2

ETL-0488

Parallel vision algorithms

First annual technical report

Hussein A. H. Ibrahim, Editor
John R. Kender
Lisa G. Brown

Department of Computer Science
Columbia University
New York, New York 10027

October 1987

Prepared for:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

U.S. Army Corps of Engineers
Engineer Topographic Laboratories
Fort Belvoir, Virginia 22060-5546

Parallel Vision Algorithms

Annual Technical Report

Contract DACA76-86-C-0024

John R. Kender, Principal Investigator
Hussein A. H. Ibrahim, Research Scientist
Lisa G. Brown, Research Programmer
Department of Computer Science
Columbia University
New York, N.Y. 10027
(212)280-2736

Technical Report - CUCS-271-87

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CUCS-271-87	5. MONITORING ORGANIZATION REPORT NUMBER(S) ETL-0488
6a. NAME OF PERFORMING ORGANIZATION Columbia University 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION U.S. Army Engineer Topographic Laboratories
6c. ADDRESS *(City, State, and ZIP Code)* Computer Science Department New York, NY 10027	7b. ADDRESS *(City, State, and ZIP Code)* Fort Belvoir, Virginia 22060-5546
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency 8b. OFFICE SYMBOL *(If applicable)* ISTO	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DACA76-86-C-0024
8c. ADDRESS *(City, State, and ZIP Code)* 1400 Wilson Boulevard Arlington, VA 2209-2308	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Parallel Vision Algorithms First Annual Technical Report

12. PERSONAL AUTHOR(S)
Hussein A. H. Ibrahim, Editor; John R. Kender and Lisa G. Brown

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Annual	FROM 10/1/86 TO 9/30/87	1987, October	73

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Computer Vision, Artificial Intelligence, Image Understanding, Multi-Resolution, Stereo, Texture, Strategy Computing.

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The "Parallel Vision Algorithms" annual report covers the project activities during the period from October 1, 1986, through September 30, 1987. The objective of this project is to develop and implement, on highly parallel computers, vision algorithms that combine stereo, texture, and multi-resolution techniques for determining local surface orientation and depth. Such algorithms will immediately serve as front-ends for autonomous land vehicle navigation systems. During the first year of the project, efforts have concentrated on two fronts. First, developing and testing the prarallel programming environment that will be used to develop, implement and test our parallel vision algorithms. Second, developing and testing multi-resolution stereo, and texture algorithms. This report describes the status and progress on these two fronts. We describe first the programming environment developed, and mapping scheme that allows efficient use of the connection machine for pyramid (multi-resolution) algorithms. Second, we present algorithms and test results for multi-resolution stereo, and texture algorithms. Also the initial results of the starting efforts of integrating stereo and texture algorithms are presented.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Rosalene M. Holecheck	22b. TELEPHONE *(Include Area Code)* (202) 355-2700 22c. OFFICE SYMBOL CEETL-RI

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Summary

The "Parallel Vision algorithms" Annual report covers the project activities during the period from October 1st, 1986 through September 30, 1987. The objective of this project is to develop and implement, on highly parallel computers, vision algorithms that combine stereo, texture, and multi-resolution techniques for determining local surface orientation and depth. Such algorithms will immediately serve as front-ends for autonomous land vehicle navigation systems. During this first year of the project, efforts have concentrated on two fronts. First, developing and testing the parallel programming environment that will be used to develop, implement and test our parallel vision algorithms. Second, to develop and test multi-resolution stereo, and texture algorithms. This report describes the status and progress on these two fronts. We describe first the programming environment developed, and the mapping scheme that allows efficient use of the Connection machine for pyramid (multi-resolution) algorithms. Second, we present algorithms and test results for multi-resolution stereo, and texture algorithms. Also the initial results of the starting efforts of integrating stereo and texture algorithms are presented.

Preface

This report, submitted to the Defense Advanced Research Projects Agency (DARPA) of the Department of Defense (DOD), in response to Contract DACA76-86-C-0024, presents the progress during the first year of the "Parallel Vision Algorithms" project at Columbia University. The "Parallel Vision Algorithms" project is sponsored by DARPA as part of its Strategic Computing Program and contracted through the U.S. Army Engineer Topographic Laboratories (ETL).

The objective of this project is to develop and implement, on highly parallel computers, vision algorithms that combine stereo, texture, and multi-resolution techniques for determining local surface orientation and depth. Such algorithms will immediately serve as front-ends for Autonomous Land Vehicle navigation systems, one of the Strategic Computing Program's application areas.

This report is prepared for the U.S. Army Engineer Topographic Laboratories, Fort Belvoir, Virginia, and the Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia under contract DACA76-86-C-0024. The Contracting Officer's Representative is Rose Holecheck. The Program manager at DARPA is LTC. Robert Simpson. Questions regarding this document should be forwarded to Prof. John Kender 212-280-8197.

Table of Contents

List of Figures

List of Tables

1. Introduction

The objective of this project is to develop and implement, on highly parallel computers, integrated parallel vision algorithms that combine stereo, texture, and multi-resolution techniques for determining local surface orientation and depth. Such algorithms will immediately serve as front-ends for autonomous land vehicle navigation systems. During this first year of the project, efforts have concentrated on two fronts. First, developing and testing the parallel programming environment that will be used to develop, implement, and test our parallel vision algorithms. Second, to develop and test multi-resolution stereo, and texture algorithms. This report describes the status and progress on these two fronts. We describe first the programming environment developed, and the mapping scheme that allows efficient use of the Connection machine (CM) for pyramid (multi-resolution) algorithms. Second, algorithms and test results for multi-resolution stereo, and texture algorithms are presented. A methodology for integrating the results of different texture modules is described. Also the initial results of the starting efforts of integrating stereo and texture algorithms are presented.

The initial plans called for the testing and implementation of the parallel algorithms on the NON-VON Supercomputer (which was being developed at that time). With NON-VON project being terminated, the Connection Machine has been chosen as the target machine to develop and test our algorithms. A simulator of the CM* has been installed, tested, and development of some algorithms on the simulator have started in September 1987. An account has been obtained on CM1 at Syracuse University, and we plans for using this machine to demonstrate the developed algorithms.

*With permission from Thinking Machines Inc.

2. Programming Environment for Multi-resolution Algorithms

Hussein Ibrahim and Lisa brown

In this section, we briefly describe the different programming environments in which development and testing of stereo and texture algorithms are performed. Some of our initial results have been developed and tested on a simulator of the NON-VON machine, a fine-grained SIMD tree-structured machine. The NON-VON machine is an example of a good architecture to implement multi-resolution algorithms because of its fine granularity and hierarchical architecture. Versions of an instruction-level NON-VON simulator are available on both the VAX 11/750, and the DEC-20 machines. A machine with 2 K processing elements can be simulated and exercised. A functional-simulator of NON-VON that assumes floating point arithmetic in the processing elements have also been used by Dong Choi in his work on depth interpolation. The simulator has been used mainly to test convolution algorithms and computing zero crossings and matching operations. The results obtained from these tests can be applied to our target testbed machine (the Connection Machine) because of its fine granularity and the implicit hierarchical structure of its hypercube network.

A second programming environment, in which we will be testing our parallel algorithms, has also been setup. This environment consists of a set of primitive functions which will allow us to simulate parallel operations on a multi-resolution pyramid of images. The operations allowed in this environment will make it possible to implement them either on a pyramid machine or easily simulate them on a mesh-connected machine. The present set-up includes mechanisms for constructing and saving pyramids, the capability of filtering each level using all the standard techniques, and inter-level operations that not only descend or ascend the pyramid but operate cooperatively in both directions including within levels. Several classic pyramid operations have been used to test the environment such as stereo matching, edge refinement, pyramid search and similar logarithmically-improved pyramid operations.

Lastly, we have started working on the details of mapping the multi-resolution pyramid data structures on the Connection Machine, with the goal of reducing the amount of communication required to simulate a full pyramid machine. This can be achieved by making use of the hypercube interconnection network employed in the connection machine. We describe this work in the following subsection.

2.1 Pyramid Algorithms on the Connection Machine

Hussein Ibrahim

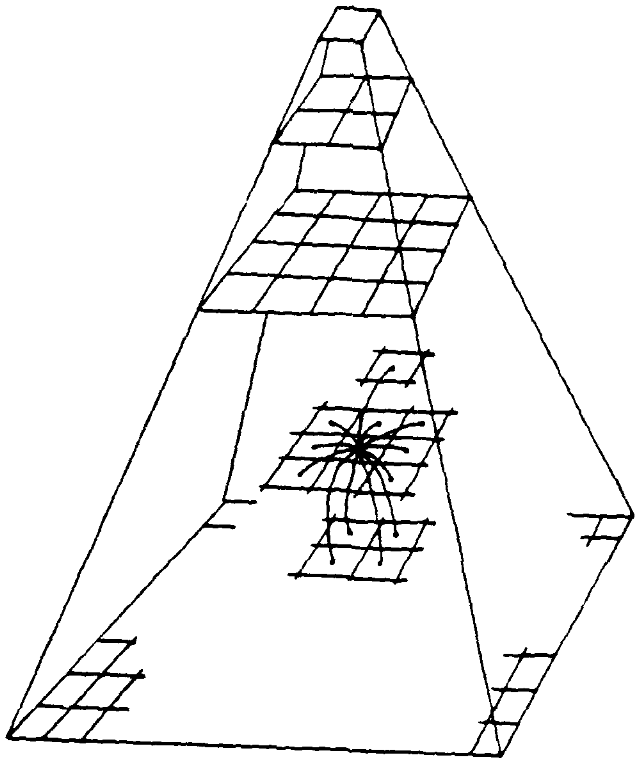
2.1.1 Introduction

Pyramid architectures have been proposed to implement efficiently (in real time) image analysis tasks, specially multi-resolution, and top-down/bottom-up image analysis tasks. The pyramid architecture consists of a set of mesh-connected layers of processing elements (PE's) successively decreasing in size by a factor of four. Each PE on an intermediate layer is connected to four children on the layer below it, to a single parent in the layer above it, and to four neighbors in the same layer, as shown in Figure 2-1. Hardware implementation of pyramid architectures is expensive because of the large amount of wiring that is required in such machines. At the same time, the Connection Machine, a highly parallel fine-grained machine with mesh and hypercube interconnection networks, has been developed at Thinking machines Inc. and a 64K version is available for the vision community. The CM executes its tasks in single instruction stream, multiple data stream (SIMD) mode.

In this section, we describe an addressing scheme that maps efficiently the pyramid architecture onto the Connection Machine (CM). This results in an efficient implementation of pyramid algorithms on the CM. The Connection Machine uses two modes of communication; the first one is mesh-communication where the whole array of PE's (64K) forms a two-dimensional orthogonal mesh (256 x 256). In this mode, each PE may communicate with its four neighbors in the east/west/north/south directions (NEWS network), as shown in Figure 2-2.

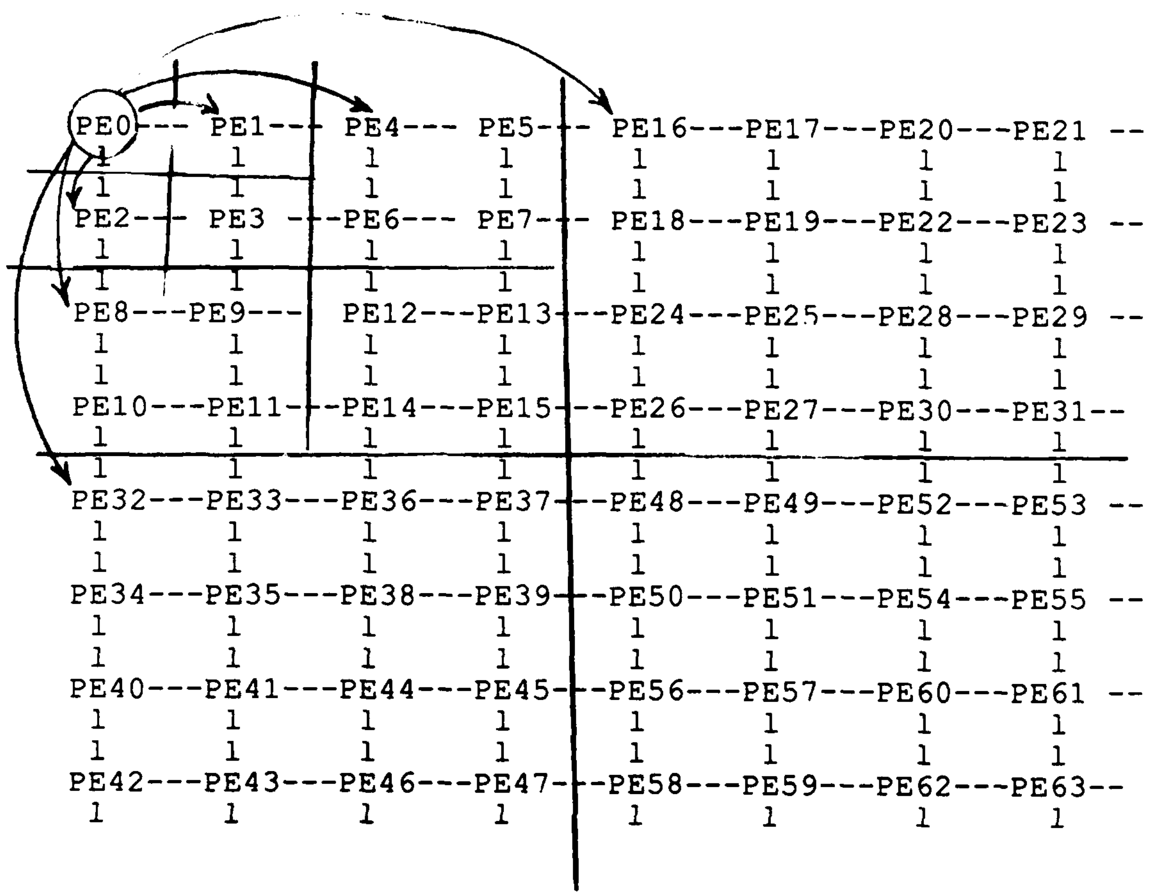
The second mode of communication is the hypercube communication, in which each PE can communicate with all the PE's whose binary addresses differ from its own address exactly in one bit location. In the 64K machine, this means each PE is also connected to 16 PE's, using the binary 16-dimension hypercube interconnection network. For example, each PE communicates in the direction of the first dimension with the PE whose address is computed from its own address by flip-flopping the least significant bit (1 --> 0, 0 --> 1). In general, each PE communicates along the n-th dimension with the PE whose address differs from its own only in the n-th least significant digit. For example, PE0 is connected to PE1 along the first dimension, to PE2 along the second dimension, to PE4 along the third dimension, ..., and to PE(2**15) along the 16th dimension.

Figure 2-1: The Pyramid Architecture



Organization of the Pyramid Machine

Figure 3-2: The address scheme of the Connection Machine PE's


PE0--- PE1--- PE4--- PE5--- PE16---PE17---PE20---PE21 --
PE2--- PE3 ---PE6--- PE7--- PE18---PE19---PE22---PE23 --
PE8---PE9--- PE12---PE13---PE24---PE25---PE28---PE29 --
PE10---PE11---PE14---PE15---PE26---PE27---PE30---PE31--
PE32---PE33---PE36---PE37---PE48---PE49---PE52---PE53 --
PE34---PE35---PE38---PE39---PE50---PE51---PE54---PE55 --
PE40---PE41---PE44---PE45---PE56---PE57---PE60---PE61 --
PE42---PE43---PE46---PE47---PE58---PE59---PE62---PE63--

--- : Mesh Connections.

$\longrightarrow$: An example of Hypercube Connections.

In the following sections, we describe the addressing scheme, and how various pyramid communication modes can be simulated on the CM.

2.1.2 Mapping Scheme

To map the pyramid architecture on the connection machine, the CM mesh network of PE's simulates the base of the pyramid, while the PE's on the internal levels of the pyramid are going to be simulated according to the following scheme. The lower right PE of each 2 x 2 cube will simulate the parent of the four PE's in this 2-D cube. Similarly, for the second level above the leaf level in the pyramid, the PE next to the one in the lower right one will be used to emulate the parent of the PE's in the 4 x 4 cube (4-D hypercube) A general formula that specifies the CM addresses of the PE's on the intermediate levels of the pyramid is given as following:

$4^i n - 2^{i-1}$, where *i* is the level number (1 for the level above the leaf level), n changes from 1 to the number of PE's on that level. $(2^{2.(8-i)})$.

Figure 2-3 shows this mapping for an 8 x 8 mesh. The hypercube connections are not shown in this configuration, but they exist according to the scheme described before.

