JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 1, 105–132 (1984)

A Hierarchical Cellular Logic for Pyramid Computers

STEVEN L. TANIMOTO

Department of Computer Science, FR-35, University of Washington, Seattle, Washington 98195

Hierarchical structure occurs in biological vision systems and there is good reason to incorporate it into a model of computation for processing binary images. A mathematical formalism is presented which can describe a wide variety of operations useful in image processing and graphics. The formalism allows for two kinds of simple transformations on the values (called pyramids) of a set of cells called a hierarchical domain: the first are binary operations on boolean values, and the second are neighborhood-matching operations. The implied model of computation is more structured than previously discussed pyramidal models, and is more readily realized in parallel hardware, while it remains sufficiently rich to provide efficient solutions to a wide variety of problems. The model has a simplicity which is due to the restricted nature of the operations and the implied synchronization across the hierarchical domain. A corresponding algebraic simplicity in the logic makes possible the concise representation of many cellular-data operations.

1. INTRODUCTION

1.1. *Motivation from Biological Vision*

Computation structures are sometimes suggested by nature. For example, the network of neurons in the human brain has been a guiding phenomenon for such formalisms as connectionist models [9]. The hierarchical computation system described in this paper is suggested in part by certain hierarchical structures in biological vision systems.

The mammalian neural visual pathway starting with the retina and ending in area 17 of the striate cortex can be divided into a number of strata or layers: the transducers (rods and cones), bipolar cells, ganglion cells (where lateral inhibition occurs in the retina), lateral geniculate, primary visual cortex, and secondary visual cortex [17]. This stratification, in part, motivated the perceptron formalism [20], as well as the recognition cone model [31].

Further support for the notion of hierarchy in the visual system comes from the work of Hubel and Wiesel, in which various kinds of receptive fields, including simple and complex varieties, were found in the striate cortex of the

0743-7315/84 $3.00

cat. These fields suggest a progression from simple, local information to more and more abstract and global information, as one moves up in the hierarchy of levels [14]. A recent model for cell receptive fields in the striate cortex is one in which a "heptarchy" (a hierarchy of cells in which cells are grouped into sevens) plays a key role [2]. The development of conceptual models for vision should go hand in hand with computational models that suggest and help to clarify algorithms for hierarchical parallel systems. The logic introduced in the present paper is intended to help meet this need.

1.2. *Motivation from Image Processing*

The computational model presented in this paper is motivated also by some practical concerns of image processing. There is a need for powerful methods to extract useful features and descriptions from images in order to recognize the objects present and to understand their relationships.

Some have suggested that a way to get ample computational power for processing a digital image is to place a processor at each cell (pixel) of the image, and allow the processors to communicate directly with their immediate neighbors in the array [4]. This model is also implicit in most cellular-logic approaches to image processing [18]. It is contended here that while such a distribution of processors is an important improvement over one processor or a row of processors, it still does not provide much support for the computation of global (nonlocal) characteristics of an image. In order to ameliorate the effects of this limitation, hardware based on the processor-per-pixel approach has included special attachments to compute sums over all cells and the inclusive OR of a signal over all the cells, thus giving some global capabilities to the system [4].

As is shown in this paper, a hierarchical structure can retain most of the advantages of the processor-per-cell arrangement, while obtaining a capability for global computations that is mathematically elegant. Pyramidal structures that are simultaneously parallel and serial allow gradual formation of more and more global descriptions of image data, in parallel [32].

The hierarchical cellular logic developed here underlies a parallel processing architecture under study at the University of Washington [29], and this architecture bears similarities to one proposed by Dyer [6].

The logic presented permits accurate description of many of the pyramidal techniques appearing in the literature. Such techniques include the computation of pyramids for intensity, color, edge, line, and texture features [27, 16, 13, 23].

2. Definitions

2.1. *Hierarchical Domains and Pyramids*

In order to define the data objects and operations that work on those objects, we begin with the general structure of data objects. Such objects are

called pyramids, and have a structure referred to as a hierarchical domain, which is, in turn, a set of cells.

A *cell* is defined to be a 3-tuple, whose components may be considered to be coordinates. The general cell, (k, i, j), occurs in "level" k, "row" i, and "column" j.

The *hierarchical domain* with $L + 1$ levels is the set of cells

$$\{(k, i, j) \text{ such that } 0 \leq k \leq L, \text{ and } 0 \leq i < 2^k, \text{ and } 0 \leq j < 2^k\}.$$

The kth *level* consists of those cells of the hierarchical domain, whose first coordinate is k. One immediately notices that each level contains a different number of cells. Level L is the largest, and it is also called the *base,* or the finest level. Level 0 consists of the single cell (0,0,0) and is called the *root.*

A function which maps each cell of a hierarchical domain to a value (in some given range) is called a *pyramid.* If the range of values is {0,1} then the function is called a *binary pyramid* or a *bit pyramid.* If the range is {0, . . . ,255} then it is a *byte pyramid.* Pyramids with somewhat different, but related definitions can be found in the literature [26, 1].

2.2. *The Pyramidal Neighborhood*

Each cell of a hierarchical domain has a neighborhood of cells considered to be adjacent to it. This neighborhood generally consists of cells on three levels: the level containing the cell, the level above (level $k - 1$), and the level below ($k + 1$). Cells that are in the base, at the sides, or at the root have incomplete neighborhoods. In order to simplify the discussion, we pretend that "dummy cells" exist around the outside of the hierarchical domain, thereby completing the neighborhoods of border cells.

The number of neighbors of each cell is 13 and they are defined for the general cell (k, i, j) as follows:

N1	"father"	$(k - 1, i \text{ div } 2, j \text{ div } 2)$
N2	"northwest"	$(k, i - 1, j - 1)$
N3	"north"	$(k, i - 1, j)$
N4	"northeast"	$(k, i - 1, j + 1)$
N5	"west"	$(k, i, j - 1)$
N7	"east	$(k, i, j + 1)$
N8	"southwest"	$(k, i + 1, j - 1)$
N9	"south"	$(k, i + 1, j)$
N10	"southeast"	$(k, i + 1, j + 1)$
N11	"northwest son"	$(k + 1, 2i, 2j)$
N12	"northeast son"	$(k + 1, 2i, 2j + 1)$
N13	"southwest son"	$(k + 1, 2i + 1, 2j)$
N14	"southeast son"	$(k + 1, 2i + 1, 2j + 1)$

Together with the home cell, that is, (k, i, j), which may be labeled $N6$, the pyramid neighborhood comprises 14 cells. The 9 of these cells in level k make

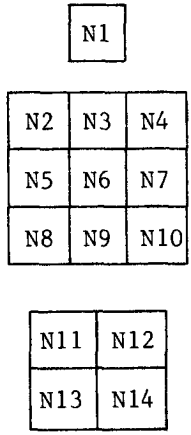

FIG. 1. The neighborhood of a cell in a hierarchical domain.

up the lateral neighborhood. The remaining 5 cells are the quadtree neighbors. The pyramidal neighborhood is diagrammed in Fig. 1.

Each cell has three siblings. The siblings of a cell are the other three sons of the cell's father. The coordinates of the siblings of a cell (k, i, j) can be obtained by complementing the least significant bit of the row coordinate (i) and of the column coordinate (j); the three combinations different from the original give the three siblings.

It is interesting to note that hierarchical domains may be defined analogously for base levels that are not square grids, or in such a fashion that the levels taper more gradually than is the case here [2, 1].

2.3. *A Matching Operator*

Let X be a bit pyramid. Then, given any cell $C = (k, i, j)$, the vector of X values for the neighborhood of C is

$$NX = [X(N_1), X(N_2), \ldots, X(N_{14})],$$

where N_1 through N_{14} are the pyramidal neighbors of C. In the case that N_i happens to be a dummy cell (because of the case when the cell C lies on the border), we define $X(N_i)$ to be 0.

