"Reconstruction of 3D Surfaces from Single Image Random Dot Stereograms"

4th Year Project Report

John W.M. Wilson

May 31, 1995

Departments of Artificial Intelligence and Computer Science

Abstract

This dissertation documents a system which reconstructs three-dimensional surfaces from single image random dot stereograms. It describes how Eric Grimson's algorithm for reconstructing surfaces from double image random dot stereograms was re-implemented and extended to perform the same operation with single image random dot stereograms. The theory behind stereograms is explained. The design and implementation of the extensions is described in detail and examples given of the finished system in use. An account is given of the experimentation carried out with the system and the results are explained. Finally, possible developments of the system are discussed; these include the introduction of colour to input stereograms and other pointers for future research.

Acknowledgements

My thanks go to my supervisor Gillian Hayes for her invaluable suggestions and constructive criticism on both the program and on this dissertation. Thanks also to Andrew Fitzgibbon for his expert advice with *MATLAB* and to Bob Fisher for his helpful comments and without whose proposal there would have been no project.

Contents

Chapter 1

Introduction

This project deals with Single Image Random Dot Stereograms (*SIRDS*). *SIRDS* are those pictures apparently composed of a meaningless jumble of dots which one sees in poster shops and books such as "Magic Eye" [17]. These pictures, which now appear on everything from greetings cards to drinks cans have become very popular over the past few years. It is this renewed interest in *SIRDS* which, in part, has inspired this project.

The concept behind *SIRDS* is that they contain a "hidden" image which most people can see after a few minutes of staring at the picture with their eyes defocused. These pictures have a novelty value due not only to the fact that they contain a cunningly hidden image but also because the resulting image has the illusion of depth and thus appears to be three-dimensional. An example of a *SIRDS* (depicting a goblet) is shown in Figure 1.1.

Such random dot images were originally developed, not for recreational purposes but by researchers attempting to learn more about how the human visual system works. They wanted to see if it was possible for people to see three-dimensional images without the aid of the visual cues that we use in everyday life; recognisable objects and perspective for example.

1.1 Motivation

Before *SIRDS* came into existence (around 1990), there existed the similar concept of *double image* random dot stereograms, an example of which (depicting a glass) is pictured in Figure 1.2. These consisted of two separate random dot images (one for each eye) and unlike *SIRDS*, had to be viewed using a special device called a *stereoscope* (it is possible but much harder to fuse them with the naked eye) but gave an identical effect – disclosing a hidden object with apparent depth.

Just before the advent of *SIRDS*, Eric Grimson embarked on research into reconstructing three-dimensional surfaces from depth information obtained from double image random dot stereograms. The product of this research, a full account of which can be found in [8], was a

