Fast Line Detection in a Hybrid Pyramid

Ze-Nian Li Danpo Zhang
School of Computing Science
Simon Fraser University
Burnaby, B.C. V5A 1S6

Abstract

A hybrid pyramid multiprocessor vision machine has been built for real-time vision applications. The pyramid has 512 one-dimensional single-instruction multiple-data (SIMD) array processors at the bottom and a 63-node transputer-based multiprocessor system on the top. Parallel hardware links are developed to enable parallel data communications between the array processors and the transputers. This paper presents algorithms and implementation of fast Hough line detection in the hybrid pyramid. Live input images are preprocessed in the array processors. Transputers on the top merge edge pixels into short line segments and then into longer lines hierarchically in the hybrid pyramid. The lines are represented using their Hough parameters ρ and θ. To reduce the complexity of the line merging process, a Dynamically Allocated Quadtree (DAQ) is introduced to represent the Hough parameter space. Comparative experimental results are presented. The fast pyramidal line detection algorithm has been integrated into a robotic workcell, where a stream of varied wooden blocks are presented to the vision system via a conveyor belt and sorted in real-time, which demonstrates that the hybrid pyramid vision machine has its great potential in industrial automation.

1 Introduction

Pyramid structure is now quite popular and favorable especially in parallel computer vision, because it supports the multiresolution approaches and capitalizes on advantages of both the mesh and the pipeline architectures. There have been many attempts in building pyramid machines. The Image Understanding Architecture at the University of Massachusetts-Amherst is by far the most ambitious pyramid-like machine under development. The large pyramid will eventually consist of three levels with 512×512 processors at the bottom, 64×64 in the middle, and 8×8 on the top, and would achieve roughly a terra-op in performance on 32-bit integer arithmetic [Rise90]. Most researchers, however, are still striving with their "prototype" pyramid machines with a base of 16×16 or 64×64.

We have built a hybrid pyramid multiprocessor vision machine for real time processing of object recognition tasks. The pyramid has the AIS-4000 1-D array processor [Schm88] at the bottom and 63 MIMD transputers on the top in the form of an augmented binary tree. It is thus called a $2DHybridPyramid$ (as opposed to common 3D pyramids built with multiresolution 2D arrays). For economic reason, the machine does not employ hundreds of transputers and is thus affordable. This paper will present algorithms and implementation of real-time Hough line detection in the hybrid pyramid. Section 2 describes the hybrid pyramid machine. Section 3 introduces the simple pyramidal merge algorithm for Hough line detection and the improved algorithm using DAQ's. Comparative experimental results are presented in Section 4. A conclusion is given in Section 5.

2 The 2D Hybrid Pyramid

Figure 1 depicts the SFU 2D hybrid pyramid vision machine. The AIS-4000 has fine-grained parallelism with 512 processors arranged in a single-instruction multiple-data (SIMD) 1-D array architecture. It can readily simulate a 2-D array and handle over 3 billion operations per second. Each transputer is a T-800 with 2 MBytes memory. Three of the four links of each T-800 node are used to form a binary tree. The remaining single link of each T-800 is connected to a programmable cross-bar switch which provides flexibility for additional desirable connectivity. For example, they can be used to form horizontal links for augmenting the binary tree as shown in Figure 1 . The single root node at level 0 interfaces with a host Sun workstation. Since the AIS-4000 is basically a 1D array of SIMD processors, we are connecting the transputer nodes into an augmented binary tree of six levels, although the transputer links do allow other flexible and reconfigurable structures. The 2D pyramid simulates 3D pyramid operations, when necessary. The 32 leaf transputer nodes (at level 5) are connected to the AIS-4000 via 64 custom-made parallel links. The key to combining the AIS-4000 and the transputer nodes into a pyramidal architecture is a high speed communication link between the AIS-4000 and the transputers. After some initial study, we ruled out the use of any single fast link because of its limitation of data transfer rate, and more importantly, its inability to transfer data to more than one transputer (or AIS processor) simultaneously.

A parallel link, called PARLink was recently built in our lab [Ens91]. The AIS-4000 has eight SIMD nodes packaged into one custom VLSI chip called a Pixie. Each Pixie chip interfaces to one 32K by eight bit static RAM chip for local image storage. Since the AIS-4000 has 512 SIMD nodes, this means there are 64 clusters of Pixie and RAM chips. This architecture lends itself to the construction of 64 parallel links connecting each cluster up to the transputer pyramid.