Let P be a "pattern" consisting of 14 symbols, each of which is either 0, 1, or D. That is,

$$P = [P_1, P_2, \ldots, P_{14}],$$

where P_i is 0, 1, or D. Although a pattern is a vector of 14 elements, we shall write it as a string of 14 symbols, to save space.

The AND_Match of P with NX is given by

$$\underset{n=1}{\overset{14}{\text{AND}}} (P_i \triangleq X(N_i)),$$

where

$$x \triangleq y \text{ is } \begin{cases} 1 & \text{if } x = D \text{ or } x = y \\ 0 & \text{otherwise.} \end{cases}$$

The OR_Match of P with NX is given by

$$\underset{n=1}{\overset{14}{\text{OR}}} (P_i \veebar X(N_i)),$$

where

$$x \veebar y \text{ is } \begin{cases} 1 & \text{if } x = y \text{ and } x \neq D \\ 0 & \text{otherwise.} \end{cases}$$

Each of these operators is extended to operate on the entire pyramid X by applying it to the neighborhod of each cell. Thus we may write

$$Y = \text{AND\_Match}(P, X)$$

and

$$Z = \text{OR\_Match}(P, X),$$

where Y and Z are the pyramids whose values are the results of matching each neighborhood vector of X with P, using AND_Match or OR_Match, respectively.

2.4. *Use of the Operator to Effect Shifting, etc.*

By suitably choosing the values of P, the matching operator can be made to perform several useful functions directly. Some examples are shifting a pyramid in one of the eight lateral directions, and detecting local extremities and local concavities in a shape.

If we take $P = DDDDDD1DDDDDDD$, and assign a name to the resulting function of X, we obtain

$$\text{ShiftWest}(X) = \text{AND\_Match}(DDDDDD1DDDDDDD, X).$$

This function happens to be equivalent to

$$\text{OR\_Match}(DDDDDD1DDDDDDD, X)$$

since in either case, the effect is to copy into a cell of Y the value of its neighbor to the east in X. Similar shifting functions can be defined for the other seven lateral directions.

A local extremity of a pattern of 1's in a two-dimensional array may be defined as a 1 surrounded by five, six, or seven contiguous 0's. For example, an extremity surrounded by five 0's is shown below:

$$\begin{matrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{matrix}$$

Thus defined, there are 24 different configurations for extremities, taking the eight possible orientations for each of the three kinds into account. A function which detects the case illustrated is

$$\text{AND\_Match}(D100110100DDDD, X).$$

Local concavities may also be defined on the basis of 3×3 arrays. One kind of concave configuration is described by

$$\begin{matrix} D & 1 & D \\ D & 0 & D \\ D & 1 & D \end{matrix}$$

The detection of this set of (64) two-dimensional configurations is computed by the function

$$\text{AND\_Match}(DD1DD0DD1DDDDD, X).$$

A set of functions such as these may be used to compute an approximation of the convex hull by detecting various local concavities, filling them in with 1's, and repeating the process until no further concavities are detected.

In some subsequent sections, operations involving interlevel interaction are illustrated. A pattern often used for such operations is

$$\text{Psons} = DDDDDDDDDD1111.$$

For notational convenience, functions based upon the matching instructions will usually be written as functions of a single bit-pyramid parameter. An example is the following:

$$\text{ANDUP}(X) = \text{AND\_Match(Psons)}(X) = \text{AND\_Match(Psons, } X).$$

2.5. *Binary Operations, Constant Pyramids, and Restricted Applications of Functions*

All boolean operations are defined on bit pyramids by applying the operations to the individual values of the pyramids. Thus, $Z = X + Y$ is the bit pyramid whose value for cell (k, i, j) is the boolean sum (inclusive OR) of the corresponding values in X and Y. Similarly, $X * Y$, $-X$, $X - Y$, $X \oplus Y$, and $X \mid Y$ represent boolean AND, NOT, difference, exclusive OR, and NAND, respectively.

There are several constant bit pyramids that are worthy of names. The empty pyramid contains 0's at all cells, and the ones pyramid is everywhere 1. Letting context distinguish the symbols' meanings as binary numbers from their meanings as bit pyramids, the empty pyramid and ones pyramid may be written as 0 and 1, respectively.

A pyramid which is everywhere zero except in one level, k, where it is always 1, is called the characteristic pyramid for level k, and is denoted Q_k.

A bit pyramid that has value 1 precisely at cells which are North sons (i.e., cells whose row coordinate is even) is denoted *QN*. Similar pyramids based on other son types are denoted *QS*, *QE*, *QW*, *QNW*, *QNE*, *QSW*, and *QSE*. Here *QNW* is defined as $QN * QW$, and thus contains 1's only at cells which are northwest sons.

Let F denote some bit-pyramid-valued function of bit pyramid X, and let Q also be a bit pyramid. Then the "application of F to X, restricted to Q" is denoted and defined as follows:

$$[F \mid Q](X) = (F(X) * Q) + (X * -Q).$$

By restricting F to Q, modifications of values of X by F occur only where Q is equal to 1, and the values of X are passed unchanged wherever Q is 0. This permits one bit pyramid, in this case Q, to control the application of a function (here F). Binary operations on pyramids may also be so restricted.

$$X[+ \mid Q]\ Y = ((X + Y) * Q) + (X * -Q)$$

denotes the application to bit pyramids X and Y of the binary operation + restricted to bit pyramid Q. Note that while + is a commutative operation, in general, $[+ \mid Q]$ is not. Similarly, restricted applications are defined for $*$, $\oplus$, $|$, and binary difference. Negation can be applied with restriction as

$$[- \mid Q]\ X = (-X * Q) + (X * -Q)$$

but this is more simply written as $X \oplus Q$ in most cases. Another way of

expressing negation is

$$-X = \text{AND\_Match}(DDDDD0DDDDDDDD, X)$$

and so we have

$$[\text{AND\_Match}(DDDDD0DDDDDDDD)\,|\,Q]X = X \oplus Q.$$

2.6. *Comparison of HCL to a Model of Dyer and Rosenfeld*

A related model of computation is that of Dyer and Rosenfeld [8]. There a "pyramid cellular acceptor" (PCA) is defined to be a stack of two-dimensional cellular automata (in proportions identical to those of a hierarchical domain), such that each cell senses the states of its neighbors (1 father, 4 lateral, and 4 sons). The PCA model was used to prove several interesting results about the possibility of recognizing certain sets of patterns in $O(L)$ time steps. Another result due to Dyer and Rosenfeld is that a pyramid cellular device can compute the number of occurrences of a particular symbol in the base level using a number of time steps equal to $2L$. This is an important result; the algorithm for bit counting is a key step in many algorithms for computing global features of two-dimensional arrays of data. Reeves described an addition module that could be combined with others of its kind into a quadtree that performs the bottom-up counting [19]. It is shown subsequently in this paper that the relatively restricted model, HCL, provides an effective way to realize this approach.

The present model, "hierarchical cellular logic" (HCL), takes account of somewhat more of the economic resrictions in designing computing equipment than does PCA. In particular, where allowing each cell to be an automaton, PCA assumes that an arbitrary state-transition table can be provided that will map an arbitrary configuration of states in a neighborhood to an arbitrary new state all in one computational step. By contrast, in HCL, we do not deal with states directly, but with data objects which if stored in memory would contribute to "state." The implicit changes in state, using HCL, are constrained to correspond to a number of primitive operations. These operations can be implemented in hardware at a relatively reasonable cost.

The most important difference between HCL and PCA, however, is that HCL provides an algebraic formalism for expressing computations in hierarchical domains, whereas PCA provides a general and intuitive model for obtaining a rough description of the time required by a pyramidal parallel processor for an algorithm. One can fairly argue that PCA corresponds to an MIMD (multiple instruction steam–multiple data stream) cellular architecture, whereas HCL corresponds to an SIMD (single instruction stream–multiple data stream) architecture.