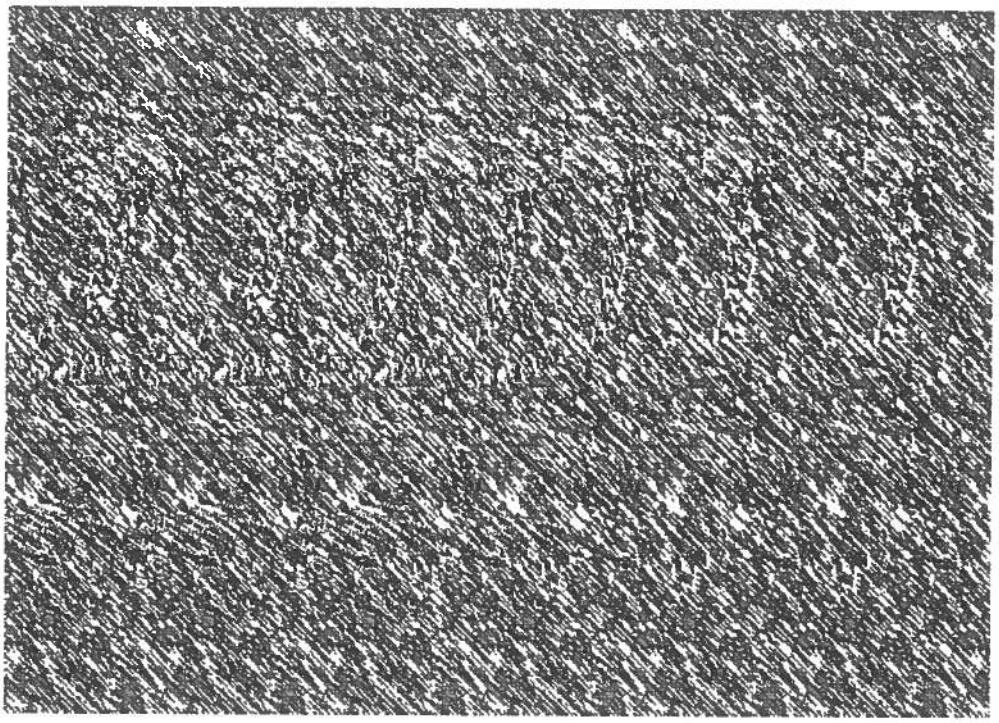


Figure 1.1: **Single Image Random Dot Stereogram.** This is a stereogram of a goblet, by Peter Chang and Gareth Richards [20].

process which worked by deriving information about the hidden object(s) by looking for places in the image corresponding to changes in properties of the objects in the scene – for example irradiance changes which may correspond to the crossing of an object's edge.

Grimson's original motivation for developing his algorithm had two aspects. The first was to advance the understanding of the computational processes involved in the human visual system. The second was to illustrate the computational paradigm in which these processes may be represented. In particular he wished to address the question: *"how does three-dimensional reconstruction take place in the human visual system?"*.

The motivation for the undertaking of *this* project, partly sparked by the current popularity of *SIRDS*, is to establish whether it is possible to extend Grimson's algorithm to work on *SIRDS*.

1.2 Project Goals

The principal aim of this project, as set out in the original project proposal by Bob Fisher, is to "develop continuous spline surface descriptions of shapes embedded in Single Image Random Dot Stereograms".

Grimson's algorithm does exactly this, except with *double* image random dot stereograms. Therefore, the project will essentially involve the re-implementation of Grimson's algorithm, adapting it as necessary, such that it works on *single* image stereograms. Throughout this report,

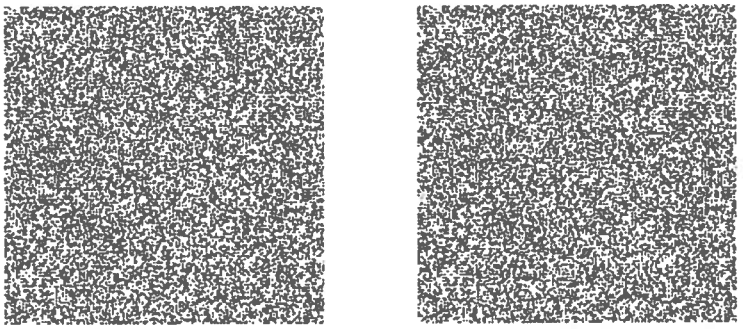


Figure 1.2: **Double Image Random Dot Stereogram**

therefore, "stereogram" should be taken to refer to a *single image random dot stereogram.*

Grimson's algorithm can be broken down into four main stages:

1. extract zero-crossings from convolved (smoothed) versions of the scene (at 4 scales),
2. establish correspondence between features in a coarse-to-fine sequence,
3. estimate the three-dimensional position of the matched image features,
4. construct a smooth surface that interpolates the estimated depth points.

The core of the process is formed by Marr and Poggio's "Stereo Algorithm" [15] (stages 1 and 2 above), which uses zero-crossings extracted from the image to construct a depth map of the scene (stage 3), and thus to recreate the surface embedded in the stereogram.

