

Automated Stereophotogrammetry

by

Gregory L. Brookshire

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the
degree of
Master of Science
in
Electrical Engineering

APPROVED:

Dr. Morton Nadler,
Chairman

Dr. F. G. Gray

Dr. James B. Campbell

June 11, 1987
Blacksburg, Virginia

Automated Stereophotogrammetry

by

Gregory L. Brookshire

Dr. Morton Nadler, Chairman

Electrical Engineering

(ABSTRACT)

The principle function of any automated stereophotogrammetry algorithm is to identify corresponding points in the stereo pair. In the present approach the widely known technique of successive refinement of parallax based on hierarchical coarse-to-fine resolution steps is used. To eliminate the four-eight connectivity problem of orthogonal arrays, a pseudo-hexagonal array is used at all but the final resolution. Candidate matching points are determined from the oriented edge-vector graph. Candidates will be nodes on the graph, which represent observable features at the gray scale level. Matching is based on the correlation of the edge vectors rather than the gray scale, except at the single pixel resolution where parallax accuracy is refined by a quick gray scale correlation of small windows. A discussion of speed and future studies is given.

ACKNOWLEDGEMENTS

I would like to thank Dr. Morton Nadler for giving me the opportunity to complete this project. His technical advice and wisdom provided many insights as the work progressed, and without his funding my graduate experience may not have been possible. Brian Telfer and Wanglu Peng should also be commended for their preliminary efforts on this project. I would like to thank my other committee members, Dr. Campbell and Dr. Gray, for their time and consideration. I would also like to thank Tracey Dobbins, my future wife, for her emotional support and encouragement during the last year and a half.

The work presented in this thesis was funded by the U. S. Army Research Office Geosciences Division under grant number DAAG29-85-K-0250.

TABLE OF CONTENTS

1.0	Introduction	1
1.1	Background	1
1.2	Stereophotogrammetry	3
1.2.1	Stereo Aerial Photos	4
1.2.2	ETL's Method	9
1.3	Brief Discussion of our Algorithm	9
1.3.1	Hierarchical Resolutions	10
1.3.2	Operations Performed at Each Resolution	12
2.0	Hexagonal Arrays	15
2.1	Pro's and Con's of Hex Arrays	15
2.2	The Pseudo Hexagonal Array	19
3.0	Edge Vectors	21
3.1	Calculation of Hex Edge Vectors	21
3.2	Edge Filtering	24
3.3	Discussion of Coherence of Edge Vectors	27
4.0	Nodes	29
4.1	Node Definition	31
4.2	Node operator	33
5.0	Matching	36

5.1	The Matching Function	36
5.2	Correlation Confidence Intervals	39
6.0	Interpolation	44
6.1	The Algorithm that I Developed	45
6.2	An Example	47
6.3	A Proposed Improvement	49
7.0	Expanding the Disparity Array	51
8.0	Gray Level Matching	52
8.1	The Ping-Pong Correlation Scheme	53
9.0	Results	58
9.1	Sample Run of the Algorithm	58
9.2	Execution Speed	77
Appendix A.	Speed Analysis	83
Appendix B.	Ignoring Y-Disparity	84
Appendix C.	Matching Trees	86
Appendix D.	Gipsy Programs	88
Appendix E.	Plotting	91

E.1	Downloading to the HP	92
E.2	Getting on the HP	93
E.3	HP Graphics	95
E.4	VECPlot - The Vector Plotter	96
E.5	CONPlot - the contour plotter	98
E.6	Using the HP Plotter	100
Bibliography		102
Vita		104

LIST OF ILLUSTRATIONS

Figure 1. Left Image of Stereo Pair	5
Figure 2. Right Image of Stereo Pair	6
Figure 3. Parallax in Stereo Aerial Photographs	7
Figure 4. The Basic Algorithm	11
Figure 5. The General Algorithm	14
Figure 6. Orthogonal and Hexagonal Grids	16
Figure 7. Ortho and Hex Connectivity	18
Figure 8. The Pseudo-Hexagonal Array	20
Figure 9. Edge-Vector Calculation	22
Figure 10. Vector Association Filter	25
Figure 11. Coherence of Edges	28
Figure 12. The Concept of a Node	30
Figure 13. Examples of Nodes	32
Figure 14. My Node Finding Algorithm	34
Figure 15. Match Windowing Scheme	38
Figure 16. The Interpolation Windowing Scheme	46
Figure 17. An Example Interpolation	48
Figure 18. The Ping-Pong Correlation Method	55
Figure 19. Network of Gray Scale Matching Points	57
Figure 20. The Test Pair at Pseudo-Hex 18	59
Figure 21. The Full Edge Field	60
Figure 22. Full Field with Edge Magnitudes	61
Figure 23. Associated Edge Field	62
Figure 24. Nodes on the Associated Edges	64

Figure 25. Matches on Image 1	65
Figure 26. Matches on Image 2	66
Figure 27. Match Disparity from Res 18	67
Figure 28. Interpolated Match Disparity	68
Figure 29. Contour Plot after Res 18	69
Figure 30. Expansion of Disparity Array	71
Figure 31. Interpolation of Expanded Array	72
Figure 32. Pseudo-Hex Res 6 Gray Scale	73
Figure 33. Pseudo-Hex Res 6 Nodes	74
Figure 34. Pseudo-Hex Res 6 Matches	75
Figure 35. Contour Plot after Res 6	76
Figure 36. The Final Contour Plot	78
Figure 37. The Calculated Contour from ETL	79
Figure 38. A Surface Plot Comparison	80
Figure 39. Effect of Matching Trees	87

LIST OF TABLES

Table 1. Matching Function Fisher-Z	42
Table 2. Individual Execution Times (sec)	81

1.0 INTRODUCTION

Everyone knows the child's game of closing one eye and then trying to touch the tips of your two index fingers. The problem is that with one eye closed you lose your ability to see in three dimensions. Or, in other words, with only one eye to see from the three dimensional world is reduced to a two dimensional image. It takes two eyes, or cameras, separated by some distance to see depth information. In the animal world, seeing with two eyes is called stereo vision. In the aerial photo world it is called stereophotogrammetry. Thus, to retrieve elevation information from aerial photos, one must process a stereo pair. The algorithm that is being presented represents an automated technique to find corresponding points in a stereo pair and thus arrive at an elevation contour.

1.1 BACKGROUND

The candidate matching points, or nodes, in the present algorithm are identified by intersecting edges on the oriented-edge graph extracted from a reduced, pseudo-hexagonal, gray-scale resolution. Several others have

developed image matching schemes using similar concepts. Moravec [9] locates **feature** points (essentially corners) in **one** image and uses multiple resolutions to find matching points in the two images. Wang [12] uses a relaxation procedure to match **corners** detected in a stereo pair. Barnard and Thompson [11] also use a relaxation-based algorithm which matches **feature** points, but feature points are found in **both** images and **only** feature points are matched. The feature points mentioned above are identified on the gray-scale image by points having locally high contrast or sharpness. All use some type of gray-scale correlation for their matching function. Baker and Binford [10] generate a disparity map by matching corresponding edges. They also reduce their search area by confining it to the epipolar lines. Epipolar lines are formed by the plane containing the two camera foci and an object point. Medioni and Nevatia [13] match strings of connected edges called **segments**. They stress that such feature-based systems are faster, more accurate, and less sensitive to photometric variations than intensity-based systems. The present algorithm adds several new concepts to those mentioned above. The present algorithm is based on a pseudo-hexagonal raster obtained from the original gray-scale. It identifies **nodes**, or feature points, on the oriented-edge graph

rather than the gray-scale image. And, at all but the final resolution, matches are based on correlated edge directions of the full edge-vector field. Preliminary work on the present algorithm was done by Nadler and Signor on an orthogonal array. In this study, their work was translated to the hexagonal array and several refinements were made to the algorithm.

1.2 STEREOPHOTOGRAMMETRY

Aerial photographs are typically taken from an airplane several kilometers, or several tens of kilometers above the ground. A camera mounted inside the plane takes pictures through an opening in the bottom of the plane. Automatic camera systems exist which can take a sequence of photographs which scan a line parallel to the path of the plane. There may be a non-zero angle between the lens and the perpendicular line from the plane to the ground. This angle adds geometric distortion to the photo, called trapezoidal distortion. Ideally, one would want to process the photo that has its camera axis completely vertical or perpendicular to the ground. Distortion is also caused by the relief of the area. Points with higher elevation will be closer to the lens and thus appear at a greater scale in the image. These and other dis-

tortions must be considered when processing aerial photos. The present algorithm concentrates on finding matching points in the stereo pair. The distortion corrections can then be more efficiently applied to the field of matching points rather than to the original gray scale array.

1.2.1 Stereo Aerial Photos

A stereo pair of photos consists of two photos taken of the same general area, but from different vantage points. Figure 1 on page 5 and Figure 2 on page 6 show the stereo pair that was used as a test image for the investigation of the present algorithm. Figure 3 on page 7 shows the ideal situation when stereo photos are taken from two different positions of the plane with the camera axis perpendicular to the ground, and with the image origins along the same line parallel to the flight path. In Figure 3 on page 7 the flight path is parallel to NN' . The region of overlap in the image pair is where the depth information is contained. From Figure 3 on page 7, the physical line segment RS produces the two image segments rs and $r's'$ in images O and O' , respectively. What is important to notice is that for a single image, R could be anywhere on the line OR ($O'R'$)

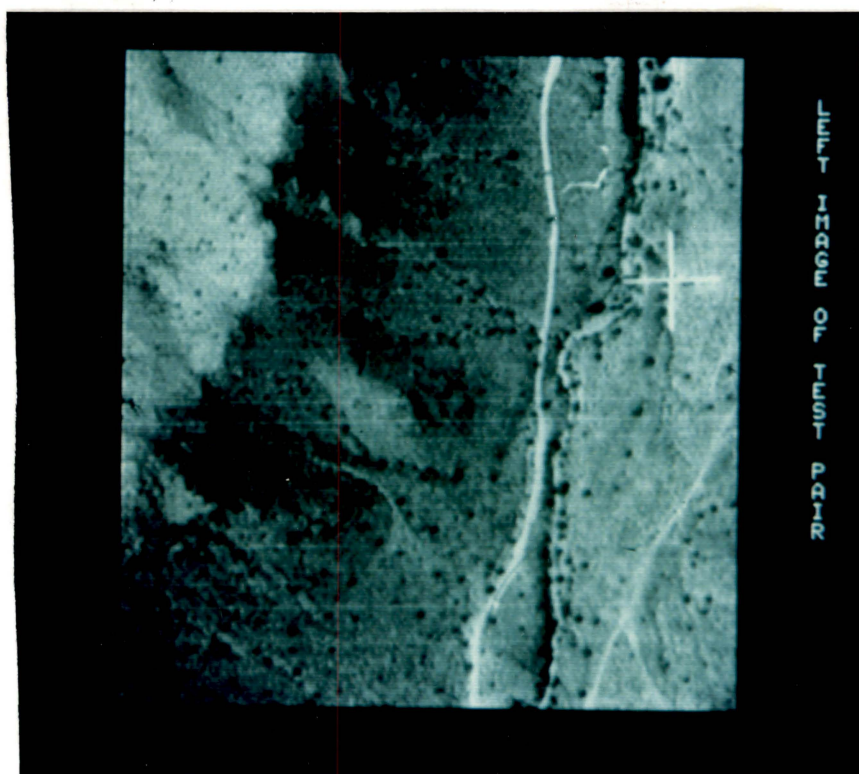


Figure 1. Left Image of Stereo Pair

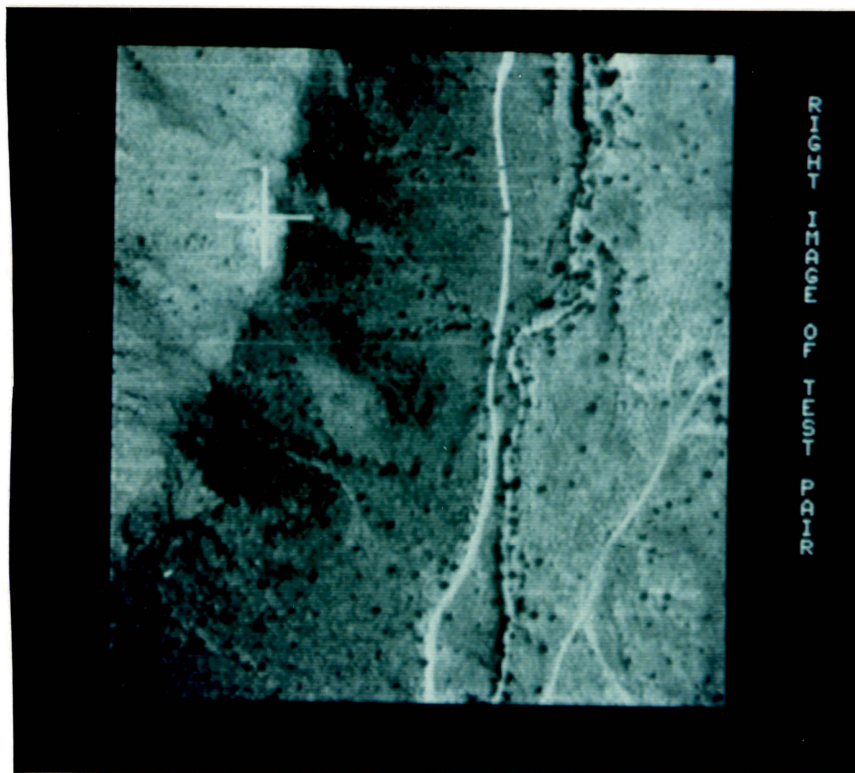


Figure 2. Right Image of Stereo Pair

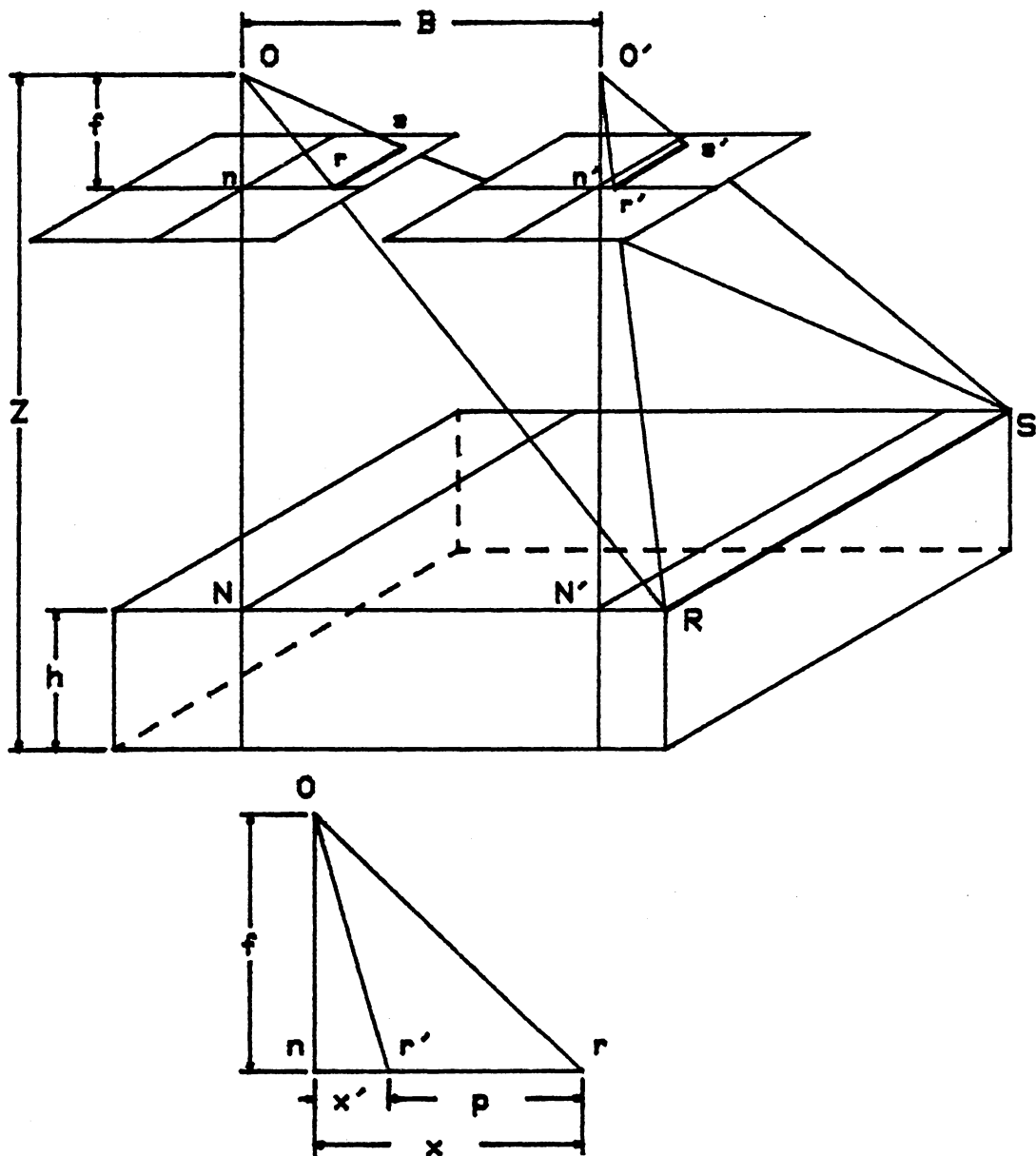


Figure 3. Parallax in Stereo Aerial Photographs: the parallax of the line RS is defined to be p

and still produce the same image r (r'). It takes the image pair to pinpoint the location in 3-space. In fact, the parallax p of R can be directly related to its elevation h . From similar triangles it can be shown that

$$h = Z - Bf/p$$

where B is the distance from O to O' , and it is assumed that both cameras have the same focal length f . Note that the parallax of each point is parallel to all the others. The parallax would be in the OO' direction for this figure.

Aerial photos are generally digitized from a print rather than being digitized directly from the camera. The pixel resolution of a digital image scanner is an important parameter to know when calculating parallax. At the resolution of today's scanners, the gray scale array from a large aerial photo would be very large. Generally subimages of the photographs are digitized. When digitizing a stereo pair, one typically aligns the second image to eliminate y-disparity (pixel offset in the y-direction) and to maximize the overlap in the image pair. This helps reduce the search window size when trying to match points in the images. These artificial offsets of

the images must be corrected when calculating true image parallax.

1.2.2 ETL's Method

In brief, the current technique for stereo matching at the Engineering Topographic Laboratory (ETL) involves the following. First, parallax is obtained manually, using some type of stereoscopic device, for a set of points along the left and top edge of the pair. Next, using a normalized, cross correlation on the original image gray-scale with 17 by 17 windows, matching points are found for every fifth pixel. The interactive parallax and the parallax of previously matched points are used to obtain an initial guess for the offset of the search window. Written in CDC Fortran IV Extended on a Cyber-730, this algorithm provides a data rate of 15 points/second. However; this does not include the time for the interactive matching.

1.3 BRIEF DISCUSSION OF OUR ALGORITHM

The matching algorithm that we use is almost completely automated. It does require some preliminary rough matching to choose the initial gray scale resolution.

It does not require any interactive matching to obtain initial offsets. The algorithm investigated here is based on matching corresponding topological features in the two images. Figure 4 on page 11 describes the basic algorithm and gives the data that are passed between processes. First, an oriented edge graph is obtained on a pseudo-hexagonal array at a reduced resolution. Then, from the edge graph, prospective matching points called nodes are identified in both images. Finally, the edge vector field surrounding a node in the first image is correlated with similar fields surrounding nodes in the second image to find a match. These steps are repeated at successively finer and finer resolutions.