The mapping scheme described above ensures that each PE in the pyramid intermediate levels is emulated exclusively by one PE in the CM (one to one mapping). The proof is as follows. The address mapping formula is $4^i n - 2^{i-1}$. On the same intermediate level, *i* is constant, thus changing *n* will result in a different address each time. Now to prove that no two intermediate level PE's are mapped to the same CM PE address, First assume that this is true. It means that for PE n1 on level i1, there exist a PE n2 on level i2 such that

$$4^{i1}n1 - 2^{i1-1} = 4^{i2}n2 - 2^{i2-1}.$$

The equation can be re-arranged as follows:

$$4^{i2}(4^{i1-i2}n1 - n2) = 2^{i2}(2^{i1-i2-1} - 2^{-1}).$$

or

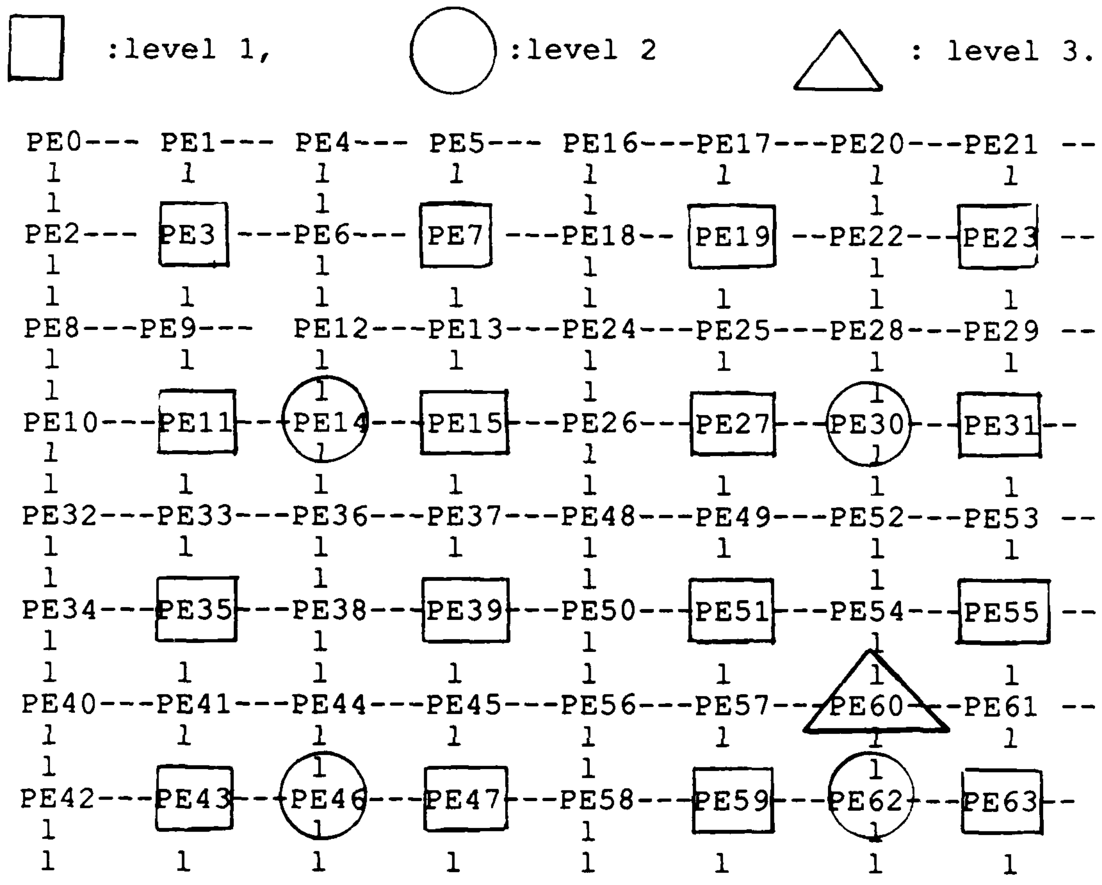
$$2^{i2}(4^{i1-i2}n1 - n2) = (2^{i1-i2-1} - 0.5).$$

The left hand side of the equation is always an integer as i1, i2, n1, and n2 are all integers, while the right hand side is not. Thus the initial assumption is not true. This proves that this mapping is one to one.

Now there are two types of communication in the pyramid that need to be simulated, the top/down communication (parent/child communication), and the lateral communication in the

Figure 3-3: The layout of the pyramid intermediate levels

The original CM mesh represents level 0.



```
PE0--- PE1--- PE4--- PE5--- PE16---PE17---PE20---PE21 --
 l      l      l      l      l      l      l      l
 l             l             l             l
PE2--- PE3 ---PE6--- PE7 ---PE18-- PE19 --PE22---PE23 --
 l             l             l             l
 l      l      l      l      l      l      l      l
PE8---PE9---  PE12---PE13---PE24---PE25---PE28---PE29 --
 l      l      l      l      l      l      l      l
 l             l             l             l
PE10---PE11---PE14---PE15---PE26---PE27---PE30---PE31--
 l             l             l             l
 l      l      l      l      l      l      l      l
PE32---PE33---PE36---PE37---PE48---PE49---PE52---PE53 --
 l      l      l      l      l      l      l      l
 l             l             l             l
PE34---PE35---PE38---PE39---PE50---PE51---PE54---PE55 --
 l             l             l             l
 l      l      l      l      l      l      l      l
PE40---PE41---PE44---PE45---PE56---PE57---PE60---PE61 --
 l      l      l      l      l      l      l      l
 l             l             l             l
PE42---PE43---PE46---PE47---PE58---PE59---PE62---PE63--
 l             l             l             l
 l      l      l      l      l      l      l      l
```

intermediate levels of the pyramid.

2.1.3 Simulating Pyramid Intermediate Levels Mesh Communication

In this subsection, we describe how the mesh communication on the pyramid intermediate levels can be simulated on the CM using the addressing scheme introduced in the previous section. Note that Simulating the mesh communication on the base of the pyramid is performed directly using the NEWS network of the CM. In what follows we how a send-east operation within level one of the pyramid is simulated in the CM, and then generalize this procedure for other pyramid levels (communication in the other directions is performed similarly).

To simulate mesh communication within level one, the information is sent first forward along the third dimension. Sending information forward along the *n*th dimension means that only PE's whose *n*th bit is 0, send their information to the PE's whose binary address is similar except that the *n*th bit is 1. Thus, sending forward along the third dimension will involve PE3 sending the information to PE7, but PE7 will not send its information to PE3. When PE7 sends its information to PE3, we will refer to that as sending the information backward along the third dimension. We will use the following notations to express these two types of hypercube communication.

-->3 (send information forward along the third dimension.)

3<-- (send information backward along the third dimension.)

Thus sending information forward along the third dimension will ensure that half the PE's on level one have sent their information to their east neighbors (PE3 to PE7, PE19 to PE23, ..., etc). Now for PE7 to send its information to PE19, it first sends the information along the 5th dimension to PE23, and then PE23 send it back along the third dimension to PE19. This results in half the remaining PE's sending their information to their east neighbors. In our own terminology this is represented as: (-->5 followed by 3<--). To continue sending the rest of information east, similar scheme is followed. The complete procedure to send east in the first level of the pyramid in a 16-dimension hypercube is given by the following figure:

Note that these communication steps can be pipelined in such a way that there will be no repetition of the same communication steps. To do that a temporary variable has to be created to prevent overwriting a traveling value. For example if we want to send the value of variable A to variable B. In any send forward operation the variable A is sent to variable B, while in a send

-->3

-->5, 3<--

-->7, 5<--, 3<--

-->9, 7<--, 5<--, 3<--

-->11, 9<--, 7<--, 5<--, 3<--

-->13, 11<--,9<--, 7<--, 5<--, 3<--

-->15, 13<--,11<--,9<--, 7<--, 5<--, 3<--

backward operation variable B is sent to replace the variable B in the receiving PE. The pipelined version of the above procedure is as follows:

-->15	-->n-1
13<--, -->13	n-3<--, -->n-3
11<--, -->11	n-5<--, -->n-5
9<--, -->9	.. ..
7<--, -->7	.. ..
5<--, -->5	
3<--, -->3	2i+1<--, -->2i+1

n: dimension of the hypercube, i: level number.

During each of these communication steps all of the PE's are actively sending or receiving information. Note also that, the above scheme can be augmented easily to perform wraparound shifting as well. This is accomplished by a sequence of send backward communication instructions. For example, in the 16-dimension hypercube this starts with 15<--, where the A variable contents is sent to to Variable B backward along the 15th dimension, and then a sequence of send backward of the contents of variable B to variable B in the receiving PE (13<-- , 11<-- , ... , 3<--), except for the last one where the contents of variable B is sent to variable A only in the receiving PE's.

In general, simulating send-east communication within level i in the pyramid, involves following the above procedure except that we stop when information is sent forward along the the 2*i+1 dimension. That means that all the intermediate mesh communications in the pyramid can be performed simultaneously by disabling the receiving PE's on any level when they have completed receiving the required information. This can be achieved by storing in each PE the pyramid level number of the PE it simulates in the pyramid architecture. The execution time of the procedure to simulate all pyramid mesh communications is then proportional to the number of dimensions in the hypercube.

2.1.4 simulation results

In what follows the simulation results for a 5-levels pyramid (16 x 16 CM) are shown. In this simulation we assume no wraparound in the shift operation. The communication steps for east shift operation is shown for levels 1 through 4.

East Shift Operation

```
Random Values for non-leaf levels in 5-level pyramid
-------------------------------------------------------------
 Level  1                                Level  2           Level 3  Lev.4
248 184 190 180 173 159 209  55           87 151 164 162    253  73  6
164 132  43  55  16  93 243  52           47 132 244 218    209  69
174 140 141 114  13  34 160 239          181 111 149 105
 74 156 177 205 171 114  37  82          212 248 150   9
222 185   7   0 240  35 161   9
203   3 110 140  43 130 204  44
 42 135   9  65  42 153 198  34
 23 171 242 185 159  20  15 173
                        Contents of the A variable

Shifting East Random Values for non-leaf levels ( A --> B)
-------------------------------------------------------------------
 Level  1                                Level 2           Level 3  Lev.4
Step 1:   --> 7    (A --> B)
  0   0   0   0 248 184 190 180    0   0  87 151     0 253     0
  0   0   0   0 164 132  43  55    0   0  47 132     0 209
  0   0   0   0 174 140 141 114    0   0 181 111
  0   0   0   0  74 156 177 205    0   0 212 248
  0   0   0   0 222 185   7   0
  0   0   0   0 203   3 110 140      PE's on levels 3,4 are
  0   0   0   0  42 135   9  65      disabled after this step
  0   0   0   0  23 171 242 185
                        Contents of the B variable

Step 2: <-- 5    (B --> B)
  0   0   0   0 190 180 190 180     0   0 151 151
  0   0   0   0  43  55  43  55     0   0 132 132
  0   0   0   0 141 114 141 114     0   0 111 111
  0   0   0   0 177 205 177 205     0   0 248 248
  0   0   0   0   7   0   7   0
  0   0   0   0 110 140 110 140
  0   0   0   0   9  65   9  65
  0   0   0   0 242 185 242 185
                        Contents of the B variable

Step 3: --> 5    (A --> B)
  0   0 248 184 190 180 173 159    0  87 151 164
  0   0 164 132  43  55  16  93    0  47 132 244
  0   0 174 140 141 114  13  34    0 181 111 149
  0   0  74 156 177 205 171 114    0 212 248 150
  0   0 222 185   7   0 240  35    PE's on level 2
  0   0 203   3 110 140  43 130    are disabled after
  0   0  42 135   9  65  42 153    this step.
  0   0  23 171 242 185 159  20
                        Contents of the B variable
```

```
Step 4: <-- 3    (B --> B)
  0   0 184 184 180 180 159 159
  0   0 132 132  55  55  93  93
  0   0 140 140 114 114  34  34
  0   0 156 156 205 205 114 114
  0   0 185 185   0   0  35  35
  0   0   3   3 140 140 130 130
  0   0 135 135  65  65 153 153
  0   0 171 171 185 185  20  20
                        Contents of the B variable

Step 5: --> 3    (A --> B)
  0 248 184 190 180 173 159 209
  0 164 132  43  55  16  93 243
  0 174 140 141 114  13  34 160
  0  74 156 177 205 171 114  37
  0 222 185   7   0 240  35 161
  0 203   3 110 140  43 130 204
  0  42 135   9  65  42 153 198
  0  23 171 242 185 159  20  15
                        Contents of the B variable
```

2.1.5 Simulating Top/Down Pyramid communication on the CM

To simulate the pyramid top/down communication on the CM, three hypercube communication steps are required to simulate the level-to-level communication in the pyramid. For example, on the bottom-level of the pyramid, going one level up, each PE sends its message first forward along the first dimension, then forward along the second dimension. For example, PE3 is the parent of PE's 0,1,2,3, and PE0 sends its information to its parent node by sending it first to PE1, and then to PE3. In our notation that consists of two steps -->1, followed by -->2. Note that sending information from PE1 to its parent involves sending its information forward along the second dimension (--> 2), and sending the information from PE2 to its parent involves sending forward along the first dimension (--> 1). Similarly going from level 1 to level 2 involves communication forward along dimensions 3 and 4 respectively Thus, PE3 sends its information to PE7 and then P15. PE14 is the parent node of PE's 3,7,11,15. Thus, PE15 sends the information now to PE14 backward along the first dimension.

Remember that in the pyramid architecture, each parent is connected to four children on the level below it. We refer to these four children as UL(upper left), UR(upper right), LL(lower left), and LR(lower right). Thus, in general, to send information from an UL child on level i to its parent on level i+1, information is send forward first along dimension 2*i + 1, then forward along dimension 2*i + 2, and finally backward along dimension i. Note that going along dimension 0 means no operation. In our notations this is represented by the following sequence:

```
--> 2*i + 1
--> 2*i + 2
i <--
```

Sending the information from child UR to its parent is executed by the following sequence:

```
--> 2*i + 2
i <--,
```

while sending information from child LL to its parent involves the following sequence:

```
--> 2*i + 1
i <--
```

Sending information from the LR child to its parent involves only one communication step (i <--). It is clear that sending information from a single child to its parent executes in at most 3 steps.

Sending information from all UL children to their parents is executed by the following sequence:

```
--> 1 , --> 2 , 0 <--
--> 3 , --> 4 , 1 <--
--> 5 , --> 6 , 2 <--
...
--> n-1 , --> n , <-- (n-1)/2
```

Note that the first step in all sequences can be performed simultaneously in one hypercube cycle if each PE stores its level number and uses it to compute the address of the receiving PE. Thus, executing a global send parent kind of communication takes at most 3 cycles if each PE has its pyramid level number stored into it. Sending from all children to their parent executes in at most 8 cycles. In conclusion simulating the parent/child communication executes in fixed time regardless of the hypercube dimensions.

3. Parallel Stereo Algorithms - A Multiresolution Approach

Hussein Ibrahim and Antoni Lee

3.1 Introduction

This section reports the initial investigation results of the implementation of a multi-resolution approach to stereopsis based on various components of the computational algorithm by Marr and Poggio [10]. Stereopsis refers to the use of the two viewpoints, provided by the left and right eyes, to establish the depth of surfaces around the viewer. The computational paradigm of visual processing proposed by Marr and Poggio regards the human visual system as an information processor performing computations on internal symbolic representations of visual information. These symbolic representations are the zero-crossings of edges produced by the inherent local intensity variations in the images of the left and right eyes. The zero-crossings of the left and right images make up what is called the left and right primal sketches. Each zero-crossing descriptor from one primal sketch image should match at most one zero-crossing descriptor from the other primal sketch image. Given the relative disparity of these descriptors, depth information for selected points in the image can be computed. Using the depth information, surface descriptions can be interpolated. (This is referred to as the raw 2 1/2-dimension sketch of the image.)

The Marr-Poggio algorithm utilizes the Laplacian of a gaussian convolution operator to detect zero crossings. Since the basic descriptors are relatively simple, there may be several possible descriptions in one image which could correspond to a particular descriptor in the other image. This gives rise to the false targets problem. The false targets problem is directly proportional to the range and resolution of depth information over which a match is sought. To ameliorate correspondence matching, Marr and Poggio proposed a technique by which matching descriptions are obtained at several levels of resolution. The rough depth information obtained at a coarse resolution are then used to guide the matching at a fine resolution by changing the orientation of the eyes (*vergence control*).

In the following sections, we describe the efforts for the parallel implementation of a multi-resolution approach to stereo and present briefly the convolution algorithms, the computation of zero crossings and the matching algorithms. It should be noted here that the implementation efforts were performed mostly on a simulator of the tree structured NON-VON Supercomputer.

The simplicity of implementing these algorithms on a pyramid machine is obvious, as the NON-VON architecture is a subset of the pyramid architecture. Testing the algorithms on a simulator of a fine-grained parallel machine using realistic images is an extremely time and space consuming task. We have limited our tests to small synthetic images intended basically to prove the validity of the approach. Testing real image will be performed once we start implementation on the real CM.

3.2 Image Convolution: The First Step

The Marr-Hildreth theory of stereopsis uses band-limiting Gaussian filters to obtain images of varying spatial resolution (that is, images of different frequency spectra). These images are then convolved with a Laplacian operator which extracts edge information from the Gaussian filtered images. The Laplacian of a gaussian can be approximated by the *difference of gaussians* (D.O.G.) In our approach, the multi-resolution capability of the pyramidal architecture is used to obtain images of various resolution levels. In other words, the low pass filtering by Gaussian filters has been replaced by another form of low-pass filtering, namely resolution reduction. Wong et al ([14]), have used an unweighted 2 x 2 averaging operator followed by resampling to compute the reduced resolution versions of the original image. They showed how their operator reduce the aliasing, which occurs due to spurious noise. Glaser et al ([5]) have used a more complicated weighted 4 x 4 averaging operator, to approximate the Guassian-like low pass filters. The unweighted 2 x 2 averaging operator can be implemented on the pyramid architecture using the same procedure that builds the multi-resolution pyramid. The weighted operator also can be implemented easily on the pyramid architecture. We have implemented the simple unweighted average operator, which is equivalent to building the multi-resolution pyramid (also referred to as low-pass pyramid in [5]) of the image.

The Laplacian convolution operator is then applied to the stored multi-resolution images. The resulting set of images represent a band-passed images at various resolutions. (Referred to as band pass pyramids in [5].) Applying the Laplacian operator (a convolution operation) at each level involves communicating values between neighbor PE's in the two-dimensional image stored at that level. This is equivalent to shifting the image stored at each level in the south, north, east, and west directions. The convolution of an *n* x *n* image $\mathbf{I}_{in}$ with a given *m* x *m* mask C is mathematically given by the following discrete summation:

$$I_{out}(i,j)=\sum_{x=-m/2}^{+m/2}\sum_{y=-m/2}^{+m/2} I_{in}(\mathbf{i}-\mathbf{x},\mathbf{j}-\mathbf{y})C(\mathbf{x},\mathbf{y})$$

This operation is computationally intensive; for example, for each pixel in the image, there are m^2 multiplications and $m^2 - 1$ additions. The time complexity for the entire operation in a typical sequential machine is $O(m^2 x n^2)$.

On a parallel machine, computational complexity is reduced substantially due to spatial parallelism. Since convolution is a local (window) operation, it can be performed in a parallel fashion, each pixel computing the convolution result from the multiplication/addition of neighboring pixels. Assuming a 3 x 3 mask is used, a naive approach to compute the convolution of the stored image can be performed by repeatedly computing an element in the convolution sum, and then transmitting it to the PE, in which it is accumulated. This is not efficient since each PE is sending information to the center PE which, in most cases, is not physically adjacent. In the 3 x 3 example above, each corner PE (with respect to local 3 x 3 window) will send information through another PE in order to communicate with the center PE. Ideally, the algorithm should try to minimize the total amount of communication between PE's.