3. Simple Sequences of CLO's

3.1. *The "Until No Change" Operator "*"*

The functions and operations introduced thus far may be considered to be primitives with which more general transformations may be described. Operations may be combined by functional composition, and more generally, the substitution of terms involving functions for variables in expressions which themselves involve functions.

A special case of composition is the repetition of one function for a definite or an indefinite number of applications. For a definite number, n, of applications of a function F, we write

$$F^n(X) = F(F^{n-1}(X)),$$

where

$$F^0(X) = X.$$

To represent the result of applying and reapplying F until no further change occurs in the result, the "star" meta-operator may be used.

$$F^*(X) = \begin{cases} F^m(X) & \text{if } \exists\ m \text{ such that } F^m(X) = F^{m+1}(X) = \ldots \\ \text{undefined} & \text{otherwise.} \end{cases}$$

This expression is undefined if the sequence does not eventually reach a stable value. By starring a function application, we can often concisely describe computations that would otherwise require explicit description of a loop index and termination condition.

The star notation hides some information about iteration. On the one hand, this may seem to obscure the number of computation steps required to compute an expression. This depends upon the particular expression, as for example, the following two expressions represent the same bit pyramid:

$$[\text{AND\_Match(Psons)} \,|\, (1 - Q_L)]^L(X)$$

$$[\text{AND\_Match(Psons)} \,|\, (1 - Q_L)]^*(X).$$

On the other hand, the star allows the number of iterations of a function application to be data dependent. This suggests that the time required to perform an operation can and should usually be less than the maximum number of steps that will ever be required. It is also useful in avoiding the need to calculate or describe the maximum number of iterations that might be required.

The star meta-operation can be compared to the propagation operation of the CLIP4 system [5]. The similarity is that both imply repetition of a local operation. In the CLIP, the local operation is restricted somewhat. In HCL, the repeated operation need not be local; it need only be expressible as an HCL expression (and to be well defined, should represent a terminating repetition). It is possible in the CLIP, as well as in HCL, to write expressions (programs in the case of the CLIP) with undefined values. One of the most obvious uses of the star in HCL is to simulate propagation. This is shown later for the problem of extracting connected components from binary images. Another use of the star is in constructing bit pyramids from images. In this case, data flow is upward in the hierarchical domain, and there is nothing comparable using the CLIP propagation operation.

Implementation of the star meta-operation typically involves translation of the starred expression into a sequence of operations followed by a conditional branch instruction. The condition is computed by exclusive-oring the results of two successive iterations and testing whether the bit pyramid obtained is uniformly zero.

At this point one should discuss the issue of computational complexity to some extent. An HCL expression is a representation for two things: a bit pyramid (the value of the expression), and an algorithm (the sequence of steps that must be taken to compute that value). The number of operations in the expression itself indicates the "program complexity" of the operation. If there are no occurences of *, then the time complexity of the expression can be determined from the expression itself. If there are occurrences of *, then the number of time steps is generally data dependent, and can only be expressed in terms of characteristics of the data. Our general assumption is that a primitive HCL operation (a single unrestricted boolean operation on bit pyramids or an unrestricted matching operation on a bit pyramid) requires one time unit. One may also assume that restriction of an operation requires one step to be "set up," but that if a restricted operation is repeated and that the same restriction remains in effect during the repetition, then no additional time is required for the restriction during successive iterations.

3.2. *Building Pyramids*

If we assume that B is a bit pyramid representing a binary image stored in its base level (and B is elsewhere equal to 0), there are some bit pyramids that may be derived from B that are useful in image processing. One of these is the "AND pyramid":

$$\text{ANDPYR}(B) = [\text{AND\_Match(Psons)} \mid (1 - Q_L)]^L(B).$$

Here the AND_Match function is repeatedly applied L times, each time constructing one more level of data in the resulting pyramid. At each step, a value of 1 is placed in a cell if and only if all four of the sons of the cell contain

1. The operation is restricted, however, to take place only in levels 0, 1, . . . , $L - 1$, so that the data in level L are not wiped out. Note that the superscript (repetition factor) L might better be replaced with *, since the eventual result is stable, and it might well stabilize to the correct result long before L iterations are actually computed. For example, if B holds a checkerboard image (whose squares are one cell in size), then the very first application of AND_Match produces an identical pyramid, which happens to be the correct value of ANDPYR(B).

The "OR pyramid" is similarly defined:

$$\text{ORPYR}(B) = [\text{OR\_Match(Psons)} \,|\, (1 - Q_L)]^L(B).$$

Both the OR and AND pyramids may be viewed as pyramids derived from a binary image by counting, at each cell, the number of sons equal to 1 and then comparing this count to a threshold. If at least one son is equal to 1, then the cell gets a 1 in the OR pyramid. For the AND pyramid, the threshold is 4. The threshold may be also set at 2 or 3, although the HCL expressions for these pyramids are longer. For example, the case of 3 is as follows: Let PMNW = $DDDDDDDDDD111$, PMNE = $DDDDDDDDD1D11$, PMSW = $DDDDDDDDD11D1$, PMSE = $DDDDDDDDD111D$, and let function

$$\begin{aligned}\text{Fthree} = {} & \text{AND\_Match(PMNW)} + \text{AND\_Match(PMNE)} \\ & + \text{AND\_Match(PMSW)} + \text{AND\_Match(PMSE)}.\end{aligned}$$

Then we define

$$\text{COUNT\_OF\_3\_PYR}(B) = [\text{Fthree} \,|\, (1 - Q_L)]^L(B).$$

The "count-of-two" pyramid has an HCL definition based on a similar function Ftwo, which is a sum of six AND_Match functions, having one term for each of the six ways in which two of the four sons can be 1's. The count-of-two pyramid may have some importance in picture processing as a kind of compromise between an OR pyramid and an AND pyramid. The OR pyramid may be regarded as a particularly liberal one, in which each cell (in levels 0 to $L - 1$) is assigned a 1 if (as few as) one son (or more) has a 1. On the other hand, the AND pyramid is a rather conservative structure, where all of a cell's sons must have value 1 in order for the cell to get a 1. The count-of-two pyramid appears to more closely achieve a balance of zeros and ones in upper levels than does either of the other two.

One characteristic of the different kinds of bit pyramids defined above is the manner in which they map objects that are connected at one level of the hierarchical domain, to other levels. In an OR pyramid, isolated objects (composed of 1's) get represented in coarse levels by 1's also, until a level

is reached where these representations are merged. With an AND pyramid, isolated holes (composed of 0's) get represented in coarse levels until they are merged. In the case of the OR pyramid, any region of 1's that is connected (in either the 4- neighbor or 8-neighbor sense) in level k, corresponds to a similarly connected set of 1's in level $k - 1$. That is, the fathers of the 1-cells in level k form a 4-connected region in level $k - 1$. In an AND pyramid, we can say the same thing about background regions (composed of 0's).

Also, in an AND pyramid, two cells containing 1's at level k, connected by a path of 1's, have their sons similarly connected (in level $k + 1$). Conversely, in an OR pyramid, two such connected cells have fathers that are so connected. These statements are true considering either 4-connectedness or 8-connectedness.

Chains of 1's on a background of 0's constitute an important class of binary imagery. They often are the results of edge detection. The three kinds of pyramids mentioned above handle such chains in completely different ways. The AND pyramids eliminate such chains at the $L - 1^{\text{th}}$ level, assuming such chains are of width 1. The OR pyramids preserve such chains (until a level is reached at which the chain becomes merged with parts of itself or other things). Not only that; if such a chain is originally 4-connected, then its reductions in the OR pyramid are also 4-connected.