1.3.1 Hierarchical Resolutions

One of the fundamental aspects of the algorithm is that of calculating parallax at hierarchical resolution steps. The reduced resolution is obtained by reducing a pixel window to its gray scale-average. The windowing scheme that we use is called pseudo-hexagonal and will be explained in detail later. The window sizes decrease geometrically from 54 to 18 to 6 to 2. The largest window is chosen to reduce the maximum parallax to ± 1 pixel. After resolution 2, a nor-

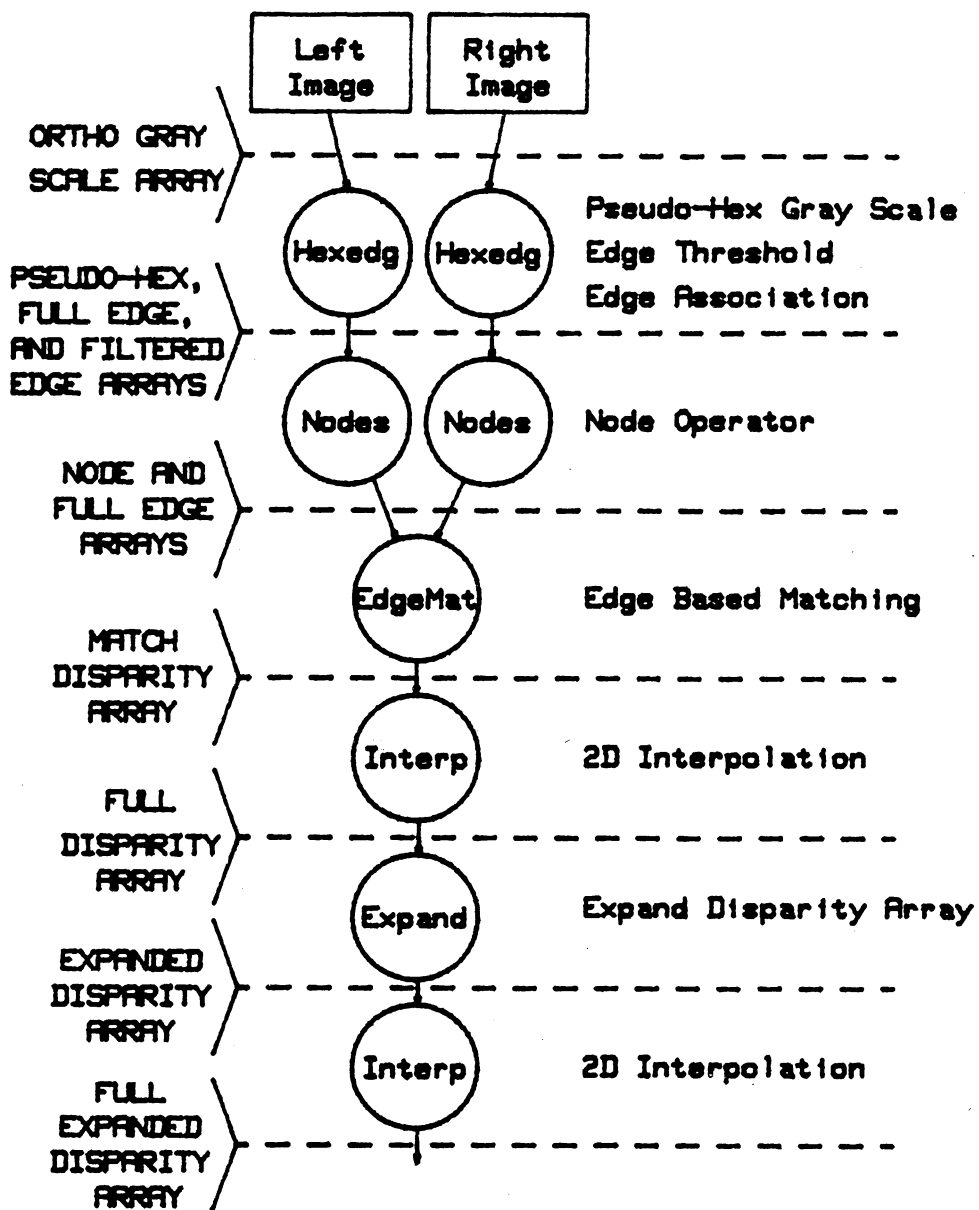


Figure 4. The Basic Algorithm: these steps are applied at each pseudo-hex resolution level

malized cross-correlation is done on the original images, using a very small window. The idea is to use the parallax at one resolution to narrow the search window at the next resolution. This can be done because each step further reduces the uncertainty of the true parallax of a particular point. The smaller search windows lead to shorter execution time.

1.3.2 Operations Performed at Each Resolution

As was mentioned above, at each resolution step the same sequence of operations is performed. The only parameter that changes is the size of the pseudo-hex array that it operates on. Figure 4 on page 11 gives the details of the basic algorithm that is applied at each step. The process begins with a stereo pair of images. Both images are processed by the HEXEDG operator which reduces the resolution using a supplied pseudo-hex window size, applies an edge magnitude threshold, and finally associates the edges to find true edge features. The next step takes the edge vector array and simply applies the NODES operator which marks points which will be used for matching attempts by the next process. EDGEMAT does the edge vector correlation of the nodes and thus finds the disparity (or parallax) of the matched nodes. Next,

to prepare for the next resolution, a two dimensional INTERPolation is done on the disparities. The array is then EXPANDED by the step size and INTERPolated again.

Figure 5 on page 14 shows a diagram of the entire algorithm as it is applied to an image pair starting at resolution 18. Note that after the Res 2 step (two-by-two pseudo-pixels), a gray scale correlation is performed on the original images. Each part of the algorithm will be discussed in great detail in the following chapters.

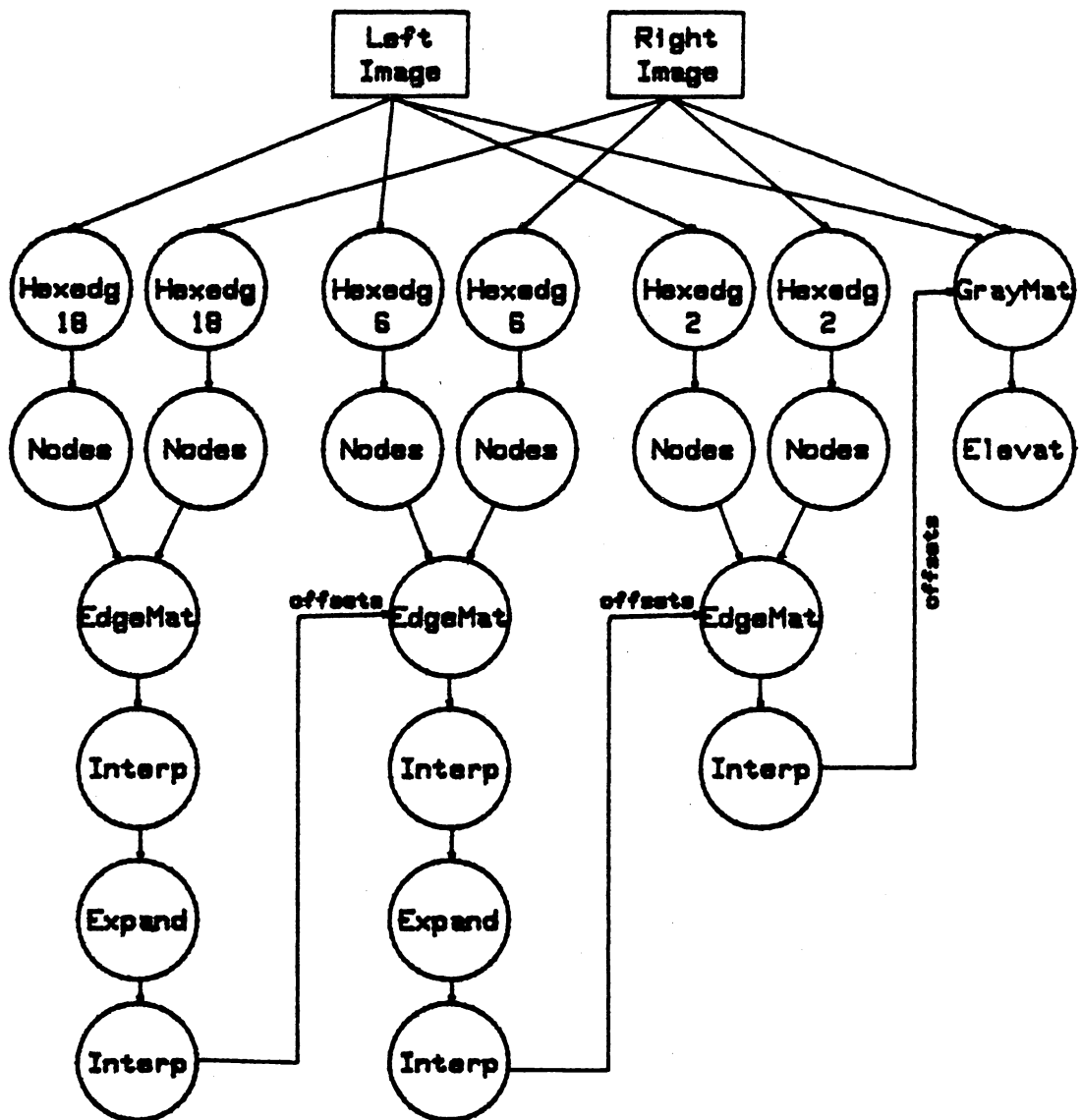


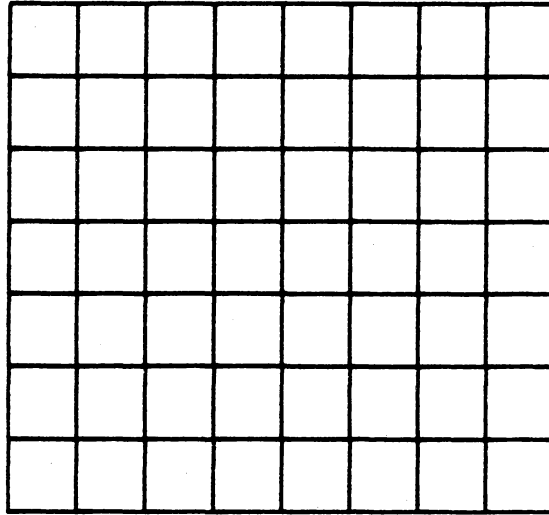
Figure 5. The General Algorithm: showing three pseudo-hex resolutions beginning at 18

2.0 HEXAGONAL ARRAYS

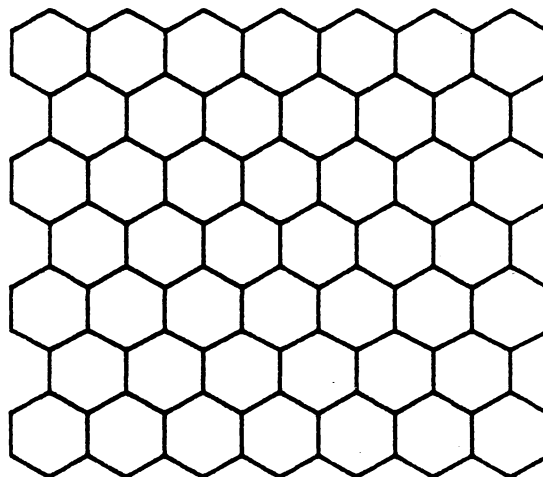
One of the fundamental concepts of the algorithm is that of a hexagonal array. As the name implies, a hexagonal array has as its fundamental element a hexagon. This is in contrast to an orthogonal array whose fundamental element is a square. The orthogonal array is the common array used in most of today's video and image processing systems. Figure 6 on page 16 shows a sample of both types of arrays. Even at first glimpse, the hexagonal grid seems to appear more natural in some way. The hexagon appears repeatedly in nature, such as in snowflakes and honeycombs.

2.1 PRO'S AND CON'S OF HEX ARRAYS

The four-eight connectivity problem of the orthogonal array is widely known in image processing. This is the precise problem which the hex-array eliminates, and makes it appear more regular. A pixel in the orthogonal array has four neighbors which share an edge and four more neighbors which share a corner. When designing an algorithm for an orthogonal array, one must decide whether a pixel has four neighbors



ORTHOGONAL ARRAY

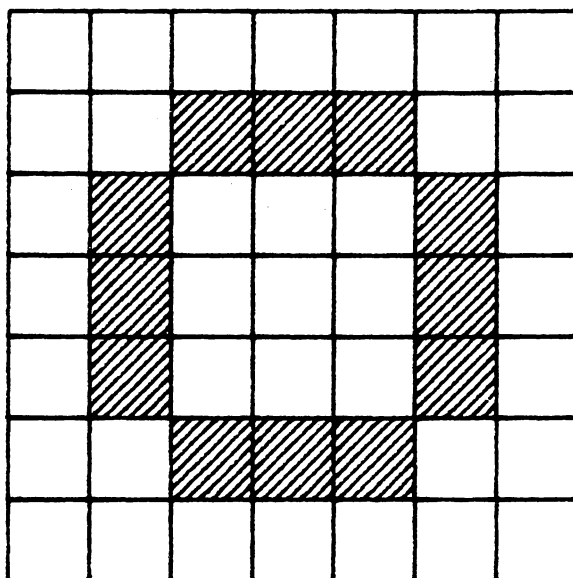


HEXAGONAL ARRAY

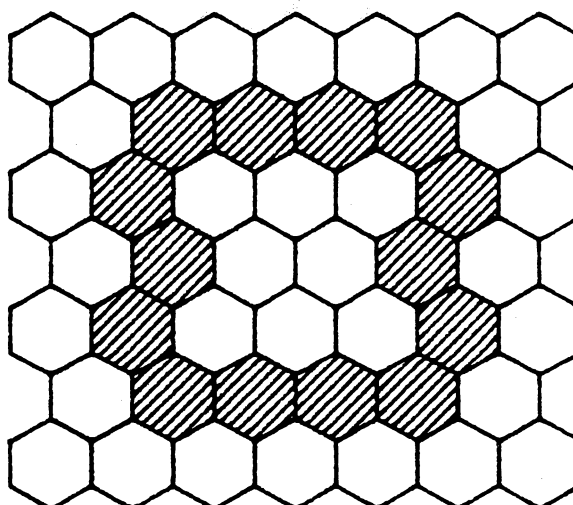
Figure 6. Orthogonal and Hexagonal Grids

through edges, or eight neighbors through edges and corners. In contrast, a pixel in a hexagonal array has exactly six neighbors, with all of which it shares an edge. Figure 7 on page 18 illustrates this problem of connectivity. In the orthogonal grid, if the image is considered to be a square, then that assumes eight connectivity. But, eight connectivity also says that the inside of the square is connected to the outside, thus, the figure represents four line segments arranged in the shape of a square. This is the orthogonal four-eight connectivity paradox. As illustrated in the hexagonal array, the connectivity is completely deterministic. In the present algorithm this feature allows complete certainty and accuracy in determining the edge contours.

One problem encountered when writing image processing algorithms on hexagonal arrays is where to get the hexagonal data from. Today's image scanners output purely orthogonal arrays. Dr. Nadler has had a patent issued on a CCD scanner which generates a pseudo-hexagonal image. Another method to obtain hexagonal data is to use a pseudo-hexagonal array. A pseudo-hex array is made up of square pixels, but every other row is offset by a half a pixel width.



Orthogonal Connectivity Problem

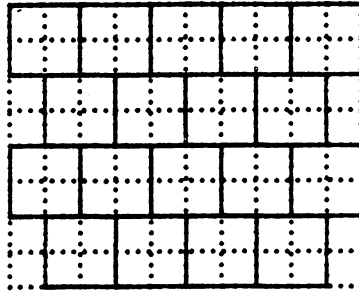


Deterministic Hexagonal Connectivity

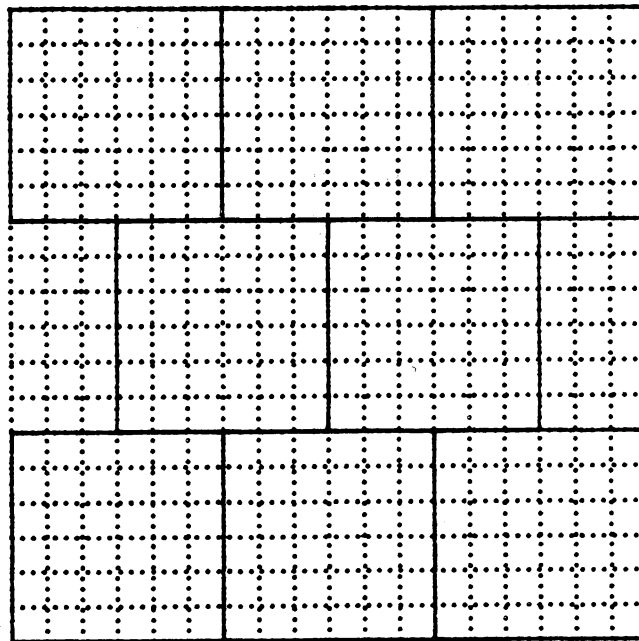
Figure 7. Ortho and Hex Connectivity

2.2 THE PSEUDO HEXAGONAL ARRAY

The present algorithm operates on pseudo-hex arrays derived from an orthogonal image. Figure 8 on page 20 illustrates the pseudo-hex principle at two resolution levels. The dotted orthogonal grid represents the original gray-scale image. The larger, staggered, solid grid is the pseudo-hex array. The figure shows a 'Res 2' array derived from 2 by 2 windows, and a 'Res 6' array derived from 6 by 6 windows. The different window sizes are used to produce the hierarchy of steps in the algorithm. The value of a pseudo-hex pixel is given by the gray-scale average over the window in the orthogonal grid. The staggering of the rows gives each pseudo-pixel six neighbors through an edge. Thus, the pseudo-hexagonal array approximates a purely hexagonal array. One should note a small difference in dimensions. The angle between pixels on different rows in the pseudo-hex grid is 63.4 degrees. This angle is obtained from the arc-tangent, where $dy=1.0$ and $dx=0.5$. In a purely hexagonal array the angle between similar pixels is, of course, 60 degrees.



Sample Res 2 Pseudo Hex



Sample Res 6 Pseudo Hex

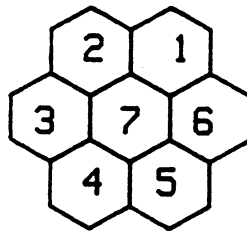
Figure 8. The Pseudo-Hexagonal Array

3.0 EDGE VECTORS

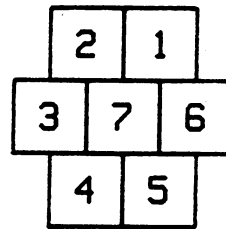
The second step in the algorithm is the application of the edge-vector operator to the two pseudo-hex gray scale images. Bowker [3] gives a detailed description of the edge-vector and its uses. The edge-vector is simply the gradient vector rotated by ninety degrees counter-clockwise. Figure 9 on page 22 shows the details of the edge operator devised for use on the hexagonal grid. Note how the connectivity of the hex window gives a unique interpretation of the hex gradient. On an orthogonal grid one could calculate gradients using either the four or the eight connectivity. Under many circumstances the two would give gradient vectors of different directions. Since the node operator is based entirely on edge directions, the choice of connectivity would directly affect the overall results of an orthogonal algorithm. The use of a hex array eliminates this problem.