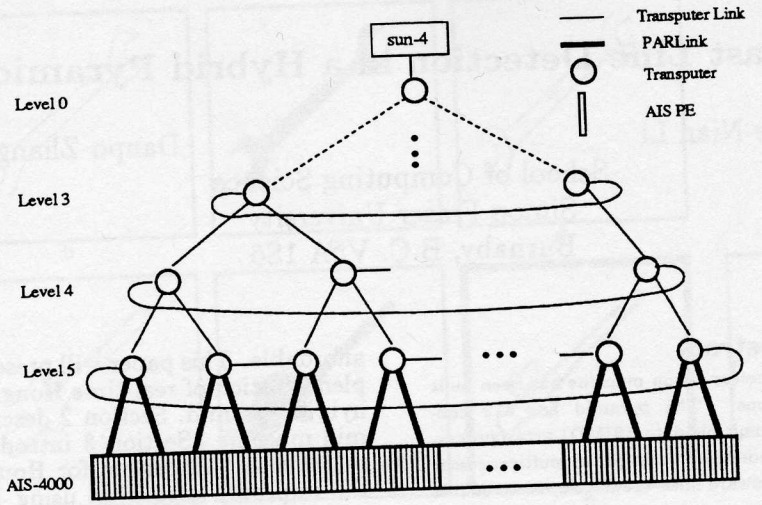


Figure 1: The SFU hybrid pyramid vision machine

Two of these links are connected to one transputer node for a binary tree configuration. The transputers are ordinarily connected with 20 Mbit/sec serial links according to an INMOS protocol. Therefore the hardware requirements for each PARLink are to read eight bit parallel data from the static RAM chips of the AIS-4000 and translate this into the INMOS serial format. This requirement is reversed for communication from the transputers to the AIS-4000. The core of each PARLink is the INMOS C012 chip, and only a few additional buffers and control signals required. All PARLinks are controlled by some custom microcode subroutines supplied by the AIS Inc. Connections are made directly to the data lines of the static RAM chips. Control signals are obtained from the AIS-4000 digital I/O channels as well as direct connections to the microcontroller. A throughput of 1 MByte/sec can be achieved for each PARLink, which allows an image to be passed in 4 msec, with a total bandwidth of 64 MBytes/sec for all of the PARLinks.

The completion of the PARLink enabled the integration of two parallel subsystems, i.e. the AIS SIMD array processor and the MIMD transputer network, into a hybrid pyramid. The massive parallelism characterized by the array processors is naturally embedded in this pyramid machine, because each level of the pyramid is itself an array. Moreover, much more interesting and complex algorithms can be implemented in the hybrid pyramid, since the transputers are powerful computing machines and they operate in a hierarchical MIMD mode.

3 The Pyramidal Line Detection Algorithm

The pyramid vision machine supports the multiresolution approaches and capitalizes on advantages of both the mesh and the pipeline architectures. A simple way of pipelining in the pyramid is to assign different functions to the multiple levels which are of progressively reduced resolutions, and the pipelining occurs between these levels. For a real-time system in which image data come in from the bottom continuously, this multilevel bottom-up information flow and abstraction can provide substantial speedup.

For example, the pyramidal Hough algorithm for line detection by Jolion and Rosenfeld [Joli89] can readily be implemented in this fashion. Briefly, each transputer at the bottom level of the pyramid examines a vertical stripe of the edge image and merges edge pixels into short lines. The short lines are then passed up to the parent nodes and merged recursively up in the pyramid. A practical threshold K for the number of lines reported by each transputer can be set to avoid excessive number of short lines and noise being detected.

3.1 A Simple Line Merging Algorithm

The central part of the pyramidal line detection algorithm is line merging. The following is a modified version of the line merging algorithm originally suggested by Jolion and Rosenfeld [Joli89]. In our line representation, we avoid their use of a reference point $O = (\rho cos\theta, \rho sin\theta)$ for a line, and the use of $OP = \lambda(-sin\theta, cos\theta)$ for a point P on the line. As a result, the checking of line collinearity is more efficient, since it does not bear the computational cost of the expensive trigonometric functions.

Line Segement Representation

A line L can be uniquely defined by a pair of Hough parameters (ρ, θ) [Duda72]. In addition, coordinates of their end points are recorded. A line having an orientation of $45° \leq \theta < 135°$ or $225° \leq \theta < 315°$ is considered as x-major, otherwise, it is y-major. A parameter *flag* is used to indicate whether a line is x-major or not. Finally, another parameter *num* will