In a count-of-two pyramid, we cannot make a statement quite as strong; however, some of the fathers of cells in level k that form a 4-connected region in the shape of a digital curve are sure to form an 8-connected region in level $k - 1$. The count-of-two pyramid maps 4-connected chains at level k to 8-connected chains at level $k - 1$. However, an 8-connected chain may be either eliminated, broken, or mapped intact to the next level in a count-of-two pyramid, depending upon width, direction, and orientation of the chain. As a consequence of this, one must use a combination of $s -$ values if one wants 8-connected representations at level k of 4-connected chains in level L. Levels L $- 1$ through $k + 1$ of a pyramid may be constructed taking $s = 1$, producing 4-connected reductions, and level k may be constructed with $s = 2$, obtaining an 8-connected reduction in which diagonal chains are no thicker than necessary. This makes it more likely that separate chains can still be distringuished at level k. Figure 2 shows three kinds of pyramids constructed from the same two-dimensional pattern in level L.

3.3. *Pyramidal Projection, Ouverture and Fermeture*

While constructing OR and AND pyramids requires that data flow upward in a hierarchical domain, another operation involves the downward flow of data. The term "projection" was selected by Hanson and Riseman [12] for downward flow in hierarchical structures such as a hierarchical domain. A specific meaning may be assigned to the term as follows:

$$\text{Project}(X) = \text{AND\_Match}(\text{Pfather}, X),$$

1	0	1
0 1	0 0	0 0
1 1	0 0	1 1
0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 0	0 0 0 0
0 1 1 1	0 0 1 0	0 1 1 1
0 1 1 0	0 0 0 0	0 1 1 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1
0 0 0 1 1 1 1 1	0 0 0 1 1 1 1 1	0 0 0 1 1 1 1 1
0 0 0 1 1 1 1 0	0 0 0 1 1 1 1 0	0 0 0 1 1 1 1 0
0 0 1 1 1 1 0 0	0 0 1 1 1 1 0 0	0 0 1 1 1 1 0 0
0 0 0 1 0 0 0 0	0 0 0 1 0 0 0 0	0 0 0 1 0 0 0 0
(a)	**(b)**	**(c)**

FIG. 2. Binary pyramids constructed by comparing the number of sons with 1's to parameter s: (a) ORPYR(B), $s = 1$; (b) ANDPYR(B), $s = 4$; and (c) "count-of-two" pyramid for B, $s = 2$.

where Pfather $= 1DDDDDDDDDDDDD$. The projection of X is the result of copying each home cell's father value down to it. Notice that if X contains a region of 1's in level k, then the corresponding region in Project(X) will occur in level $k + 1$, and will be twice as long and twice as wide and have four times as many cells. As is shown next, projection is an important step in defining some transformations on binary images.

"Ouverture" and "Fermeture" are two operations which can be used to simplify a binary image. The *pyramidal ouverture* of k steps is defined as

$$\text{Ouverture}_k(X) = \text{Project}^k(\text{ANDPYR}(X))$$

and this operation has the effect of removing 1's from the image in any given level of X. The 1's are first removed at cells whose siblings in X are 0, then at cells whose fathers' siblings in ANDPYR(X) are missing, etc. The higher the value of k, the more 1's are removed.

The *pyramidal fermeture* of k steps is similarly defined:

$$\text{Fermeture}_k(X) = \text{Project}^k(\text{ORPYR}(X)).$$

A pyramidal fermeture of 1 step has the effect of adding 1's to a binary image, so that cells having 1's end up with siblings having 1's also.

Both ouverture and fermeture as defined here are stable in the sense that additional applications of either operation do not change the result of the first application (provided k remains the same in each case). That is, assuming a fixed k

$$\text{Fermeture}(X) = \text{Fermeture}^*(X) = \text{Ouverture}^*(\text{Fermeture}(X))$$

and

$$\text{Ouverture}(X) = \text{Ouverture}^*(X) = \text{Fermeture}^*(\text{Ouverture}(X)).$$

An interesting application of ouverture and fermeture is in the construction of a sort of quadtree representation for a binary image. In a quadtree, a binary image is represented by a tree structure, whose nodes are of three types: full, void, and mixed. Furthermore, each node has either 0 or 1 incoming arcs and 0 or 4 outgoing arcs. Only the root has 0 incoming arcs. And only leaf nodes have 0 outgoing arcs. Every mixed node must have four outgoing arcs, and the nodes they lead to must neither be all four void, nor all four full [15]. The quadtree represents the binary image by each leaf node representing a square block of 0's (if the node is void) or 1's (if the node is full). Mixed nodes correspond to blocks which contain both 0's and 1's. The root of the quadtree corresponds to the entire image. Its four sons correspond to the NW, NE, SW, and SE quandrants of the image, etc.

The quadtree for a binary image B may be represented using two bit pyramids; let us call them E and F. Any cell of the hierarchical domain falls into one of four categories under this scheme. The four categories are shown with their corresponding indications using E and F below:

Category	E bit	F bit
No node	0	0
Void node	0	1
Mixed node	1	0
Full node	1	1

The two pyramids are easily computed. It is assumed that B is a bit pyramid containing the input image in level L. Then we compute: $E = \text{ORPYR}(B) - \text{Ouverture}(B)$ and $F = \text{ANDPYR}(B) - \text{Project}(\text{ANDPYR}(B)) + (\text{Fermeture}(E) - E)$. Here Ouverture is the 1-step pyramidal ouverture, Fermeture is the 1-step pyramidal fermeture, "+" indicates boolean addition, and "−" indicates boolean subtraction. The steps in this computation are alternatively described as follows:

```
Let X = ORPYR(B)          ; = set of full & mixed nodes unpruned
Let Y = ANDPYR(B)         ; = roots of full subtrees
```

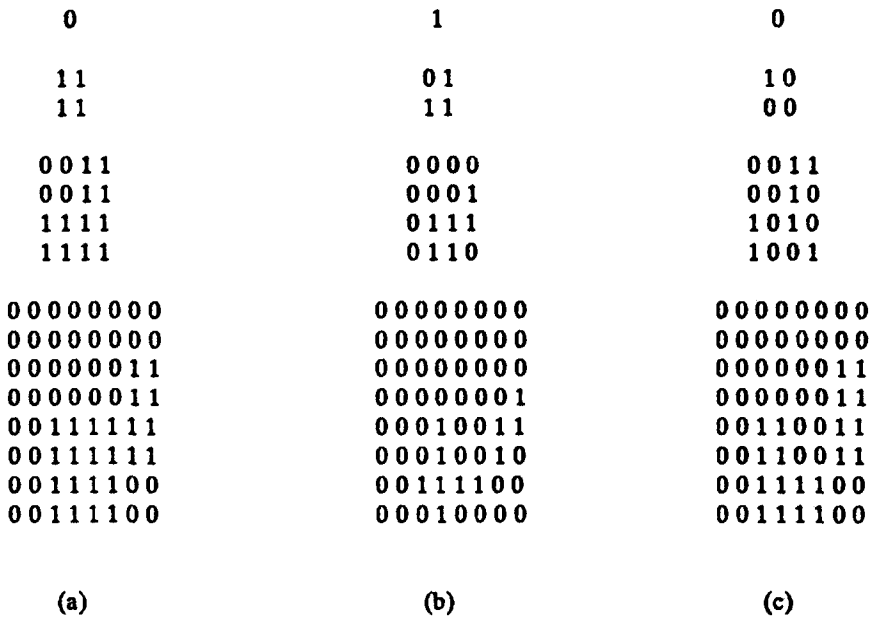

FIG. 3. Construction for an embedded quadtree: (a) Fermeture(B), the pyramidal fermeture of 1 step applied to binary image B of Fig. 2, (b) the E pyramid, and (c) the F pyramid.

```
Let Y' = Project(Y)          ; = Ouverture(B) = nonmax roots of f.s.
Let E = X - Y'               ; = ORPYR(B) - Ouverture(B)
Let Z = Y - Y'               ; = maximal roots of full subtrees
Let W = Project(X) - E       ; = set of void nodes
Let F = Z + W                ; = union of full nodes with void nodes.
```

The result of these steps on the image used in Fig. 2 is shown in Fig. 3.