3.1 CALCULATION OF HEX EDGE VECTORS

Figure 9 on page 22 shows a 7-pixel hexagonal window and the corresponding pseudo-hex window. The window numbering scheme corresponds to that used throughout



Hexagonal



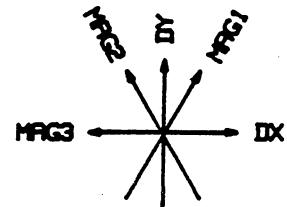
Pseudo-Hex

Hexagonal Gradient Components...

$$\text{MAG1} = \text{HEX1} - \text{HEX4}$$

$$\text{MAG2} = \text{HEX2} - \text{HEX5}$$

$$\text{MAG3} = \text{HEX3} - \text{HEX6}$$



Orthogonal Components...

$$\text{DX} = \text{MAG1} * \cos 60 - \text{MAG2} * \cos 60 - \text{MAG3}$$

$$\text{DY} = \text{MAG1} * \sin 60 + \text{MAG2} * \sin 60$$

Edge-Vector Angle and Magnitude...

$$\text{THETA} = \text{ARCTAN}(\text{DY} / \text{DX}) + 90$$

$$\text{MAG} = \text{SQR}(\text{DX}^2 + \text{DY}^2)$$

Angle Quantization...

$$-15 \leq \text{THETA} < 15 \quad \text{DIR} = 1$$

$$15 \leq \text{THETA} < 45 \quad \text{DIR} = 2$$

...

$$315 \leq \text{THETA} < -15 \quad \text{DIR} = 12$$

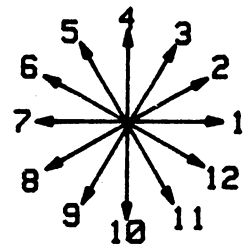


Figure 9. Edge-Vector Calculation

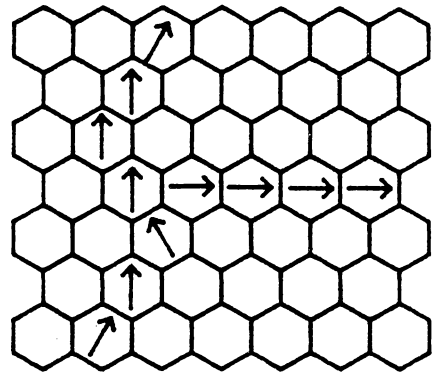
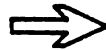
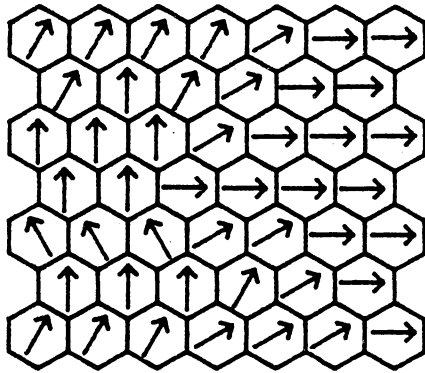
the GIPSY software implementation. In the figure, HEX_n refers to the pseudo-hex gray scale of pixel numbered n in the window. First the gradient components along the three fundamental hex-axes are calculated. These are MAG₁, MAG₂, and MAG₃. The three axes are separated by angles of sixty degrees. The next step converts the hex components to components along the orthogonal axes labeled DX and DY. This facilitates computation of the resultant edge vector's angle, THETA, and magnitude, MAG, using orthogonal trigonometry. In the quest for efficiency this step may be skipped in the future if some type of hexagonal trigonometric functions are developed.

The final step in the process is the quantization of the angle into thirty degree levels given by the variable DIR in the figure. This quantization provides great efficiency in the matching process which involves the correlation of edge directions based on the sum of the cosines of angle differences. Quantization allows a lookup table to be used for the cosine function. It should be noted that a similar quantization in an orthogonal array provides only eight levels, leading to a reduction in accuracy.

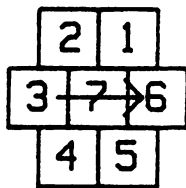
3.2 EDGE FILTERING

After the full edge-vector field is calculated it is stored for later use in the matching process. To find the oriented-edge graph two filters are applied to the full vector field. The first is simply a threshold on the edge magnitudes. All vectors with a magnitude less than the threshold are removed. There are several reasons for applying this filter. The first is obviously to remove random noise which could give false matching points. The second reason stems from the underlying principle of the algorithm which is to match topological features. Such features should be made up of a pattern of mainly strong (high magnitude) vectors. Topological gray-scale features described by weak vectors would be unreliable to match even manually. The edge thresholds were chosen to give similar match densities at each resolution. At resolutions 18, 6, and 2, the thresholds used were 10, 15, and 20, respectively.

The second filter is called vector association. The idea, as shown in Figure 10 on page 25, is to reduce a field of parallel vectors to a single string of vectors. Any given topological edge will likely produce a parallel 'stream' of vectors several pixels



Case 1



$$MSUM0 = M7 + 1/2 (M3 + M6)$$

$$MSUM1 = M1 + M2$$

$$MSUM2 = M4 + M5$$

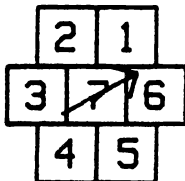
RETAINED IF

$$MSUM0 \geq MSUM1 * \text{MAX} (\text{COS} (D7 - D1), \text{COS} (D7 - D2))$$

AND

$$MSUM0 \geq MSUM2 * \text{MAX} (\text{COS} (D7 - D4), \text{COS} (D7 - D5))$$

Case 2



$$MSUM0 = M7 + 1/4 (M1 + M3 + M4 + M6)$$

$$MSUM1 = M2 + 1/2 (M1 + M3)$$

$$MSUM2 = M5 + 1/2 (M4 + M6)$$

RETAINED IF

$$MSUM0 \geq MSUM1 * \text{COS} (D7 - D2)$$

AND

$$MSUM0 \geq MSUM2 * \text{COS} (D7 - D5)$$

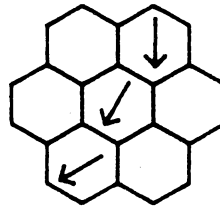
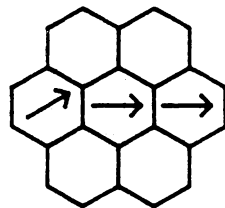
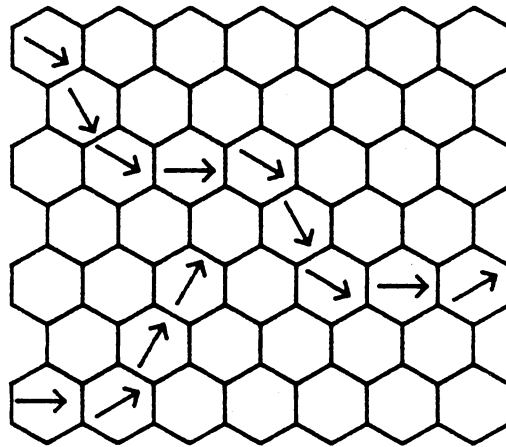
Figure 10. Vector Association Filter

wide. The association operator attempts to find the location of the true edge by reducing the stream to be only a single vector wide. The association operator also reduces the effects of noise. As shown in Figure 10, the association operator uses the same hex window as the edge operator. There are two cases to the association operator, determined by the direction of the vector under consideration $E(7)$, at location seven in the window. $E(7)$ may point towards an edge of its window or towards a vertex. By symmetry, each case contains six rotations. To illustrate the first case in Figure 10 on page 25, let $DIR(7) = 1$. Then, $E(7)$ is retained if the vector string that it belongs to ' $E(3),E(7),E(6)$ ' is stronger than both of the vector strings on either side, namely ' $E(2),E(1)$ ' and ' $E(4),E(5)$ '. The second case, when $DIR(7) = 2$, is a thirty degree rotation of the first. The vector now points towards a vertex of its hex pixel. This makes its vector string a bit more complicated. In fact, the vector strings used are somewhat merged together. The center string becomes ' $E(3)\&E(4),E(7),E(1)\&E(6)$ ' and the side strings are ' $E(3),E(2),E(1)$ ' and ' $E(4),E(5),E(6)$ '. In both cases only the component of the side strings in the direction of $E(7)$ are used for comparison. These components are found by the cosine function in the

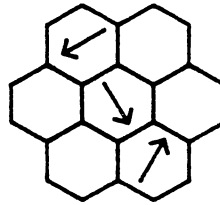
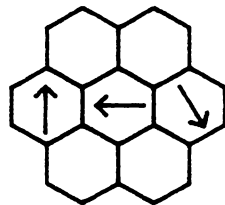
inequalities. $E(7)$ is retained if its string magnitude is greater than both of the side string components. Note that because of the symmetry of the edge directed case (case one), there are two cosines to choose from. The MAX function provides more stringent requirements on MSUM0. Thus, it produces thinner edges than say a MIN or an average would.

3.3 DISCUSSION OF COHERENCE OF EDGE VECTORS

Another edge filter, suggested by a preliminary study of the present algorithm was also investigated as part of this work. Figure 11 on page 28 shows the concept of edge coherence. The coherence filter eliminates edges with direction codes that are not within one direction of both its predecessor and its successor. The idea was to aid in finding only the main edges of the oriented-edge graph, and to eliminate noisy regions that a low threshold retains. This filter was abandoned because it eliminated some of the nodes in the oriented graph. This is in direct contradiction to the purpose of the algorithm, since it is the nodes which identify the topological features that we are trying to match.



Coherent Edges

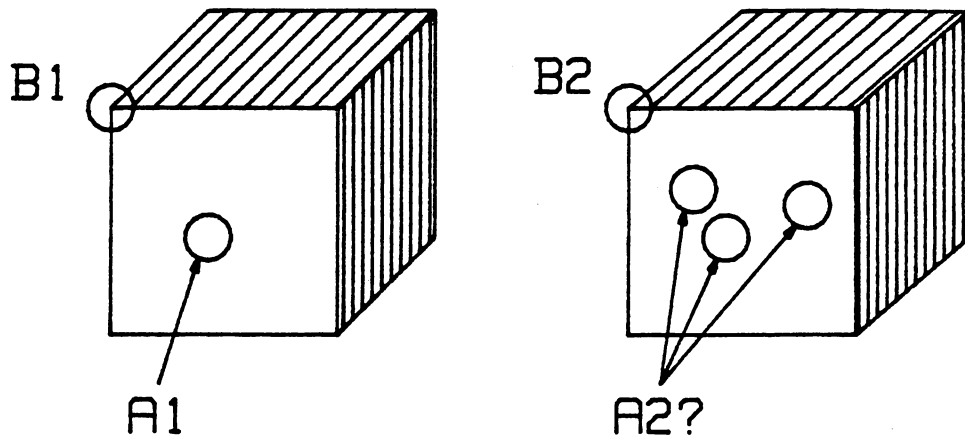


Non-Coherent Edges

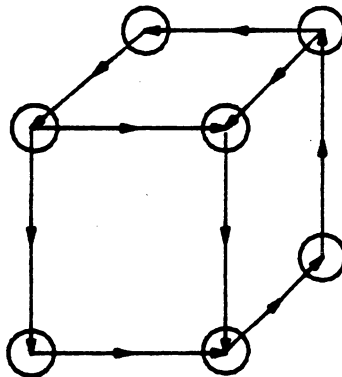
Figure 11. Coherence of Edges

4.0 NODES

If a human being were trying to match points in a stereo pair of photographs, there would be a certain type of point that he would try to match. For instance, consider the stereo pair in Figure 12 on page 30 of a cube with a different shading on the three sides shown. One would not pick a point near the center of a face in the first image and try to match it to a point in the second image. To do so would require much more information than is immediately available in the vicinity of the point. If the match were based on a local gray-scale correlation, many possible matching points would be indicated. The type of point that one would pick to match would be a corner of the cube. Here there is enough information to find an exact match. The correlation function would have a maximum value. Thus, the match would be reliable. Most of the authors mentioned in the Introduction recognized these benefits of finding feature points in the gray-scale image. In the present algorithm this is the exact type of topological feature that the node operator is designed to identify in the oriented-edge graph. Thus, by only trying to match the nodes in one graph to the nodes in the other,



Matching Points in a Stereo Pair



Oriented Edge Graph with Nodes

Figure 12. The Concept of a Node

the algorithm finds the most reliable matches in a minimal amount of time.

4.1 NODE DEFINITION

From Figure 12 on page 30 one can see that a node is identified on the edge graph by intersecting edges or by an edge that makes a sharp angle turn. Figure 13 on page 32 shows a few examples of nodes on hex grids. Note that converging and diverging edges are included in the definition of a node. Every node definition is based on two edges. In most cases there is some ambiguity as to which pixel should be labeled the node. The convention used is to choose the pixel closest to the intersection of the edges. A second look at the node examples should clarify this point. In the figure the center pixels are labelled as nodes. In the converging/diverging cases both vectors would be eventually labeled as nodes. The ambiguity of node labeling will have a direct influence on the windowing scheme in the matching algorithm discussed in the next chapter.

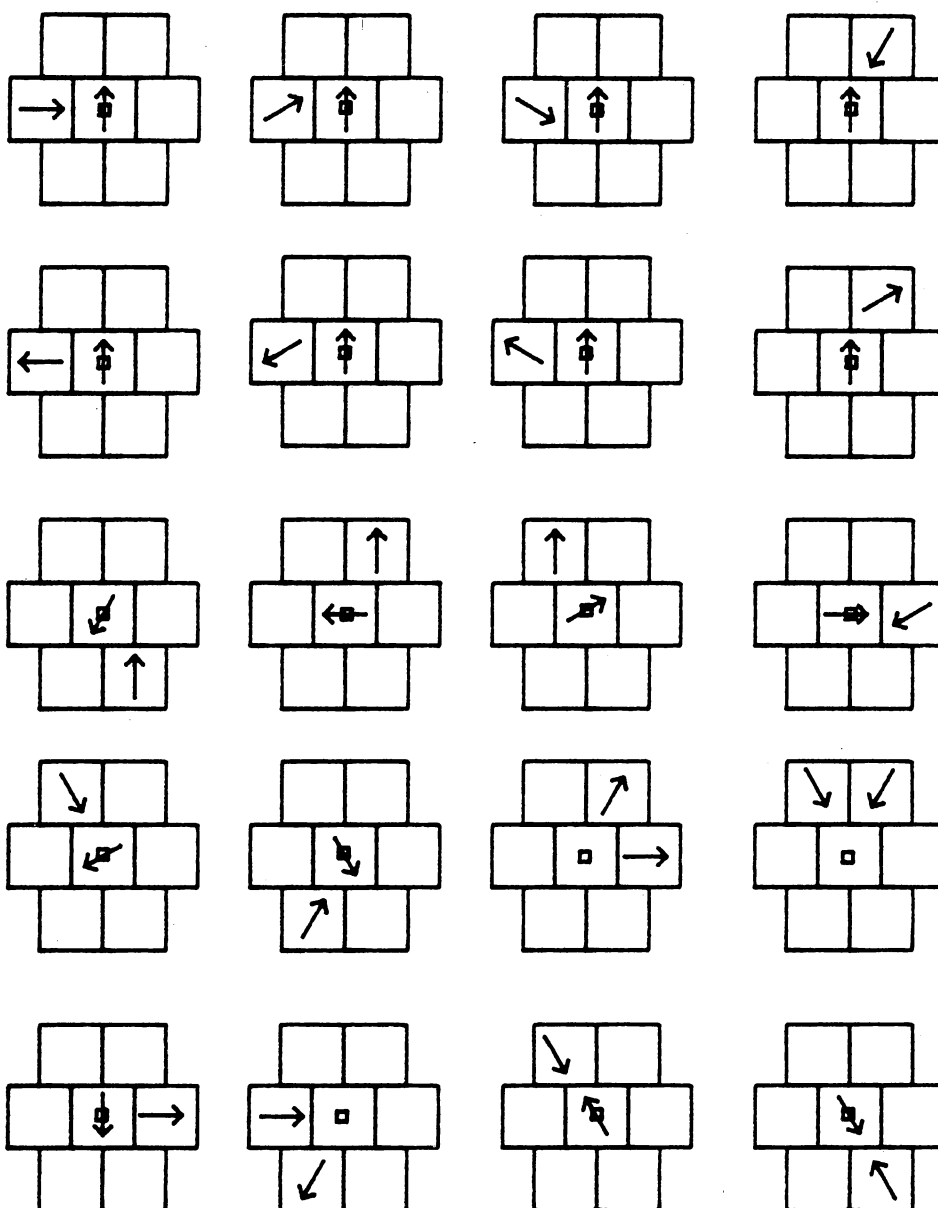


Figure 13. Examples of Nodes

4.2 NODE OPERATOR

If one tried to enumerate the combinations and permutations of edges that could be classified as a node, it would be a very large number. Thus, trying to enumerate all possible conditions in software to identify nodes would be futile. One would like to capture the concept of a node in a simple and concise way that could be easily coded. The node operator that I developed attempts to do just that. Figure 14 on page 34 shows the algorithm that I use. It simply searches for a neighboring edge that makes an angle of at least sixty degrees with, and is either convergent upon or divergent from, the center edge. This definition covers all the cases mentioned so far except the acute angles. These are covered by a short enumeration. The Figure 13 on page 32 shows this type of node near the bottom of the page. It is the only node that is not marked on one of the vectors, since the edge extensions intersect on the center pixel. Note that this definition will also mark both diverging or converging vectors as nodes.

In Figure 14 on page 34 the statement that checks if the edge-direction code difference is greater than six is necessary since edge codes 1 and 12 differ

N = 0

IF D7 <> 0 THEN

REPEAT

N = N + 1

IF D(N) <> 0 THEN

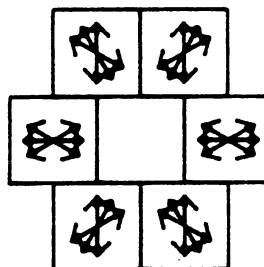
DIFF = D7 - D(N)

IF DIFF > 6 THEN

DIFF = 12 - DIFF

IF DIFF >= 2 THEN

IF D(N) IS ONE OF:



THEN NODE = TRUE

UNTIL NODE = TRUE OR N = 6

Figure 14. My Node Finding Algorithm

by 11, but the angles that they represent differ by only thirty degrees. Also note that the three angles of convergence and divergence for each neighboring pixel location are necessary to maintain symmetry because of the thirty (instead of sixty) degree edge angle quantization.

5.0 MATCHING

After the nodes have been found in both oriented-edge graphs, the next step is to try to find corresponding nodes in both images. A significant time savings is made by only attempting to match nodes, and the beauty of the algorithm is that it is precisely these points which provide the most reliable matches.

5.1 THE MATCHING FUNCTION

The matching correlation function is based on the directions of the full vector field around a node. Using the full field, rather than the thresholded and associated field, takes into account as much information as possible, keeping in mind that the edges themselves filter some noise as they are calculated. The function sums unit-vector components of vectors in image 2, that are in the direction of the corresponding vectors in image 1. Mathematically, it sums the cosines of the differences in direction of corresponding vectors. The correlation function for a single point is as follows:

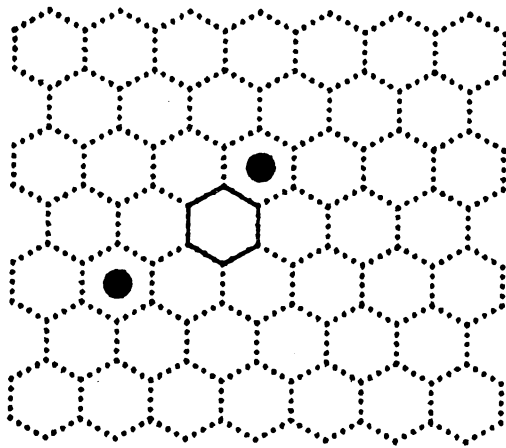
```

Correl = 0
For all i in window
    Correl = Correl + COS(Dir(1,i)-Dir(2,i))
Next i

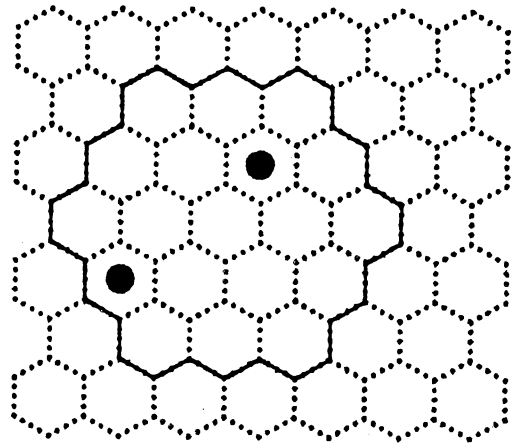
```

Thus, vectors in exactly the same direction contribute $\text{COS}(0)=1$, those differing by 1 direction code contribute $\text{COS}(1*30)=.866$, those differing by 2 direction codes contribute $\text{COS}(2*30)=.5$, etc. Note that vectors in completely opposite directions actually subtract one from the correlation function (e.g. $\text{COS}(180)=-1$). The correlation function has a range from $-(\text{window size})$ to $+(\text{window size})$.