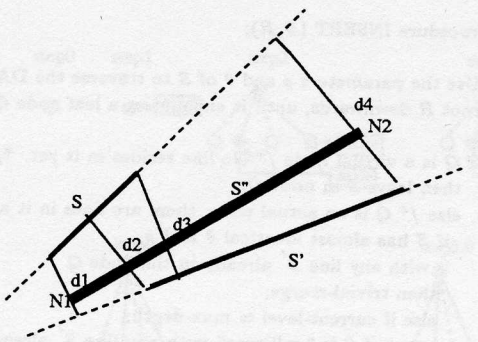


Figure 2: Detection of collinearity and merging

represent the current number of pixels on the line segment. In summary, a line segment is represented by:

$$S = (\rho, \theta, x_1, y_1, x_2, y_2, flag, num)$$

Collinearity Checking of Line Segments

The collinearity of two similarly oriented line segments is determined by simply measuring the maximum distance between them. Consider $S \subset L$ and $S' \subset L'$, the distance between the two line segments is defined by:

$$d(S, S') = max[d_1(P_1, L'), d_2(P_2, L'), d_1'(P_1', L), d_2'(P_2', L)],$$

where P_1, P_2, and P_1', P_2' are endpoints of S and S', and d_1, d_2, and d_1', d_2' are their distances to the other line, respectively (see Fig. 2).

Algorithm (Simple Merging)

```
if two line segments S and S' have almost identical θ and ρ
  then trivial-merge;
  else if their θ's differ too much
   then trivial-reject;
   else      /* Collinearity checking */
    if d(S, S') is less than a pre-determined threshold
      /* non-trivial merge */
     then merge S and S' into a new line segment S";
end;
```

As shown in Figure 2, for x-major lines, two of the four end points that have the smallest and the largest x coordinate values will be used to determine the new end points N_1 and N_2. The relative position of N_1 (or N_2) with respect to S and S' is affected by the numbers of the pixels on S and S', i.e., the line with a larger num will possess a heavier weight and hence pull the new end point toward it. Similar methods will be used for other situations, e.g., merging y-major lines, or merging x-major and y-major lines.

It should be emphasized that the cost of trivial-merge or trivial-reject is minimal. If two lines cannot be trivially merged or rejected, then the collinearity checking had to be invoked which is computationally expensive.

The Jolion and Rosenfeld paper [Joli89] claims that their line merging algorithm has a time complexity of O(log N), where N is the size of the image. Beside the severe limitation on the number K of lines allowed to be reported by each processor, a major assumption is that the algorithm is implemented in a massive pyramid with a 2D array processor at its base that is nearly as large as the image size, i.e., the height H of the pyramid is not much less than $M = \log_2 N$. While working with a smaller pyramid, e.g., the short ("half-scale") pyramid as ours, the algorithm will slow down drastically. Assume the image has a resolution $2^M \times 2^M$ and the base of the pyramid has $2^H \times 2^H$ nodes, then $h = M - H$. If h is not small, then the leaf nodes are overloaded and become the bottleneck.

In a previous paper [Li91], we demonstrated that several effective and flexible pyramidal pipeline techniques are especially suitable for our 2D pyramid. Under the flexible pipelining, processors at each level can have relatively balanced load. However, the increase of the speed has a upper bound of 100%, because there are not quite 100% more parent nodes that can share the work of the leaf nodes in a binary tree configuration.

The simple merging algorithm is implemented and the results will be shown in Section 4. As some of the timing results show, when the number of lines in the test image increases, the time for line detection increases quadratically. This is because the worst case time complexity for merging two groups of lines is O(PQ) where P and Q are numbers of lines in these two groups. The algorithm will be especially slow if each pair of the lines must undergo a collinearity check.

3.2 An Improved Line Merging Algorithm Using DAQ

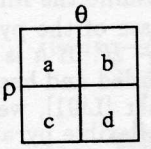
As pointed out in the previous subsection, the original simple line merging algorithm has several drawbacks. One of its major problems is the unstructured organization of the partially detected line segments. When a parent processor attempts to merge partial results from its two child processors, it has to deal with two potentially long linked lists. Since lines are represented by points in the Hough parameter ($\rho - \theta$) space, a potential cure is to partition the parameter space into subspaces of smaller sizes. For instance, if the number of the lines in each of the two child parameter spaces can be split evenly into M subspaces, the worst case time complexity will become $M \times (P/M \times Q/M) = PQ/M$. Nevertheless, in order to quickly construct and traverse the subspaces, an efficient and dynamic indexing mechanism for the Hough space must be developed.