We have implemented two algorithms that make use of the minimum possible communication to accumulate the convolution value of an n x n image with an m x m mask, where $m < n$. The first of the two algorithms is applied to the general case and follows the approach described in [7]. The second algorithm takes advantage of the circular symmetry of templates used in image processing, such as the difference of guassian template. In both algorithm, we assume that each PE stores exactly one pixel of the image, and at each step it computes and accumulates one term of the cross correlation sum. The eventual output is also stored one value per PE.

3.2.1 The Spiral Convolution Algorithm

This algorithm minimizes the communication time by effectively making all communications local; in other words, data is only transmitted between adjacent PE's. Instead of shipping the multiplication results to their target PE, pixel values are transported to the target PE in a pipelined fashion, where they are multiplied and summed. This is achieved by an ordered sequence of image shifts. If the template is an odd square (the most common case), an outwardly spiraling shift order as shown in figure 3-1-a, is always possible, thus involving every required image position without any wasted shifts.

To keep the input image intact, a copy of this image is used in shifting operations. The algorithm computes the first term of the convolution sum by multiplying the image value of the

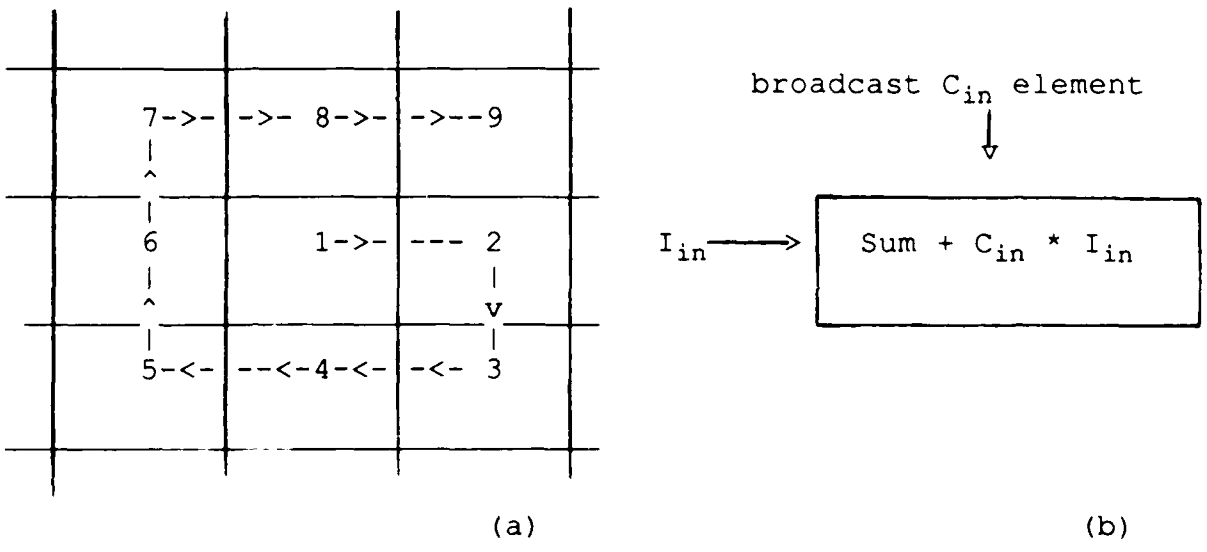


Figure 3-1: Data Flow in Spiral Convolution Algorithm

PE, where the convolution sum is to be compiled, by the template value corresponding to it. This corresponds to the central PE in case of odd square templates. Then, a sequence of shift images are performed according to the following scheme. We start at the template element, where the sum is computed, and then we walk over the template in such a way as to cover all the elements of the template (the spiral is the most direct way). For each move, the corresponding shift step is in reverse direction. For example if we move right, then the shift step is left, and if we move up, then the shift step is down, and so on. This gives us the ordered sequence of image shifts, that if performed will bring all the image elements under the central template element in the same order of the scanning spiral. For each shift step, the corresponding template element is multiplied by the shifted value and added to the compiled value at the PE. These steps make up the basic "inner product" kernel associated with the convolution operation. In essence, the "spiral algorithm" maps the convolution operation into a linear pipeline of (m xm) such kernel operations, with the first operation being slightly different in that the accumulated result is not sent to the next PE, but rather stays in the same PE as the convolution result. A 3 x 3 kernel is shown in Figure 3-1, along with the sequence of shifts required to follow the spiral path. All PE's perform this kernel operation in parallel, communicating solely with their adjacent neighbor.

The algorithm currently implemented minimizes the communications overhead as well as the amount of RAM memory used. The number of multiplications and additions are the same as for a standard convolution. In summary, every local window operation requires: m^2 multiplications, $m^2 - 1$ additions, and $m^2 - 1$ inter-PE communication steps, where m is the size of the convolution mask. Due to the spatial parallelism, each local m x m window of PE's independently computes the convolution summation, so that the total time required to perform the entire image convolution is equal to that of performing a single convolution summation.

3.2.2 The Migration Convolution Algorithm

This algorithm takes advantage of the circular symmetry of non-directional operators in performing convolutions. In doing so, it drastically reduces the multiplication time involved in performing a convolution, as well as decreases the amount of communication time and addition time necessary. To illustrate how this algorithm works, it is helpful to first make several observations about non-directional operators:

1. If one were to take an illustration of a non-directional operator, for example, The DOG operator, one would see that all four quadrants of the operator are circularly symmetrical. An example of a 7x7 mask for the non-directional DOG Operator is shown in figure 3-2.

2. Because of this circular symmetry, it will always hold true that the set of all coefficients of the operator will be found within a single "slice" of the mask (including the elements along the diagonal from the center of the mask to the outer corner, and along the central axis it borders, be it horizontal or vertical). For example, the following is a "slice" of the mask in Figure 3-2. It contains the full set of coefficients for the D.O.G. operator described.

-.0009			
-.008	-.055		
-.027	-.123	0	
-.04	-.135	.303	1

3. Each quadrant is a mirror copy of the two quadrants adjacent to it if the mask were folded along the vertical or horizontal centers that separated the two quadrants. For example, the upper left hand quadrant is a mirror copy of the upper right hand quadrant were the mask folded in half along the central vertical (the elements along the central vertical are considered elements of both quadrants). Because of this relationship, the four rows of any quadrant reflect the spatial relationships of the coefficients in all quadrants. Each of these rows is called a vector. To illustrate this, the four vectors of the upper left quadrant are:

a. (-.0009, -.008, -.027, -.04)

b. (-.008, -.055, -.123, -.135)

c. (-.027, -.123, 0, .303)

d. (-.04, -.135, .303, 1)

The vectors of the upper right hand quadrant are these same vectors with their elements in reversed order.

-.0009	-.008	-.027	-.040	-.027	-.008	-.0009
-.008	-.055	-.123	-.135	-.123	-.055	-.008
-.027	-.123	0	.303	0	-.123	-.027
-.040	-.135	.303	1	.303	-.135	-.040
-.027	-.123	0	.303	0	-.123	-.027
-.008	-.055	-.123	-.135	-.123	-.055	-.008
-.0009	-.008	-.027	-.040	-.027	-.008	-.0009

Figure 3-2: 7x7 Non-Directional Convolution Mask

Given the above observations, it is now possible to discuss the Migration Algorithm for Convolutions.

The first step of the algorithm is to broadcast each coefficient in a "slice" of the mask to all the PE's. Each PE then stores the product of the coefficient broadcast and the intensity of the pixel corresponding to that PE. Because of observation 3 above, no further multiplications will need to be performed throughout the remainder of the convolution. The products of this step are referred to as the Coefficient_Products.

The second step of the algorithm entails migrating the Coefficient_Products to the pixel corresponding to the center of the mask. An illustration of the migration paths that this algorithm uses is shown in the diagram below. Data is first migrated towards the central vertical. Once the data is collected there, it is then migrated towards the center pixel (marked with an "+"

```
-->-->-->  v  <--<--<--
-->-->-->  |  <--<--<--
-->-->-->  v  <--<--<--
-->-->-->  +  <--<--<--
-->-->-->  ^  <--<--<--
-->-->-->  |  <--<--<--
-->-->-->  ^  <--<--<--
```

Figure 3-3: Data Migration Paths

in the above figure). This method of data migration reduces the number of communications due to the parallel nature in which data is migrating in each of the quadrants.

The data being migrated are the Coefficient-Products themselves. Which Coefficient-Product is sent to a neighboring PE at any given time is determined by the vectors that were discussed in observation 4, above. To illustrate this, consider the first vector in the upper left quadrant. The first Coefficient-Product of the vector is retrieved from RAM and sent to its East neighbor, or the PE to its right. At that PE, the received value is added to the second Coefficient-Product of the vector retrieved from that PE's RAM. That sum is then sent to its East neighbor. This is continued until the accrued sum reaches the central vertical, where the last Coefficient-Product of the vector is retrieved from RAM and added to the received sum, and then stored into a temporary RAM location. This is done for each vector of the quadrant. The result of this is that each PE along the central vertical now has four temporary values stored: one for the result of applying each vector to that row of PE's.

This process is then done on the upper right hand quadrant of the mask, however, the migration now proceeds to the West, or left. When the accrued sums arrive at the central vertical, they are added to the results obtained from the migration of data in the upper left quadrant when the same vector was applied. So, for example, the result of applying vector 1 to the first row of the upper left quadrant is added to the result of applying that same vector to the first row of the upper right quadrant.

Once all of the horizontal migration has been completed, the PE's of the central vertical each have four temporary values stored which correspond to the application of each vector to the entire row in which the PE resides. To get these values to the center of the mask, we must pass these values to the North and South. This is done by the top PE of the central vertical sending the result of applying vector one to its row South. The second PE from the top adds the received value to the result of applying vector 2 to its row, and so on, until the accrued sum reaches the center. This is exactly the way data is sent North. The bottom PE of the central vertical sends the result of applying vector 1 North, where it is added to the result of applying vector 2, etc., until the sum reaches the center PE. The result of the South and North migration are then added and the result of the convolution stored!

The result of using this algorithm for performing convolutions using non-directional operators is a reduction in the expenditure of time spent on multiplying, adding and communicating. Using this algorithm, the number of multiplications (also the size of memory space required to store the result of multiplication) is reduced to

$$\sum_{i=1}^{\lceil \frac{m}{2} \rceil} i$$

which is equal to $(m^2 + 4m + 3)/8$. This is approximately one eighth the number of multiplications required in the spiral algorithm. And, the number of Inter-PE Communications and Additions is

$$2\lfloor \frac{m}{2} \rfloor \lceil \frac{m}{2} \rceil + 2\lfloor \frac{m}{2} \rfloor$$

where m is the size of the convolution mask. This is approximately equal to $(m^2/2 + m)$

Because of NON-VON's spatial parallelism, the sum of these two equations is not only the time it takes for a single m x m window convolution, but the time it takes to convolve an entire image.

In comparing the spiral and migration algorithms, we see a classic example of a "time-space" trade-off. The migration convolution is faster than the spiral convolution since it contains less number of multiplication and communication steps. On the other hand, the spiral convolution uses less RAM locations and is more flexible since it does not require a circularly symmetric convolution mask.

3.2.3 Extracting Zero-Crossings from the Image

Once the multi-resolution pyramid has been constructed, the image at each pyramid level can be convolved, either in parallel or independently, with a Laplacian mask. The resulting image is then searched for edges by looking at its zero-crossing components. Gaussian filtering should not be needed to obtain band-limited images since the multi-resolution images have already been filtered by the resolution reduction operation. If noise is a problem, each image can be Gaussian filtered before application of the Laplacian. However, this is done for noise considerations and not as a means of band-limiting the image.

The resulting image contains edge information in the form of "zero-crossings." The algorithm implemented can locate zero-crossings in various directions, although only 4 basic (1st-order) directions exist: East-West, North-South, and the two diagonal directions. In addition, the contrast (i.e. whether convolution changes from negative to positive or vice versa) is computed as well as the gradient of the convolution in a given direction. This additional information should help during the matching phase of the stereo algorithm, since orientation information is essential in getting accurate matches, especially in noisy environments. The zero-crossings would be found for both left and right images. an oriented edge exists if the output (convolved) image changes sign in a particular direction; for example, for vertical edges (i.e. horizontal zero crossings), the following criteria is used in the detection of edges:

- if two adjacent output pixels have opposite sign (excluding 0 for either one), then an edge exists between the two pixels
- if an output pixel is equal to 0 and its left and right neighboring pixels are of opposite sign, then an edge exists at the 0-valued pixel

These two conditions are shown in Figure 3-4. Edges may be found in any of the 8 primary directions by using the same criteria for pixels lying in the given orientation.

```
- - - + + +              - - - 0 + + +
     ^                         ^

+ + + - - -              + + + 0 - - -
     ^                         ^
condition 1              condition 2
```

Figure 3-4: Horizontal Zero Crossings

The current implementation assumes that the convolved image is stored in every PE's RAM on a one-to-one basis (i.e.pixels map directly into PEs). The algorithm finds zero-crossings in any of 8 orientations, labels them as to contrast, and finally calculates and stores in a given RAM location the gradient of the convolution in the given orientation. The orientation chosen is a function of the matching algorithm; in Grimson's paper [6], it is stated that only horizontal zero-crossings contain valid information for stereo matching. In any case, it is often useful to find the orientation as well as the location of the edges in an image.

The orientation of an edge can easily be found by finding in which direction the gradient of the convolution is maximized. Furthermore, by finding the two directions in which the gradient peaks, one can interpolate between the two directions to get an average orientation. An example taken from an actual run is shown below.

```
A typical 3x3 window                The gradient in 8
of a convolved image                directions is:

| 0 | 0 | 0 |               0 deg ==> 0     180 deg ==> 88
| 0 |-88|-88|              45 deg ==> 88    225 deg ==> 0
|-88|264|176|              90 deg ==> 88    270 deg ==> 352
                          135 deg ==> 88    315 deg ==> 264
180 deg <---> 0 deg
```

Since the maximum value of the gradient occurs at 270 degrees, then this is the orientation of the edge. As one can see, this type of information would be very helpful during the matching operations.

The algorithms have been successfully tested on the NON-VON simulator using artificially-constructed images. We are starting now the implementation of this approach on the CM simulator in anticipation of using the real CM machine on real images. Figure 3-5 shows the 16x16 test image used for our tests. This image has been then convolved with several filter masks, and the resulting images have been analyzed. A typical mask used is a "pseudo-Gaussian" mask, which is constructed using an exponential formula such as $2exp[\frac{-r^2}{\sigma^2}]$. A 3x3 mask for $\sigma = 1$ (and also normalized to 1) is shown below:

```
| 1 | 2 | 1 |
| 2 | 4 | 2 |  * 1/16
| 1 | 2 | 1 |
```

The Laplacian mask utilized in our experiments is shown below:

```
|  0 | -1 |  0 |
| -1 |  4 | -1 |
|  0 | -1 |  0 |
```

Using the Guassian mask on the original image (Figure 3-5) results in the image shown in Figure 3-6. Applying the Lapalacian mask to the convolved image, and using the resulting image as input to the zero-crossing algorithm, one can locate edges in any of eight particular directions. Figure 3-7 shows both horizontal and vertical edges of the Laplacian-Gaussian convolved image.

Convolution of an image at levels other than the base level allows the parallel filtering of the multi-resolution images in the pyramid. Figure 3-8 shows the reduced resolution images at levels 2 and 4 (where the mesh level is 0). Notice that for level 2, the image is still quite distinguishable, but for the next level, only the basic outline of the image is apparent.

The Gaussian and Laplacian filtering of these two images occur in parallel with that of the original, base level image. The resulting Gaussian filtered image and edge map of the 8x8 image are shown in Figures 3-9 and 3-10. Each edge map also holds the edges' contrast and convolution gradient in the direction chosen. In matching applications, it would essential to find the gradient of the convolution in several directions so as to increase the likelihood of a good match. To do this would require using the zero-crossing function in as many directions as desired, each time storing the gradient information in a separate RAM location.

3.2.4 Stereo Matching on Pyramid Architectures

In [6], a method for using coarse descriptions to control eye vergence movements is explained. In this scheme, zero-crossings from coarse resolution images are first matched and the disparity found at this level used to shift one image with respect to the other image, thus emulating the eye's vergence movements. After a new image is brought in, it is again searched but at a finer level than before, until the disparity is found at the highest possible resolution.

A pyramid architecture facilitates matching and vergence operations. Assuming that a multi-resolution image pyramid has been built, the matching would start from the higher levels of the pyramid and work its way down the pyramid. The matching operation makes use of the location of the edge, its contrast (i.e. polarity), and orientation (as given by the gradient of the convolution taken in different directions), the matching algorithm attempts to locate an edge with similar characteristics in the opposite image.

A set of rules is used to determine the existence of a "good" match:

- the matching edge must be located around the same relative horizontal (row) neighborhood (assuming negligent horizontal disparity)
- given several possible matches, the largest disparity located on the same horizontal row will be chosen ahead of any other
- if no match exists on the same horizontal row, then the next highest priority goes to the match with the largest disparity in the closest row
- a match is found for edges whose contrasts are the same and whose orientations are within certain limits from each other

Given an edge on the left image, the search neighborhood is that area in the right image which will be explored for possible matching edges. If the search neighborhood is very large (e.g. the entire image), it is likely that invalid matches or 'false targets' will bias the results. On the other hand, too small a neighborhood can result in no matches at all, an equally unacceptable situation. The algorithm implemented searches an area 'w' wide and 3 rows high. The width 'w' is an adjustable parameter, while the height is fixed at 3 rows. A typical 5x3 search is shown below.

$$\begin{tikzcd}
g \arrow[r] & f \arrow[r] & e \arrow[r] & d \arrow[r] & c \arrow[d, dash] \\
i \arrow[r] \arrow[d, dash] & h \arrow[r] & E & a \arrow[l] & b \arrow[l] \\
j & k \arrow[l] & l \arrow[l] & m \arrow[l] & n \arrow[l]
\end{tikzcd}$$

The matching algorithm compares each edge found in the left image with edges lying inside a given wx3 rectangle surrounding the corresponding right image pixel. The two images are assumed to be at the same pyramid level of the pyramid machine(i.e. of the same resolution).