1.3 Principal Results

The finished system implements an adaptation of the Marr-Poggio stereo algorithm. The main change here is that instead of extracting zero-crossings at four scales, they were only extracted at one scale, as experimentation showed this to be adequate. Grimson's surface interpolation algorithm was not implemented; instead *MATLAB* was used for this purpose. The system takes as input a stereogram in *HIPS* format. In contrast to Grimson's method, described in Section 1.2, the system follows the following steps:

1. extract zero-crossings from *one* convolved version of the scene,
2. establish correspondence between features by matching zero-crossings,
3. three-dimensional position is derived directly from matched image features,
4. construct a smooth surface that approximates the depth points.

The above scheme was implemented and the finished system successfully extracts a set of three-dimensional points from the scene and after noise removal, fits a surface to these points. The system has been shown to work on a variety of *SIRDS*.

Experimentation with the completed system showed that it is more difficult to obtain a complete depth description from a *SIRDS* than from a double image stereogram. This is due to the fact that the images seen by each eye are merged into one picture as opposed to being in two independent pictures. As we shall see later, this circumstance introduces gaps in the data extracted; we explain this problem fully and in Chapter 7, a solution is proposed.

1.4 Overview

The remainder of this report describes the process of adaptation and re-implementation of Grimson's algorithm. In Chapter 2, we examine Grimson's algorithm in more detail and explore the possibilities for extending it to work on single image random dot stereograms. We also give a short explanation of how *SIRDS* work and give some of the theory and history behind them. Chapter 3 covers the research and design stages involved in the building of the system. The implementation of the system is described in Chapter 4. Here we look at how the initial design was altered during coding as the idiosyncrasies of *SIRDS* became apparent and the resulting implemented strategies. Chapter 5 documents the testing of the various modules of the system and Chapter 6 describes experimentation carried out using the final system. Finally, in Chapter 7, a critical look is taken at the project as a whole and the possibilities for future extensions and improvements are explored.

Chapter 2

Background

While this second chapter would normally be devoted to a review of the literature relevant to the subject in hand, since the project involves the re-implementation of a specific algorithm (*i.e.* Grimson's), such a review would not be particularly meaningful. However, we shall aim to expose alternatives to the Marr-Poggio stereo algorithm, favoured by Grimson. We shall also give a detailed explanation of the main features of Grimson's algorithm and an explanation of how *SIRDS* work.

2.1 Stereograms and Stereopsis

Someone with normal binocular vision has a slightly different view of the world in each eye due to the fact that their eyes are separated by around 3 inches. When both eyes look at a distant object, two processes cooperate to bring two images into the brain.

The first of these processes ensures that the images are on the centres of the retinae by directing the eyes at the object. For an object directly in front of the face, assuming the eyes start off pointing in a direction perpendicular to the face, the eyes are turned inwards towards it. This process is known as *convergence*. The second process in concerned with focusing the individual images.

The brain must then fuse the different images together to produce the perception of depth; this fusion is known as *stereopsis*. Julesz [11] showed that the sense of depth could arise solely from stereopsis – *i.e.* without any visual cues, such as recognisable objects with perspective and colour – using separated patterns of random dots (double image random dot stereograms). Although Julesz is credited as being the first to generate random dot stereograms, it is believed that they were discovered entirely by accident when a Spitfire flying over Cologne in 1940 took some pictures of the river Rhine which was, at the time covered in ice. Because the ice was floating downriver and the pictures were taken at slightly different times, the ice patterns were slightly different in the two stereo pictures taken. The result of this was that when the two