3.4. *Cartesian Coordinates*

Let N be the number of rows in the base of the hierarchical domain under consideration. The number of columns must then also be N, and $N = 2^L$. Using $L = \log_2 N$ bit pyramids, the row or the column coordinates of each base cell may be represented. Let the values at a cell (k, i, j) in bit pyramids $I(L-1), I(L-2), \ldots, I(0)$ be the binary representation for the row (i) coordinate of that same cell (k, i, j). Then $I(b)$ is given by

$$\text{Project}^b(QS).$$

Similarly the bth bit of the column coordinate is in the bit pyramid

$$\text{Project}^b(QE).$$

If some procedure should require that cartesian coordinates of each cell be available at that cell, in L steps, the successive projections of either QS or QE (or both using $2L$ steps) may be obtained. Alternatively, the bit pyramids $I(L-1), \ldots, I(0)$, and $J(L-1), \ldots, J(0)$ may be considered constants, since they are not data dependent, and could reasonably be counted for free in analyzing the time requirements of a procedure that uses them.

3.5. *Bit-Serial Arithmetic*

The computation of arithmetic operations on multibit pyramids may be accomplished in two ways. One of these is through straightforward bit-serial manipulations. The other is to divide the hierarchical domain into groups of cells called "clerks" which then work as units to compute arithmetic operations [25]. These two kinds of arithmetic computations may be called "per cell" operations and clerk operations.

3.6. *Bottom-up Bit Counting*

As previously mentioned, the result of Dyer and Rosenfeld, that a pyramid cellular automaton can count the number of occurrences of a particular symbol in the base level using $O(\log N)$ time steps, is important. This is because counting the number of pixels (in an image) having some particular property is an important way of obtaining features of an image for use in pattern recognition. Reeves [19] has expanded on the proof appearing in [22], indicating that two, rather than one, carry bits are needed at each cell.

What is not immediately obvious, however, is that this result carries over to the HCL computation model. Because the HCL model only computes one function at a time (and thus is SIMD), its algorithms must follow more stringent restrictions. It is shown here how bit counting can be expressed in HCL and yet achieve the $O(\log N)$ upper bound.

The proof is divided into two parts. First, a special bit-serial addition operation, "ADD_SONS_BITS," is defined which can be employed at one or more levels in a hierarchical domain. Then a scheme is presented which uses this operation many times to compute the entire sum.

The job of ADD_SONS_BITS is to take six inputs and compute three outputs, at each cell of the hierarchical domain. Four of the inputs are the binary values of the four sons of a given bit pyramid. We refer to these four values by the symbols $x1, x2, x3$, and $x4$. The other two inputs are the values of two carry bits: $C1$ and $C0$ associated with each cell. These two carry bits represent integers between 0 and 3; $C1$ is the more significant of the two. The three outputs, $S2$, $S1$, and $S0$, are the binary representation of the sum of $x1 + x2 + x3 + x4 + 2C1 + C0$. This sum is in the range 0 to 7. Since all the son values come from a single bit pyramid, ADD_SONS_BITS may be viewed as a multivalued function which inputs three bit pyramids and outputs three new bit pyramids. In order to express it in HCL, several other

bit pyramids are used. In the 13 equations below, boolean multiplication (AND) is shown implicitly; $x0 * x1$ is written as $x1\ x2$.

$$
\begin{aligned}
T0 &= C0 \oplus x0 \\
T1 &= T0 \oplus x1 \\
T2 &= T1 \oplus x2 \\
S0 &= T2 \oplus x3 \\
U0 &= C0\ x0 \\
U1 &= U0 + (-U0)\ T0\ x1 \\
U2 &= U1 + (-U1)\ T1\ x2 \\
U3 &= U2 + (-U2)\ T2\ x3 \\
S1 &= U3 \oplus C1 \\
V0 &= U0\ C1 \\
V1 &= V0 + C1\ x0\ x1 \\
V2 &= V1 + C1\ (x0\ x2 + x1\ x2) + U0\ x1\ x2 \\
S2 &= V2 + C1\ (x0\ x3 + x1\ x3 + x2\ x3) + U0\ (x1\ x3 + x2\ x3) \\
&\quad + C0\ x1\ x2\ x3 + x0\ x1\ x2\ x3.
\end{aligned}
$$

Computing these values requires 45 HCL primitive operations, assuming that the following are primitive operations;

- loading any xi for the first time;
- computing any product term consisting only of xi's, e.g., $x0\ x2\ (-x3)$ can be computed in one primitive operation;
- any binary operation on bit pyramids, e.g., $C0 \oplus x0$, assuming here that $x0$ has already been loaded.

Counting the primitive operations in the above equations gives 48. However, one operation is saved by computing $x1\ x2$ just once in the equation for $V2$. Two more operations are saved by computing $x1\ x3$ and $x2\ x3$ just once each in the equation for $S2$.

Using ADD_SONS_BITS, a formulation for the entire bit-counting operation can be made by finding appropriate names for the bit pyramids that serve as inputs to and outputs from the various occurrences of ADD_SONS_BITS. Let $F0, F1, \ldots, Ft, G0, G1, \ldots, Gt, H1, H2, \ldots, Ht$ be bit pyramids. We assume that $F0$ is the input data for the entire operation, and that the 1's to be counted are in level L, with all other levels empty. $G0$ and $H0$ are both equal to the empty pyramid. The operation ADD_SONS_BITS is

applied at each step, inputting Fi Gi, and Hi (representing son data, low carry bit, and high carry bit, respectively), and outputting $F(i + 1)$, $G(i + 1)$, and $H(i + 1)$ (which receive $S0$, $S1$, and $S2$, respectively). The number of steps required, t, is equal to $3L$. The count of 1's in $F0$ appears represented in binary in the roots of bit pyramids Ft, $F(t - 1)$, . . . , $F(L)$. That is, the root bit of Ft is the most significant bit of the sum, etc. This formulation does not require explicit erasure of the input to avoid counting the same data twice. The erasure is implicit because the level L cells are adding their dummy sons (equal to 0 by definition) to their carried-over values (zero because $G0$ and $H0$ are empty), thus ensuring that the level L cells of $F1$ contain 0. Then since there is no downward data flow in this procedure, level L remains zero.

The result of performing this algorithm is an integer (encoded in binary by the roots of bit pyramids) that we represent by COUNT(X), where X is the binary image whose 1's have been counted. Strictly speaking, COUNT(X) is not an expression of HCL, since it represents an integer scalar, not a bit pyramid. However, we shall find the function COUNT useful on a number of occasions.

If only certain bits of the count are desired, it may be possible to simplify the bit-counting procedure. In the case when only the parity of X is desired (parity = 1 iff COUNT(X) is odd), then at each step, only $F(i)$ need be computed, and after L steps, one obtains $F(L)$, whose root value is the parity bit. Another case where time can be saved is when it is known a priori that COUNT(X) requires at most b bits for its binary representation, where b is something well less than $2L$. In this case, only $L + b - 1$ iterations need be performed.

4. Global Statistics of an Image

The bit-counting procedure previously described can be applied to the task of measuring global statistics for an image or an object represented by an image [22]. Let COUNT(X) represent an integer which is the count of 1's in level L of the bit pyramid X. Expressions for various global image statistics can now be easily given.