Assuming that the two images (and their corresponding edge maps) are stored in RAM memory, the matching algorithm is composed of the following basic steps:

1. Move the right image edge profile to a buffer area in RAM. Send the data stored in the buffer towards the center PE (i.e. send 'a' towards 'E'in the diagram above)
2. Compare to see if a match exists between the left image edge profile and that received from 'a'. A match strength is assigned to this Comparison based on the edge information as described before.
3. Enable those PE's in which a match was found and broadcast an appropriate (dx,dy) disparity pair which is then stored in RAM by each enabled PE
4. Send the data again in the same direction so that 'b' will now arrive at the 'E' and repeat step two.
5. Repeat these send/match functions until for all possible matches of the window (Another spiral kind of algorithm).
6. Select the best match found among the various matches (largest strength).

Note that this algorithm utilizes the parallelism inherent in a pyramid machine to match all edges in the left image with right image edges in parallel. If the center pixels are also compared (i.e. for a disparity of 0), the total time to match an entire image is equal to (w * 3)(time to do a send/compare/broadcast operation).

At the end, every left image 'edge' PE will contain an array of disparity values (dx,dy) corresponding to possible right image matches (no pair is stored if a match was not found within the given search area). The possible matches are arranged according to the priority scheme mentioned above (that is, matches located in the same horizontal line are stored in the first array location). The best match is chosen for each edge, and the maximum disparity is computed using the pyramid structure.

This maximum value is then used to guide the search on the next level down the pyramid by limiting the search area. This continues until the matching is performed for the images at the base. Match values at different levels can be then used to compute a confidence value for the matching at each edge.

3.3 Depth Interpolation Problem - A multi-resolution approach

Dong Jae Choi and John R. Kender

The depth interpolation problem is concerned with reconstructing the surface of an object for which sparse depth information has been obtained. In other words transforming the 2 1/2 D raw sketch that has resulted from the stereo algorithm into a complete 2 1/2 D sketch containing explicit information about the surface at all points. We report here on efficient computational methods of solving this problem on fine-grained SIMD machines with local and global communication networks. The problem can be cast as solving a large system of linear equations with a symmetric positive definite matrix (SPD). Previous work on the problem, by Grimson and Terzopolus, have been studied. These methods basically employ local information to reach the solution. Our work concentrated on investigating the numerical methods that use global information adaptively to reach the solution and how they can be implemented efficiently on emerging parallel architectures.

3.3.1 Problem Formulation

The nonzero coefficients of each equation in the large linear system is specified as summations of computational molecules. Given the depth constraints and the orientation constraints, a set of computational molecules computes the nonzero coefficients of the linear system by local computations. Because of the symmetric nature of the computational molecules, it can be easily snown that the resulting matrix is symmetric. Furthermore, Terzopoulos shows the stronger result that the matrix generated is symmetric and positive definite (SPD). The matrix is also sparse. Even for interior nodes which are sufficiently distant from a boundary where the depth is discontinuous, they interact with only 12 neighbors, all of them at most only 2 nodes away. When an interior node $[i, j]$ is constrained by the depth constraint $d^h_{i,j}$, its nodal equation is given

Figure 3-5: Original 16x16 Artificial Image

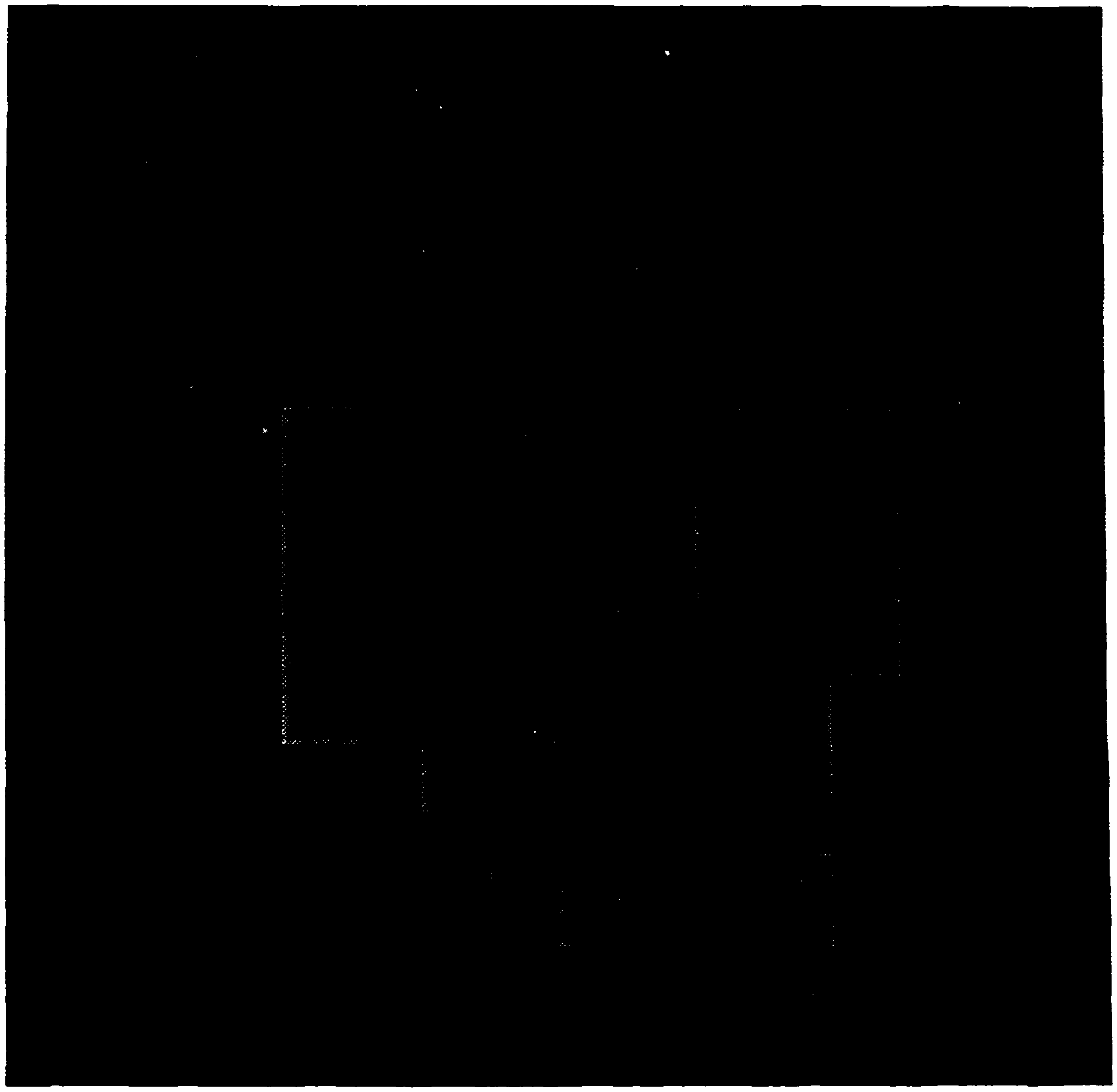


Figure 3-6: Gaussian Filtered 16x16 Image

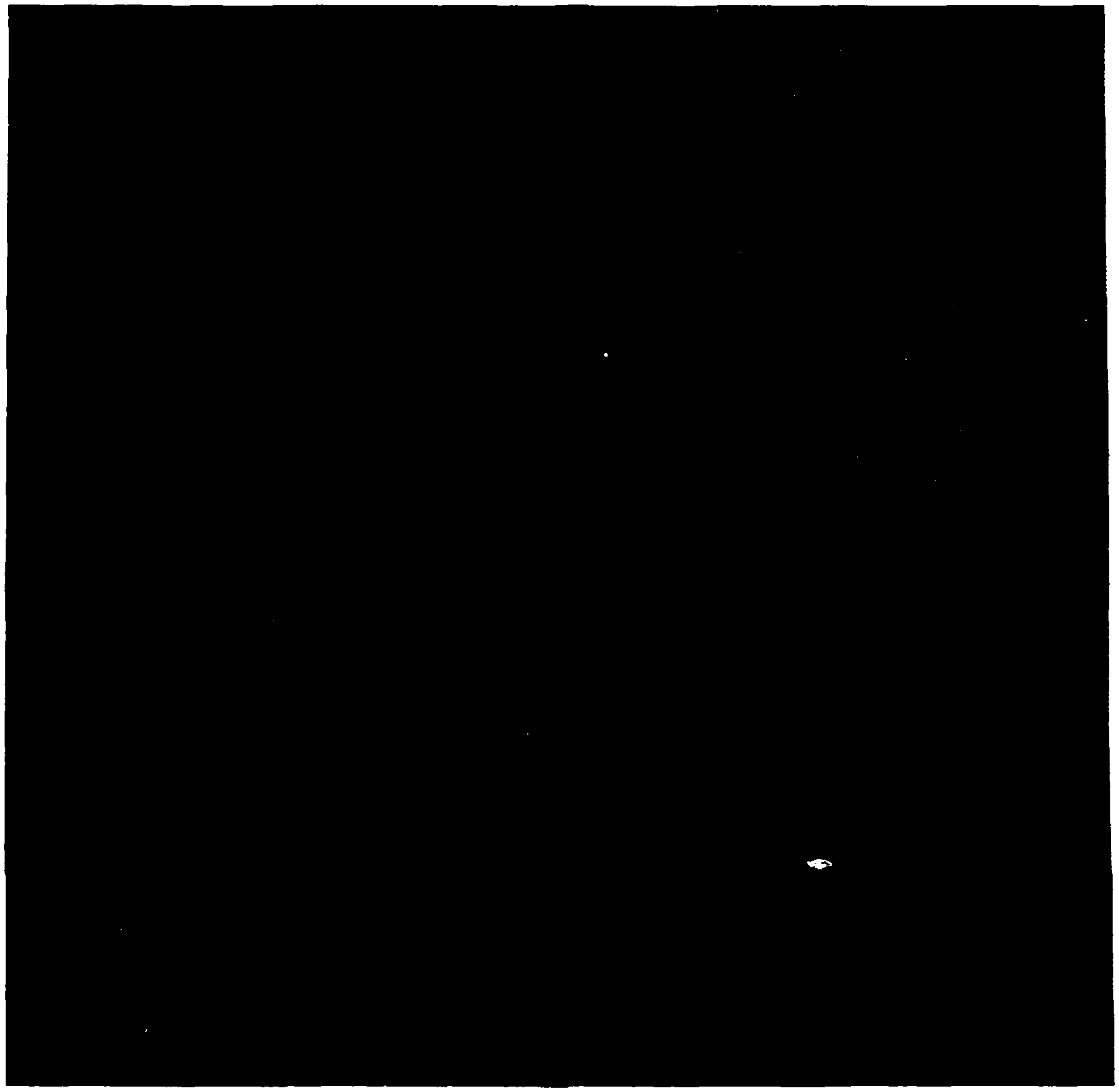
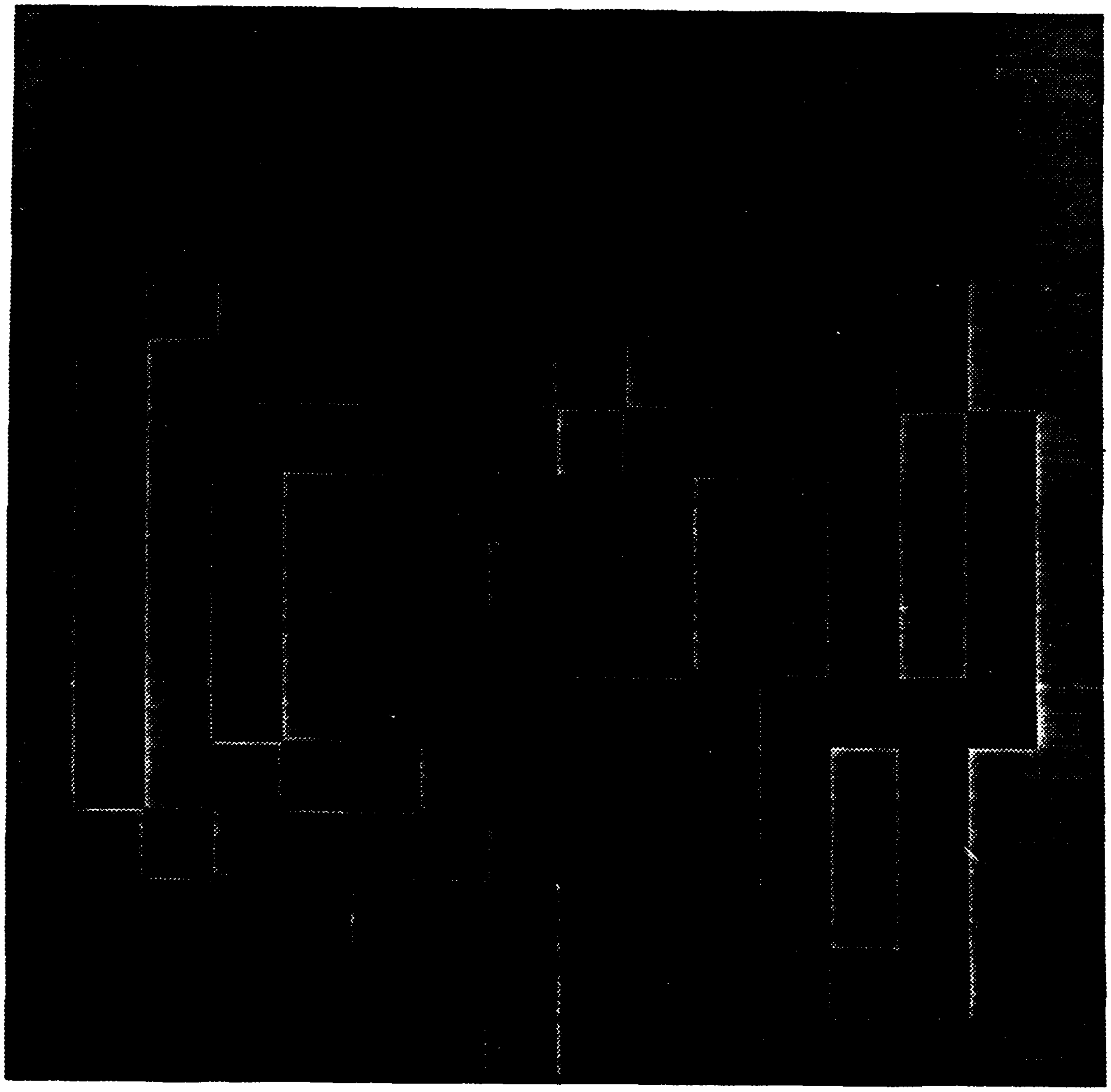
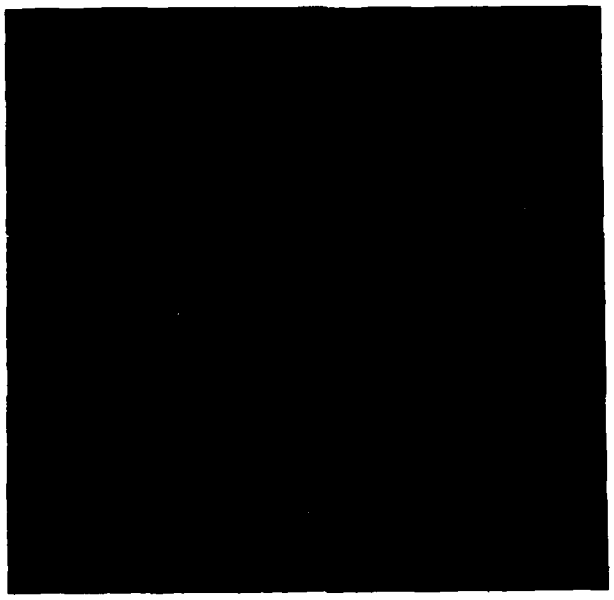


Figure 3-7: Horizontal/Vertical Edge Map of 16x16 Image





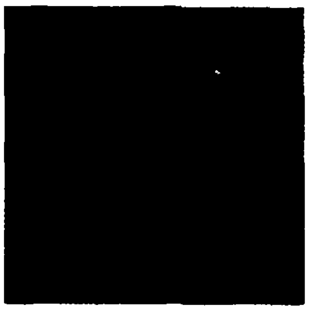


Figure 3-8: Reduced Resolution Images - 8x8 and 4x4

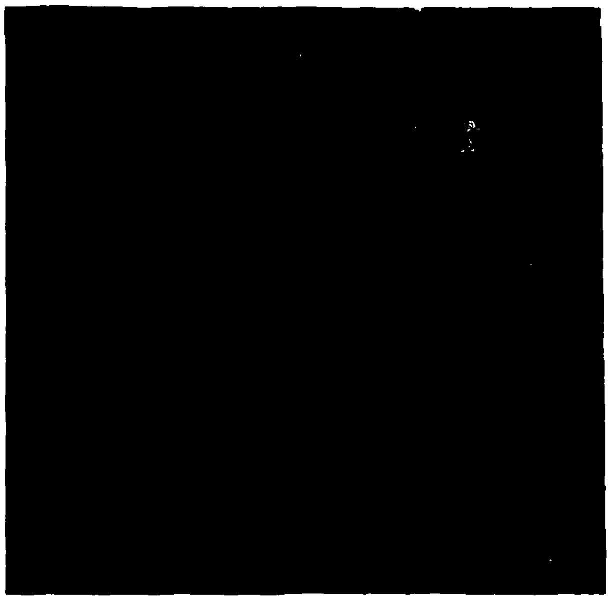


Figure 3-9: Gaussian Filtered 8x8 Image

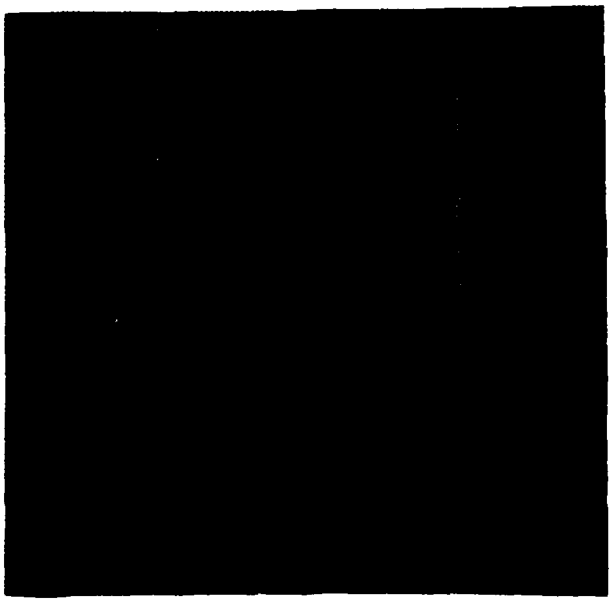


Figure 3-10: Horizontal and Vertical Edge Map of 8x8 Image

by

$$\begin{aligned}
&(20/h^2 + \beta^h_{i,j})\, x^h_{i,j} \\
&- (8/h^2)(x^h_{i-1,j} + x^h_{i+1,j} + x^h_{i,j-1} + x^h_{i,j+1}) \\
&+ (2/h^2)(x^h_{i-1,j-1} + x^h_{i+1,j-1} + x^h_{i-1,j+1} + x^h_{i+1,j+1}) \\
&+ (1/h^2)(x^h_{i-2,j} + x^h_{i+2,j} + x^h_{i,j-2} + x^h_{i,j+2}) \\
&= \beta^h_{i,j}\, d^h_{i,j}, \qquad (1)
\end{aligned}$$

where x^h_{ij} denotes the depth value computed for the node $[i, j]$. The two terms, $\beta^h_{ij} x^h_{ij}$ and $\beta^h_{ij} d^h_{ij}$, are not present when the node is not constrained with any depth constraints. The optimal value for β^h_{ij} is dependent on h but independent of $[i, j]$ and is given by $\beta^h = \gamma / h^2$, where γ is constant. The value employed by Terzopoulos for γ is either .5 or 2.0 in most cases.