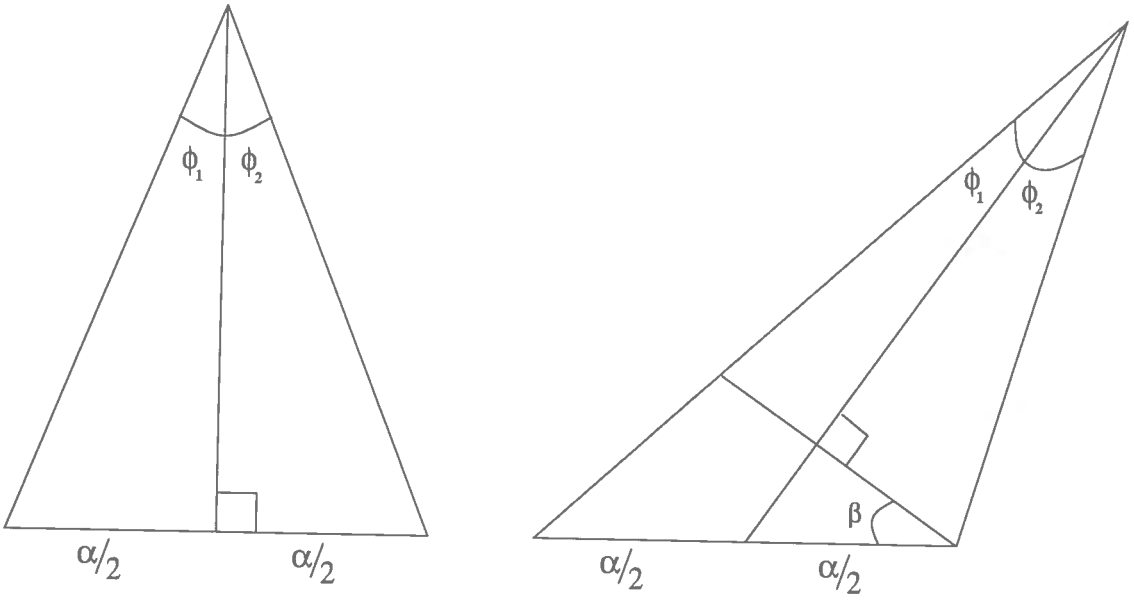


Figure 2.1: **The Geometry of Disparity**. The left diagram shows the simplest case of an object centred along the axis between the eyes; disparity is given by $\phi = \phi_1 + \phi_2$ and the interocular separation by α. The right diagram shows the general case of an object lying off axis; disparity is again given by $\phi = \phi_1 + \phi_2$ and the off axis angle is denoted by β.

pictures were stereoscopically fused there appeared to be a deep valley in the middle of the river.

The next vital development came with the discovery that a separate pair of images is not necessary for stereopsis. Tyler and Clarke [24] made the first single image random dot stereograms (*SIRDS*). They relied on the "wallpaper effect", discovered by Sir David Brewster [1844] who noticed a 3D effect from staring cross-eyed at wallpaper. This effect is due to the brain matching up different units of the repeating block pattern in the wallpaper. This "mismatch" of the two images being transmitted to the brain translated into the perception of elements of the pattern on the wallpaper at a closer distance than they really were. The important detail was that the apparent depth was controlled and determined by the period of repetition of the wallpaper.

The key to decoding stereograms is the depth information which is encoded in the picture itself. Here, depth is taken to mean the *subjective* distance, as perceived by a viewer. Humans use stereo vision to obtain depth information, which is computed using *disparity*. Disparity is the angular difference in position of the image element in the two eyes (see Figure 2.1). As suggested by Marr [13], to obtain depth information computationally, we can locate an element in one image, locate the same element in the other image and measure the difference in position between the two.