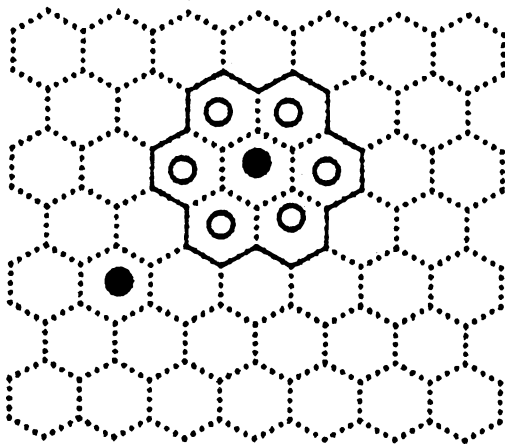
Figure 15 on page 38 shows the windowing scheme used for finding match points. The scheme compares a fixed window in image 1 centered around a node to a moving window in image 2. Each grid in the figure shows a region in image 2 that contains two nodes. The first grid shows the location of the initial offset, for some node in image 1, that was passed down from the previous resolution level (first level it is 0). The second grid shows the window around the offset that is searched for candidate matching nodes. The third grid shows the ambiguity window for one of the nodes in the search window. The correlation function will be calculated for all seven of the pixels in this window. This window is due to the ambiguity



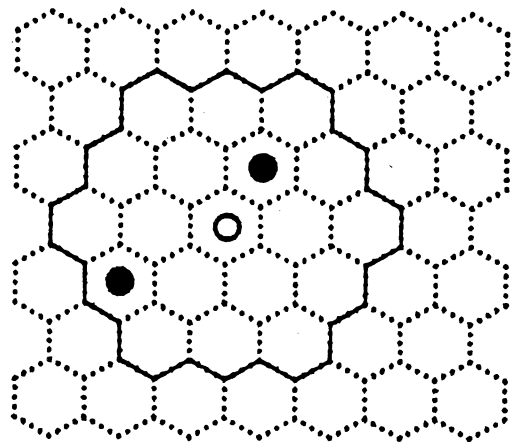
Initial Offset



Search Window



Ambiguity Window



Correlation Window

Figure 15. Match Windowing Scheme

of labeling some types of nodes mentioned in the previous chapter. The difference in perspective of the two images will cause the vectors to be oriented in a slightly different way in the two images. This could cause the node labels in the two images to differ by a single pseudo-pixel in some direction. Thus, all seven pixels in this window are possible match candidates. In the example shown in Figure 15 on page 38, 14 correlation functions will be calculated, seven for each node. The fourth grid in the figure shows an actual correlation window and its corresponding candidate at its center. This window contains 19 pixels, which makes the range of the correlation function -19 to +19.

5.2 CORRELATION CONFIDENCE INTERVALS

The choice of a correlation function threshold to determine when a valid match has been found is crucial to the success of the algorithm. If the threshold is too low, false matches will be made. If the threshold is too high, the network of matched points will not be dense enough. The thresholding scheme developed for the present algorithm is an original idea which came about during the investigation. At first I tried a single threshold, but no matter what

value that I used I would either get false matches or miss good matches. False matches were most easily observed by overlaying the matches on the pseudo-hex gray scale. Missed good matches were evident from overlaying a transparency of one edge graph on top of the other. In the windows where false matches were made it was observed that the correlation function had two peaks, namely a false match and the true match. When the peaks are close together in magnitude, an unreliable match will be made whether the true match wins out or not. What was needed was a confidence interval to give some sense of reliability to the matches that were made. Intuitively, the interval need not be as stringent for high correlation values as it is for low ones. The difference between a 95% correlation and a 90% correlation is more significant than the difference between 65% and 60%. This suggest a dual-threshold, dual-interval approach.

The Fisher-Z Transformation provides a mathematical basis for both the dual-threshold and the dual-confidence-interval. The Fisher-Z Transformation is given by

$$z = 1.15 * \log_{10}((1+r)/(1-r))$$

where r is the correlation coefficient (percentage). As stated in Moroney [6], the Fisher-Z Transformation may be used for testing the significance of a correlation coefficient. It can also be shown that z will be distributed with standard error given by

$$\text{std_error} = 1/\text{SQR}(N-3)$$

where N is the number of pairs used to estimate r . A difference of std_error between two z values is then said to be significant.

Table 1 on page 42 shows the Fisher-Z applied to the match correlation function (column 1). The last column in the table gives the matching function interval based on one standard error difference from column 1. The first threshold value was chosen experimentally to be 12, with a confidence interval from the table of 2. The second interval comes directly from the table, namely 15 with a confidence interval of 1. Thus, the maximum and runner-up values of the matching function calculated for a node must satisfy the following:

```

Match = False
IF (QM >= 15) AND ((QM-QR) > 1)
  THEN Match = True
ELSE IF (QM >=12) AND ((QM-QR) > 2)
  THEN Match = True
END IF

```

Table 1. Matching Function Fisher-Z

N = 19

Standard Error = $1/\text{SQR}(19-3) = 0.25$

$z = 1.15 * \log_{10}((1+r)/(1-r))$ where $r = \text{Mat_Corr} / N$

Match Correlation	r	Fisher-Z	Mat_Corr -Std_err	Confidence Interval
1	.053	.121	-	-
2	.105	.243	-	-
3	.158	.366	-	-
4	.211	.492	1	3
5	.263	.620	3	2
6	.316	.752	4	2
7	.368	.889	5	2
8	.421	1.033	6	2
9	.474	1.184	7	2
10	.526	1.346	8	2
11	.579	1.520	9	2
12	.632	1.711	10	2
13	.684	1.925	11	2
14	.737	2.170	12	2
15	.789	2.461	14	1
16	.842	2.825	15	1
17	.895	3.324	16	1
18	.947	4.153	17	1

where QM is the maximum value and QR is the runner-up value. If Match = True then the pixel with match correlation QM is the corresponding point in image 2 for the node in image 1. This matching process is carried out for every node in image 1. The result will be a random network of matching point disparities. A spatial interpolation is then performed to fill in the blank areas.

6.0 INTERPOLATION

As the hierarchical steps progress and the resolution gets finer, more details will be present in the pseudo-hex gray scale. The increase in detail will cause more nodes to be present. Inevitably, there will be nodes in the areas between matching points in the network at the previous resolution. To provide an initial offset for such nodes to the matching algorithm, a two-dimensional interpolation must be performed on the previous network of disparities. Many 2-D interpolation algorithms exist, but all of the ones that I discovered were global algorithms. They required information about the entire network, or a major portion of it, to calculate a value for one point. Besides being global these algorithms are computation-intensive. Keeping in mind future hardware implementations of the present algorithm and to take advantage of a parallel architecture, a local interpolation algorithm was needed. By local, I mean an algorithm that requires only information in an immediate small window.

6.1 THE ALGORITHM THAT I DEVELOPED

The algorithm that I devised is illustrated in Figure 16 on page 46. The algorithm searches an expanding window for known, or previously calculated, disparities. The windows are divided into 12 radial segments in the same directions as the quantized edge-vector angles. As the window expands, only a single closest disparity in each segment is retained. One wouldn't take into account the valley behind a mountain when he was trying to interpolate an elevation in front of the mountain. Also, if there were a concentration of disparities in one segment, they would tend to distort the interpolation if they all were considered. This is a spatial interpolation, not a numerical average.

The window expansion terminates when a disparity is found in two segments that differ in angle by at least 120 degrees, or when the window reaches some preset maximum size. This ensures that there is a reasonable spatial distribution of known points before an average is performed. If the angle requirement is met, then the retained disparities are combined in a weighted average. Disparities in the largest window are counted once, those in the second largest

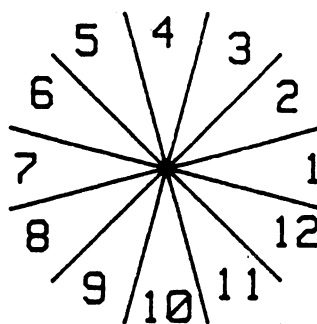
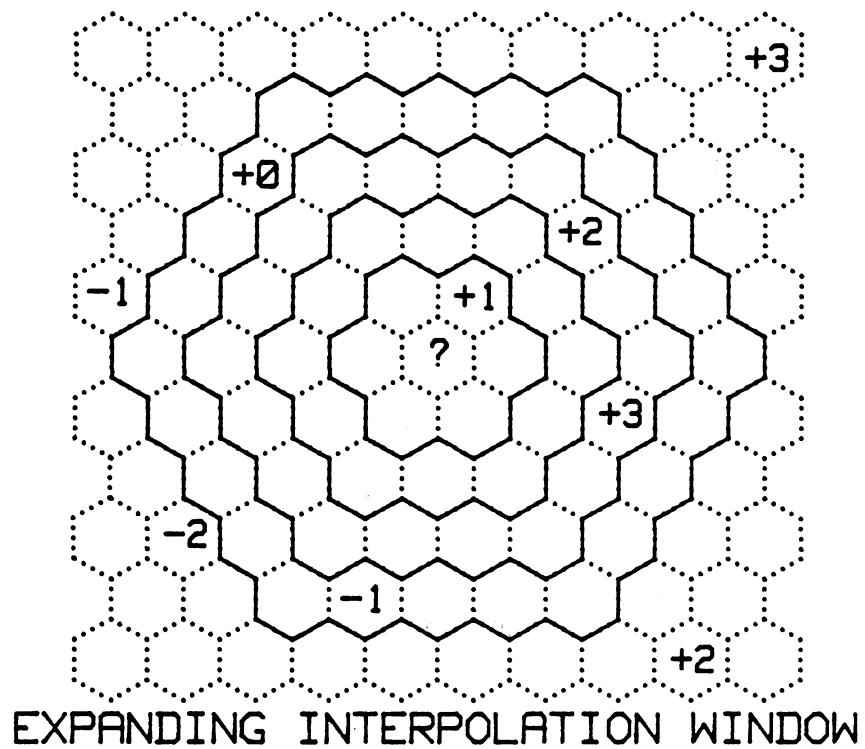
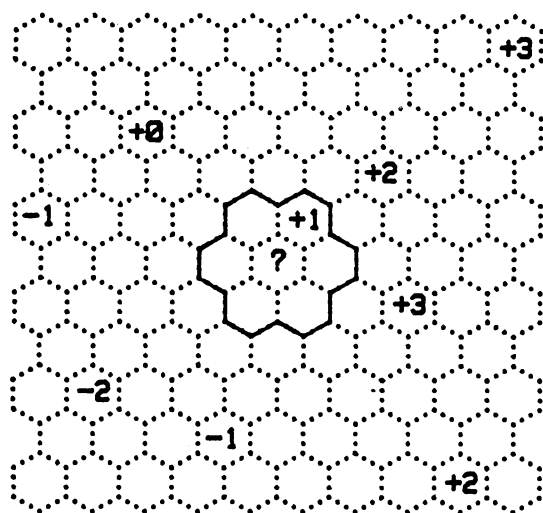


Figure 16. The Interpolation Windowing Scheme

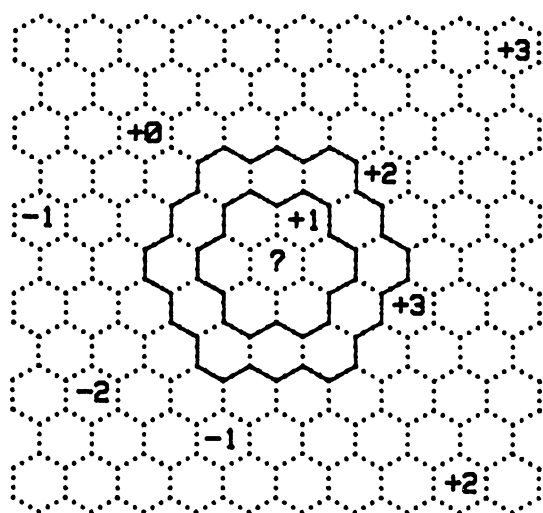
twice, etc. Any disparity in the first window will be counted N times, where N is the size of the largest window (hopefully not the maximum window). This weighting causes the average to be more in favor of the closer points, which is precisely the effect desired. If the window expands to the maximum allowable size, then all pixels in the first window are averaged regardless of angle. This allows holes in the network to close slowly until the window can span them and perform a proper interpolation. If the maximum window size terminates the expansion and there are no points in the first window, then the point remains a hole. These last two cases require repeated passes over the grid until all holes are filled.

6.2 AN EXAMPLE

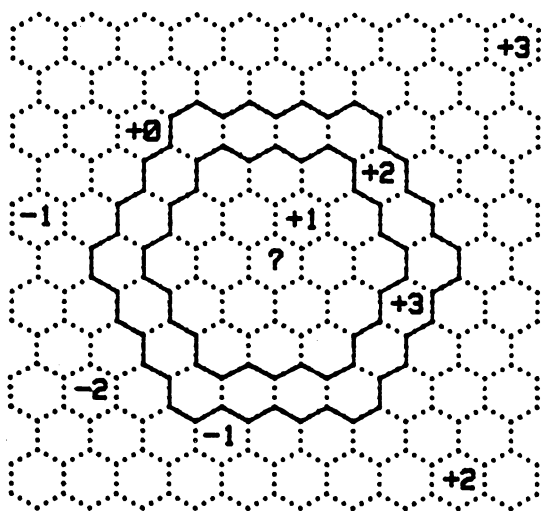
Figure 17 on page 48 gives a step by step illustration of the expanding window. A value is being interpolated for the pixel at the center of the window marked with the question mark. The first window searches the 6 immediate neighbors. In the figure, a +1 is found at an angle of 60 degrees. It will be stored in radial segment 3 as being found in the first window. The second window looks one pixel out in each direction. In the example of Figure 17 on page



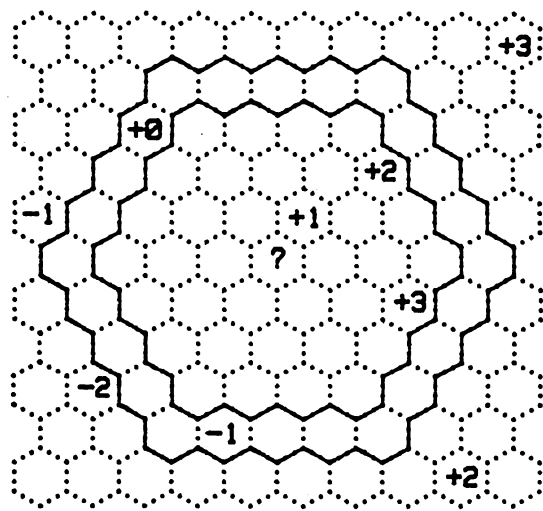
First Window



Second Window



Third Window



Fourth Window

Figure 17. An Example Interpolation

48 this window is empty. The third window looks out one pixel in all directions again. This time the window contains a +2 in segment 2, and a +3 in segment 12 (see Figure 16 on page 46 for the segment labels). So far radial segments 3, 2, and 12 are filled. The maximum angle between these segments is only 90 degrees, so the search continues. The window expands once more to the fourth window. Two points are found in this window, a +0 in segment 5 and a -1 in segment 10. Now there are points in segments 3, 2, 12, 5 and 10. Several of these differ by 120 degrees or more, and thus, there is a good spatial distribution of points, and the search stops. The weighted average is now calculated based on a window size of four:

$$\text{Interp} = \frac{4(1) + 2(2) + 2(3) + 1(0) + 1(-1)}{4 + 2 + 2 + 1 + 1}$$

where in integer arithmetic, Interp is 1. Looking back at the figure, this seems like a reasonable value.

6.3 A PROPOSED IMPROVEMENT

There is one possible improvement in the above two-dimensional interpolation algorithm that might be made

in this application. Recall that the only points where a disparity value from the previous resolution is needed in the current resolution are at the locations of nodes in image 1. If this information can be fed into the interpolation, some time could be saved interpolating disparities for pixels where it is never used. Note that with a fixed maximum window, one could not just try to interpolate the network of disparity at next-step node locations because it may be necessary to calculate some intermediate points to make some of the interpolations. Also note that the difference in scale of the grids will cause some loss in accuracy. A compromise improvement will be given at the end of the next chapter.

7.0 EXPANDING THE DISPARITY ARRAY

Referring back to Figure 4 on page 11 one sees that the next step in the basic algorithm is to expand the interpolated disparity array to the next resolution size. This step was introduced to provide for more accuracy in the disparity interpolation near large elevation gradients. This step simply expands the array by the step size, three, in both the x and y directions. In the expanded array every third point in every third row will have a disparity. As shown in Figure 4 on page 11, this array is then interpolated to fill in the holes.

As mentioned in the previous chapter, some time could be saved in the algorithm if the node information at the next resolution were used to limit the interpolation of disparity. This second interpolation, would be an ideal place to include such information. In the expanded disparity array every third pixel is a known disparity. Thus, with a reasonable window size one could simply make one interpolation pass and interpolate only at the next resolution nodes. This will save a considerable amount of time per resolution.

8.0 GRAY LEVEL MATCHING

After matching at pseudo-hex res 2, the parallax of the matching points should be within one orthogonal pixel of their true values. To further refine the parallax a modified normalized gray-scale correlation is performed on the original images. Because of the certainty given by the res 2 matches, this correlation can be carried out using a very small search window. The present algorithm uses a 5 by 5 correlation window on a 5 by 5 search window. A 5 by 5 window is used rather than a 3 by 3 to give a factor of reliability to an obtained match. Since a match should be within ONE pixel of the res 2 parallax, one should verify this by checking that the correlation is actually lower at points TWO pixels away.

The normalized cross-correlation is described in detail in Duda and Hart. It will be described in brief here. A fixed window in array A1 is correlated to a moving window in an array A2 by calculating the following quantity at each point in the search window:

$$\frac{\text{SUM: } A1(i,j)*A2(i+di,j+dj)}{\text{SQR(SUM: } A2(i+di,j+dj)**2)}$$

where d_i and d_j vary from -2 to +2 for a five by five correlation window. The denominator of the equation normalizes each quantity to the energy present in its A2 window. The pixel at the center of the moving window with the highest value is then considered to be the best match for the pixel at the center of the fixed window in A1.

A peculiarity of the normalized cross-correlation is that the match pair could be a function of whether A1 is correlated to A2, or A2 is correlated against A1. That is, if you correlate P0 in A1 to A2 and get a maximum at PA, then correlate PA back onto P0 and get a maximum at PB, P0 and PB may not be the same point. If one must have a match for P0, then this is not a problem, but larger windows might be used. If one doesn't necessarily need a match at P0, but somewhere near P0, then the above example suggests an algorithm to obtain a more reliable matching pair.

8.1 THE PING-PONG CORRELATION SCHEME

The previous example suggests a type of 'ping-pong' algorithm to find the most reliable match. Figure 18 on page 55 illustrates this modified correlation

technique which is used in the present algorithm. Starting from P_0 in image 1, the 5 by 5 correlation window is used in a 5 by 5 search window in image 2, with its center given by the offset at res 2. Let P_A be the pixel with the maximum correlation value. In step (B) of the figure, the window around P_A is then correlated back onto a 5 by 5 search window around P_0 . Let P_B be the maximum correlation at this step. If $P_B = P_0$ and P_A is within one pixel of the res 2 offset, then P_0 and P_A form a matching pair at res 1. Otherwise, step (C) in the figure is performed. In this third, and final correlation, P_B in image 1 is correlated back onto P_A in image 2. If the maximum correlation is again at P_A , then P_B and P_A are considered a matching pair, otherwise no match is found. This technique adds reliability to the small window correlation. When compared to larger window sizes, it still maintains its computation time advantage. With 5 by 5 windows, the ping-pong method has an execution time on the order of $3 \times 25 \times 25$, or 1875, while a single correlation using 9 by 9 windows has execution time on the order of $1 \times 81 \times 81$, or 6561.