4.1. *Area and Perimeter*

Area and perimeter are useful image features for pattern recognition [21]. Their expressions are

$$\text{Area}(X) = \text{COUNT}(X)$$

and

$$\text{InteriorPerimeter}(X) = \text{COUNT}(X * \text{OR\_Match}(\text{Pintedge}, X)),$$

where Pintedge = *D*0000*D*0000*DDDD*. InteriorPerimeter thus counts the number of cells which contain 1's but which have a lateral neighbor that contains 0. One may also define

$$\text{ExteriorPerimeter}(X) = \text{InteriorPerimeter}(-X)$$

to count the number of cells which contain 0 but have a neighbor containing 1.

4.2. *Euler Numbers*

The Euler number of an image is defined to be the number of connected components (of 1's) minus the number of holes. It is well known that this is equal to Csingles − Cpairs + Cblocks, where Csingles is the count of 1's in the image, Cpairs is the number of adjacencies of 1's in the image, and Cblocks is the number of times a group of four 1's can be found in a 2 × 2 configuration. Using HCL (and the COUNT function), we have

$$\begin{aligned}
\text{Csingles}(X) &= \text{COUNT}(X) \\
\text{Cpairs}(X) &= \text{COUNT}(\text{AND_Match}(DD1DD1DDDDDDDD, X)) \\
&\quad + \text{COUNT}(\text{AND_Match}(DDDDD11DDDDDDD, X)) \\
\text{Cblocks}(X) &= \text{COUNT}(\text{AND_Match}(DD11D11DDDDDDD, X)).
\end{aligned}$$

4.3. *Directionality Counts*

In order to have a rough indication of the shape of an object depicted in a binary image, one may wish to know the distribution of angles at which its boundaries lie. Directionality information may also be used to describe the texture of an image [11]. HCL expressions for one set of such indicators are the following:

$$\begin{aligned}
\text{Verticality}(X) &= \text{COUNT}(\text{AND_Match}(DD10D10DDDDDDD, X)) \\
&\quad + \text{COUNT}(\text{AND_Match}(D01D01DDDDDDDD, X)) \\
\text{Horizontality}(X) &= \text{COUNT}(\text{AND_Match}(D00D11DDDDDDDD, X)) \\
&\quad + \text{COUNT}(\text{AND_Match}(DDDD11D00DDDDD, X)).
\end{aligned}$$

These measures work by counting the number vertical or horizontal edge segments in an image, where an edge segment is a 0 to 1 transition reinforced by a similar occurrence directly adjacent. They indicate the amount of vertical (or horizontal) edge activity in a binary pyramid. More elaborate sets can easily be constructed.

5. Graphics Operations

Two applications of HCL to computer graphics are described here. One is the rapid filling of a bounded region of an image with a given label. The other is the rapid plotting of curves and surfaces.

5.1. *Fast Fill*

A number of commercially available graphics systems provide an operation called "polygon fill," "paint," or "region fill," which causes a connected set of pixels of a two-dimensional digital image to be labeled with a given value. This operation is often the bottleneck in interactive graphics programs that present pictures made up of colored-in regions.

The region-filling problem may be stated as follows: One is given two binary images, A and B, such that A has a 1 at only a single cell (the "seed" pixel), and such that B consists of one or more connected components (of 1's), one of which includes the seed cell. From these, it is desired to produce a third binary image, C, which has 1's precisely at those cells which comprise the connected component in B which includes the seed pixel in A. This problem may be stated in other ways in which one may allow nonbinary values at cells or have a region of 0's to be filled in rather than a region described by 1's bounded by 0's, etc. Such problems are minor variations on the one treated here.

A straightforward solution in HCL to the region-filling problem is

$$C = \text{FlatFill}(X,Y) = [\text{OR\_Match(Plateral)} \mid Y]^*(X),$$

where Plateral = $D111111111DDDD$. In each iteration of OR_Match function application, the labeling moves one cell further out from the seed cell. However, this propagation is restricted to take place only at cells where Y is a 1. Thus the number of iterations required is equal to the length of the longest path in Y between the seed cell and another cell belonging to the same connected component in Y. The diameter of a region is the longest such distance possible in the region, for any placement of the seed cell within the region. Note that the distance in this case is the minimum number of "chess king move" steps required to get from one cell to another. The FlatFill function requires a number of steps on the order of the diameter of the region to be filled.

An alternative HCL solution allows (in the general case) the propagation to use higher levels of the hierarchical domain, and it achieves much faster filling when the region is large and relatively compact.

$$\text{PyramidFill}(X,Y) = [\text{OR\_Match(Pall)} \mid \text{ANDPYR}(Y)]^*(X),$$

where Pall = 11111111111111. PyramidFill requires a preliminary step, the

construction of ANDPYR(Y), which itself may require up to L iterations of an AND_Match operation. However, this operation may be performed just one time and many regions in Y may be labeled by PyramidFill. For a large, compact, region (one relatively free of holes, cracks, and isthmuses), the labeling propagates out from the seed cell not only laterally but up into higher levels of the hierarchical domain. Successive iterations have the effect of spreading the label across the region very rapidly because the diameter of the region is much reduced at such levels. The labeling then propagates back down, filling in large areas of the region in level L at once.

The results of computing FlatFill and PyramidFill are similar. The results in level L are identical. As explained, PyramidFill labels some cells in upper levels, and these are also part of its result. They can be eliminated by ANDing with Q_L. For regions of the appropriate type (described subsequently), PyramidFill requires a number of steps proportional to the logarithm of the diameter. Not counting the construction of ANDPYR(Y), in the worst case, PyramidFill uses the same number of steps as FlatFill.

Regions that can be efficiently filled with PyramidFill are those whose "pyramidal diameter" is small. The *pyramidal diameter* of a region R is the longest pyramidal distance between some pair of points in R. The *pyramidal distance* between a pair of cells A and B in R is the length of a shortest path between A and B that stays within the cells labeled 1 in the AND pyramid constructed from R. The path is a chain of arcs each of which links one cell to one of its pyramidal neighbors. More intuitively, a region can be efficiently filled by PyramidFill if it is relatively convex or blobular and does not contain long thin portions in its shape, and does not contain lots of small holes or deep intrusions. However, in the worst case, the running time of PyramidFill is approximately equal to that of FlatFill.

A study of the space complexity of quadtrees [7] treats related issues. In the case of PyramidFill, however, the fact that labeling is modeled as parallel in the pyramid, and the fact that full pyramidal interconnections are assumed rather than only the quadtree interconnections, makes a separate analysis necessary. No further speedup of PyramidFill can be made by decomposing the region R into smaller, possibly convex, ones. The time required by PyramidFill is usually much less than the number of cells in R. In the worst case, it is essentially equal to R (ignoring the setup step of computing ANDPYR(Y)).

5.2. *Function Plotting*

Any function $y = f(x)$ which can be reexpressed as a decision function $F(i, j)$ which has an algorithm requiring a fixed (data-independent) sequence of steps (and is thus appropriate for an SIMD system) involving simple arithmetic can be plotted in level L of a hierarchical domain using a number of steps depending only upon the precision of the plot desired. The number L of bits

representing each cell's row or column coordinate determines the length of the bit-serial arithmetic computations that may be necessary to evaluate $F(i, j)$ at each cell. This assumes that the cells work "independently," rather than in groups called "clerks" [25].

5.3. *Progressive Refinement*

It is interesting to note that in interactive graphics applications in which there is a bottleneck in transmission bandwidth between an image source and the user's CRT, a hierarchical technique for transmission may sometimes be appropriate. The potential advantage of the scheme to be described is that rough versions of the entire image arrive on the display early, so that the user may obtain an overall impression of the image long before all its details arrive [24]. The transmitting agent first creates a bit pyramid Trans(X) from the original binary image stored in level L of X.

$$\text{Trans}(X) = [\text{AND\_Match(PNWson)} \,|\, (1-Q_L)]^*(X),$$

where PNWson = *DDDDDDDDD*1*DDD*. Then the transmitter sends selected data from this pyramid in a particular order: Beginning with the root, and ending with level L, the levels are scanned in a raster-scan order, and the bits visited are transmitted in a stream, but the cells which are northwest sons are skipped over, and their values not sent. The root is considered in this special case not to be a northwest son, so that its value is sent.