3.2.1 The Dynamically Allocated Quadtree (DAQ)

Quadtree has become a popular data structure. Its various applications and variations were reviewed extensively by Samet in his 1984 paper [Same84]. A quadtree for representing the Hough parameter space

is a tree where each node represents a certain range of the parameter space $(\rho_{min}, \rho_{max}, \theta_{min}, \theta_{max})$. A finer resolution of this parameter space can be found in 4 children of this node, each child represents one of four subspaces of the current node:

Child a: $(\rho_{min}, \frac{\rho_{min}+\rho_{max}-1}{2}, \theta_{min}, \frac{\theta_{min}+\theta_{max}-1}{2})$
Child b: $(\rho_{min}, \frac{\rho_{min}+\rho_{max}-1}{2}, \frac{\theta_{min}+\theta_{max}+1}{2}, \theta_{max})$
Child c: $(\frac{\rho_{min}+\rho_{max}+1}{2}, \rho_{max}, \theta_{min}, \frac{\theta_{min}+\theta_{max}-1}{2})$
Child d: $(\frac{\rho_{min}+\rho_{max}+1}{2}, \rho_{max}, \frac{\theta_{min}+\theta_{max}+1}{2}, \theta_{max})$



An index to the quadtree that contains a string composed of characters a, b, c, and d can be used for each node. Root (level 0) node has an empty string of index, while 4 children of the root (level 1) have index a, b, c, d, respectively. The nodes at the next level (level 2) will have index $aa, ab, ac, ad, ba, ..., dc, dd$. Thus, a unique index is associated with each node for fast access of a certain node in the tree. A list of line segments is allocated within each leaf node of the quadtree, whereas no line segment is associated with any internal tree nodes.

In their early papers [O'Ro81, Sloa81], O'Rourke and Sloan presented Dynamically Quantized Hough Spaces and Dynamically Quantized Pyramids. Both focused on the quantization problem, i.e., using a limited number of cells to represent the parameter space so that fine precision is maintained where it is needed. In this paper, quadtrees are used for different purposes, namely, they are used for dynamic construction of Hough subspaces and, subsequently, for quick indexing in the merging process. We call them Dynamically Allocated Quadtrees (DAQ's) because they are dynamically constructed by the transputer leaf nodes. We always equally divide a 2D Hough space into four quadrants when it is necessary to split a DAQ node. In this way, an efficient algorithm can be developed for combining two DAQ's at the parent node.

3.2.2 Building initial DAQ's

The following recursive procedure INSERT is performed by the transputer leaf nodes in the pyramid when they combine edge pixels into short line segments and build the initial DAQ's for these line segments in their subimages. Max-depth is a adjustable parameter which defines the maximal depth of the DAQ. Since the collinearity checkings for non-trivial merges are only performed among line segments at max-depth level of the DAQ, the insertion routine is reasonably fast [1].

[1] It should be pointed out that the recursive procedure INSERT provides a concise description. In real implementation, some modifications/optimizations can make the procedure more efficient.

```
Procedure INSERT (S, R);

Use the parameters ρ and θ of S to traverse the DAQ from the
root R downwards, until it encounters a leaf node Q of the DAQ

if Q is a virtual node /* No line resides in it yet. */
  then leave S in node Q;
  else /* Q is an actual node, there are lines in it already. */
   if S has almost identical θ and ρ
     with any line S' already in the node Q
    then trivial-merge;
    else if current-level = max-depth;
     then if S is "collinear" with any line S' already in the Q
       then non-trivial merge;
       else link S into the list of the lines at Q;
     else      /* split Q */
       generate 4 virtual child nodes for Q;
       for all lines NewS ∈ {S, all lines in Q}
         INSERT (NewS, Q);
end; {INSERT}
```

Figure 3 illustrates a sequence of insertions taken place in a transputer leaf node. Numbers in boxs indicate ρ and θ of lines. Initially, the root of the DAQ is a virtual node which has no line associated with it yet. A line (60, 250°) is inserted at Step 1. Step 2 sees the second inserted line which causes the split of the root node. A new line (61, 250°) is trivially merged with the old line (60, 250°) at Step 3/3.1, and a new longer line (60, 250°) is generated. At Step 4, another line (12, 170°) is inserted. Because its ρ value 12 is fairly close to ρ value 40 of the previous line in node b, new children nodes are recursively generated until the two lines reach the deepest allowable level (assume max-depth = 4) and fit in nodes bccc and bccd. Finally, the last inserted line (36, 171°) finds its place at node bccd at Step 5/5.1. Since it is almost collinear with line (40, 170°), the two lines are merged into a new line (39, 170°).