The density of the matching points at res 1 depends on the application. In the present algorithm the

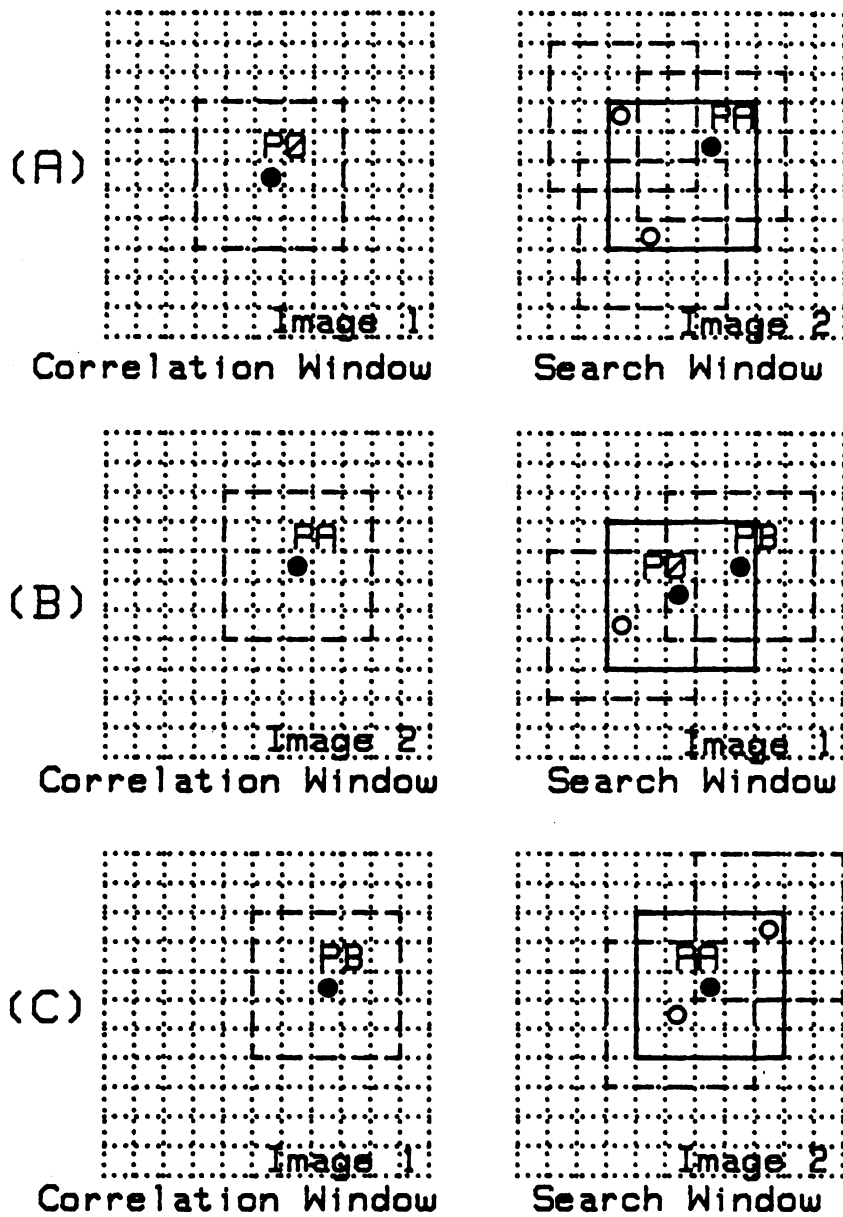


Figure 18. The Ping-Pong Correlation Method

ping-pong correlation is applied to every 4th pixel on every 4th row. This step size was chosen because it is a multiple of the previous resolution size of two. Figure 19 on page 57 shows the network of gray-scale matching points found in the test pair. Note that some of the grid points are missing and that some of them are misaligned. The missing points are where the ping-pong method was not able to lock on to a reliable match, and the misaligned points are where part (C) (see Figure 18 on page 55) of the algorithm found a better set of matching points than the original 4 by 4 grid point.

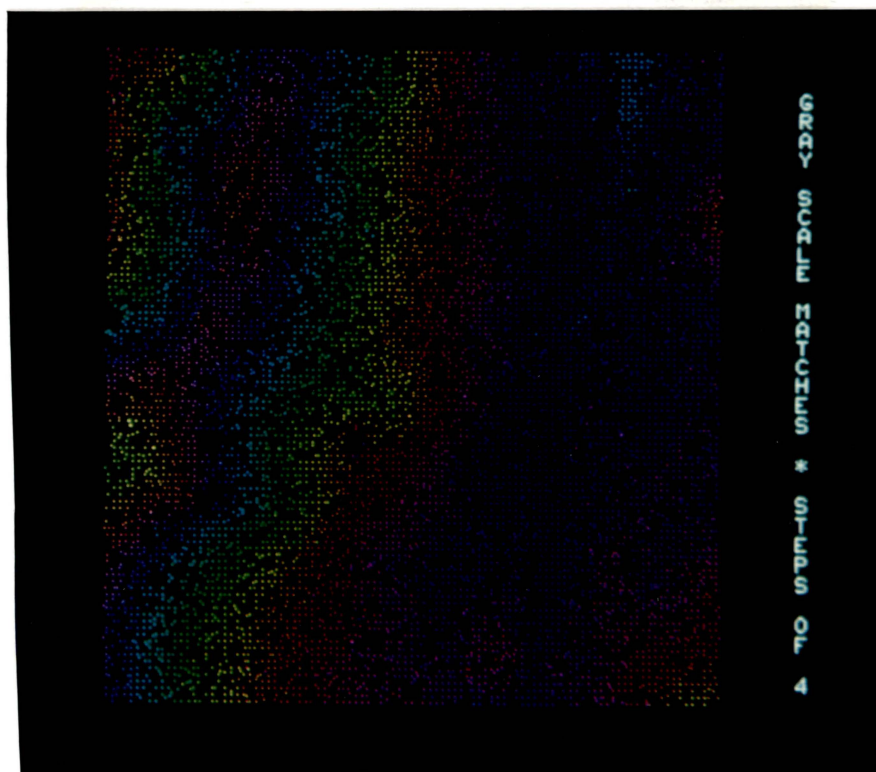


Figure 19. Network of Gray Scale Matching Points: with colors indicating the X-disparity contours

9.0 RESULTS

9.1 SAMPLE RUN OF THE ALGORITHM

The first step in the algorithm is the reduction to pseudo-hex resolution. Figure 20 on page 59 shows the test pair, from Figure 1 on page 5, at pseudo-hex resolution size 18. Note the offset rows of pixels characteristic of the hexagonal array. At this resolution the maximum parallax between the photos is reduced to plus or minus one pseudo-pixel, which is necessary for the first resolution step.

The second step in the algorithm is the application of the edge operator. Figure 21 on page 60 shows the full edge vector field of the left image (image 1) from Figure 20 on page 59. Note the streams of parallel vectors which tend to follow the pseudo-hex gray-scale features. Figure 22 on page 61 shows the same vector field, but with the vector lengths plotted proportional to the vector magnitudes. The gray-scale features are much more prominent in this plot.

The next step is the application of the edge filters. Figure 23 on page 62 shows the resulting edge field

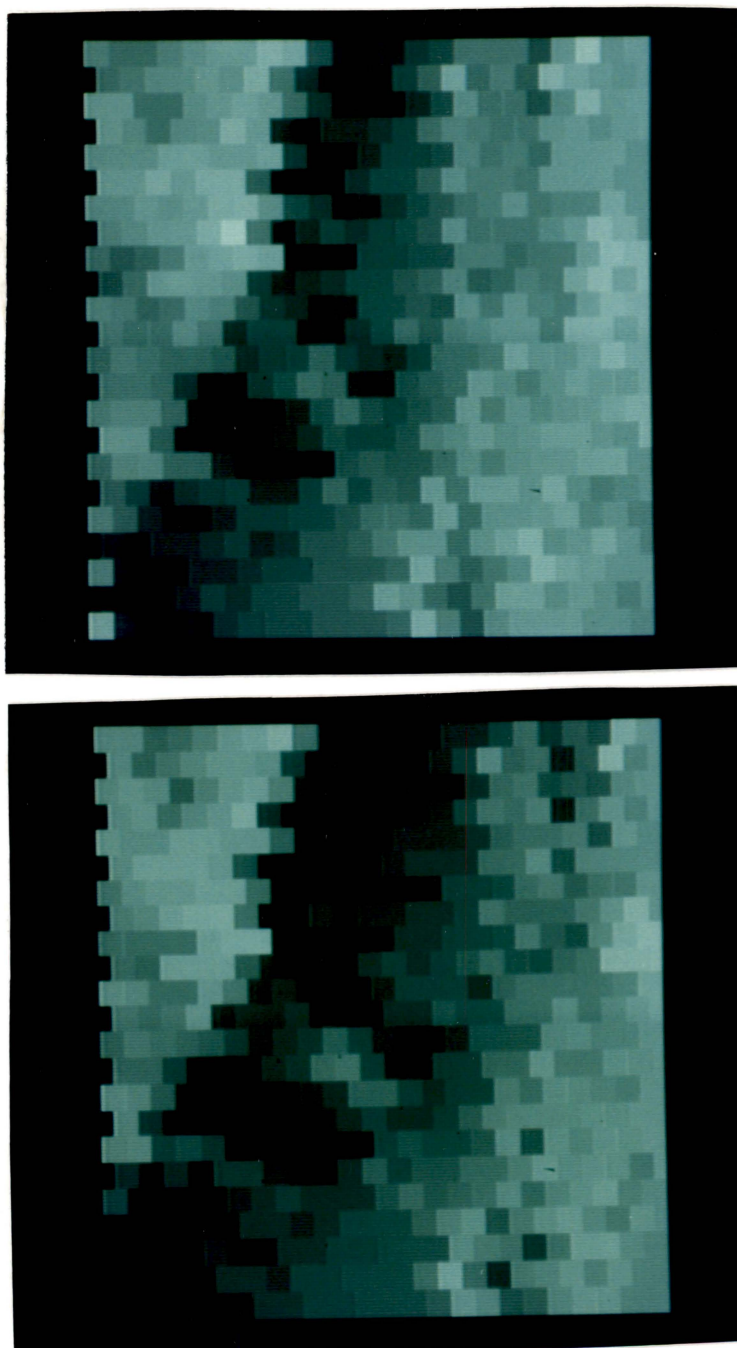


Figure 20. The Test Pair at Pseudo-Hex 18

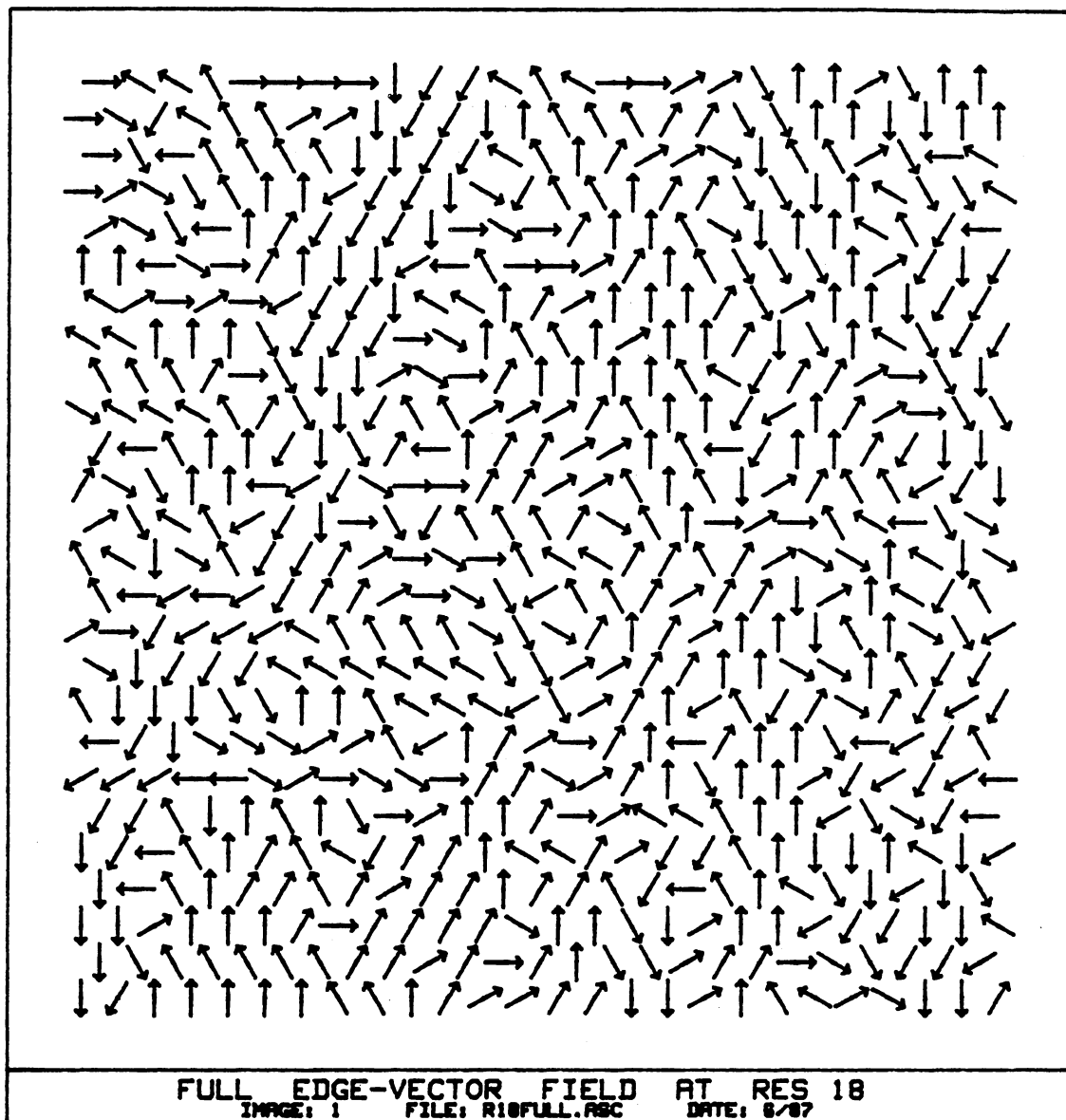


Figure 21. The Full Edge Field

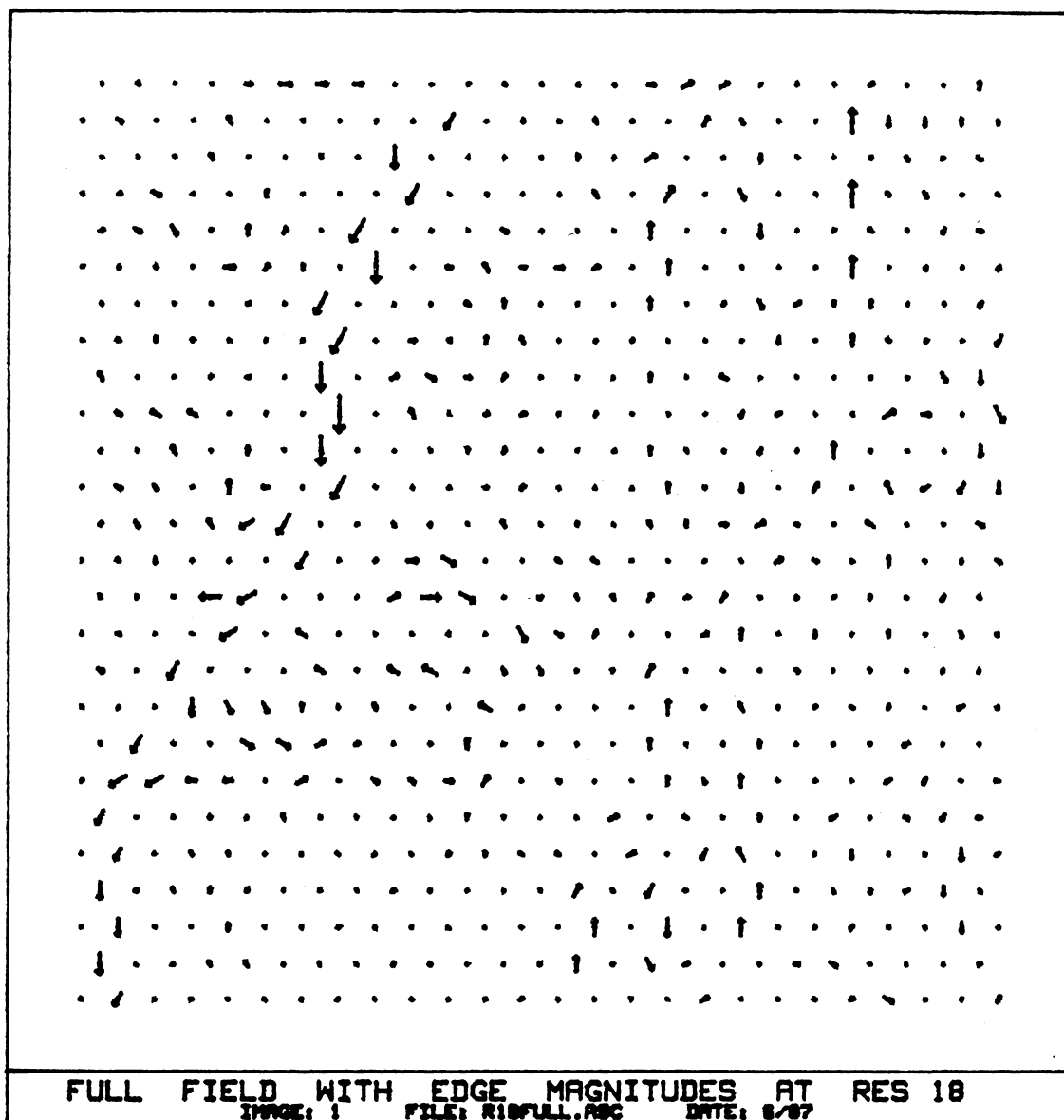


Figure 22. Full Field with Edge Magnitudes

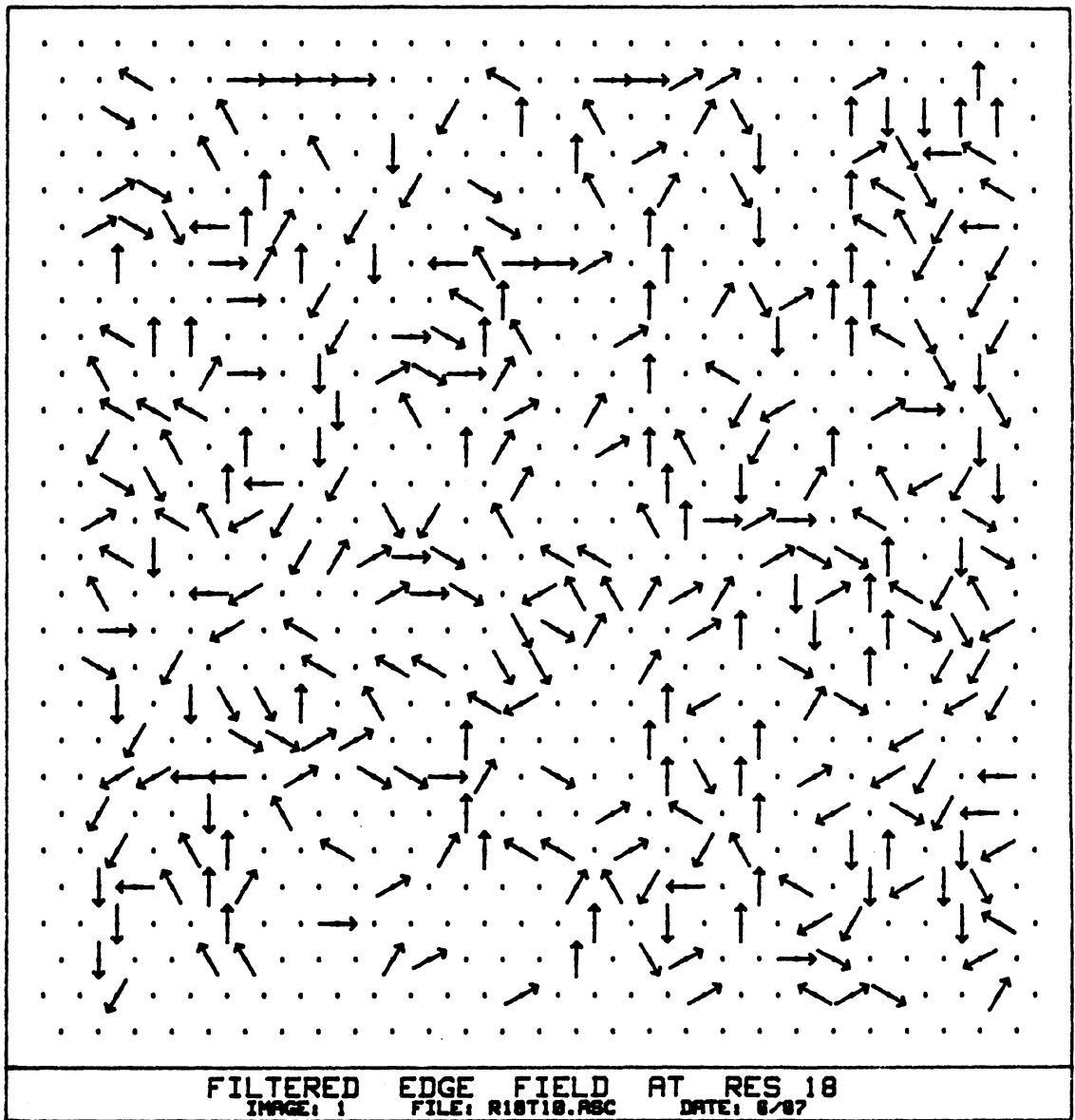


Figure 23. Associated Edge Field

after the threshold and association filters have been applied. An edge threshold of 10 was used at res 18. The most prominent gray-scale features are described by these edges.