The receiver, in synchrony with the transmitter, fills a hierarchical domain with the received values, again passing over northwest sons. However, the receiver interleaves two other activities with the filling process. One of these activities is the projection of values downward in the hierarchical domain, and the other is the reading of values out of level L to refresh a display. The projection downward serves two purposes. One of these is to get data to level L where they can be displayed. The other is to replace northwest son data that were extracted from each sibling group before transmission. Assuming that Y is the pyramid resulting just after cell (k, i, j) has been filled, and that Y' is the pyramid just after the projection has been done (and before the next cell is filled), we have

$$Y' = \text{RProject}_k{}^{L-k}(Y),$$

where

$$\text{RProject}_m(X) = [\text{AND\_Match(Pfather)} \,|\, (1 - Q_m)](X).$$

This new projection function differs from Project in not allowing level m to change. That is, RProject copies data down from level m without destroying the contents of level m itself. After each projection, the image to be displayed

is in level L. If the original image shows large, compact solid-colored objects, these objects will be visible in the display long before all the data have arrived. It is easy to see, also, that the data bits transmitted are exactly those of the original image, but that the ordering is different from the usual raster-scan order.

6. Image Analysis Operations

HCL offers two potential benefits in the description of image analysis operations. One benefit is concise and accurate description. The other is fast algorithms implied by the HCL expressions, assuming that the HCL primitives have been implemented efficiently enough.

6.1. *Binary Edge Detection*

Two edge detection procedures for handling noisy binary images have recently been described [10]. They may both be described effectively with HCL expressions. For brevity, only the first is described here. Let B be the bit pyramid containing the input binary image in level L. Then the following definitions compute the clean edge image in E:

$$
\begin{aligned}
E0(X) &= [\text{AND\_Match(Psons)} \,|\, (1 - Q_L)](X) \\
E1(X) &= E0(X) * \text{OR\_Match(Pedge, } E0(X)) \\
E2(X) &= \text{OR\_Match(Plateral, } E1(X)) \\
E3(X) &= \text{Project}(E2(X)) \\
\text{EdgeHull}(X) &= \text{AND\_Match(Plateral, } E3(X)) \\
\text{EdgeHull2}(X) &= \text{EdgeHull}(-X) \\
E4(X) &= \text{OR\_Match(Plateral, EdgeHull2}(X)) \\
\text{CleanEdges1}(X) &= \text{EdgeHull}(X) * E4(X) \\
E &= \text{CleanEdges}(B).
\end{aligned}
$$

Here $E0(X)$ generates level $L - 1$ only of ANDPYR(X). $E1(X)$ generates a rough edge representation. EdgeHull(X) represents the edge image with exterior noise removed. EdgeHull2(X) represents edges with interior noise (only) removed. The final clean edge image is obtained by growing the internally cleaned edges enough to obtain overlap with the externally cleaned ones, and then taking the intersection of these two bit pyramids.

6.2. *Bright Spot Detection*

The problem of generating good seed points for region extraction can be solved easily with a hierarchical approach. There are several versions of the problem and there are even more solutions.

The simplest version of the problem assumes that a binary image is given in level L of a bit pyramid Y. This image presumably consists of a collection of nontrivial connected regions of 1's and is presumably to be used as a mask for region filling as described earlier in this paper. It is desired to obtain, in a very small number of steps, a bit pyramid X containing a single 1 which must lie in the interior of one of the connected regions of Y. That is, X must satisfy $X = X * Y$, and $\text{COUNT}(X) = 1$. A simple strategy for constructing X is to construct $\text{ORPYR}(Y)$ and then steer a 1 down the hierarchical domain from the root, so that at each step it is within $\text{ORPYR}(Y)$, and therefore finishes up in Y. Such a procedure clearly uses a number of steps proportional to L. Some care must be taken to ensure that no more than one son of the currently selected cell becomes selected at the next step.

The basic step of this procedure must place a 1 in level $k + 1$ at one of the four sons of the cell marked 1 in level k. It begins by marking any northwest son in level $k + 1$ whose father is marked 1, provided the cell itself has value 1 in $\text{ORPYR}(Y)$. Then it marks northeast sons that satisfy that condition plus the condition that their west neighbor is not marked. The other two types of sons are similarly handled.

$$
\begin{aligned}
X_0 &= Q_0 \\
X_{k+1,\text{NW}} &= \text{ORPYR}(Y) * \text{QNW} * \text{Project}(X_k) \\
X_{k+1,\text{NE}} &= (\text{ORPYR}(Y) * \text{QNE} * \text{Project}(X_k)) * -\text{ShiftEast}(X_{k+1,\text{NW}}) \\
X_{k+1,\text{SW}} &= (\text{ORPYR}(Y) * \text{QSW} * \text{Project}(X_k)) * -\text{ShiftSouth}(X_{k+1,\text{NW}}) \\
&\quad * -\text{ShiftSouthWest}(X_{k+1,\text{NE}}) \\
X_{k+1,\text{SE}} &= (\text{ORPYR}(Y) * \text{QSE} * \text{Project}(X_k)) \\
&\quad * -\text{ShiftSouthEast}(X_{k-1,\text{NW}}) \\
&\quad * -\text{ShiftSouth}(X_{k+1,\text{NE}}) * -\text{ShiftEast}(X_{k+1,\text{SW}}) \\
X_{k+1} &= X_{k+1,\text{NW}} + X_{k+1,\text{NE}} + X_{k+1,\text{SW}} + X_{k+1,\text{SE}}.
\end{aligned}
$$

It is interesting here to note that, although it is beyond the scope of this paper to demonstrate it, a similar scheme may be made to work with byte pyramids or real-valued pyramids to find a bright spot in an image in L steps. Each of the L steps labels a search successor by moving the label to the son having the maximum value in the "search pyramid" (in the example above, $\text{ORPYR}(Y)$ is the search pyramid), which may be constructed using an averaging operator or a maximizing operator [29].

6.3. *Thresholding and Threshold Detection*

Let $B(n-1), B(n-2), \ldots, B(0)$ be n bit pyramids representing a byte pyramid BB. (Normally $n = 8$ but this is not necessary.) Let $T(n-1)$, $T(n-2), \ldots, T(0)$ be the n bit pyramids of the binary representation of a threshold byte pyramid TT. The thresholding of BB by TT is defined:

$$\text{Threshold}(BB) \text{ at } (k,i,j) = \begin{cases} 1 & \text{if } BB \text{ at } (k,i,j) \geq TT \text{ at } (k,i,j) \\ 0 & \text{otherwise.} \end{cases}$$

HCL expressions for Threshold(BB) can be derived, but are omitted here.

In a hierarchical domain there are several interesting approaches for solving the problem of threshold selection [21]. One of these is to use bit counting to determine global statistics (e.g., all or portions of a histogram) from which a global threshold may be computed [29, 22]. The other is to an average pyramid and then let TT at (k, i, j) be a weighted combination of the average pyramid values along the path from (k, i, j) to the root.

6.4. *Filtering with Means and Medians*

For an intermediate level k of a hierarchical domain, each cell (k, i, j) may be considered as the root of a subpyramid (actually another hierarchical domain). Any algorithm designed to run in a hierarchical domain may be made to run, in parallel, on such a collection of subdomains. Image filtering generally requires the calculation, at each lateral neighborhood of the image, of a new value, such as the mean or median over the neighborhood. Through the use of a sorting procedure [30], efficient median filtering may be effected using SIMD operations on pyramids.