3.2.3 Combining two DAQ's

The function of a parent node is to combine the two DAQ's from their two children to generate a new DAQ. In the combining process, collinear short line segments are merged into longer lines. Suppose there are two trees DAQ1 and DAQ2, an efficient algorithm is to first traverse only one tree, say DAQ2, and examine each leaf node Q_2 in sequence. The procedure INSERT is used to place the lines in Q_2 into their corresponding nodes in DAQ1. As a result, DAQ2 is appended to DAQ1.

```
Procedure COMBINE (DAQ1, DAQ2);

  Traverse DAQ2 and examine each leaf node Q2 in sequence;
  for each Q2
   if there is a (real, virtual, or internal) node Q1
        in DAQ1 at the corresponding position as Q2 in DAQ2
    then Q = Q1;
    else follow the child-parent link in DAQ1 to find
```

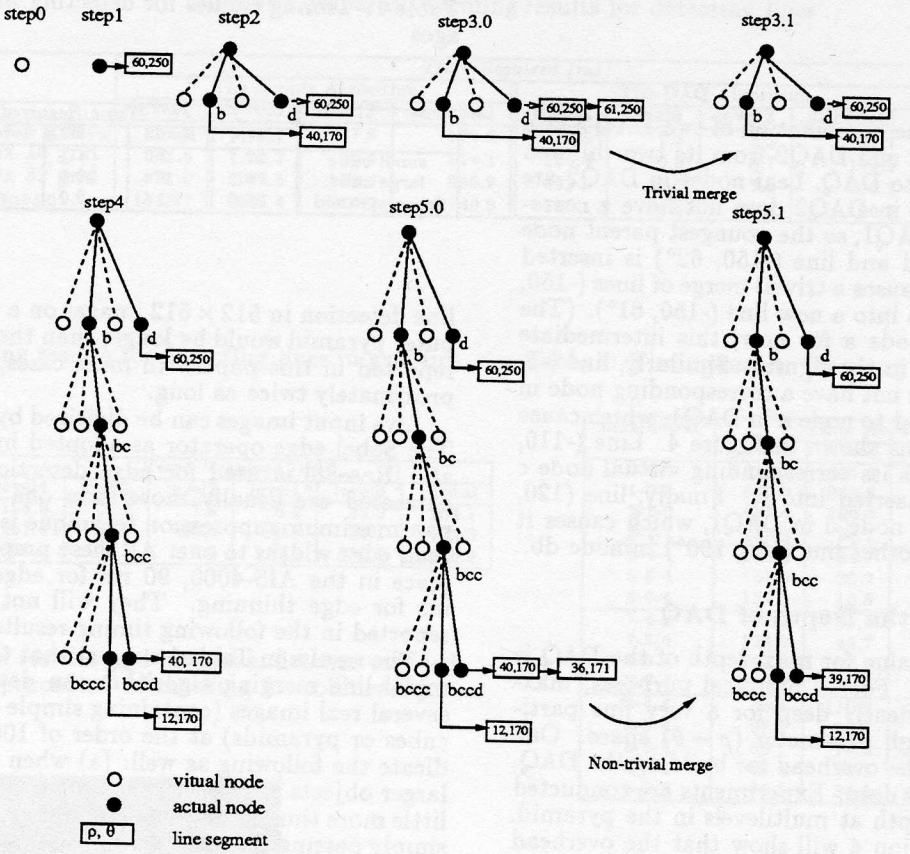


Figure 3: Building a DAQ at a transputer leaf node

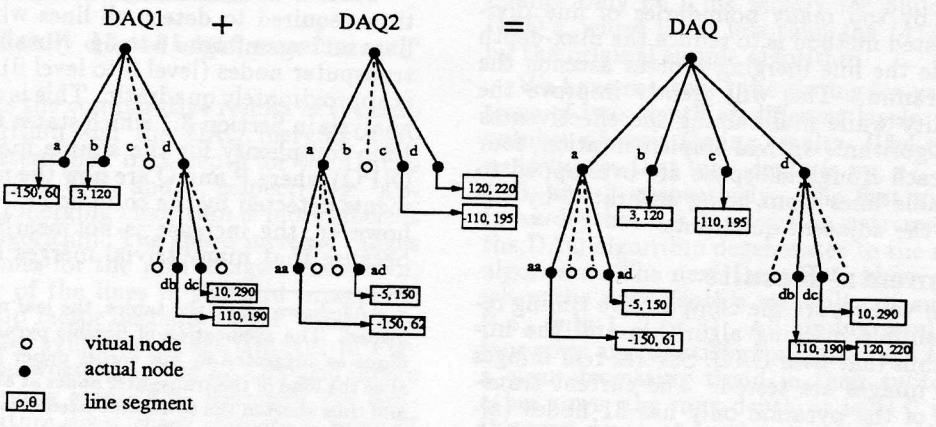


Figure 4: An example of combining two DAQ's

```
        the youngest parent node Q1';    Q = Q1';
    for all lines S in Q2
      INSERT (S, Q);
end; {COMBINE}
```