The main problem encountered when using Marr's method is called the "correspondence problem". This is essentially the problem of identifying which element in one image matches with which object in the other image. This problem is particularly relevant when dealing with

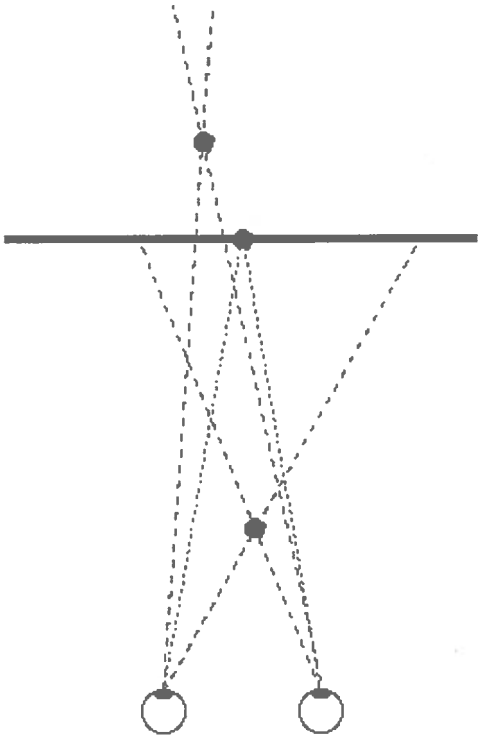


Figure 2.2: **Two Stereogram Viewing Methods.** For wide-eyed viewing the convergence point is behind the stereogram and for cross-eyed viewing, it is in front [3]. When the stereogram is viewed normally, the convergence point is on the stereogram but focus is at real depth.

random dot stereogram images, as, since they are entirely composed of dots, each element could potentially match with many in the other image. In other circumstances, this problem could conceivably be lessened by performing object recognition before matching. In the case of stereograms, however, this is not feasible as there is no monocular structure in the image at object level.

A solution to this problem which *does* work with stereograms is to consider changes in irradiance. A one-dimensional change in irradiance gives rise to a high first derivative and a zero-crossing in the second derivative. Thus, we can reduce the problem to detecting zero-crossings, which can, as is described later, be matched, allowing their disparity to be computed.

Grimson implemented a system which took, as input, two bi-level images (a left and a right image). By bi-level we mean an image composed of pixels of two grey levels – typically black and white. His system used Marr and Poggio's Stereo Algorithm (described in detail later) to obtain depth information from the images which was then used as the basis for approximating or interpolating a continuous spline representation of the surface.

There are two methods for viewing a stereogram: *wide-eyed* and *cross-eyed*, as depicted in Figure 2.2, the only difference between the two methods being the point of intersection of the two hypothetical lines drawn from the eyes – *i.e.* the point of convergence. For wide-eyed viewing,

the convergence point is behind the stereogram and for cross-eyed viewing, it is in front. So, when the eyes defocus to the correct extent, they are looking at different parts of the stereogram; the brain does not know this and thinks that a real object is being seen – interpreting the different images seen by each eye as depth information in the scene.

This phenomenon is demonstrated by Figure 2.3 which shows a stylised representation of the workings of a stereogram. The diagram depicts a stereogram with three hidden images (of the letters E, F and G) at different apparent depths. The distances shown in the diagram are typical of stereogram pictures. The two E's on the stereogram are quite close together – this will result in their perceived image appearing closer to the viewer than the perceived image of the G's for example, which are further apart.

The exact disparity of a point in a stereogram is difficult to compute as we do not have available to us measures of camera angles and distances. Hence we shall use a qualitative approximation to disparity – that of horizontal displacement of any two *pixels*. So, for example, the disparity of the E in Figure 2.3 is the distance between its two occurrences in the stereogram picture.

We also make the assumption [8] that the objects being viewed do not lie too far off axis. In this case, the off axis angle (β in the left-hand diagram of Figure 2.1) will be negligible. This is a reasonable assumption to make because if an object lies too far off axis, it will only be seen by one eye.

2.2 A History of Stereograms

Although most people will only have become aware of stereograms very recently, they have been in existence for many years [3].

The creation of stereograms stems from mankind's desire to create more lifelike pictures on a flat surface. The first event in the series of developments leading up to the creation of *SIRDS* occurred in 1838 when Sir Charles Wheatstone invented the stereoscope. The stereoscope was a piece of binocular apparatus through which one could view a seemingly three-dimensional drawing. This illusion was made possible by the fact that the viewer's left and right eyes would see a picture drawn from a slightly different angle (a so-called *stereo pair* of images), thus giving the false impression of depth.