7. Conclusions

A logic has been presented in which operations are applied to objects called bit pyramids which themselves are functions on spaces called hierarchical domains. The logic is motivated on the one hand by the need for good computation models for the parallel processing of images and, on the other hand, by a desire to capture some notion of hierarchy, which is manifest in biological vision systems, in a parallel computation system. The system presented here more closely matches the characteristics of proposed pyramidal computer architectures than do previous models, which have been based on automata theory.

The logic permits many image processing operations to be formulated in an algebraic way. While such formulations are precise and unambiguous, they are relatively concise and facilitate such manipulations as simplification and verification. Because of the logic's relationship of pyramidal computers, expressions of the logic are easily translated into programs, and the resulting

procedures are very often highly efficient, making good use of the parallelism and hierarchy available.

It has been shown here that a hierarchial parallel system of computation can count the number of 1's in its base level (of size $N \times N$) in time proportional to log N, even when such a system is restricted to SIMD operation, thus improving the previously known result that an essentially MIMD system could achieve that bound.

Properties of OR, AND, and count-of-two pyramids have been given related to the preservation of connectedness of objects across levels. A fast parallel method for the region-fill problem has been presented. The bright-spot detection problem has been formulated and a solution presented. Finally, additional applications of the hierarchical cellular logic in graphics and image processing have been illustrated.

ACKNOWLEDGMENTS

This paper has been improved through the constructive comments of Mr. B. Bennett, Professor C. Dyer, Mr. T. Ligocki, and Professor L. Uhr. Parts of the work represented here were performed while the author was a visiting professor at the Institut de Programmation, Université de Paris-VI, made possible by Professor J.-C. Simon, and at the University of Linköping, Sweden, made possible by Professor G. Granlund. The author gratefully thanks his hosts for their support and encouragement.

REFERENCES

1. Burt, P. J. Tree and pyramid structures for coding hexagonally sampled binary images. Tech. Rep. TR-814, Computer Science Center, University of Maryland, College Park, Md., Oct. 1979.
2. Crettez, J.-P., and Simon, J.-C. A model for cell receptive fields in the visual striate cortex. *Computer Graphics and Image Processing* **20** (1982), 299–318.
3. Dubitsky, T., Wu, A. Y., and Rosenfeld, A. Parallel region property computation by active quadtree networks. *IEEE Trans. Pattern Recognition and Machine Intelligence* **PAMI-3,** 6 (1981), 626–633.
4. Duff, M. J. B. CLIP4: A large scale integrated circuit array parallel processor. *Proc. Third Int. Joint Conf. on Pattern Recognition,* 1976, pp. 728–733.
5. Duff, M. J. B. Propagation in cellular logic arrays. *Proc. Workshop on Picture Data Description and Management,* Asilomar Conf. Grounds, Pacific Grove, Calif., IEEE Computer Soc., Aug. 1980, pp. 259–262.
6. Dyer, C. R. A VLSI pyramid machine for hierarchical parallel image processing. *Proc. PRIP '81, The IEEE Computer Soc. Conf. on Pattern Recognition and Image Processing,* Dallas, Tex., Aug. 1981, pp. 381–386.
7. Dyer, C. R. The space efficiency of quadtrees. *Computer Graphics and Image Processing* **19** (1982), 335–348.

8. Dyer, C. R., and Rosenfeld, A. Cellular pyramids for image analysis. Tech. Rep. No. 544, Computer Science Center, University of Maryland, College Park, Md., 1977.
9. Feldman, J. A., and Ballard, D. H. Connectionist models and their properties. *Cognitive Sci.*, 1982.
10. Gangolli, A. R., and Tanimoto, S. L. Two pyramid machine algorithms for edge detection in noisy binary images. *Inform. Process. Lett.* **17** (Nov. 8, 1983), 197–202.
11. Granlund, G. H. The GOP parallel image processor. In Bolc, L., and Kulpa, Z. (Eds.). *Digital Image Processing Systems*. MacMillan & Co. London, 1981.
12. Hanson, A. R., and Riseman, E. M. Design of a semantically directed vision processor. Tech. Rep. No. COINS-74-1, University of Massachusetts, Amherst, Mass., 1974.
13. Hanson, A. R., and Riseman, E. M. Processing cones: A computational structure for image analysis. In Tanimoto, S. L., and Klinger, A. (Eds.). *Structured Computer Vision: Machine Perception through Hierarchical Computation Structures*. Academic Press, New York, 1980, pp. 101–131.
14. Hubel, D. H., and Wiesel, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.* **160** (1962), 106–154.
15. Hunter, G. M., and Steiglitz, K. Operations on images using quad trees. *IEEE Trans. Pattern Analysis and Machine Intelligence* **PAMI-1,** 2 (April 1979), 145–153.
16. Levine, M. D. Region analysis using a pyramid data structure. In Tanimoto, S. L., and Klinger, A. (Eds.). *Structured Computer Vision: Machine Perception through Hierarchical Computation Structures*. Academic Press, New York, 1980, pp. 57–100.
17. Polyak, S. L. *The Vertebrate Visual System*. Univ. of Chicago Press, Chicago, Ill., 1957.
18. Preston, K., Jr., Duff, M. J. B., Levialdi, S., Norgren, P. E., and Toriwaki, J.-I. Basics of cellular logic with some applications in medical image processing. *Proc. IEEE* **67,** 5 (1979), 826–855.
19. Reeves, A. R. Methods for global feature computation in SIMD arrays. *Computer Graphics and Image Processing* **14,** (1980), 159–169.
20. Rosenblatt, F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, Washington, D.C., 1962.
21. Rosenfeld, A., and Kak, A. C. *Digital Picture Processing*. Academic Press, New York, 1976.
22. Rosenfeld, A. *Picture Languages*. Academic Press, New York, 1981.
23. Shneier, M. Two hierarchical linear feature representations: Edge pyramids and edge quadtrees. *Computer Graphics and Image Processing* **17,** 3 (Nov. 1981), 211–224.
24. Sloan, K. R., Jr., and Tanimoto, S. L. Progressive refinement of raster images. *IEEE Trans. Comput.* **C-28,** 11 (Nov. 1979), 871–874.
25. Stout, Q. F. Drawing straight lines with a pyramid cellular automaton. *Inform. Process. Lett.* **15,** 5 (Dec. 1982), 233–237.
26. Tanimoto, S. L., and Pavlidis, T. A hierarchical data structure for picture processing, *Computer Graphics and Image Processing* **4** (1975), 104–119.
27. Tanimoto, S. L. A pyramid model for binary picture complexity. *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, Troy, N.Y., June 1977, pp. 25–28.
28. Tanimoto, S. L., and Klinger, A. (Eds.). *Structured Computer Vision: Machine Perception through Hierarchical Computation Structures*. Academic Press, New York, 1980.
29. Tanimoto, S. L. A pyramidal approach to parallel processing. *Proc. 10th Annual Int. Symposium on Computer Architecture*, Stockholm, Sweden, June 14–17, 1983, pp. 372–378.

30. Tanimoto, S. L. Algorithms for median filtering of images on a pyramid machine. In Duff, M. J. B. (Ed.). *Computing Structures for Image Processing*. Academic Press, New York, 1983, pp. 123–141.

31. Uhr, L. Layered "recognition cone" networks that preprocess, classify and describe. *IEEE Trans. Comput.* **21** (1972), 758–768.

32. Uhr, L. Psychological motivation and underlying concepts. In Tanimoto, S. L., and Klinger, A. (Eds.). *Structured Computer Vision: Machine Perception through Hierarchical Computation Structures*. Academic Press, New York, 1980, pp. 1–30.