Figure 24 on page 64 shows the nodes that were found on the filtered-edge field. The algorithm attempts to find matching points in image 2 for all of these nodes in image 1. Reliable matches may not be available for all of these nodes.

Figure 25 on page 65 and Figure 26 on page 66, show the matches found on the full vector fields for image 1 and image 2, respectively. Since the disparity at this level is at the most one pseudo-pixel, the corresponding matches in the two plots are easily found by observation. Note the similar vector orientations in the windows around corresponding points.

The next step in the algorithm is to do the spatial interpolation of the x-disparities. The y-disparities are assumed to be near zero. Figure 27 on page 67 shows the x-disparity of the matching points, and Figure 28 on page 68 shows the resulting spatial interpolation. In the photos, red regions represent low elevation and blue regions represent high elevation.

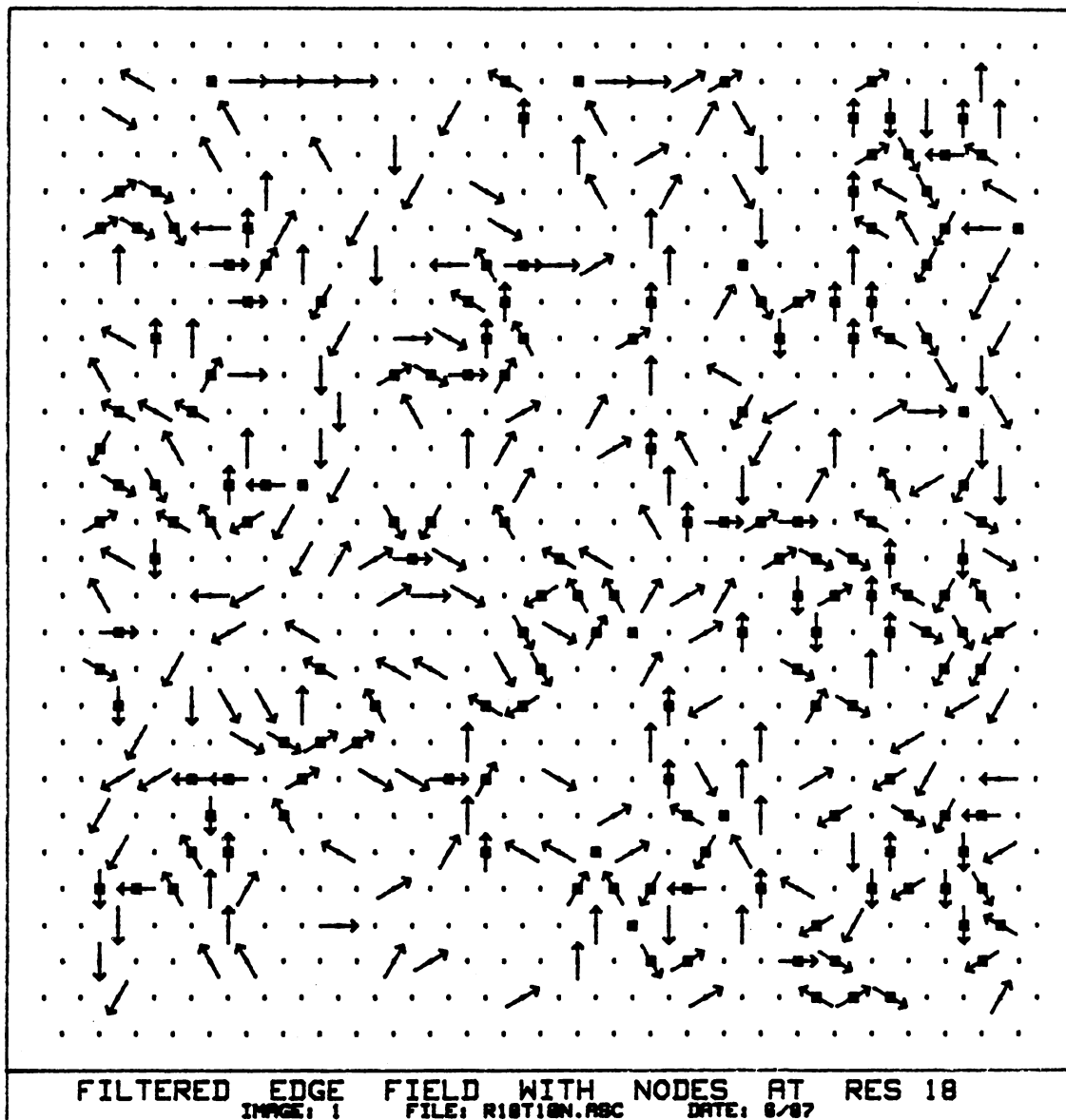


Figure 24. Nodes on the Associated Edges

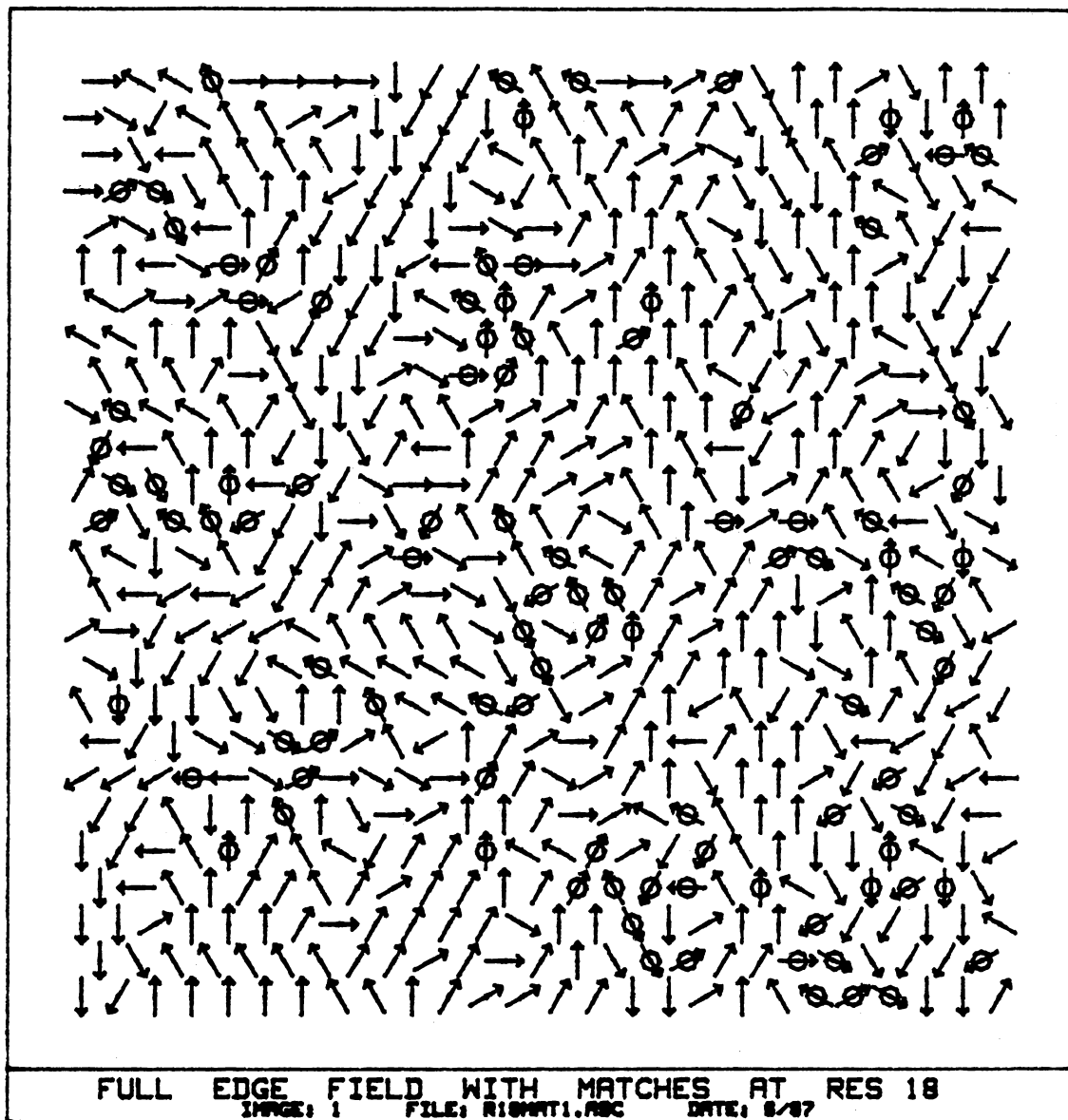


Figure 25. Matches on Image 1

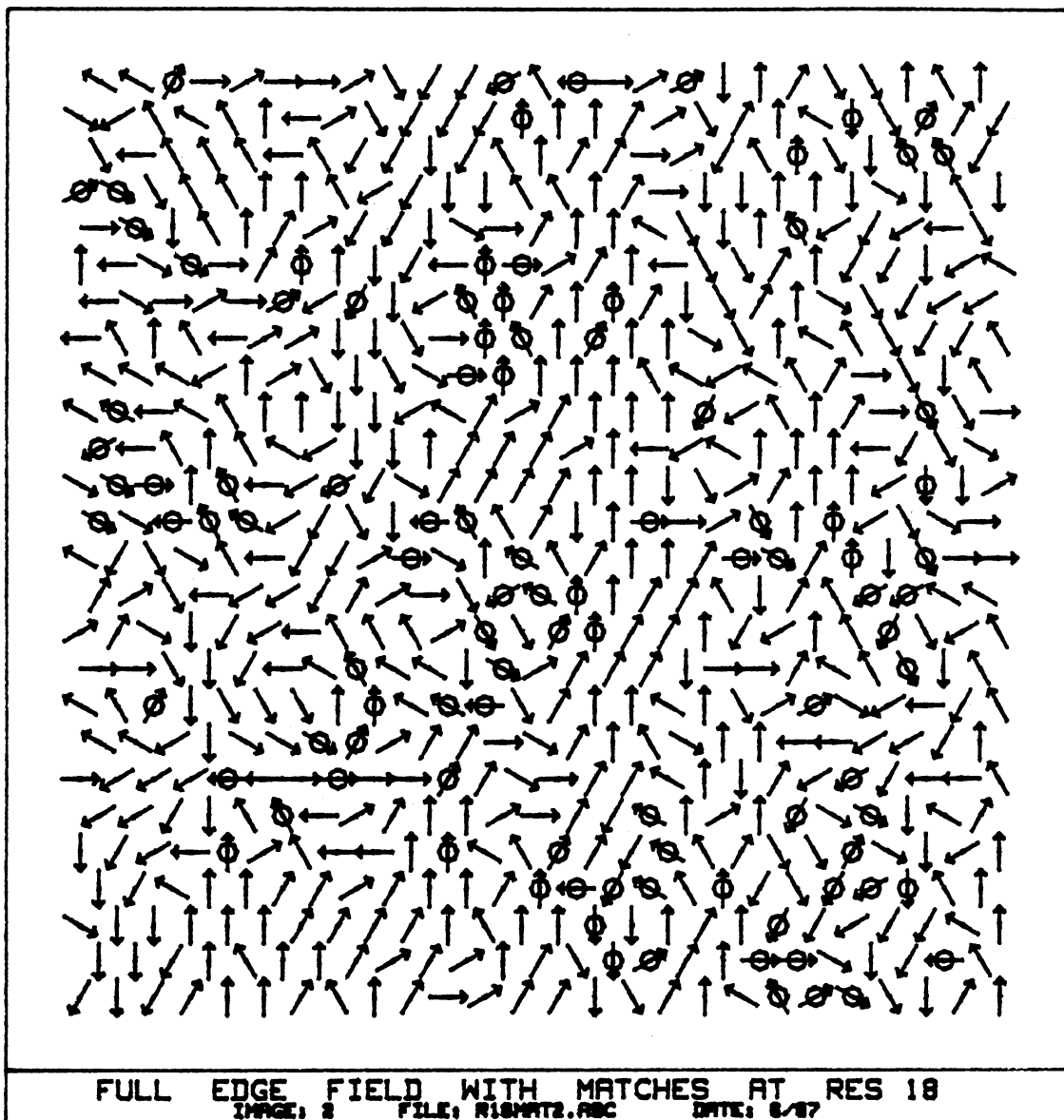


Figure 26. Matches on Image 2

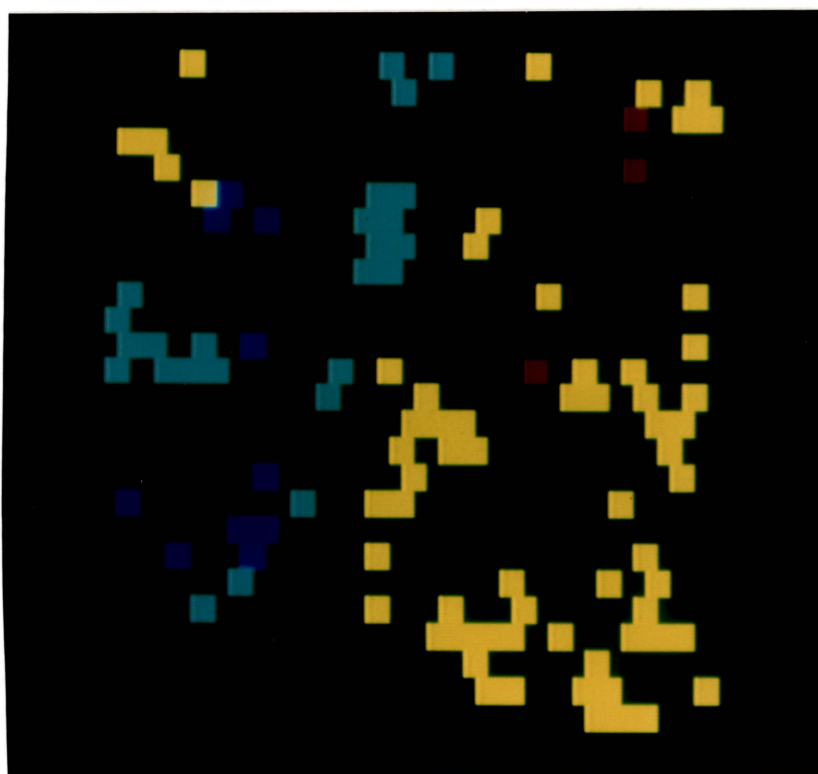


Figure 27. Match Disparity from Res 18

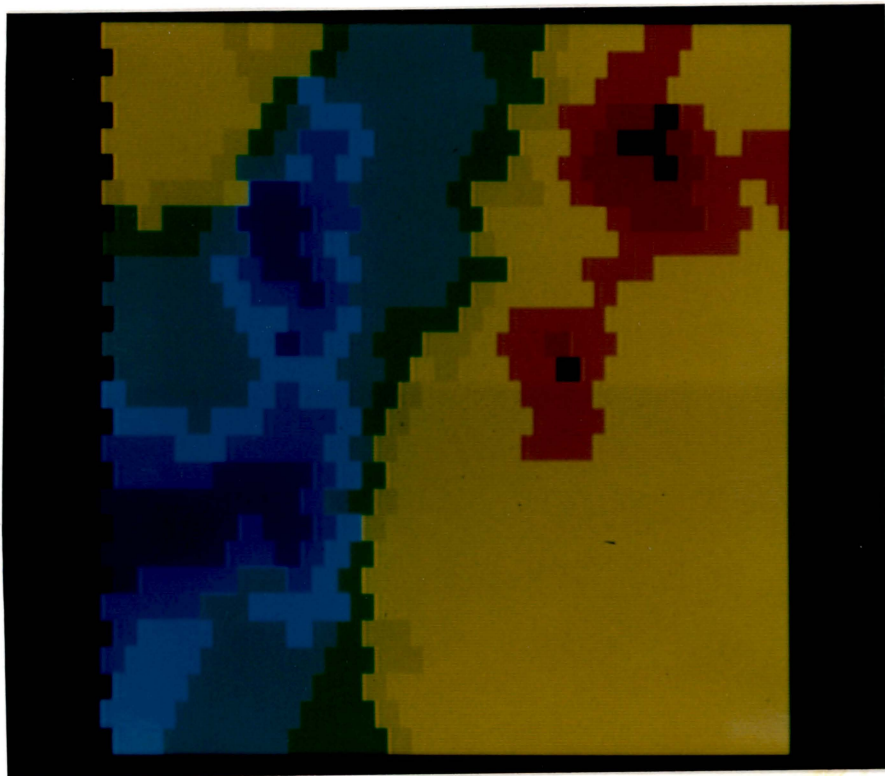
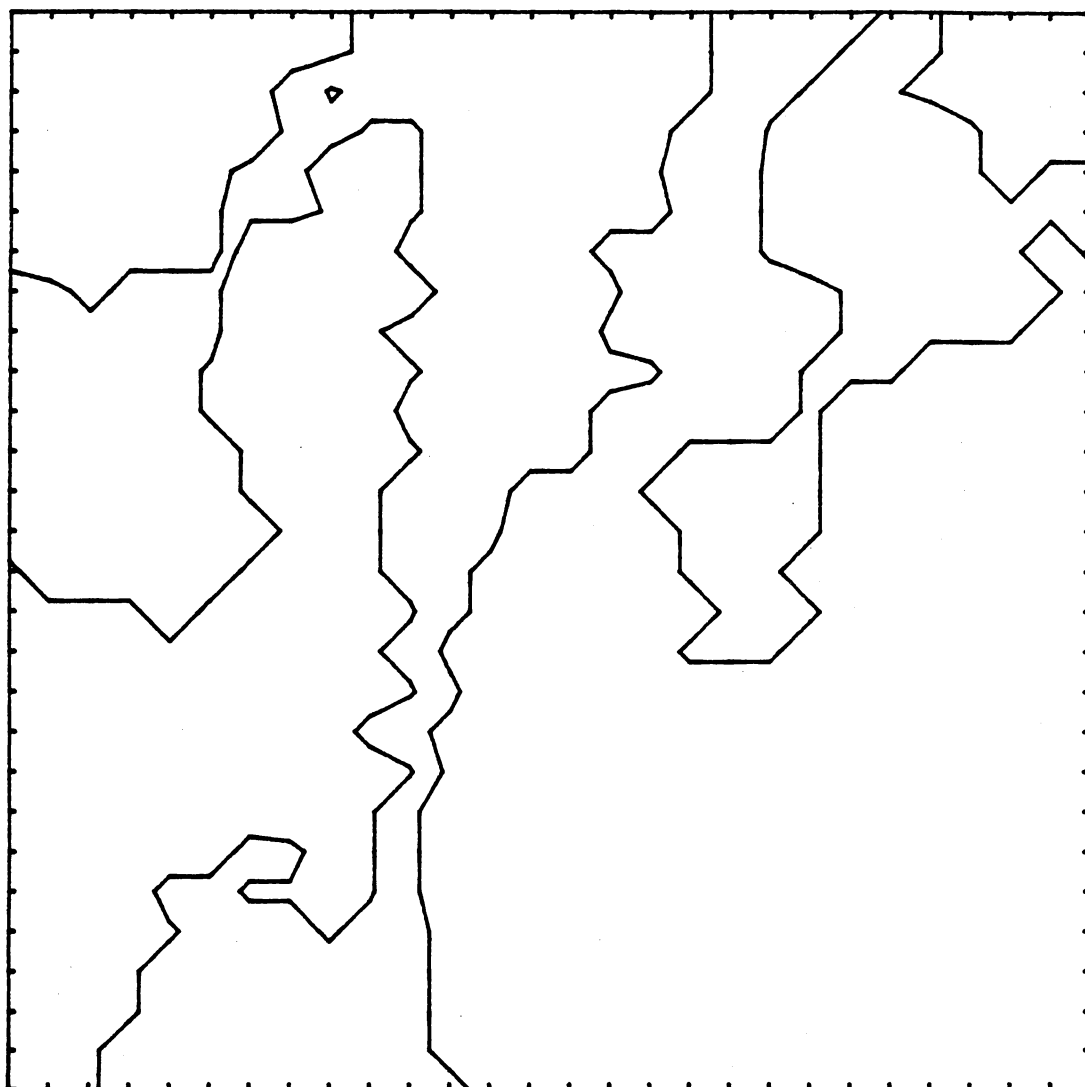


Figure 28. Interpolated Match Disparity



X-DISPARITY CONTOUR AFTER RES 18
Files: CON10.ASC Array has 28 rows and 28 columns
Minimum: 122 Maximum: 131 Interval: 2.25 Density: 4

Figure 29. Contour Plot after Res 18

In this first interpolation the disparity is multiplied by three to interface to the next resolution. From the interpolated disparity, the contour plot in Figure 29 on page 69 can be made.

