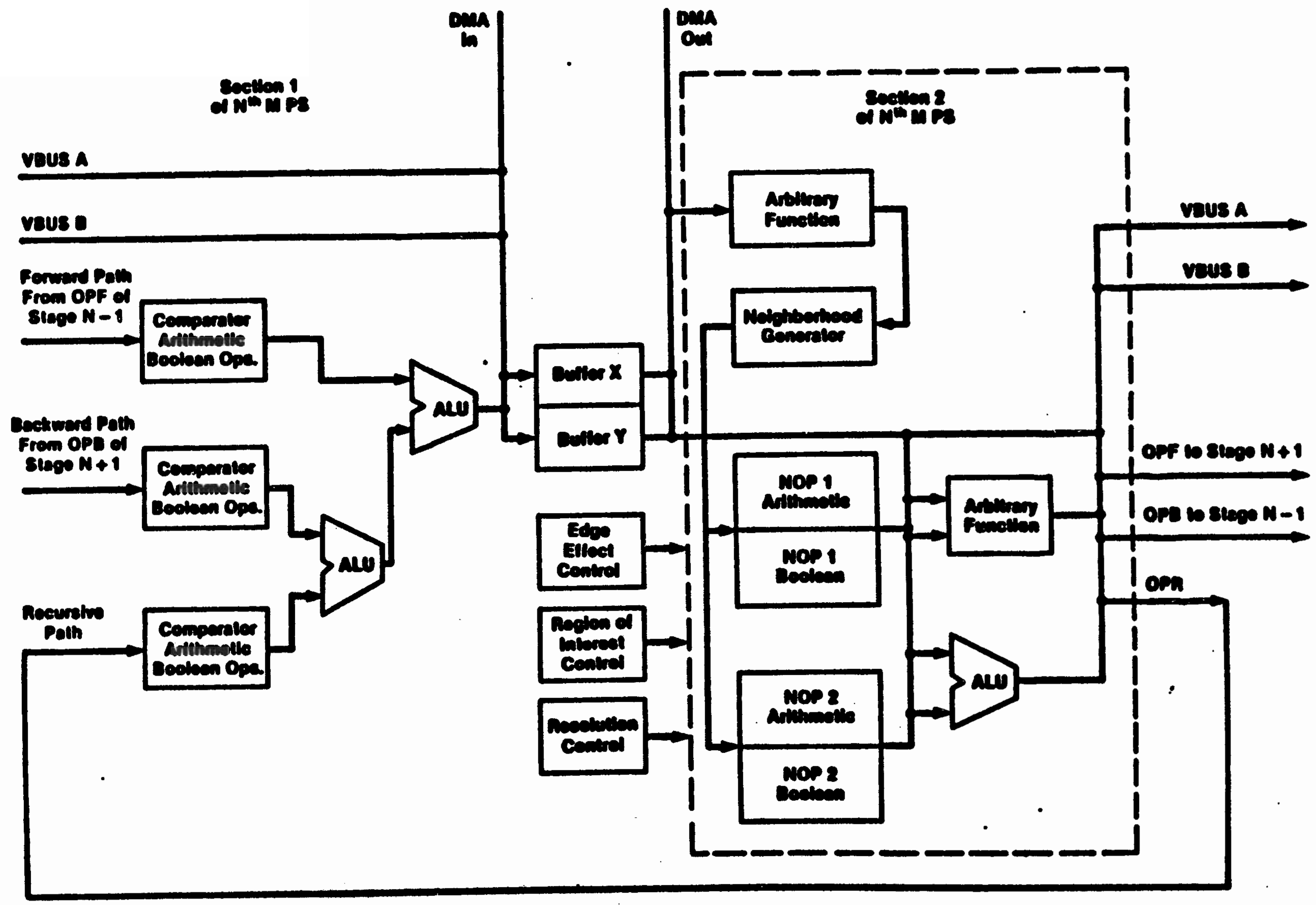

FIGURE 7: The internal architecture of a processing stage

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 one-bit results are obtained from a single pass of the data through the pipeline, yielding an orthogonal eight-bit Boolean vector as output.

Two neighborhood operators (NOP1 and NOP2) are applied independently and simultaneously. They operate on the same data stream, using neighborhood 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 lookup-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 twelve. For example, the six most significant bits of each. If only one argument is used, all eight bits are available, the remaining bits being interpreted as "don't care". The other function is an ALU with two eight-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 eight 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.

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 three by three can be obtained, allowing, for example, the construction of "Gaussian" pyramids using the hierarchical discrete correlation procedure of Burt (1980).

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 three by three 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 two-by-two 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. (When the three by three operator is replaced with a larger one, this asymmetry will 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 non-existent pixels are replaced by the pixels in the border row or column. For a three by three 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.

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 is 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.

Stage Control Units

Once the pipeline modules have 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 pre-set 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 singlestepped.

Programming

During the design phase of PIPE, small programs were written for the machine to evaluate the utility of its architecture. Some of these are described in Kent et al, 1984. Ultimately PIPE will be programmed by utility programs currently under development. A group at Cooper Union under the direction of George Chaiken is working on emulators and on assemblers for setting up operations of the the processing stages, while John Roach of Virginia Polytechnic Institute has been developing a Prologue-based system which will automatically generate the sequences of stage operations required to carry out image processing tasks defined in a high-level language. This system has already successfully derived programs of the level of complexity of Sobel operators.

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(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.

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. 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 (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.

While several theoretical designs have been proposed for hierarchical (pyramid) processors (e.g., Dyer, 1981, Uhr et al., 1981), there is apparently only one that has actually been constructed (Tanimoto, 1984). This is a SIMD machine consisting of a pyramidal array of processing elements connected to a general-purpose computer. Each processing element connects to thirteen 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.

Conclusions

This paper has described a new image pre-processor, 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 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 device 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.

P. J. Burt, Fast hierarchical correlations with Gaussian-like kernels. Proc. Fifth International Joint Conference on Pattern Recognition, Miami, Florida, 1980.

C. R. Dyer, A quadtree machine for parallel image processing. Knowledge Systems Laboratory Technical Report KSL 51, University of Illinois at Chicago Circle, 1981.

E. Kent, M. Shneier, and R. Lumia, PIPE - Pipeleined Image Processing Engine. J. Parallel and Distributed Computing, 1984 (in press).

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 multi-processor system for image processing. Proc. Fifth International Conference on Pattern Recognition, Miami, Florida, 1980.

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.

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.

L. Uhr, M. Thompson, and J. Lockey, 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, 31-34.

FIGURE CAPTIONS

<u>Figure 1.</u> The PIPE processor and its relations to other elements of the NBS image-processing configuration.

<u>Figure 2.</u> Major connections between processing stages in PIPE.

<u>Figure 3.</u> Generation of image-flow paths from simultaneous application of independent neighborhood operators.

<u>Figure 4.</u> The interstage combining logic.

<u>Figure 5.</u> Interactions between arithmetic and Boolean images.

<u>Figure 6.</u> Staggering of read/write operations between stages.

<u>Figure 7.</u> The internal architecture of a processing stage.