Figure 4 shows the combination of two DAQ's at a parent node. DAQ1 and DAQ2 from its two children will be combined into DAQ. Leaf nodes in DAQ2 are examined. Node aa in DAQ2 does not have a corresponding node in DAQ1, so the youngest parent node a in DAQ1 is found and line (-150, 62°) is inserted into node a, which causes a trivial merge of lines (-150, 60°) and (-150, 62°) into a new line (-150, 61°). (The new line stays in node a for now, this intermediate result is not shown in the figure.) Similarly, line (-5, 150°) in DAQ2 does not have a corresponding node in DAQ1 and is inserted to node a in DAQ1, which cause the split of node a as shown in Figure 4. Line (-110, 195°) in DAQ2 finds its corresponding virtual node c in DAQ1, and is inserted into it. Finally, line (120, 220°) is inserted to node d in DAQ1, which causes it to be linked to the other line (110, 190°) in node db.

3.2.4 Choosing the Depth of DAQ

The choice of the value for max-depth of the DAQ is an important issue. For all practical purposes, max-depth = 8 is sufficiently deep for a very fine partitioning of the Hough parameter ($\rho - \theta$) space. One of the concerns is the overhead for building the DAQ, especially when it is deep. Experiments are conducted by varying max-depth at multilevels in the pyramid. Our results in Section 4 will show that the overhead for building a deep DAQ is low and acceptable and has a very good payoff.

Another concern is the detectability of lines in the image. If deep DAQ's are maintained at all levels in the pyramid (including the root transputer), then the Hough space is always divided into many fine subspaces. Excessive number of lines might be reported by the root transputer, even if many of them only deviated by a tiny $d\rho$ and/or $d\theta$. It is because they are separated by too many boundaries of fine divisions. A suggested method is to reduce the max-depth gradually while the line merging process ascends the transputer pyramid. This will greatly improve the line detectability while maintaining the effectiveness of the DAQ algorithm. In real implementation, four quadrants of each Hough subspace are overlapped to prevent mergable lines from being separated by the boundaries of the adjacent quadrants.

4 Experimental Results

This section will report the comparative timing results for the simple merging algorithm and the improved algorithm that uses DAQ. Several real images and synthetic images are tested. The current transputer portion of the pyramid only has 31 nodes (although it will be upgraded to 63 nodes soon), of which 16 are transputer leaf nodes. It can be viewed as half of the final scale. The test images used in this paper are thus of the size of 256 × 256. The timing for line detection in 512 × 512 images on a 63-node transputer pyramid would be longer than the timing results reported in this paper. In most cases, it will be approximately twice as long.

Table 1: Timing results for detecting lines in real images

	Time Required (ms)					
	level4	level3	level2	level1	level0	Total
small cube	37.1	2.0	1.1	0.1	1.9	42.2
large cube	76.2	6.4	10.0	8.6	3.9	105.1
large pyramid	85.0	10.4	9.0	8.6	11.0	124.0

Live input images can be digitized by the AIS-4000. The Sobel edge operator as adopted by Rosenfeld, et al. [Rose88] is used for edge detection. The edges extracted are usually more than one pixel wide. A non-maximum suppression technique is used to reduce their edge widths to one. All these preprocessings take place in the AIS-4000, 90 ms for edge detection, 20 ms for edge thinning. They will not be repeatedly reported in the following timing results.

The results in Table 1 suggest that the simple pyramidal line merging algorithm can detect all lines in several real images (containing simple objects such as cubes or pyramids) at the order of 100 ms. They indicate the following as well: (a) when the lines of the larger objects are longer, the parent nodes will need a little more time to do some real merging job instead of simply passing the results up to the root; (b) when the object has more lines, e.g., as in the "large pyramid", it will take longer to merge them[2].

To better illustrate the latter point, we are presenting a set of results below from some synthetic images. Three synthetic grid images are used. The first image has an 8 × 8 grid and hence 8+8 = 16 lines. Similarly, the second has 16 + 16 = 32 lines, and the third has 32 + 32 = 64 lines,

Table 2 shows the significant increase in the total time required to detect all lines when the number of lines increases from 16 to 64. Notably, for the non-leaf transputer nodes (level 3 to level 0), the time increase is approximately quadratic. This is consistent with the analysis in Section 3.1 which states that the worst case time complexity for the simple merging algorithm is O(PQ) where P and Q are now the numbers of line segments detected by the child nodes. For the leaf nodes, however, the increase is not nearly as drastic. It is because that many trivial merges happened to occur

[2] As shown in all the tables, the leaf nodes are clearly overloaded. The application of flexible pyramidal pipelining technique as suggested by our earlier paper [Li91] can readily balance the load of the transputer nodes at all levels of the pyramid and thus shorten the total time needed for the entire pyramidal line detection process. However, the main purpose of this section is to show the comparative results from the simple merging algorithm and the one using DAQ. For the simplicity and clarity of a fair comparison, the pipeline technique is not used.