3.3.2 Our approach

We have followed the Terzopoulos' formulation on visible surface reconstruction and have used the computational molecules proposed by him. However, we present an alternative depth interpolation process using the theoretically better iteration methods, which speed convergence and are amenable to certain classes of parallel computers. We were motivated to use these iteration methods by the observation that the Chebyshev and conjugate gradient methods are provably optimal in terms of computational complexity. Multiresolution techniques have been incorporated to speed the execution.

The depth interpolation problem has been cast as solving a large system of linear equations,

$$Ax = b \tag{2}$$

where A is an $n \times n$ SPD matrix and x and b are $n \times 1$ vectors. We use x to denote the depth vector where n represents the number of depth continuous nodes in the region.

Using one of several known basic iterative methods, the equation (2) can be solved by the following iterative process

$$x^{(i+1)} = Gx^{(i)} + k, \qquad i = 0, 1, 2, \ldots \tag{3}$$

where G is the iteration matrix for the method and k is an associated vector. The iterative method (3) is convergent if for any initial approximation $x^{(0)}$ the sequence $x^{(1)}, x^{(2)}, \ldots$ defined by (3) converges to the unique solution $\alpha = A^{-1}b$.

There are several well known basic iterative methods: the Jacobi, the Gauss-Seidel, the successive overrelaxation (SOR), and the symmetric successive overrelaxation (SSOR) methods. However, methods other than these basic iterative methods are used in practice because of the slow convergence rates of the basic iterative methods. The rates of convergence can be accelerated by two major classes of accelerations: polynomial acceleration methods or nonpolynomial acceleration methods. Note that the multi-grid method used by Terzopoulos is one of the nonpolynomial acceleration methods.

In our implementation, we chose the Jacobi method as the underlying basic iterative method , and the adaptive Chebyshev method as the method of polynomial acceleration. In the Jacobi method, the iteration matrix G is related to the matrix A by

$$g_{i,j} = \begin{cases} 0 & \text{if } i=j \\ -a_{i,j}/a_{i,i} & \text{otherwise} \end{cases} \tag{4}$$

where $A = (a_{i,j})$ and $G = (g_{i,j})$ for $1 \le i, j \le n$.

The convergence rate of this method is fastest when the largest eigenvalue, $M(G)$, and the smallest eigenvalue, $m(G)$, of the iteration matrix G for the related basic method are known.

We will return to experiments with this method shortly.

Another method which uses the global information of the matrix is the conjugate gradient method. We can also solve the equation (2) iteratively by constructing a sequence $\{x^{(i)}\}$ converging to the solution $\alpha = A^{-1}b$. The following equations have been derived to solve this problem:

$$\begin{aligned} r^{(i)} &= Ax^{(i)} - b, \\ z^{(i)} &= x^{(i)} - c_i r^{(i)}, \\ y^{(i)} &= x^{(i-1)} - z^{(i)}, \\ x^{(i+1)} &= z^{(i)} - u_i y^{(i)}, \end{aligned} \tag{5}$$

where

$$c_i = \frac{(r^{(i)}, r^{(i)})}{(r^{(i)}, Ar^{(i)})},$$

$$u_0 = 0,$$

$$u_i = \frac{(y^{(i)}, r^{(i)} - c_i A r^{(i)})}{(y^{(i)}, Ax^{(i-1)} - Ax^{(i)} + c_i A r^{(i)})}, \qquad i \ge 1. \tag{6}$$

3.3.3 Parallel Implementation

A parallel architecture to support the particular structure of our application demands the following characteristics:

- Many of the operations described in the previous section ought to be performed simultaneously on a subset of nodes properly chosen at each moment. Identical operations are carried out upon the data at each selected node. This first property naturally leads to a fine grained processor and a single instruction multiple data stream (SIMD) mode of execution.
- Secondly, the matrix involved is sparse. In particular, in the depth interpolation problem even an interior node far removed from the region boundary interacts only with 12 neighboring nodes. Therefore, mesh interconnections between nodes are sufficient for handling all the local communication needs for matrix multiplication.
- Thirdly, what is needed as well is a fast global summary capability. In the two iteration methods, we need to compute various vector norms, a matrix norm, and inner products. This global communication need can be met well by any global network mechanisms, e.g., the tree topology or the boolean *n*-cube topology, superimposed on the underlying mesh.

We need 528 bits/PE in the adaptive Chebyshev acceleration method, while 688/PE are needed in the case of the conjucate gradient method.

Tables 3-1, and 3-2 show the summary of typical operations required in both methods.

The methods were applied to a synthetic image of a portion of a cylinder whose axis was parallel to the *j* direction. The synthetic depth for node $[i, j]$ was set to

$$\alpha_{[i,j]} = (1.0 - (i - r/2)^2 / (r)^2)^{1/2},$$

where $0 \le i \le 127$ and $r = 127.0$. The shape of the boundary was a square, with size 128×128. The density of the depth constraints were set to 15%, 30% and 50% respectively.

The root mean square error (RMSE) at the *i*th iteration is defined as follows:

$$RMSE^{(i)} = ((\sum_{j=1}^{n} [x_j^{(i)} - \alpha_j]^2) / n)^{1/2}.$$

In Table 3-3, we show the number of iterations *i* to attain the specified fraction of the initial RMSE value. The results are tabulated side by side for three different iteration methods, the conjugate gradient, the adaptive Chebyshev acceleration, and the Gauss-Seidel method. We observe that the conjugate gradient method performs best in the sense that it takes the least number of iterations. The adaptive Chebyshev acceleration method comes next and the Gauss-Seidel method performs worst. But we should note that each step of the iteration of the first two methods is completely parallelized so that overall execution is much faster compared to the Gauss-Seidel method where the computation is done in serial fashion. As the depth constraints become sparser, the depth interpolation problem itself becomes inherently harder to solve and takes more iterations. Even here, the degradation in the Gauss-Seidel method turns out to be the worst.

We have derived before that the total number of machine cycles per iterations are 47197 for the adaptive Chebyshev acceleration method and 73299 for the conjugate gradient method, respectively. In Table 3-4 we show the normalized number of iterations for two methods in the first two columns. For the adaptive Chebyshev acceleration method, we use the same numbers as in Table 3-3. For the conjugate gradient method, we have multiplied the number of iterations in Table 3-4 by 73299 / 47197 = 1.5530. After normalization, the conjugate gradient method still performs better than the adaptive Chebyshev acceleration method. This is in part due to the errors in the initial estimates of the eigenvalues. The initial estimates of the smallest and the largest eigenvalues, m_E and M_E, were $-\|G\|_\infty = -3.0$ and 0.0, respectively.

We show the number of iterations i in Table 3-3 and the normalized values in Table 3-4. We have similar results, but the overall number of iterations are smaller, because of the parameter change in the nodal equations.

For the Chebyshev, acceleration method run with more accurate initial estimates, we proceeded in similar fashion. For the initial estimates of m_E, we used -2.3. When the initial estimates of m_E and M_E were $-\|G\|_\infty = -3.0$ and 0.0, we obtained .9795, .9932, and .9981 as the final estimates of M_E values at the iteration steps of 90, 153, and 295 when the density of the depth constraints were varied to 50%, 30%, and 15%, respectively. As the improved initial estimates of M_E, we used .97, .98, and .99, respectively. We note that the adaptive Chebyshev accelaeration method performs better than the conjucate gradient method if near optimal values for the minimum and maximum eigen values were used as the initial values.

Operations	*local*	*global*	TOTAL (machine cycles)
Addition, Subtraction	16	$6(\mathit{log}\ s)$	$(6(\mathit{log}\ s)+16)\times 348$
Multiplication	11	1	12×563
Division	1	0	1×993
Mesh Communication	16	0	16×160
Tree Communication	0	$12(\mathit{log}\ s)+3$	$(12(\mathit{log}\ s)+3)\times 192$

Table 3-1: Summary of Operations (Chebyshev Accel. Method)

Operations	*local*	*global*	TOTAL (machine cycles)
Addition, Subtraction	30	$8(\mathit{log}\ s)+3$	$(8(\mathit{log}\ s)+33)\times 348$
Multiplication	18	5	23×563
Division	2	0	2×993
Mesh Communication	32	0	32×160
Tree Communication	0	$16(\mathit{log}\ s)+4$	$(16(\mathit{log}\ s)+4)\times 192$

Table 3-2: Summary of Operations (Conjugate Gradient Method)

RMSE	Conjugate Grad.			Chebyshev Accel.			Gauss-Seidel		
	50%	30%	15%	50%	30%	15%	50%	30%	15%
0.5	4	7	12	17	23	36	8	15	36
0.2	9	15	28	27	39	67	21	42	109
0.1	13	22	42	33	52	94	33	66	187
0.05	18	29	56	42	66	123	45	94	288
0.02	24	40	82	53	85	164	62	138	470
0.01	29	49	100	60	102	203	77	178	647
0.001	46	77	152	90	153	295	131	339	1323

Table 3-3: Number of Iterations (cylinder)

RMSE	Conjugate Grad.			Chebyshev Accel.					
	50%	30%	15%	50%	30%	15%	50%	30%	15%
0.5	6.2	10.9	18.6	17	23	36	6	8	12
0.2	14.0	23.3	43.5	27	39	67	10	17	32
0.1	20.2	34.2	65.2	33	52	94	15	26	58
0.05	28.0	45.0	87.0	42	66	123	20	39	91
0.02	37.3	62.1	127.3	53	85	164	28	64	128
0.01	45.0	76.1	155.3	60	102	203	36	75	159
0.001	71.4	119.6	236.1	90	153	295	69	121	266

Table 3-4: Normalized Number of Iterations (cylinder)

4. Parallel Texture Algorithms

In this part of the report, we report progress on the development and testing of parallel texture algorithms. The methods used for determining surface orientation from textural cues rely upon the distortions caused by (1) the effects of different perspectives as the surface recedes and (2) the projection of the surface onto the image. The first effect, originally suggested by J.J.Gibson as the primary effect used for surface perception by humans, causes that part of the surface which is farther away from the viewer to be represented by a smaller portion of the image. This is because of the oblique angle between the viewer and the surface when perspective is considered. Although this effect is clearly a powerful cue for human perception, machine perception methods based upon it, have been constrained by the need for strong assumptions based on the existence of texels (texture elements) and some texel regularity such as uniform area or equal spacing between texels. Given some such assumption, a texture gradient measure is defined from which it is possible to infer the geometry which gave rise to it. For this reason, shape from texture methods which rely on this type of distortion can be categorized as gradient methods.

The second effect, often called 'foreshortening', is seen as a compression in the direction of inclination of the surface. A well-known example is the transformation of an image of a circle into an ellipse whose parameters depend on the orientation of the surface. In particular, a circle on a surface with slant σ and tilt τ when orthographically projected becomes an ellipse whose direction of its minor axis is the same as the tilt and whose ratio of minor axis b to its major axis a is the cosine of the slant. This effect occurs in the same way for all parts of the surface regardless of how far away from the viewer any part is. Hence, it is often not necessary to get involved with the complicated geometry of perspective projection in order to use this distortion to infer surface orientation. Since it is not necessary to make any assumptions about the existence of texels, methods based on this distortion have been more statistical in nature and hence more accurate for large pictures of a single uniform natural texture. A good example of this type of shape from texture method was developed by A.P.Witkin.

Within our objective to integrate our stereo and texture algorithms with those of other ALV participants, we have tested an algorithm for on/off road segmentation using the data we have received from ERIM's ALV Data Collection. The algorithm uses micro-edge densities, following the road from previous pictures and experimenting with variable size windows which change in parallel to accommodate the current locations of the road edges. Figure 4-1 shows the application of this algorithm to one of the road scenes obtained from ALV data collection. In addition, we are exploring the use of the density gradients to determine relative orientation. At present we have been limited by low texture resolution and have been in contact with Martin Marietta about possible improvements.

A texture segmentation scheme based on a combination of pyramid linking spatial grey level dependence statistics have been tested on the pyramid programming environment. The algorithm is based on constructing links between adjacent levels in the multi-resolution pyramid of the image. The average pyramid is constructed such that each node on an intermediate level is set equal to the of its own children and the children of its east, south, and east-south siblings. It follows that each node has at most 16 children, and it contributes to at most 4 parents. In the linking process, each node establishes a link between it and the parent which has a value most near to its own. Once this is established, we traverse the pyramid top/down and color the children with the same color of the parent they are linked to. This iteration is repeated, but this time the average value is computed based on the children with links established to the parent. the iteration should terminate when no change in coloring is happening (basically the segmentation is the same). Figure 4-2 shows the application of this algorithm to segmenting a road image of the ALV data collection. The figure shows the original image and the segmented image after three iterations.

In what follows, we report on the progress of testing and implementing two texture algorithms, and a system that integrate texture algorithms to determine surface orientation.

4.1 Analysis and Parallel Implementation of Witkin's Method

Lisa Brown & Hussein Ibrahim

In this section, we discuss a parallel implementation of A.P.Witkin's algorithm for the recovering of surface orientation from texture for the highly parallel, SIMD, tree-structured NONVON supercomputer. Witkin's approach to the surface from texture problem is well-suited to a wide spectrum of textures and relies upon the following assumptions:

1. The image is an orthographic projection representing a planar surface.
2. The texture itself is considered to be the distribution of edge directions and this distribution is assumed to be uniform prior to the effects of projection. More importantly, the method will work to the extent in which the distribution of edge directions that compose the texture do not resemble the effects of projection.