To prepare for the next resolution size, the disparity array must now be expanded and interpolated. Figure 30 on page 71 shows the results of the expansion at res 18, and Figure 31 on page 72 shows the resulting interpolation.

The same steps are repeated again at res 6. Figure 32 on page 73 shows the test pair at pseudo-hex res 6. An edge threshold of 15 was used at res 6. Figure 33 on page 74 shows the nodes found on image 1 at res 6. Figure 34 on page 75 shows the matches found on image 1 at res 6. The resulting, refined contour is shown on the plot in Figure 35 on page 76.

At res .2 the plots get too large to be presented in this text. Also the pseudo-hex res 2 photos are indistinguishable from their originals and will also be omitted. An edge threshold of 20 was used at res 2.



Figure 30. Expansion of Disparity Array

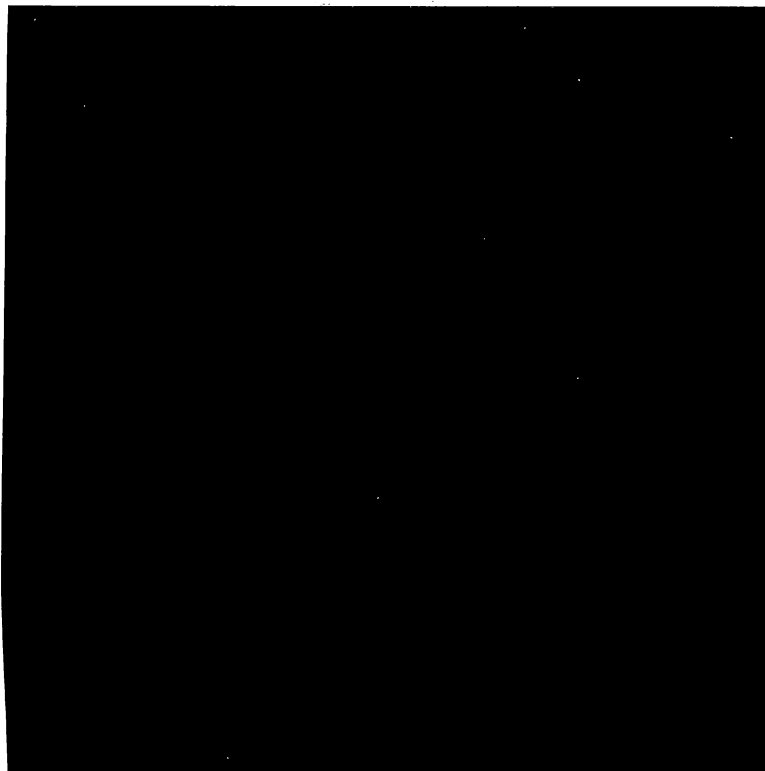


Figure 31. Interpolation of Expanded Array



Figure 32. Pseudo-Hex Res 6 Gray Scale



Figure 33. Pseudo-Hex Res 6 Nodes

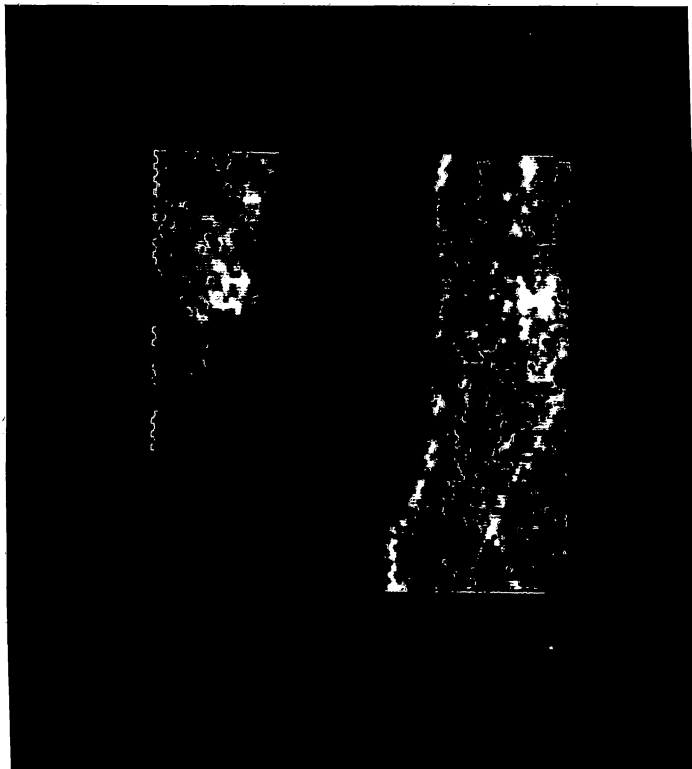
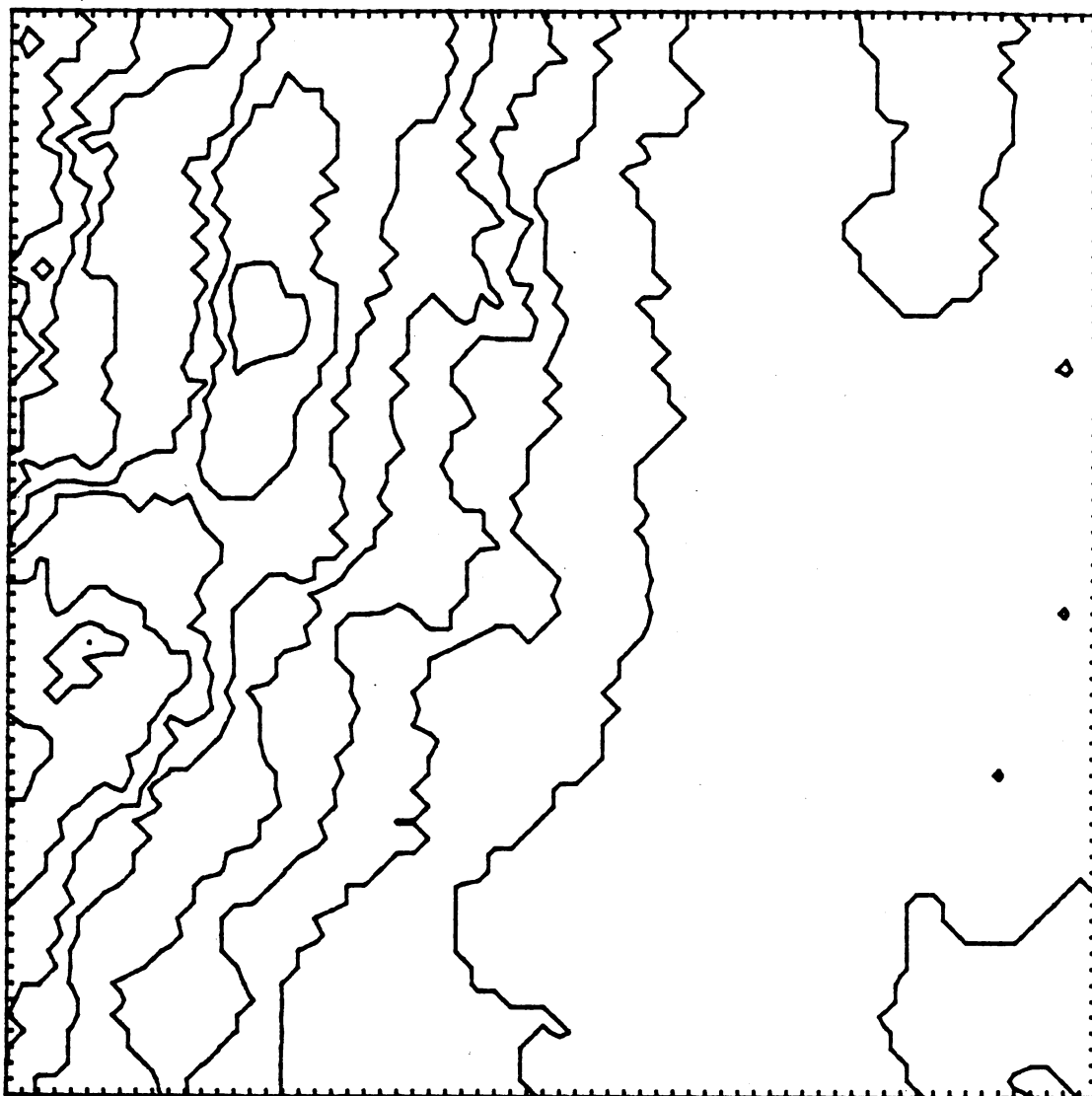


Figure 34. Pseudo-Hex Res 6 Matches



X-DISPARITY AFTER RES 6
File: CON6.ASC Array has 60 rows and 60 columns
Minimum: 112 Maximum: 148 Interval: 2.7 Density: 18

Figure 35. Contour Plot after Res 6

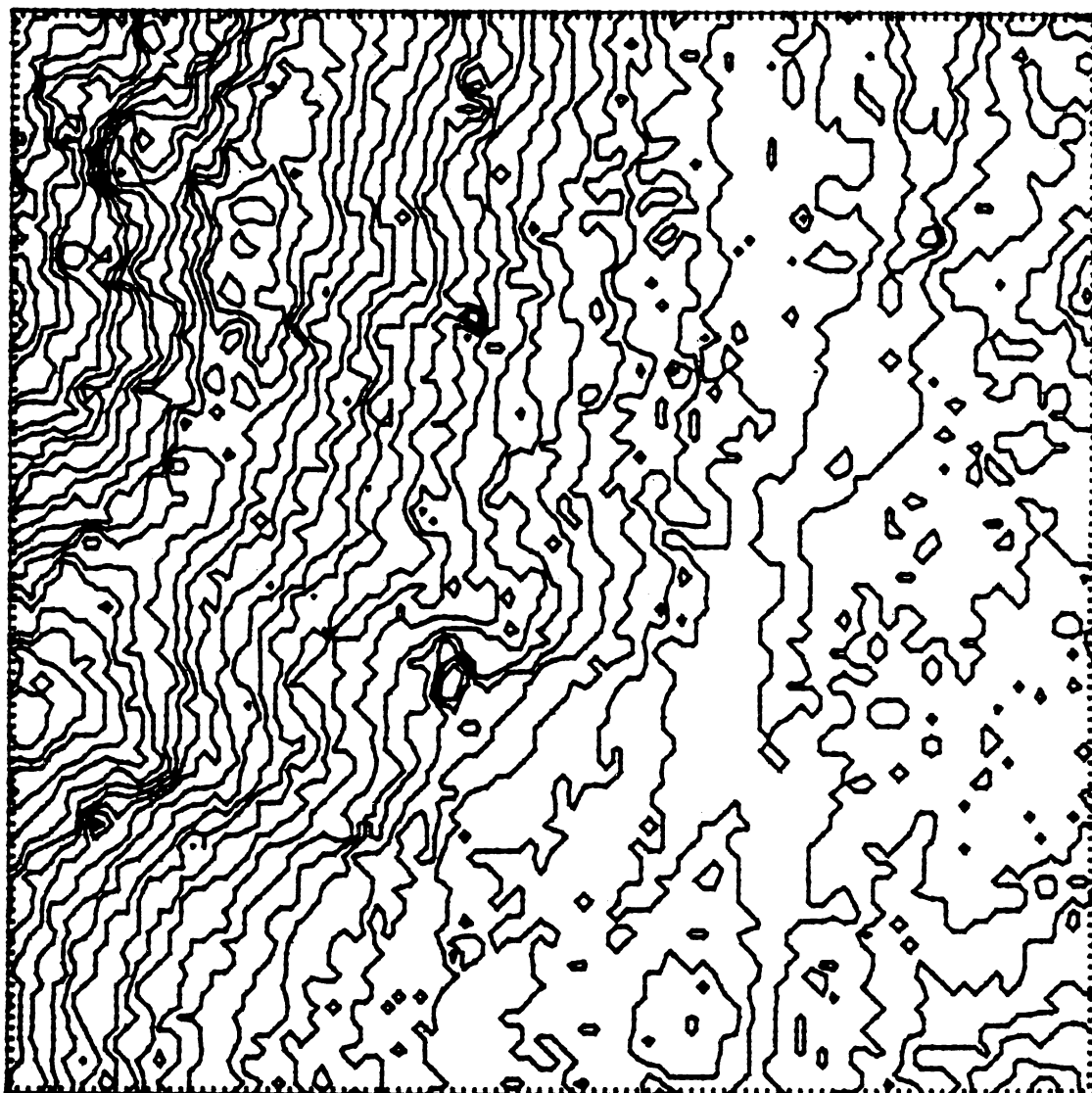
The final contour plot obtained from the 4 by 4 gray scale matching on the original images is shown in Figure 36 on page 78.

For comparison the ETL elevation contour plot is given in Figure 37 on page 79. The contour lines are generally parallel throughout the image.

For a three dimensional comparison, photos of surface plots of the contours are given in Figure 38 on page 80. The algorithm contours are based on x-disparity which directly relates to the image relief. The ETL data is actually in tenths of meters of elevation. The surface plots are indeed very similar.

9.2 EXECUTION SPEED

The computing environment under which the present implementation of the algorithm runs is described in Appendix A. The CPU times for each part of the entire algorithm are compared in Table 2 on page 81. One can see the increase in time as the resolution size decreases. Naturally, the array sizes get larger as this happens. These data are based on an original image size of 512 by 512. The pseudo-hex array sizes are then 28, 85, and 256. The network of matching



FINAL X-DISPARITY CONTOUR
File: CON1.ABC Array has 181 rows and 181 columns
Minimum: 87 Maximum: 153 Interval: 1.82183446275 Density: 29

Figure 36. The Final Contour Plot

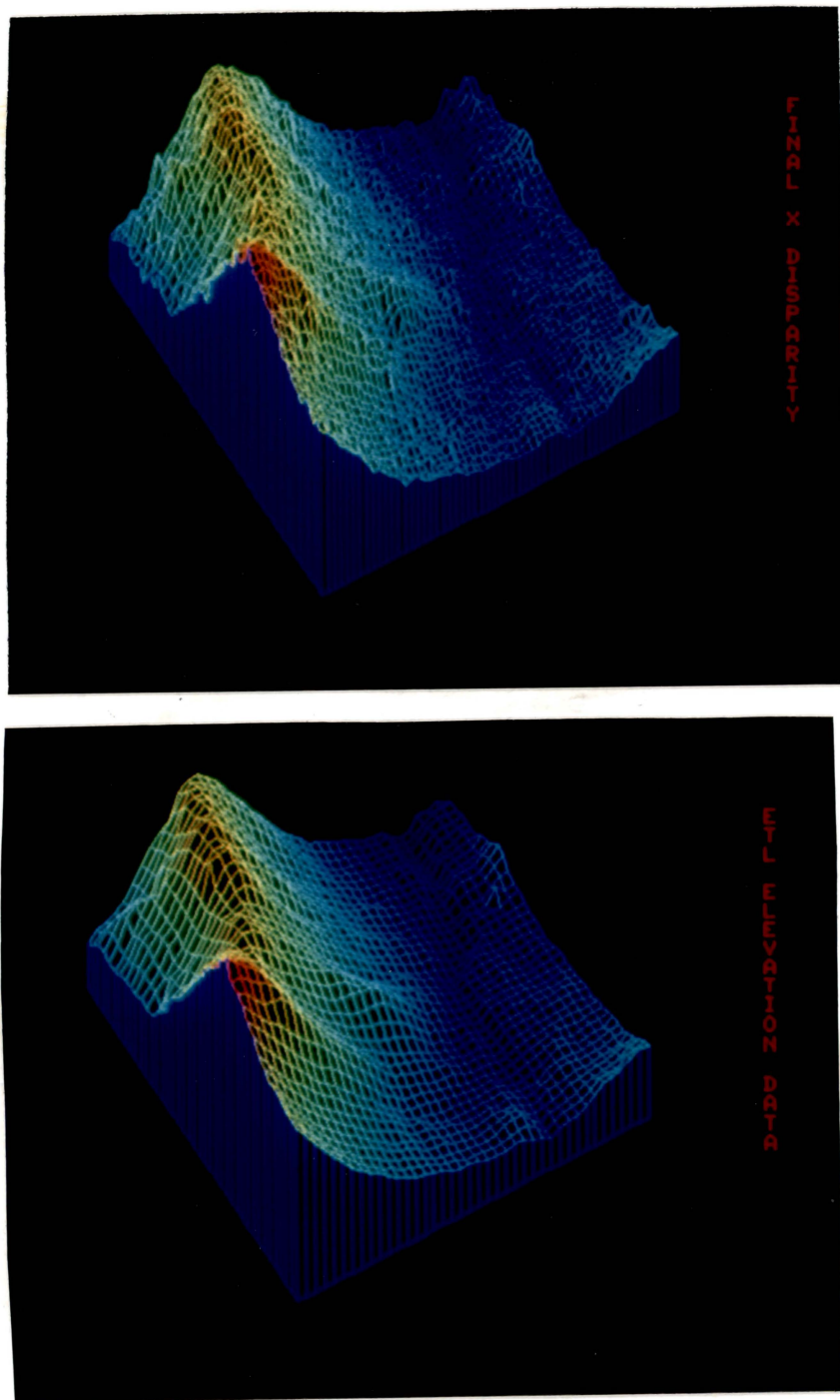


Figure 38. A Surface Plot Comparison

Table 2. Individual Execution Times (sec)

	Res 18	Res 6	Res 2	Res 1
Hex1	8.17	11.77	36.54	-
Hex2	8.01	11.98	37.70	-
Nod1	1.78	4.84	27.23	-
Nod2	1.73	4.84	27.94	-
Match	13.64	134.75	743.16	520.12
Interp	26.53	120.58	1125.61	-
Expand	1.28	2.97	-	-
Interp	27.73	181.80	-	-

points at the gray-scale level contains approximately 125×125 (500/4), or 15,000 points. This gives a data rate on the order of 5 pts/sec. Compared to ETL's rate of 15 pts/sec on a Cyber, which at present technology is a super computer, 5 pts/sec on a Vax is significant. The Cyber-730 may run anywhere from 10 to several hundred times faster than a Vax-11/785, depending on the nature of the program. ETL's data rate does not even include the extensive time required for the initial interactive matches. At the very least, the algorithm presented in this thesis is an order of magnitude faster than ETL's current algorithm.

APPENDIX A. SPEED ANALYSIS

The software implentation of the algorithm was done in GIPSY. GIPSY is an image processing environment incorporating RATFOR (a structured extension of Fortran). At Virginia Tech, GIPSY runs on the SDA Vax-11/785, typically rated as the standard 32-bit, 1 MIP machine. Experimental results in the SDA lab show that the Vax runs at about 0.8 MIPS while executing a typical GIPSY program. All of the data files in the algorithm are stored in 8-bit integer SIF (Standard Image Format) files. The majority of computations operate directly on Fortran 32-bit integers. The only real arithmetic done in the present algorithm are the edge magnitude and angle calculations, the match quality sum, the radial angles in the interpolation, and normalizing of the gray-scale correlation. The cosines in the matching algorithm are stored in a table, since the angles are multiples of 30 degrees.