Table 3: Comparative timing results for detecting lines

	Time Required (ms)								speedup
	The Simple Algorithm				The DAQ Algorithm				
	level4	level3	level2	subtotal	level4	level3	level2	subtotal	
8 x 8 grid	155.8	8.1	7.6	171.5	72.6	7.2	2.9	82.7	2.1
16 x 16 grid	251.5	32.2	30.4	314.1	105.4	10.8	3.7	119.9	2.6
32 x 32 grid	651.0	128.5	119.4	898.9	222.2	30.5	7.5	260.2	3.5
Image1	1413.7	365.4	370.2	2149.3	66.1	79.6	53.8	199.5	10.8

Table 2: Timing results for detecting lines in grid images

	Time Required (ms)					
	level4	level3	level2	level1	level0	Total
8 x 8 grid	155.8	8.1	7.6	8.2	25.0	204.7
16 x 16 grid	251.5	32.2	30.4	33.1	95.1	442.3
32 x 32 grid	651.0	128.5	119.4	129.5	390.5	1418.9

at the leaf level for the grid images and hence avoided the expensive procedure of checking for collinearity.

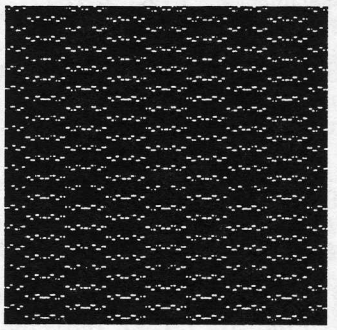


Figure 5: An artificially created edge image "Image1"

Table 3 presents the comparative results of the simple merging algorithm and the DAQ merging algorithm. The pre-determined max-depth for the DAQ's at level 4, 3, and 2 is 8, 6, and 4, respectively. Apparently, the DAQ merging algorithm is more efficient than the simple algorithm. The speed increase ranges from 2.1 to 3.5 times for the grid images. As shown, while the number of the lines in the grid images increases, the speedup factor becomes larger. This leads us to create another synthetic image (Image1 in Figure 5) which is even worse than the 32×32 grid image for the simple merging algorithm. Basically, the simple merging algorithm must check numerous pairs of lines for their collinearity. Moreover, while scanning in column-major, individual edge pixels cannot be trivially merged which makes the linked lists in the leaf

Table 4: Timing results for using various max-depth's

max-depth	*Time Required (ms)*			
	level4	level3	level2	subtotal
0-0-0	1413.7	365.4	370.2	2149.3
1-0-0	800.0	364.8	370.2	1535.0
2-1-0	431.4	180.9	382.8	995.1
3-2-1	256.8	94.1	342.4	693.3
6-4-4	157.9	34.7	40.3	232.9
6-5-4	157.9	20.2	46.2	224.3
6-6-4	157.9	10.9	53.8	222.6
7-4-4	124.2	65.9	40.3	230.4
7-5-4	124.2	44.7	46.2	215.1
7-6-4	124.2	39.4	53.8	217.4
7-7-4	124.2	19.6	103.9	247.7
8-4-4	66.1	110.8	40.3	217.2
8-5-4	66.1	85.7	46.2	198.0
8-6-4	66.1	79.6	53.8	199.5
8-7-4	66.1	64.1	103.9	234.1
8-8-4	66.1	34.2	214.8	315.1

transputer nodes even longer. The DAQ merging algorithm can finish the whole process in 199.5 ms which is more than 10 times faster than the original simple algorithm. It turns out that the overhead for building the initial DAQ trees at the leaf transputers is low and acceptable. As shown in Table3, for the quite complicated edge image "Image1", the leaf nodes at level 4 require only 66.1 ms mostly for building the initial DAQ's, which is far less than the 1413.7 ms required by the original simple algorithm.