Figure 4-1: A road scene and its segmentation using edge density counting

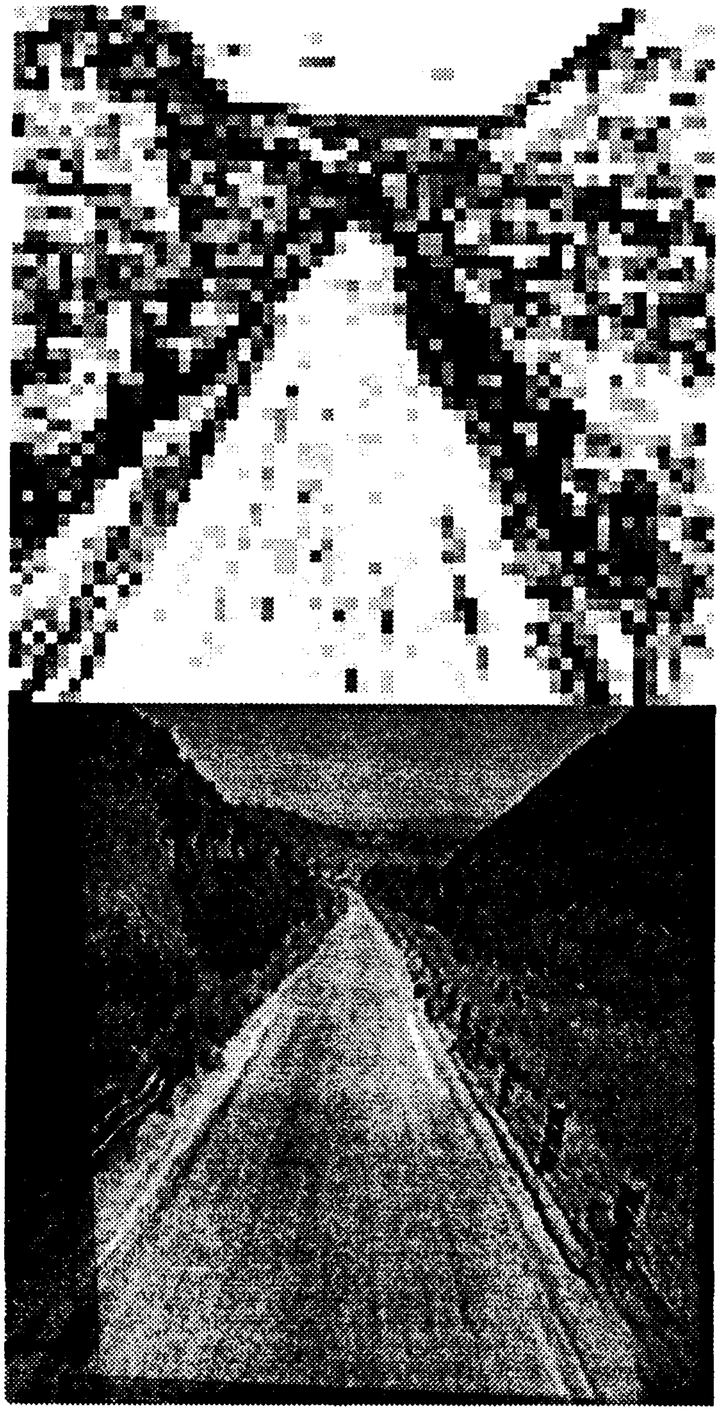
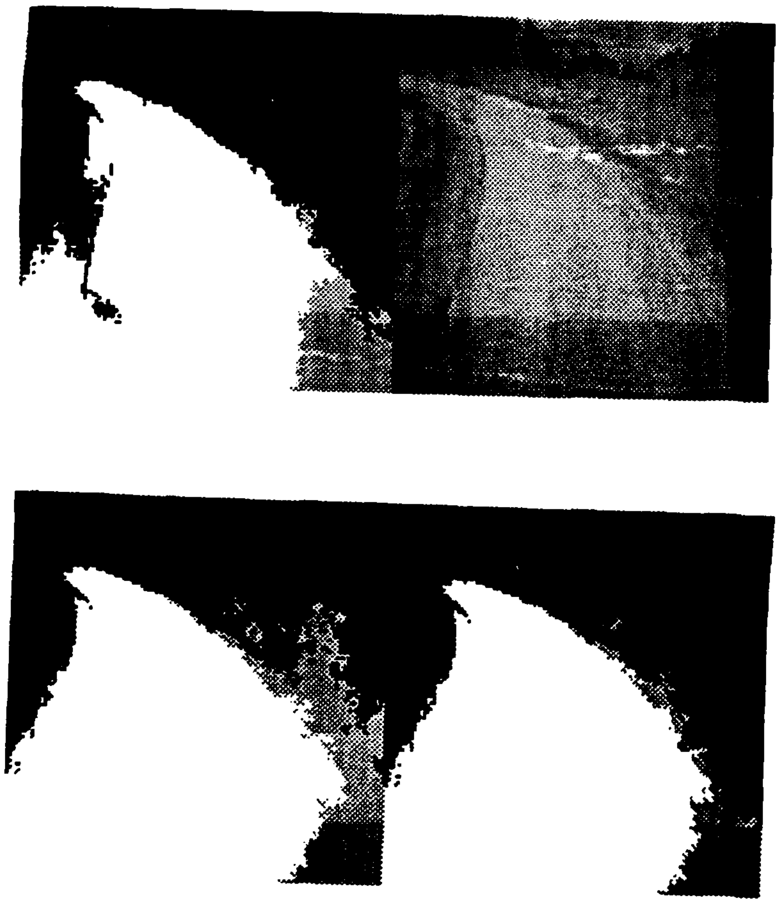


Figure 4-2: Pyramid-Linking segmentation of a road image



3. All surface orientations are equally likely. If we consider each surface as being represented by a point on the Gaussian sphere then a given surface is equally likely to be represented by any point on the sphere.

Given that these assumptions hold completely, this method has already been proven to be quite robust. The major drawbacks are that it fails when confronted with textures with regular features (assumption 2) and it is computationally inefficient on a serial machine. we have considered ways to improve the efficiency through parallelization. We take care to use representations which give accurate measurements and can be used to extend the method to a larger domain.

The idea behind Witkin's method is as follows. We assume that the distribution of edge directions is uniform prior to the projection. If the surface is slanted about an axis parallel to the X-axis by an angle σ with respect to the image plane then we expect the distribution of edge directions to become dominated by edges whose directions are close to horizontal. The larger σ is, the greater the domination of these edges. This makes intuitive sense since all the edges in the surface are being more and more compressed along their vertical component. If the surface is slanted about a different axis which makes a tilt angle τ with the X-axis, then the distribution is just shifted.

The three phases of the algorithm as implemented on NON-VON are:

1. The determination of the edges in the image in each of the edge directions considered.
2. The histogramming of the edge directions.
3. Computing the likelihood for the histogram for each possible surface orientation and finding the maximum likelihood.

Finding the edges in different direction is computationally the most important phase of the algorithm. Since this phase is only the input into the algorithm it is often overlooked; but the choice of representation and method used here greatly effect the types of information available for subsequent use. Once the histogram of edge directions is found the computation of the likelihood estimates is well defined and computationally less intensive. However, there are several different ways to determine the edges and their orientations and we would like to do this both efficiently utilizing our parallel architecture and with an eye to facilitating the detection of edges which do not satisfy the assumptions.

For the initial detection of the edges, Witkin proposed the Marr-Hildreth zero-crossing operator which he used successfully to demonstrate his algorithm. This method is also applied here using the parallel functions for the NONVON as described earlier.

The multiresolution approach described earlier can be used to improve edge detection. Marr and Hildreth [10] have developed this idea. Each level is used to find the zero-crossings for differing frequency bands and then a set of parsing rules determine the relationship between the zero-crossings at each level. The parsing rules are determined by physical constraints such as the spatial localization of intensity changes due to each physical phenomena in the visual world. Thus, zero-crossings are expected to be found at each level unless more than one phenomena have been combined from a higher frequency band. From these rules, the primitives of the primal sketch are found. This same technique might by exploited for use with Witkin's algorithm. Distribution of edge directions found at each level could be analyzed together with the parsing rules to determine the frequency band which give the most reliable results or greatest likelihoods. In fact, one of the problems encountered with zero-crossings is that more than what is normally perceived as the edges in the image are detected. This is significant since the set of edges which are optimal for Witkin's method are not known. In our current implementation however, just the base level is used for edge detection.

Given this technique for edge detection, it still remains to determine the directions of the edges. To do this, let us first consider the principles behind our edge detection scheme. The Marr-Hildreth operator was selected as the optimal smoothing filter that satisfied two physical constraints. The first constraint is one of spatial localization since it is believed that only nearby points contribute information to the intensity changes at any given point. The second constraint is a frequency localization. We would like a smooth and band-limited filter. These constraints lead directly to using Del^{2}G operator to find the zero crossings. The first thing to note is that finding a zero-crossing in a given direction does not necessarily tell us about the direction of an edge at this point. Indeed, it is possible that there is a zero-crossing in {*every*} direction at a given point when for example, there is only a single edge formed by a uniform intensity change with constant lines that run parallel to the edge. To choose among several zero-crossings at one point, we have two simple strategies:

1. Choose the zero-crossing which has the maximum slope.
2. Choose the zero-crossing whose orientation agrees with the orientations of other neighboring zero-crossings.

The second strategy is supported by the theorem that Marr and Hildreth call, the *condition of linear variation* in which for smoothed images (as in our case) the two strategies above are equivalent. The theorem tells us that the intensity variation near and parallel to the line of zero-crossings should be locally linear. We are ultimately interested in the direction of an edge at this point, and we see that according to this theorem the line of zero- crossings is in the direction of the edge since we expect linear intensity changes along and parallel to the edge and a maximum change across it.

Thus, there are two approaches that can be taken in extracting the edge directions from the zero-crossings, both which can be easily implemented in our parallel scheme. The first approach takes the edge direction as orthogonal to the orientation of the zero-crossing with the maximum gradient. The second approach considers an edge only if there are two zero-crossings of the same orientation which are adjacent to each other on the line perpendicular to the zero-crossing orientation. The direction of this edge is then the direction formed by the line on which the two zero-crossings lie. This method has the advantage that an edge is more clearly part of contour and less likely to have been contributed by noise. On the other hand, since we are only interested in texture, a contour may be irrelevant. Finally, an edge in the vertical or horizontal direction is more easily found due to the nature of the grid and this will skew our histogram values.

Once the edge directions have been found, we can take advantage of the pyramid tree communication capability to compute the histogram. Since the edge directions now reside at the base level, we simply accumulate the number of edges in each direction as we ascend the pyramid. The algorithm will require only $O(\log n)$ operations where n is the number of pixels in the image as opposed to $O(n)$ time needed by a sequential machine. There are of course several alternatives to this scheme for determining the edge directions. Two methods that we have considered are: (1) using the gradient determined by the orthogonal Sobel operators and (2) using the wedge shaped regions centered at the origin of the power spectrum of the image.

4.1.1 Orientation Likelihoods

This phase of the algorithm although computationally somewhat complex, is not nearly as computationally intensive and therefore we initially thought that it did not justify implementation on a parallel machine but could more easily be accomplished by its host computer. Basically, it consists of computing the likelihood of each possible orientation given the distribution of edge directions. While the number of possible orientations is infinite, they can be well represented by a small sized subset on the order of a hundred. However, after considering all the benefits, it becomes clear that it is worthwhile to take advantage of the parallel architecture.

In our parallel implementation each PE represents one likelihood, so that a fine grained output of the orientation likelihoods could be obtained. Once these likelihoods are found, the maximum can be found by traversing up the tree. But in addition, we have created an ideal structure for outputting {*all*} the likelihoods to other shape-from methods and for feedback between the likelihoods and the edge direction distributions found at varying bandwidths and thresholds. This latter result, can be used to identify the edges which optimally satisfy the assumptions, allowing Witkin's method to be used on more natural textures successfully. A sample of our results is shown in Figure 4-3.

A more appropriate choice of the coordinate space representation of orientation can lead to more accurate results. The singularity that occurs when the slant is small is a result of the nonuniform probability associated with each orientation region, the regions being smallest when the slant approaches zero. If instead of selecting equally spaced intervals of σ, τ, we choose equal size areas on the Gaussian Sphere, the joint pdf is simply a uniform distribution and thus a constant. The advantages are twofold: large errors at small slants are avoided and the resolution of the results becomes more consistent.

Figure 4-4 shows the decrease in the error as a function of slant by transforming the orientation space into Gaussian Sphere. Ten random samples of 2000 uniformly distributed tangent directions were projected at a constant tilt for each slant. The projected tangent directions are then histogrammed and given as input into Witkin's original method which finds the maximum likelihood of 100 surfaces: 10 slants and 10 tilts, equally spaced and into the modified method which finds the maximum likelihood of 100 surfaces approximately uniformly spaced on the Gaussian Sphere. The implementation of this method on the simulator for the NONVON computer, has exposed both the advantages and limitations of Witkin's statistical approach to texture. One obvious limitation is that not all surface orientations can be distinguished. For every surface orientation that the method finds, there is another orientation which slants in the opposite direction which is equally likely. This is obviously counter to human perception. It suggests that this method is somehow incomplete. However there are several reasons why this method is appealing. Most other methods which derive shape from texture rely upon gradient measures. These methods rely upon determining texel size, shape or spacing and assumptions concerning their uniformity. These methods are complicated by determining or locating texels and limited to very specific domains. Notice how these methods contrast with Witkin's

Figure 4-3: The center column contains the images whose orientations are top: $\sigma=0$, $\tau=45$, center: $\sigma=0$, $\tau=20$, bottom: $\sigma=45$, $\tau=45$. The left column contains the likelihood at each surface orientation where the distance from the origin is τ, and the angle is σ. The right column contains the Fourier Spectra.

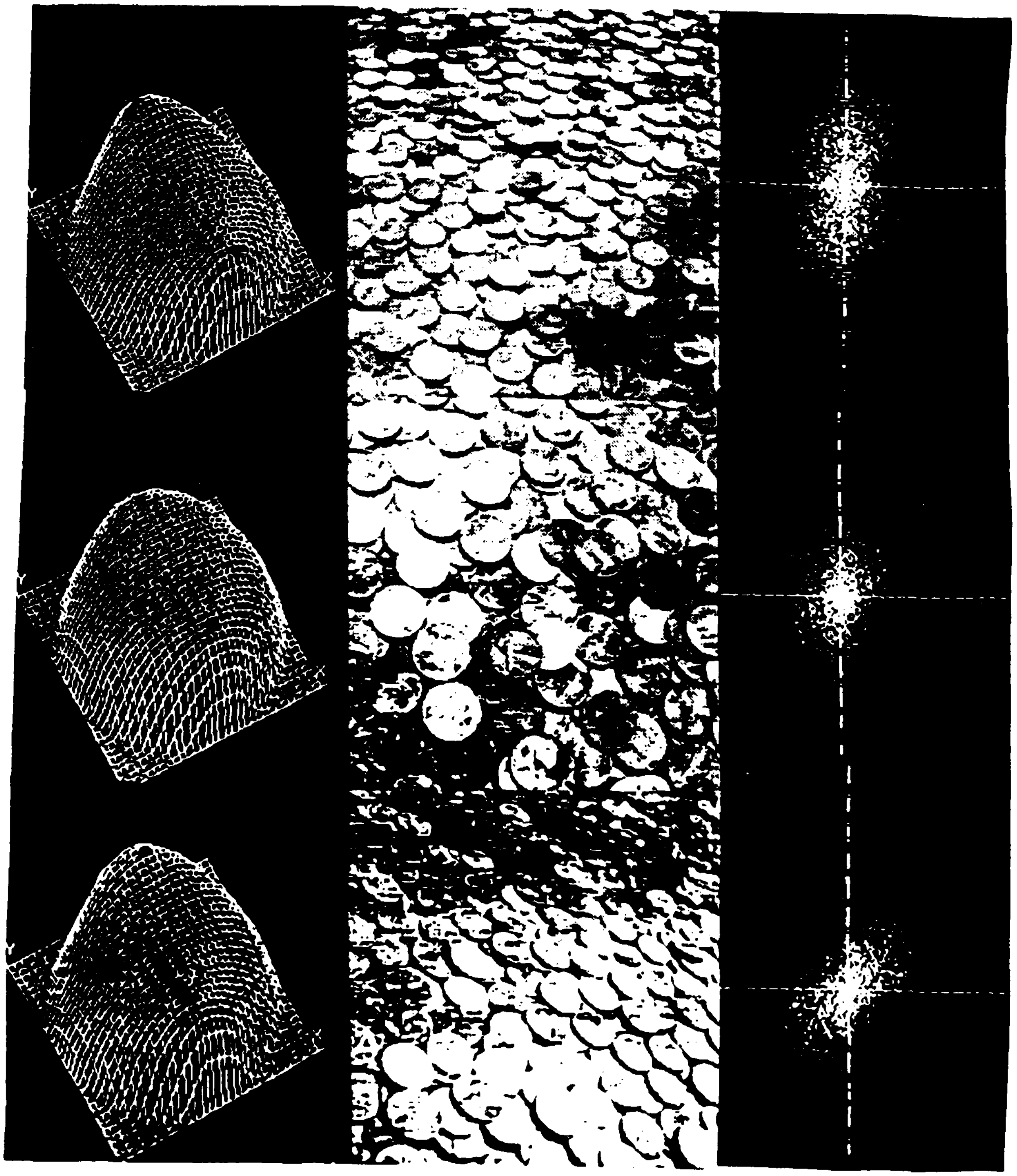
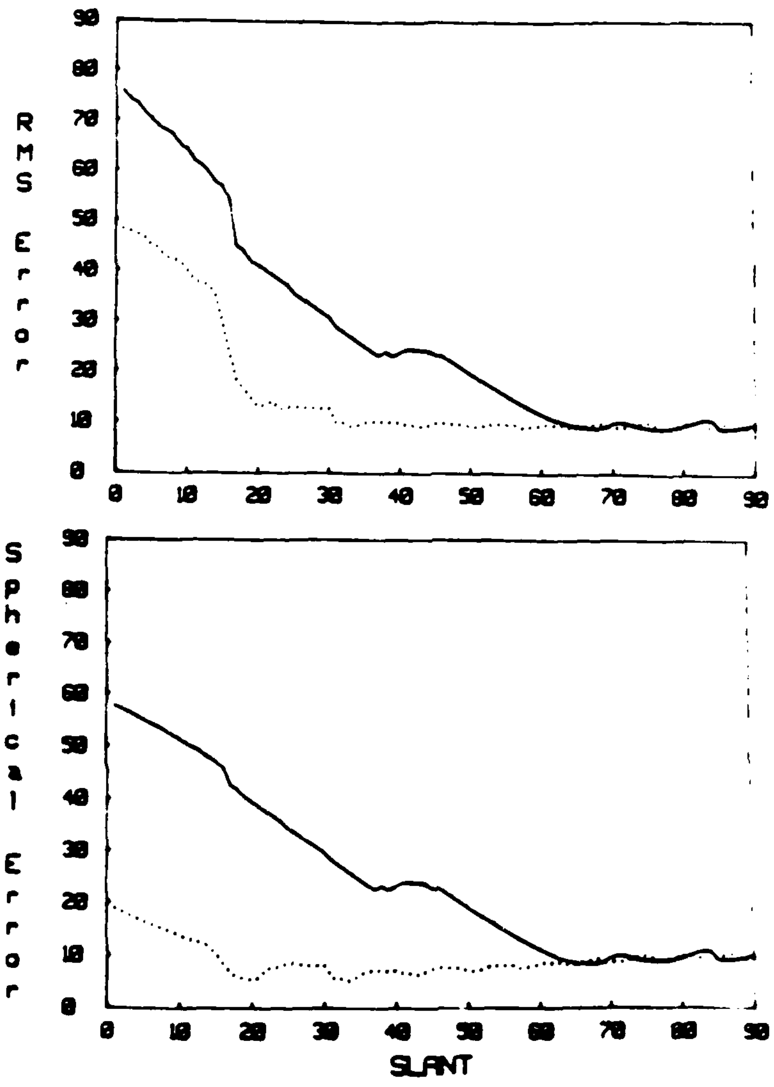


Figure 4-4: Comparison of Witkin's Original Method and Modified Version(Dotted lines)



approach which will often fail when their assumptions hold but work when they fail. For example, uniformly shaped texels will cause Witkin's method to fail if the texel shape is biased towards certain directions but when the texels are all shaped differently Witkin's is likely to be applicable. Thus, it would be useful, if Witkin's method could be applied so that instead of solely determining the orientation which is most likely to be represented by the given edge distribution, the likelihood of each orientation was used as information to be passed onto to other shape from texture methods. These other methods which measure shape from texture using gradients, can then determine precisely what Witkin doesn't: the local directionality which uniquely determines the surface orientation. This Fusion of information from different texture modules is the subject of a later section.