APPENDIX B. IGNORING Y-DISPARITY

In a stereo matching algorithm one can greatly reduce computation by using images that have their parallax reduced to a direction along one of the orthogonal axis. Typically, the parallax is reduced to the x-axis. The algorithm discussed in this thesis is designed to find disparity in any direction. The test image used was rectified and aligned so that the disparity was mainly in the x-direction. At the original image scale, some y-disparity was found, but it was only a single pixel in most cases. This one pixel variation could very well be due to the digital sampling resolution of the image scanner. When a diagonal edge is sampled to fit an orthogonal gray-scale array, it will have jags and variations from the true position along its path.

In future work, the two dimensional capabilities could be explored further. Theoretically, the algorithm should be able to operate on non-rectified images. The rectification could then be done on the network of matching points, instead of the original gray-scale where it introduces noise. Presently, the software will find y-disparity, but only the x-disparity in-

formation is used as offsets between resolutions. The computation time increase from adding y-disparity will only double the interpolation time at each resolution.

APPENDIX C. MATCHING TREES

A common topographical feature present in many aerial photographs is an ordinary tree. At typical resolutions trees are present as small, dark blobs in the original photo. At pseudo-hex resolution 2 the trees are still present, and at res 6 a few of the larger ones can be seen. In the algorithm discussed in this thesis, the trees give rise to edges which turn sharp corners as they encircle the dark blobs. The corners then are identified as nodes. Many of these nodes are then matched in the stereo pair. Figure 39 on page 87 shows the effect that matching trees has on the determination of the terrain. As seen from above, the matches will occur on the tops of the trees and distort the calculated elevation. If other matches are made nearby, then it may be possible to detect a tree. This effect must be considered when comparing true elevation to elevation calculated using an aerial, stereo-photo pair.

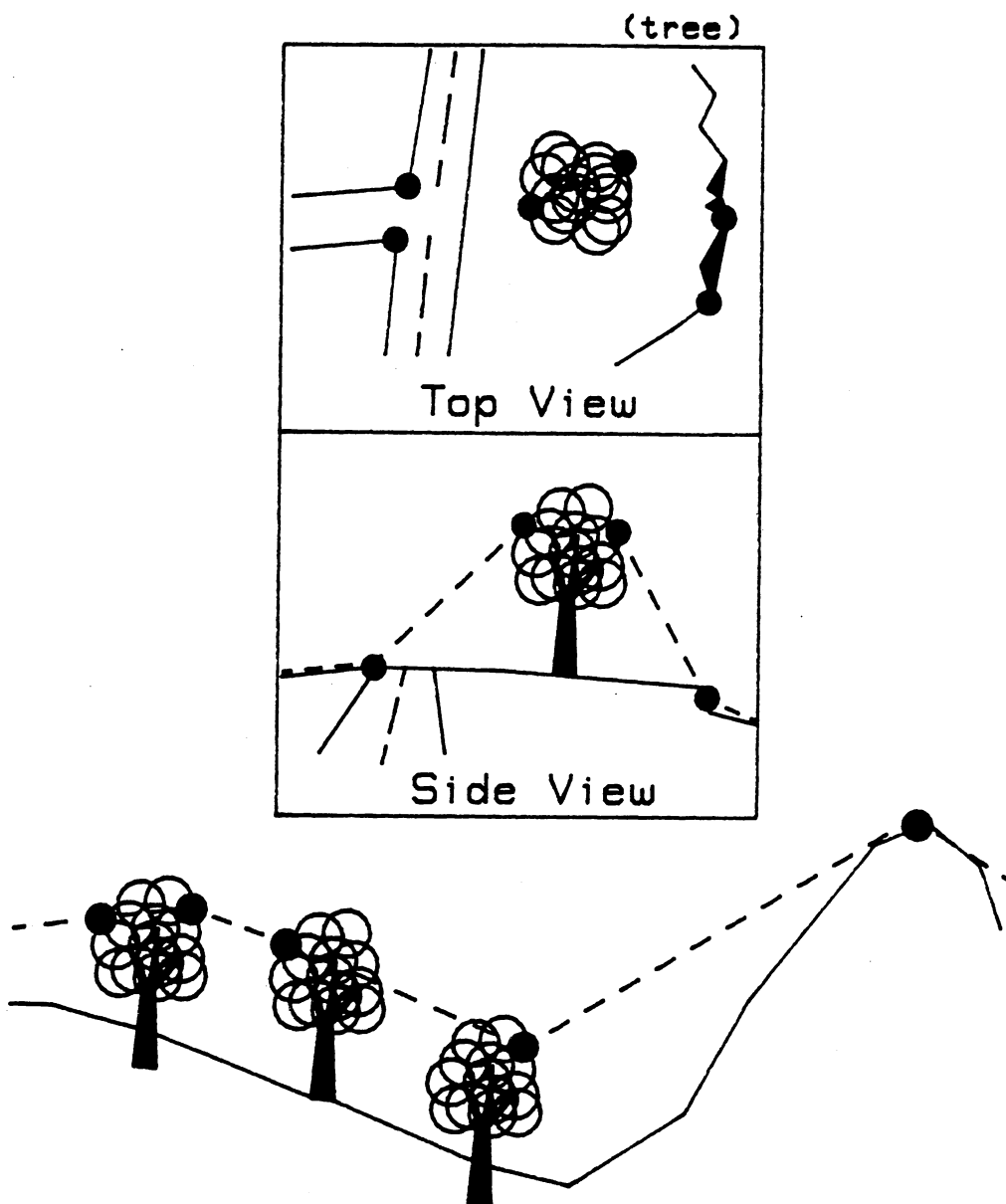


Figure 39. Effect of Matching Trees

APPENDIX D. GIPSY PROGRAMS

When I started this project several people had worked on it before me. I will mention only those authors of the sections of code that I use.

Brian Telfer wrote software to do the following:

- HEXEDG - takes a gray level image and determines the gray level pseudo hex, the edge vector magnitudes, the edge vector directions, and the associated edge vectors. Hexedg uses the following routines:
 - HEDCRU - a routine which pipelines the next three routines
 - HEXAG - converts orthogonal images to pseudo hex images at a specified resolution
 - EDGE - computes the edge vector magnitudes and directions, and eliminates those edges whose magnitudes do not exceed a specified threshold
 - ASSOC - performs coherence and association of edge vectors
- DSPHX - converts an arbitrary pseudo hex band to a displayable pseudo hex image

Wanglu Peng wrote Node and Matching programs, but unfortunately she misunderstood the concept of a node. I used her Node program as a skeleton for my own, but rewrote the node definitions. The matching program that I use is original.

- **NODES** - searches a direction band for nodes and creates a file with two bands, the first being a copy of the input direction band with node directions = direction + 64, and the second with nodes = 255. The second band is used to create a symbolic overlay for displaying nodes on gray level images.
- **NODESM** - same as NODES except edges with magnitudes above a threshold value are marked as nodes also.
- **MATCH** - takes as input two files processed by NODES(M). For each node in file 1, a window is searched for nodes in file 2. For each node found in the window, an edge correlation is performed for it and its six neighbors. The pixel with the high correlation above a threshold value is marked as a match. The output consists of four bands. The first two are X and Y match disparity, respectively. The third and fourth bands contain only matching pixels indicated by 254 for input images one and two, respectively. These bands are used as display overlays similar to the node band.
- **INTERP** - this program takes a pseudo-hex file as input and runs the expanding window interpolation algorithm on it. Zeros are considered holes. This program stores EACH point found and its absolute radial angle from the center of the window. It checks every point found for a pair that differ in angle by more than 120 degrees. It then performs the weighted average on every point. At present it only interpolates ONE band, the X band.
- **INTERP2** - this program is similar to INTERP, but it quantizes the radial angle into 12 segments, and only stores the closest point to the center of the window in every segment. This is the algorithm described in the text.
- **EXPAND** - this program simply expands a pseudo-hex array by 3 in both the x and y direction.
- **GRAYMAT** - this program performs the gray scale matching at res 1. It takes as input image one, image two, and the interpolated offsets from res 2. The output contains 3 bands. The first two are the x and y disparity. The third is proportional to the difference between the maximum and runner up correlations of each match. The program uses all 5 by 5 windows and the ping-pong method.

- ASCVEC - converts an edge field to a data file readable by VECPLOT
- ASCNOD - searches an edge direction band for nodes and puts them in a data file readable by VECPLOT with a magnitude and direction equal to 255.
- SIFDMP - this program dumps a sif file to an ascii format readable by the contour plotting program on the HP 550 system. It gives optional starting rows, columns, and step size.

APPENDIX E. PLOTTING

Interpreting a numerical array of edge directions that are labeled 1-12 is a very difficult task for the mind to perform. Shapes, trends, and contours are simply not observable. In order to observe and verify the effect of applying the edge vector, association, and node algorithms, it is necessary to have a graphic form of looking at vectors. This was the motivation behind the vector plotting program that I developed.

The Spatial Data Analysis (SDA) lab does have a plotter that is linked to the GIPSY system, but the resolution and controllability of the plotter are not sufficient for our needs. For example, the software does not support hexagonal arrays, which are the underlying principle of our work. Fortunately, the CAD lab, presently in 432 Whittemore, has a large Hewlett Packard drum plotter which can be directly controlled from HP Basic graphic commands, the only inconvenience being that the HP550 system and the SDA Vax are not linked directly. Vector data files must be downloaded from the Vax to the HP over the localnet.

E.1 DOWNLOADING TO THE HP

To download a file from Gipsy to the HP system, your workstation needs to be connected to the local network. Presently there is only one HP station in the CAD lab that has a network connection - the one nearest the console. To put the workstation in terminal mode do the following:

1. Turn power off
2. Turn power on
3. Type 2T (or 1T) to execute the HP terminal emulator
4. Press Enter at the prompts for Date, Time and Program

The screen should now have a 'terminal ready' message on it. The emulator has an extensive function menu. You can use it to configure the terminal to your needs.

- NOTE: the HP may not keep up during the downloading procedure at 9600 baud. Try a slower speed if you have problems.

Now you need to logon to the SDA Vax. The method used to do this seems to change quite a bit. Try the following:

- CALL 4000 and request: CSRN
- CONNECT to VTSDA

To prepare for downloading you need to use the function menu to do the following:

- PREFIX to your HP directory
- Set your DESTination file name (under the FILES menu) to be a fn.ASC file. BASIC will only read ascii files.
- Set the End Of Record identifier to Carriage Return, Line Feed by using the DSPLY FUNCTIONS mode and CNTL M and CNTL J
- Set the End Of File identifier to '\$' (or to your Vax prompt char)

The menu function RECORD will now copy all screen output to the DESTination file. So, to download a data file, do the following:

1. Enter the Vax TYPE filename command
2. IMMEDIATELY press the function key for RECORD

Your file is now being saved in the DESTination file. When the TYPE command is complete and your prompt appears, recording will cease and the DESTination file will be closed.

E.2 GETTING ON THE HP

The following is a procedure which can be used to access the Basic interpreter on the HP system:

1. Power on an HP graphics terminal
2. Type 1B to execute Basic 4.0

When Basic is ready, a message will appear at the bottom left of the screen. To prefix to a directory on the HP, type the following command:

- MSI "directory pathname"

For example, to prefix to the directory containing VECPLOT type:

- MSI "USERS/NADLER/BROOKSHIRE/BASIC"

The meaning of the other Basic commands such as LOAD, RUN, EDIT, CAT, etc., are straight forward. Most of them are in fact available from the function keys, along with a HELP facility. The command given in the HELP facility to use the printer is incomplete. To get a printout of a Basic program type the following:

- CREATE BDAT "/PRINTER/filename:REMOTE",10
- PRINTER IS "/PRINTER/filename:REMOTE"

At this point, all text screen output is redirected to the indicated printer file. To get a printout of a LOADED program type:

- LIST
- PRINTER IS 1

The last command returns the screen output to normal and closes the printer file. The printer file will

now be spooled on the line printer. A technique similar to the above will also direct graphic commands to the plotter.

E.3 HP GRAPHICS

This section describes briefly the graphic commands which are used in the vector plotting program VECPLOT. The HP graphic terminals have two screen memories, one for text and one for graphics. The keyboard has function keys which will toggle them on and off. The Basic commands ALPHA ON(OFF) and GRAPHICS ON(OFF) have the same effect as the function keys. As with the text screen, the graphics output can be directed to either the CRT or the PLOTTER. The way to send graphics to the plotter is identical as that for the printer, except you use the PLOTTER directory (see Vecplot for example). The following is a descriptive list of the Basic graphic commands:

- SHOW xmin,xmax,ymin,ymax - defines the coordinate mapping function from Basic to the graphic device. In other words, the left side of the screen will now be xmin, the bottom ymin, and so on.
- MOVE x,y - moves the graphic pointer (invisible) to an absolute x,y.
- DRAW x,y - traces a line from the graphics pointer to an absolute x,y. After the command, the pointer will be at x,y.

- RPLOT rx,ry - traces lines relative to the pointer. Several RPLOTS in succession will trace connected points, but will leave the pointer unchanged.
- PEN n - changes the drawing color to n on the CRT, or to location n on the plotter pen carousel.

Many of the above commands have options which I have not listed. One should consult the Basic Graphics manual for more information.

E.4 VECPLOT - THE VECTOR PLOTTER

The Basic program VECPLOT reads a file containing a list of coordinates, magnitudes, and directions. For vectors, magnitude is interpreted to be the length of the vector. All magnitudes are quantized to a scale of ten. Directions are interpreted to be multiples of 30 degrees, with 1 mapping to zero degrees (horizontal - to the right), 4 being 90 degrees (up), and so on. Conversion from Direction to Degrees, then, is as follows:

$$\text{DEG} = (\text{Direction} - 1) * 30$$

Points with Directions greater than 250 are considered to be NODEs (or matches). The magnitude of a node has no meaning at present. Nodes are indicated with

a small circle at the specified coordinate. Matches are indicated by a larger circle.

The following is an explanatory list of the prompts and requested data as they are queried by Vecplot:

- Read from a file (Y/N) - this option is given because Vecplot is capable of generating random data for debugging purposes.
- Does your data have tails (Y/N) - this option refers to vector tails. Originally our algorithm included a pass to 'bend' the edge vectors to make lines more continuous. This plotting option is implemented, but not used at present.
- Echo (Y/N) - will echo the file as it is read
- Title - a title box is included at the bottom of the plot which will hold two lines of text, the first will be the 'title' string.
- Subtitle - this will be the second line of text in the title box. The font size will be about half that of the 'title'.
- Screen size - this entry is used as the total square screen size, i.e. as xmax and ymax in the SHOW command.
- Unit Grid Size - this will be the distance between grid points. For example, with a screen size of 800 and a grid size of 20, a 40 by 40 array of vectors can be plotted.
- Hex grid (Y/N/Z) - the pseudo-hex grid shifts even numbered lines by a half a grid unit to the right and makes them one grid point shorter (see the example plots).

Y - plots a hex grid before plotting data

N - plots an orthogonal grid before plotting data

Z - uses a hex grid, but it is not plotted

- Magnitude plot (Y/N) - vectors may be plotted with relative magnitudes or with uniform length equal to one grid unit.

At this point the program will plot the titles, grid, and data on the terminal screen. Note that the graphic screen will be on, and the text screen will be off. The plot will appear exactly the same on paper as it is on the screen, except that the plotter has much better resolution. To continue the program, press any key.

- Hardcopy (Y/N) - to create a hardcopy the program redirects the graphic commands to a /PLOTTER file and redraws the plot. This takes quite a bit more time than plotting on the screen, so be patient. The program will keep you updated on its progress, and will beep when it has finished. It will then display a list of the pen numbers used for each part of the plot. Use these numbers to select pens for the plotter carousel.
- Replot (Y/N) - the program will loop back to the title prompt and keep the same titles and sizes as defaults. Answering no will exit the program.

E.5 CONPLOT - THE CONTOUR PLOTTER

The contour plotting program is also on the HP550 system in the CAD lab. It works very similarly to the vector plotting program. The program is designed to read an ascii file created by SIFDMP in GIPSY. The file must be downloaded to the HP. Presently the contour plotting algorithm assumes an orthogonal

grid, which has the usual connectivity problems (see CASE 4 of the contour subroutine). In addition to contours, this program will plot matches. The matches must be in a separate file created by ASCNOD. The prompts for CONPLOT are very similar to VECPLOT, and will only be described when they differ. They are as follows:

- Read data from file (Y) - read the contour file
- Echo (N) - echo the data as read
- Minimum Value (min) - allows the user to change the minimum data value that will create a contour line
- Maximum Value (max) - allows the user to set the maximum value contour
- Density (5) - allows the user to set the number of lines to draw between the Min/Max interval
- Extremes (0) - when set to 1, the contour routine will label local maxima and minima
- Do you want to plot matches (N) - do you have a match file to overlay? If so, prompts similar to VECPLOT let you read in the match file
- Title - allow you to enter a main title for the plot. An improvement over VECPLOT, automatically prints a subtitle which contains the minimum, maximum, difference, and density of the contour, plus the file name(s) and the array size.

At this point the plot will be displayed on the screen. When the plot has finished the terminal will beep. At this time press any key to continue:

- Hard Copy (N) - will spool the plot to the plotter exactly as it is plotted on the terminal

screen. This takes a while; the program will inform you of its progress on the screen. When it has finished spooling the file, it will sound off! The program will give a list of pen numbers used for the plot, so that you may set up the carousel.

- Plot Again (Y) - loops back and allows you to modify the plot (density, matches, title, etc.). Answering 'N' will exit the program.

E.6 USING THE HP PLOTTER

After the plotter file is closed a spooler message will appear on the HP console indicating this fact, and that the plotter requires setup (if it has not already been done). To set up the plotter do the following:

1. Set up the plotter pen carousel with the corresponding pens that you want to use for each part of the plot
2. Insert and line up paper (A - sideways, B - longways)
3. Press Chart Hold
4. Press Remote
5. Type SPool COntinue 7,8 at the console

The plotter should now begin drawing the frame and title box with the pen number indicated for 'frame' on the pen list. Beware that plotting takes a long time. The plots in this report took about 7-10 minutes to plot.

When the plot is complete:

1. Press View
2. Press Chart Unload
3. Remove the paper.

BIBLIOGRAPHY

1. Morris M. Thompson, ed., Manual of Photogrammetry: 3rd Edition. Menasha, Wisconsin: George Banta Co., 1966.
2. Brian A. Telfer, "Creating Oriented Edge Graphs in Hexagonal Raster." Independent Study Report, Virginia Tech, 1985.
3. K. Bowker, "Edge Vector Image Analysis." Second International Joint Conference on Pattern Recognition, IEEE, Copenhagen, 1974.
4. M. Nadler, "Automated Aerial Stereophotogrammetry." Research Proposal, Virginia Tech, 1984.
5. M. Nadler and G. Signor, "Automated Stereophotogrammetry Using Techniques of Structural Pattern Recognition." Digital Image Processing and Analysis, ed. Simon and Rosenfield.
6. M. J. Moroney, Facts from Figures, Penguin Books, pp. 312-315, 1965.
7. A. Goshtasby, "Piecewise Linear Mapping Functions for Image Registration." Pattern Recognition 19, no. 6, pp. 459-466, 1986.
8. K. E. Price, "Hierarchical Matching Using Relaxation." Computer Vision, Graphics, and Image Processing 34, pp. 66-75, 1986.
9. H. Moravec, "Rover Visual Obstacle Avoidance." Proceedings 7-IJCAI, Vancouver, B. C., Canada, August 1981, pp. 785-790.
10. H. Baker and T. Binford, "Depth from Edge and Intensity Based Stereo, Proceedings 7-IJCAI, Vancouver, B. C., Canada, August 1981, pp. 631-636.
11. J. Barnard and W. Thompson, "Disparity Analysis of Images." IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-2, pp. 333-340, 1980.
12. C. Wang, H. Sun, S. Yada, and A. Rosenfeld, "Some Experiments in Relaxation Image Matching Using Corner

Features." Pattern Recognition **16**, no. 2, pp. 167-182, 1983.

13. G. Medioni and R. Nevatia, "Segment-Based Stereo Matching." CVGIP **31**, pp. 2-18, 1985.
14. R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis, New York: John Wiley and Sons, 1973.

**The vita has been removed from
the scanned document**