We have measured the timing for various combinations of max-depth at different levels. Table 4 lists a number of their timing results. The notation n_1-n_2-n_3 indicates that max-depth is n_1, n_2, and n_3 at level 4, 3, and 2, respectively. The first row in the table (max-depths: 0-0-0) shows an extreme case in which the DAQ algorithm degenerates to the simple merging algorithm. The next three rows show some examples when the max-depth's are still not sufficiently large (i.e., only 1, 2 or 3). All timing results for the first four rows are unsatisfactory, although there is already a clear decreasing trend in their subtotals. For the other rows, the max-depth at level 2 is chosen as 4, the max-depth at level 4 ranges from 8 to 6, and the max-depth at level 3 is somewhere in between. Apparently, the DAQ algorithm works well as long as the max-depth's are reasonably large (i.e., ≥ 4 at all

levels). The overhead for building the DAQ's is never overwhelmingly heavy. In fact, the best timing results are from 8-5-4 and 8-6-4 when the leaf nodes at level 4 have the deepest DAQ of max-depth = 8 and only require 66.1 ms, which yield the minimal subtotals (198.0 or 199.5 ms) for all three levels.

The fast pyramidal line detection algorithm has been integrated into a robotic workcell, where a stream of varied wooden blocks are presented to the vision system via a conveyor belt. The vision system recognizes square blocks, which are subsequently removed by the robot arm. Blocks of other shapes are left on the belt. For ordinary block scenes, the algorithm only takes less than 50 ms for line detection in the transputer pyramid. A simple heuristic algorithm was adopted for cube recognition. The entire vision process is completed in less than 500 ms and thus enables the real-time sorting of the blocks.

5 Conclusion

A 2D hybrid pyramid has been assembled at SFU. Parallel hardware links are developed to enable massively parallel data communications between the array processors and the transputers and thus enable the integration of two parallel subsystems into a powerful and economic hybrid pyramid vision machine.

This paper presents algorithms and implementation of fast Hough line detection in the hybrid pyramid. The lines are represented using their Hough parameters ρ and θ, and merged hierarchically by all the transputers in the pyramid. To reduce the complexity of the line merging process, a Dynamically Allocated Quadtree (DAQ) is introduced to represent the Hough parameter space. The comparative experimental results show that the line merging algorithm using DAQ is significantly more efficient than the simple merging algorithm.

The reported research activity is a portion of our long term pyramid vision project. It is now well-known that pyramid provides a unique parallel and hierarchical platform, and it has great potential for parallel vision. It supports the multiresolution approaches and capitalizes on advantages of both the mesh and the pipeline architectures. However, a full-scale 2D pyramid with thousands of nodes is yet to be created, and its availability is going to be limited for the years to come. We have shown that, a 2D "half-scale" hybrid pyramid is not only economic, but can also be quite efficient and adequate for many parallel vision tasks.

6 Acknowledgments

This work is supported in part by the Canadian National Science and Engineering Research Council under the grants OGP-36726, EQP-42445, EQP-90642 and a Strategic Research Initiative Grant from the Centre for System Sciences at the Simon Fraser University. The authors would like to thank Dr. John Ens, who contributed greatly to the design and implementation of the PARLinks, and Frank Tong for many stimulating discussions.

References

[Ens91] J. Ens, A parallel link between an AIS-4000 and a transputer pyramid, *Simon Fraser University Technical Report* CSS-IS TR91-09, 1991.

[Duda72] R.O. Duda and P.E. Hart, Use of the Hough transform to detect lines and curves in pictures, *Communications of the ACM*, Vol.15, 1972, pp. 11-15.

[Joli89] J. Jolion, A. Rosenfeld, An O(log n) pyramid Hough transform, *Pattern Recognition Letters*, Vol.9, 1989, pp. 343-349.

[Li91] Z.N. Li, Vision in pyramids - object recognition in real-time, *Proc. 6th Int. Conf. on CAD/CAM, Robotics, and FOF*, London, 1991.

[O'Ro81] J. O'Rourke, Dynamically quantized spaces for focusing the Hough transform, *Proceedings 7th IJCAI*, 1981, pp. 737-739.

[Rose88] A. Rosenfeld, J. Ornelas, and Y. Hung, Hough transform algorithms for mesh-connected SIMD parallel processors, *Computer Vision, Graphics, and Image Processing* Vol. 41, 1988, pp. 293-305.

[Rise90] E.M. Riseman and A.R. Hanson, Progress in Computer Vision at the University of Massachusetts, *Proc. DARPA Image Understanding Workshop*, 1990, pp. 86-96.

[Same84] H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Survey*, Vol. 16, No. 2, 1984, pp. 187-260.

[Schm88] L.A. Schmitt and S.S. Wilson, The AIS-5000 parallel processor, *IEEE Trans. on Pattern Recognition and Machine Intelligence*, Vol. 10, No. 3 1988, pp. 320-330.

[Sloa81] K.R. Sloan, Dynamically quantized pyramids *Proc. 7th IJCAI*, 1981, pp. 734-736.