4.2 An autocorrelation approach to texture

Lisa Brown and Haim Shvayster

We have developed a new method for determining local surface orientation from rotationally invariant textures based on the two-dimensional two-point autocorrelation of an image. The method does not require the texture to be composed of texels or assume texture regularities such as equal area texels or equal spacings between texels. It relies only on the projective distortions of directionally homogeneous textures. The technique is applied to images of textured surfaces with a single orientation and the results are accurate even in cases where human perception is not easy. The method is simple and it uses information from all parts of the image.

In this method, we assume that the textured surface is isotropic. By this we mean that the texture is directionally homogeneous. More precisely, any measurement of length taken over an image of a surface which is not oriented with respect to the image plane (i.e. not slanted) is independent of direction.

A practical example of this concept can be portrayed by Witkin's method. Witkin measured the cumulative length of surface edges in each direction and assumed that for an image of a surface without slant this measure would be independent of direction. (Uniform probability distribution of edge directions) Then, using the effect of projecting the image orthographically he computed precisely how the probability distribution of edge directions changes for different surface orientations. Based on this, given a particular distribution of edge directions he determined the orientation of the surface of maximum likelihood. A circle can be considered both as an example of an image whose edge-lengths are independent of direction or as the polar plot of the function of edge-lengths versus direction for an image which is independent of direction.

With this formulation, there are a whole class a functions that we could utilize in order to determine surface orientation, and our task is then to select that function which is optimal for our purposes. First, let us notice some of the shortcomings and additional constraints involved in Witkin's method. The assumption in this method is that edges, are directionally homogeneous. The measurement of edges can take on countless forms, each with its own set of parameters making the determination of edges a rather arbitrarily defined concept. Furthermore, to ascertain the length of edges of a certain direction is difficult to do precisely given the discrete nature of the input. Lastly and most significantly, assuming directional homogeneity of edges constrains the textured surface much more than is necessary. Certain textures may be composed of enormous amounts of information very little of which will be measured as edges.

With these limitations in mind, we would like to find an isotropic function that will give us a more powerful method. The isotropic function, we suggest, is the well-defined and efficiently computable, autocorrelation. Traditionally, we consider the 2-D autocorrelation as a function of a point (i,j) whose value is the sum of all products of pairs of points separated by the vector r. For a given direction, the autocorrelation of an isotropic texture which is not oriented should be a constant. Thus, for any set of points which compose a centered circle, all values of the function will be constant. Furthermore, if an isotropic texture is oriented, there will exist sets of points along nested ellipses which will be constant. All of these ellipses will have the same parameters and therefore computing the parameters of any one will be sufficient to determine the surface orientation.

The following describes how the surface orientation is determined from the autocorrelation of an image. We assume that the image is of an orthographically projected planar textured surface whose texture is rotationally invariant when the surface is viewed as parallel to the image plane. As a nomenclature convention, we use the term 'slant' to specify the degree to which the surface is inclined, where no slant implies that the surface is parallel to the image plane, and a slant of 90 degrees indicates that is maximally oblique. The term 'tilt' specifies the direction of the inclination which will vary from 0 degrees when the the surface tilts away and to the right to 180 degrees when the surface tilts away and to the left. Notice that the slant of an actual surface may vary from 0 to 180 (or the tilt from 0 to 360) but when using only the effects of projective distortion (foreshortening) it is impossible to distinguish between slants that are 180 degree complements of each other when the tilt is the same (or tilts which are 360 degree compliments of each other when the slant is the same.) This is because the amount of compression for both cases is identical. To distinguish these surfaces it is necessary to use the effects of perspective distortion in which it is possible to determine which part of the surface is relatively closer.

We have shown theoretically that the 2-point 2-D autocorrelation will always be entirely composed of elliptic iso-contours, where the ellipses are all of the same family. By this we mean that they all have the same ratio of minor to major axis and the same directionality.

The second order moments of the autocorrelation image can be used to compute the ration of the minor axis to the major axis, and also to compute the angle of tilt as follows:

$$\tau=0.5tan^{-1}\frac{2\mu_{11}}{\mu_{20}-\mu_{02}}$$

$$\frac{b}{a}=\sqrt{\frac{x+y}{x-y}}$$

$$where \quad x=\mu_{20}+\mu_{02}$$

$$and \quad y=\sqrt{(\mu_{20}-\mu_{02})^2+4\mu_{11}}$$

where μ_{20}, μ_{02}, and μ_{11} are the second order moments of the autocorrelation image

The above equation computes the slope of the principal axis of the image. The principal axis is the line passing through the centroid of the image with the minimum moment of inertia. This line coincides with the major axis of an image of a single centered ellipse. In our case, several ellipses with the same eccentricity and slope are embedded within each other. Since all of these ellipses have the same principal axis, it is also the line of minimum moment of inertia for the entire autocorrelation. The equation also gives the slant of the surface in the image (the ratio of the minor and major axes).

To test our method, we used pictures of natural images taken by two different processes. Using a CCD video camera, pictures were taken in our lab and digitized by our Grinnell Image Processor. From both 35mm photographs and high quality pictures found in magazines such as {*National Geographic*}, we also created digitized images using a Sharp Image Scanner which was capable of scanning at as high a resolution as 300bpi.

To initially test our method, we computed the autocorrelation using Fast Fourier Transform and used video images of homogeneous isotropic textures such as pennies, packing bubbles, and sand. The first obstacle that became apparent was that the autocorrelation of these images uniformly contained significant noise levels that occurred dominantly in the horizontal direction. This was particularly evident at points that are far away from the origin where very low values of correlation are expected. Since all the pictures taken from the CCD camera had this noise but synthesized images did not, it was hypothesized that this noise was caused by horizontal smearing due to the CCD camera. Further tests using images of photographs of natural outdoor scenes such as sidewalks, leaves and grassy hills that were digitally scanned, showed the same effect was already present in the photographs.

Since it was evident that the problem of horizontal smearing was due to several commonly found hardware acquisition systems, we opted for a software solution. In order to diminish the effect of the horizontal noise, we designed a simple thresholding technique that could extract it, regardless of the qualities of the original image. The important thing to notice about this design was that it did not introduce a 'user-specified' threshold parameter. The thresholding process that we used was to automatically set the thresholding value - above which all values were retained unchanged - based on the average value found in the autocorrelation along the circumference of a small circle. This was an effective scheme because for a large range of circle radius tested (5 to 25) on the autocorrelation of several different images the effects of the different radii were very similar. Thus an arbitrary setting of the circle radius to 10 worked well for almost all of the pictures we tested.

This worked successfully as can be seen in a representative example shown in Figure 4-5. The top right picture in the figure shows the noise due to the horizontal smearing on the autocorrelation. The other three pictures are the same autocorrelation using the automatic-thresholding with circle radii of 5, 10 and 25. These pictures are also magnified 4 times the original so that the nested ellipses can be seen clearly and the similar elliptic parameters for all three pictures is easily discerned. The figure shows how the horizontal noise is effectively extracted but is insensitive to the circle radius.

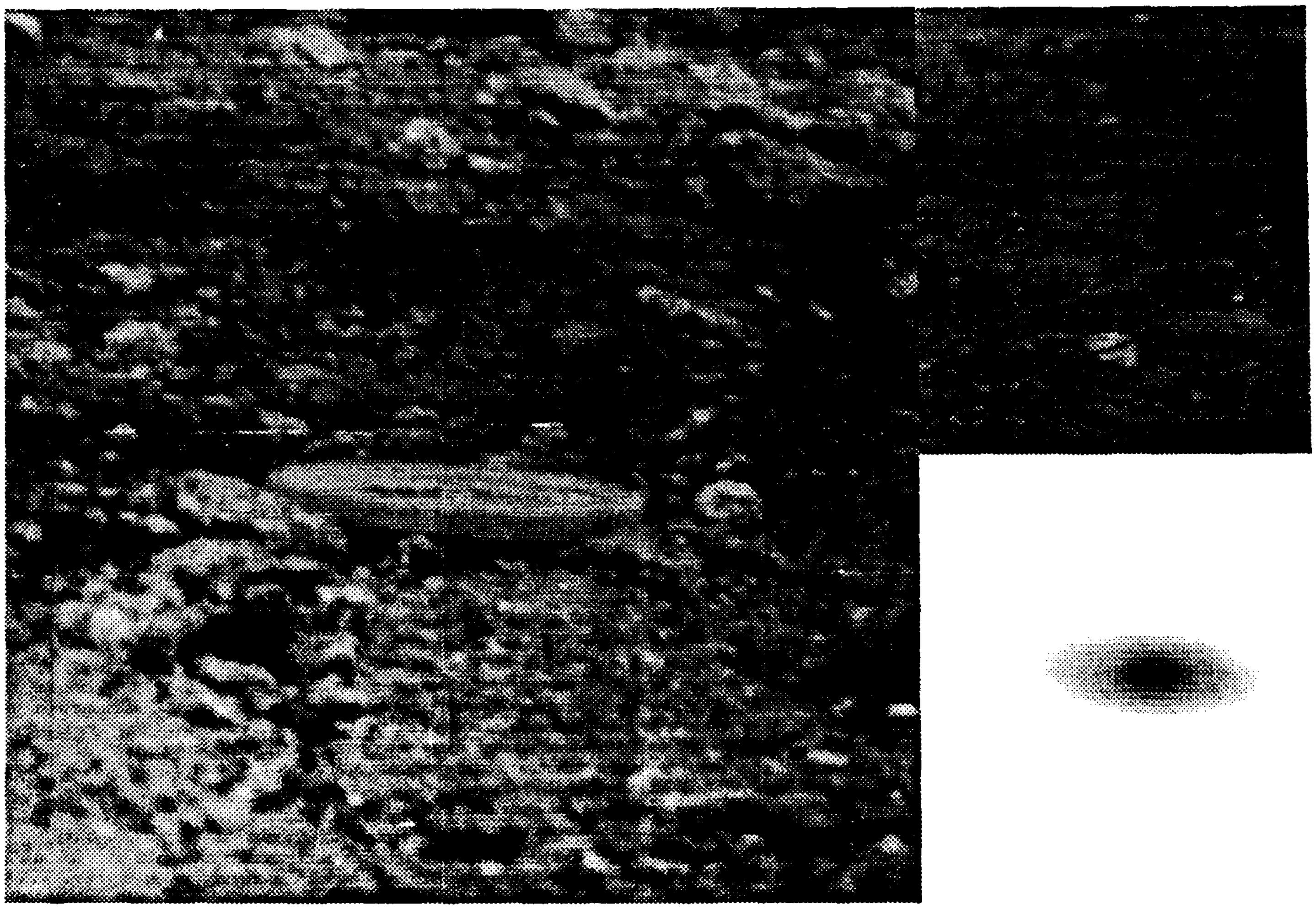
Using the automatic-thresholding technique, the results of the Principal Axis method were improved dramatically. For a large set of natural images which satisfied the assumption of an isotropic texture we compared the results with those of Witkin's. Most other shape from texture methods could not be used since few of the natural isotropic textures used contained practical texels.

4.3 An Integrated System That Unifies Multiple Shape From Texture Algorithms

Mark L. Moerdler and John R. Kender

This section describes an approach which integrates several conflicting and corroborating shape-from-texture methods in a single system. The generality of this approach is due to the interaction between textural cues, allowing the methodology to extract shape information from a wider

Figure 4-5: An Example of Computing Surface Orientation Using Autocorrelation approach



```
ACTUAL SLANT= 76, TILT= 89 DEGREES
COMPUTED SLANT= 67, TILT= 86 DEGREES
```

range of textured surfaces than any individual method. The system uses a new data structure, the *augmented texel*, which combines multiple constraints on orientation in a compact notation for a single surface patch. The augmented texels initially store weighted orientation constraints that are generated by the system's several independent shape-from-texture components. These texture components, which run autonomously and may run in parallel, derive constraints by any of the currently existing shape-from-texture approaches e.g. shape-from-uniform-texel-spacing. For each surface patch the *augmented texel* then combines the potentially inconsistent orientation data, using a Hough transform-like method on a tesselated gaussian spheres, resulting in an estimate of the most likely orientation for the patch. The system then defines which patches are part of the same surface, simplifying surface reconstruction.

4.3.1 Design Methodology

The generation of orientation constraints from perspective distortion uses one or more image texels. The orientation constraints can be considered as local, defining the orientation of individual surface patches (called *texel patches**) each of which covers a texel or group of texels. This definition allows a simple extension to the existing shape-from methods beyond their current limitation of planar surfaces or simple non planer surfaces based on a single textural cue. The problem can then be considered as one of intelligently fusing the orientation constraints per patch. Ikeuchi [8] and Aloimonos [1] attempt a similar extension based on constraint propagation and relaxation for planer and non planer surfaces for using only a single shape-from-texture method.

The method consists of three major phases, the calculation of orientation constraints and the generation of *texel patches**, the consolidation of constraints into a "most likely" orientation per patch, and finally the reconstruction of the surface.

During the first phase the different shape-from-texture components generate texel patches and *augmented texels*. Each augmented texel consists of the 2-D description of the texel patch and a

*Texel patches are defined by how each method utilizes the texels. Some methods (e.g. Uniform texel size) use a measured change between two texels; in this case the texels patches are the texels themselves. Other methods (e.g. Uniform texel density) use a change between two areas of the image , in this case the texel patches are these predefined areas.

*A *texel patch* is a 2-D description of a subimage that contains one or more textural elements. The number of elements that compose a patch is dependent on the shape-from-texture algorithm.

list of weighted orientation constraints for the patch. The orientation constraints for each patch are potentially inconsistent or incorrect because the shape-from methods are locally based and utilize an unsegmented, noisy image.

In the second phase, all the orientation constraints for each augmented texel are consolidated into a single "most likely" orientation by a Hough-like transformation on a tesselated gaussian sphere. During this phase the system will also merge together all augmented texels that cover the same area of the image. This is necessary because some of the shape-from components define "texel" similarly, and the constraints generated should also be merged.

Finally, the system re-analyzes the orientation constraints to determine which augmented texels are part of the same constraint family and groups them together. In effect, this segments the image into regions of similar orientation. In order to build a complete system one may also want to reconstruct surfaces from these surface patches [3].

The robustness of this approach is illustrated by a system that fuses the orientation constraints of two existing shape-from methods: shape-from-uniform-texel-spacing [11], and shape-from-uniform-texel-size [12]. These two methods generate orientation constraints for different overlapping classes of textures.

4.3.2 Surface Patch And Orientation Constraint Generation

The first phase of the system consists of multiple shape-from-texture components which generate augmented texels. Each augmented texel consisting of a texel patch, orientation constraints for the texel patch, and an assurity weighting per constraint. The orientation constraints are stored in the augmented texel as vanishing points which are mathematically equivalent to a class of other orientation notations (e.g. tilt and pan as gradient constraints) [13]. Moreover, they are simple to generate and compact to store.

The assurity weighting is defined separately for each shape-from method and is based upon the intrinsic error of the method. For example, shape-from-uniform-texel-spacing's assurity weighting is a function of the total distance between the texel patches used to generate that constraint. A low assurity value is given when the inter-texel distance is small (1 texel distance) because under these conditions a small digitization error causes a large orientation error. Above this threshold the assurity weighting is set high and then starts to decrease as the inter-texel distance increases. (The optimal shape of this assurity function is under investigation.)

4.3.3 Most Likely Orientation Generation

Once the orientation constraints have been generated for each augmented texel, the next step consists of unifying the constraints into one orientation per augmented texel. The major difficulty in deriving this "most likely" orientation is that the constraints are errorful, inconsistent, and potentially incorrect. A simple and computationally feasible, solution to this is to use a gaussian sphere which maps the orientation constraints to points on the sphere [13]. A single vanishing point circumscribes a great circle on the gaussian sphere; two different constraints generate two great circles that overlap at two points uniquely defining the orientation of both the visible and invisible sides of the surface patch.

The gaussian sphere is approximated, within the system, by the hierarchical by tesselated gaussian sphere based on trixels (triangular shaped faces [2, 4, 9]. The top level of the hierarchy is the icosahedron. At each level, other than the lowest level of the hierarchy, each trixel has four children. This hierarchical methodology allows the user to specify the accuracy to which the orientation should be calculated by defining the number of levels of tesselation that are created.

The system generates the "most likely" orientation for each texel patch by accumulating evidence for all the constraints for the patch. For each constraint, it recursively visits each trixel to check if the constraint's great circle falls on the trixel, and then visiting the children if the result is positive. At each leaf trixel the likelihood value of the trixel is incremented by the constraint's weight. Although this is a search process the hierarchical nature of this approach limits the number of trixels that need to be visited.

Once all of the constraints for a texel patch have been considered, a peak finding program smears the likelihood values at the leaves. Currently, this is done heuristically by a rough approximation to a gaussian blur. The "most likely" orientation is defined to be the trixel with the largest smeared value.

4.3.4 Surface Generation

The final phase of the system generates surfaces from the individual augmented texels. This is done by re-analyzing the orientation constraints generated by the shape-from methods in order to determine which augmented texels are part of the same surface. In doing this, the surface generation is also performing a first approximation to a surface separation and segmentation.

The re-analysis consists of iterating through each augmented texel, considering all its orientation constraints, and determining which constraints aided in defining the "correct" orientation for the texel patch as described in the previous phase. If an orientation constraint correctly determined the orientation of all the texels that were used in generating the constraint, then these augmented texels are considered as part of the same surface.

4.3.5 Testing The Methodology

The knowledge fusion approach outlined in the previous section has been applied to a test system that contains two shape-from-texture methods, shape-from-uniform-texel-spacing [11], and shape-from-uniform-texel-size [12]. Each of the methods is based on a different, limited type of texture. Shape-from-uniform-texel-spacing derives orientation constraints based on the assumption that the texels on the surface are of arbitrary shape but are equally spaced. Shape-from-uniform-texel-size is based on the unrelated criteria that the spacing between texels can be arbitrary but the size of all of the texels are equivalent but unknown.

Under certain conditions either method may generate incorrect constraints, which the system will ignored. On textures that are solvable by both methods, they cooperate and correctly define the textured surface or surfaces in the image. Some images are not solvable by either method by itself but can only be correctly segmented and the surfaces defined by the interaction of the cues.

Real images contain noise and shadows which are effectively ignored by the system in many cases. The system treats shadows as potential surface texels and uses them to compute orientation constraints. Since many texels are used in generating the orientation for each individual texel the effect of shadow texels is minimized. Even under the conditions where many shadow texels are found they do not effect the computed orientation of surface texels so long as the placement of the shadow texels does not mimic perspective distortion.

Noise can occur in many ways: it can create texels, and it can change the shape, size, or position of texels. If noise texels are sufficiently small then they are ignored in the texel finding components of the shape-from methods. When they are large, they are treated in much the same way as shadow texels and thus often do not affect the orientation of the surface texel patches. Since many texels are used and more than one shape-from method is employed, noise-created changes in the shape of texels can perturb the orientation results, but the effect appears negligible as shown in the experimental results.

The system has been tested over a range of both synthetic and natural textured surfaces, and appears to show robustness and generality. Figure 4-6 shows a real image of a man-made texture consisting of equally spaced, equally sized circles. The system finds fourteen texels: the twelve texels on the surface, plus two noise texels located in the background. It is able to generate the correct gradient space p and q values for each of the twelve surface texels (see figure 4-7 for the positions of the texels and figure 4-9 for the individual p and q values.) In figure 4-8 the orientations of the texel patches are displayed as needle-like surface normal vectors.

The system is also able to segment the image into three surfaces, one of which contains only the twelve correct surface texels. The noise-generated texels are each individually marked as parts of separate surfaces.

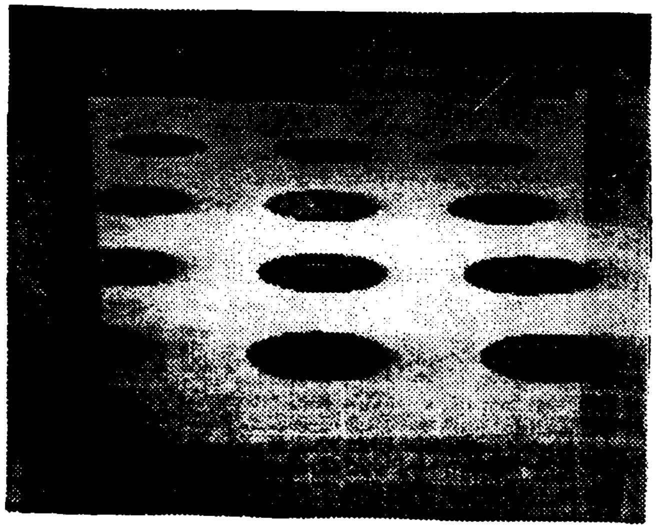


Figure 4-6: A texture of equal spaced and sized circles

Future enhancements to the system would include addition of other shape-from-texture modules, investigation of other means of fusing information (such as object model approaches), analysis of curved surfaces, studies of error behavior, and optimization of the fusion approach, especially in a parallel processing environment.

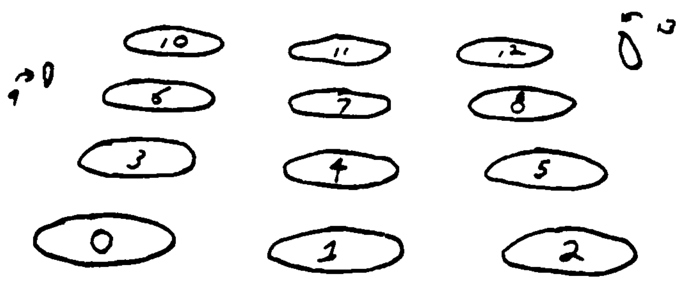


Figure 4-7: The numbering of texels for the circles texture

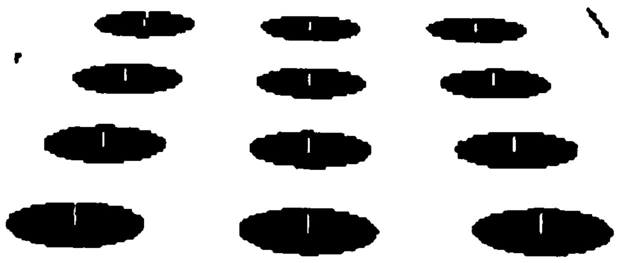


Figure 4-8: Surface normals for the circles texture

Figure 4-9: Orientation values for the circles texture

Texel Number	Measured p & q	Actual p & q	error

0 to 8	p =3.0
	q =0.0
		p = 3.0
		q = 0.0
			0^o
			0^o

9	p=11.0
	q=6.7
		Shadow Texel	

10 to 12	p =3.0
	q =0.0
		p = 3.0
		q = 0.0
			0^o
			0^o

13	p=11.0
	q=6.7
		Shadow Texel	

5. Integrating Stereo and Texture Information

Terrance E. Boult and Mark Moerdler

This section describes a work progressing in its initial phase that concentrates on combining stereo and texture techniques to determine surface orientation (depth values). As research in vision has progressed, researchers began to realize that the information available from a single "shape-from" algorithm would not be sufficient to solve the general vision problem. Prior vision research has yielded different "modalities" of information including: numerous approaches to shape-from-texture, binocular stereo, shape-from shading, and shape-from-motion. Each of these sources of shape information has different domains of applicability, different computational complexity, and different error characteristics. For any given module, there would exists numerous images (or regions thereof) for which the module would not correctly predict surface shape. Some of the sources are complementary, e.g. shape-from-shading will apply generally only in those regions where shape-from-texture will fail. Other modules can act in either a competitive or synergistic fashion, e.g. binocular stereo and shape-from-texture will generally apply in the same regions of an image, and may compete for dominance if their outputs differ, or can they mutually reinforce a consistent interpretation.

The system under development uses two levels of data *fusion*, *intra-process integration* and inter-process integration. The former is fusion of information generated by all shape-from-X approaches with certain predetermined similarities, e.g., feature based stereo algorithms with different features. The latter type of integration is the fusion of the information resulting from each of the intra-process integration phases, with any {*a priori*} knowledge, e.g., smoothness assumptions or model assumptions. However, to allow for some amount of top-down processing, there is communication between each process through a global blackboard.

The techniques for intra-process integration are dependent on the assumptions, explicit and implicit, in the underlying processes. Brief presentation are given of two such intra-process integration techniques. Inasmuch as the final result of the data fusion is assumed to be objects with smooth surfaces, the inter-process integration should depend on our model(s) of surfaces, not on the data acquisition techniques. The current system employs a regularization based surface reconstruction technique for this task. This approach allow the system to independently weight each piece of information from the intra-process integration phases. Part of this weighting is a global factor determining which of the modalities has higher priority.

The interaction among the various computational modules as well as the integration modules, is accomplished with a blackboard organization. This scheme allows bidirectional flows of information and provides a means for easy detection of when the information necessary for modules execution is present in the system. This portion of the system will not be considered further in this paper.

In a previous section we have discussed an approach to the problem of deriving the orientation information from multiple independent textual cues. The stereo-based processes of the system are based on feature matching between the two images. The system uses multiple feature definitions to insure both good localization and noise resistance. These feature are then classified as to amount of ambiguity. The system starts with the least ambiguous matches and reconstructs a disparity surface. Intra-stereo-integration is accomplished through a regularized reconstruction of the disparity field based on the supposition that the smooth surfaces in the world give rise to a smooth disparity surface. After all points are considered, the intra-process module adds its output to the blackboard. Currently this output is depth values at various points, especially along the ''edges'' of surfaces in the disparity field and at the locations of feature points.

The stereo module currently combines two different types of features. These are: (1) zero crossings of laplacian of gaussians of the images, which are subsequently thresholded (based on magnitude of crossing) and matched along approximately epi-polar lines using orientation and sign as a filters), and (2) centroids of texels defined in the shape-from-texture algorithm (with some of the other texel features used to insure only valid matches). The first of these features provide a large number of features for the matching algorithm, unfortunately the localization of these features are not highly accurate. The second set of features are not very dense, however, they provide very accurate localization of the feature.

The integration of the various features is accomplished by a multi-pass matching algorithm, where the quality of localization/ambiguity is effects the order in which points are considered, and previously matched points effect the disambiguation of other points. The basic assumption underlying the matching algorithm is that the disparity surface should be smooth. The smooth disparity fields used for this system are based on generalized two dimensional smoothing splines. The smoothness criterion is similar to one used in smooth surface reconstruction.

The system starts with the feature points which have ''unique matches'' and good localization (i.e., at the current time it begins with centroids of ''texel'' defined on the blackboard). In all neighborhoods without these features, lower quality (in terms of localization) features with ''unique'' matches are added, however they are given a lower confidence value. Thus when the smooth surface is fitted to the disparity data, the disparity values generated by lower quality features will not be as closely approximated.

After all the ''unique'' matches have been used, the module reconstructs a disparity surface. This reconstruction is based on the assumption that the disparity surface should give rise to a smooth surface in depth. Using this disparity surface, the module disambiguates other matches by choosing the potential match which comes closest to the smooth surface. The distance between the disparity predicted by the ''best'' match and the smoothed disparity surface affects the confidence of the match, which in turns affects the way the disparity surface approximates that match. The disambiguation takes place in multiple passes each of which incorporates to features that are increasingly ambiguous.

After all points are considered, the intra-process module adds its output to the blackboard. Currently, this output is depth values at various points, especially along the ''edges'' of surfaces in the disparity field, and depends on the calibration of the imaging system.

5.0.1 Inter-process integration and surface reconstruction

This section describes the inter-process integration phase of the system. This phase of the fusion process is predicated on the assumption that the world is comprised of piecewise smooth surfaces/objects. Therefore, the inter-process integration should depend on the assumed smoothness model(s) for surfaces, not on the data acquisition techniques. There are two main aspects of the inter-process integration, basic surface building, and the weighting of various modules.

The approach taken herein is based on generalized smoothing spline functions [3], and is more efficient for sparse data. The system allows for each data point to be individually weighted in the contribution to the allowed fitting error choice of these weights is influenced by two things, the confidence passed for the point from the intra-process integration processes, and the weighting assigned to the module as a whole.

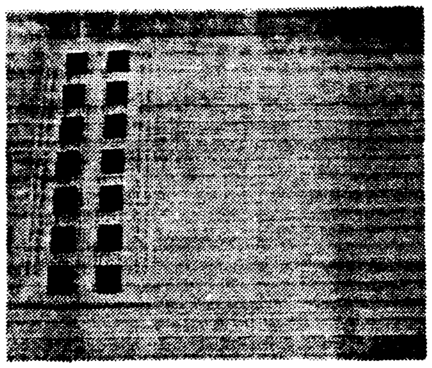
The above fusion scheme requires that each data point be given a weight. The correct selection of these weights is difficult. The study of these weighting will be of paramount importance in our future research. Currently, the system builds three surfaces

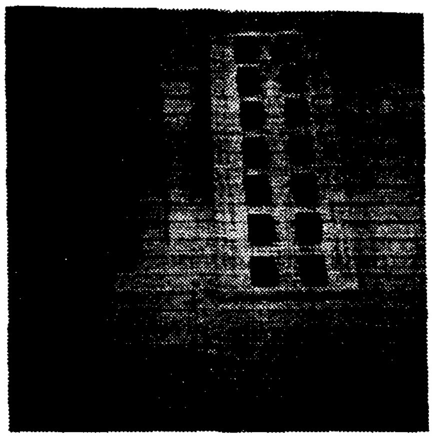
1. one surface from the output of the intra-texture-integration module using the weighting supplied by that module,
2. one surface from the output of the intra-stereo-integration module using the weighting supplied by that module, and
3. one surface combining all data. For the combination, the weights are divided by the number of data-points output by a module. This provides some means for the texture data to have an effect on the surface. Otherwise the stereo data (with 500-5000 points) would totally dominate the shape information from texture (which only provides ~10-50 data points).

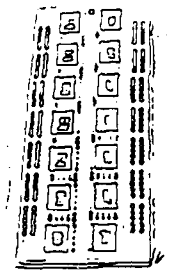
While it would be nice for the system to choose which of these surface is the ''correct'' one, this is not possible. When the information is conflicting, the ''correct'' precept is subjective, and can often be changed by will in humans. However, when the surfaces agree, the system should be able to (but currently cannot) take note, and remove the redundant representations.

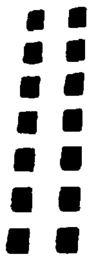
The system described above is still under development and has only been subject to limited experimental testing. One example of the system working on camera images is presented in Figure 5-1. In this example, the curved roll of paper demonstrates a surface where stereo dominates, but is significantly aided by the texture information.

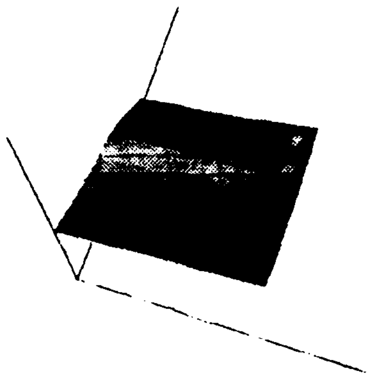
Figure 5-1: An example of fusing stereo and texture to reconstruct surfaces.

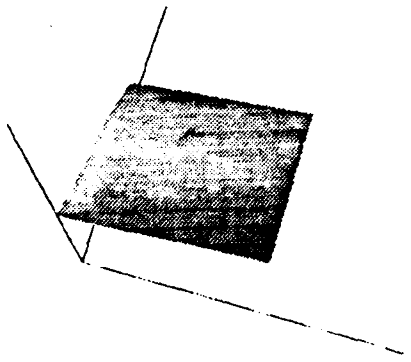

Left input image for example 2 a "circuit breadboard".


Right input image for example 1.


Laplacian of Gaussian of left image for stereo.


Texels identified in left image.


Reconstruction from only texture data.


Reconstruction from combined texture and stereo outputs.

6. Conclusions

Fast, integrated, multi-modality computer vision algorithms are the necessary basis for the development of automatic real-time navigation, photo interpretation, and other human-intensive tasks. The introduction of highly parallel computers such as the Connection Machine makes this goal feasible. However, the selection, tuning, integration, and mapping of parallel algorithms onto these architectures is a complex prerequisite to the full use of the power of the new machines.

In this report we presented the progress of research during the period of the first year. These include:

- Developed and experimentally verified a new shape-from-texture method that uses autocorrelation to compute local surface orientation for rotationally invariant textured surfaces such as natural road surfaces.
- Developed and validated a resource allocation scheme that efficiently maps pyramid-based vision algorithms onto the Connection Machine, using a combination of the hypercube and mesh connections.
- Tested micro-edge density based algorithms on road scenes and other natural scenes, establishing their effectiveness and tuning their free parameters.
- Developed a programming environment to test the multi-resolution parallel algorithms on a pyramid-like architecture. The environment consists of primitive functions that simulate all pyramid operations used in low-level and middle-level vision.
- Implemented versions of Laplacian of Gaussian convolution and zero-crossing operators that include optional mask decomposition, alternative arithmetic modes, and the cascading zero-crossings over several resolutions.
- Implemented and tested parallel versions of Witkin's shape- from-texture algorithm, and decomposed it in order to integrate it with other shape-from methods.

7. Future Work

Our Objectives for the second year of the project that starts in October 1987 are:

- Demonstrate on the Connection Machine edge extraction and linear interpolatory algorithms, using multiple resolution techniques.
- Demonstrate on the Connection Machine surface analyses of depth and orientation, using fast multi-resolution edge, stereo, and surface interpolation algorithms.
- Implement and exercise on the Connection Machine parallel texture segmentation algorithms and shape-from-texture algorithms based on texel density, and validate the newly developed auto-correlation approach.
- Demonstrate on a VAX, possibly also on the Connection Machine, the control interaction of stereo and texture mediated by surface orientation and depth calculations.

References

1. Aloimonos, J., and Swain, M. J. Shape from Texture. Proceedings of the Tenth International Joint Conference on Artificial Intelligence, IJCAI, 1985.

2. Ballard, D., and Brown, C.. *Computer Vision.* Prentice-Hall Inc., 1982.

3. Boult, T.E. *Information Based Complexity in Non-Linear Equations and Computer Vision.* Ph.D. Th., Department of Computer Science, Columbia University, 1986.

4. Fekete, G., and Davis, L. S. Property Spheres: A New Representation For 3-D Object Recognition. Proceedings of the Workshop on Computer Vision Representation and Control, 1984, pp. 192 - 201.

5. Glazer, F., Reylnolds, G. and Anandan, P. Scene Matching by Hierarchical Correlation. Proceedings of Image Understanding Workshop, June, 1983.

6. Grimson, W.E.L.. *From Iimages To Surfaces: A Computational Study of the Human Early Visual System.* The MIT Press, 1981.

7. Ibrahim, H. A. H., Kender, J. R., and Shaw, D. E. SIMD Tree Algorithms for Image Correlation. IN the Proceedings of AAAI at Philadelphia, August, 1986.

8. Ikeuchi, K. Shape from Regular Patterns (an Example from Constraint Propagation in Vision). Proceedings of the Internation Conference on Pattern Recognition, December 1980, pp. 1032-1039.

9. Korn, M. R. and Dyer, C. R. 3-D Multiview Object Representation for Model-Based Object Recognition. RC 11760, IBM T.J. Watson Research Center, March 1986.

10. Marr, D.. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* W.H. Freeman, 1982.

11. Moerdler, M. L., and Kender, J. R. Surface Orientation and Segmentation from Perspective Views of Parallel-Line Textures. Columbia University, 1985.

12. Ohta, y., Maenobu, K., and Sakai, T. Obtaining Suface Orientation from Texels under Perspective Projection. Proceedings of the Seventh International Joint Conference on Artificial Intelligence, IJCAI, 1981.

13. Shafer, S. A. and Kanade, T. and Kender, J. R. "Gradient Space under Orthography and Perspective". *Computer Vision, Graphics and Image Processing 24* (1983), 182-199.

14. Wong, R. Y., and Hall, E. L. "Sequential Hierarchical Scene Matching". *IEEE Transactions on Computers 27*, 4 (April 1978).