Shortly after this followed the invention of photography in 1839, enabling scenes viewed by the stereoscope to exhibit increased realism and detail. Random dot stereograms were finally created in 1960 when Bela Julesz [11], researching into the psychology of vision, discovered that the human brain could derive depth information from a *pair* of random dot stereograms.

This eventually led to the creation of the first *SIRDS* in 1990, when Tyler and Clarke [24] realised that a separated pair of images was not necessary for stereopsis. Apart from the fact that we are now dealing with only one image, the crucial difference between double and single image

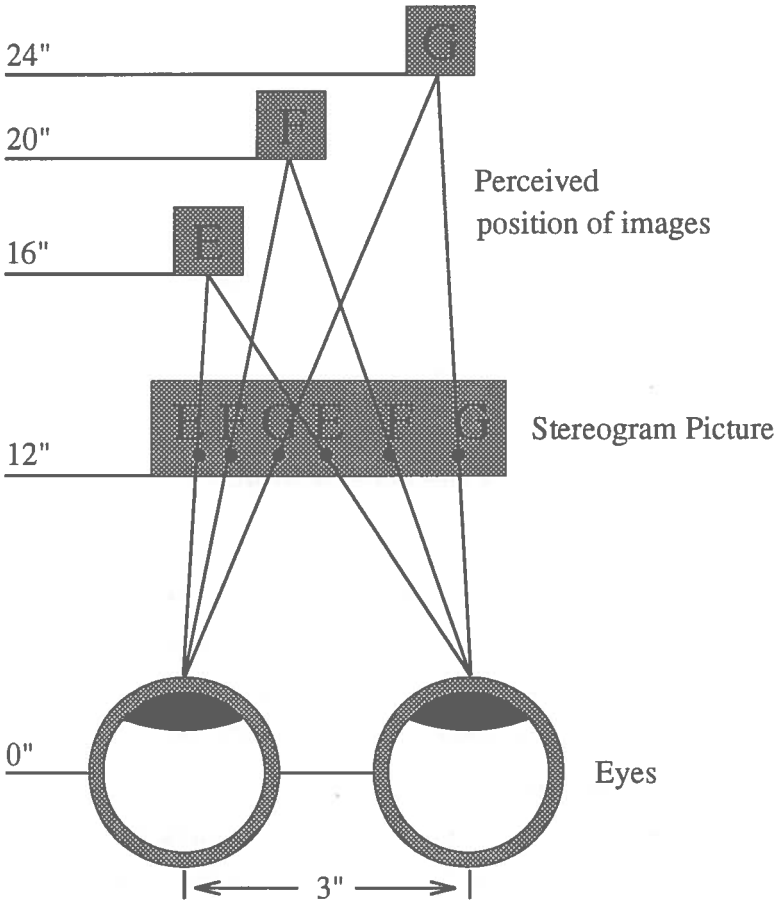


Figure 2.3: **How a stereogram picture is viewed** [12].

Chapter 4

Implementation

As described in the last chapter (Section 3.2.1), there are separate modules for image processing (LoG.c), operations on files (fileops.c), detection of zero-crossings (zero.c), matching (matcher.c) and for the main function (main.c). In addition to these, there are two header files: stereo.h and kernels.h, containing shared declarations and definitions of the convolution kernels used respectively. Here we consider the actual implementation details of each of these modules.

4.1 Convolution

As indicated earlier, a one-dimensional change in pixel irradiance gives a zero-crossing in the second derivative. One way of approximating a second derivative measurement on the image is to apply a Laplacian kernel to it – a commonly used discrete approximation to the Laplacian filter is shown in Figure 4.1. Unfortunately, the Laplacian filter is very sensitive to high frequency noise, so to alleviate this problem, we first smooth the image using a Gaussian filter. Convolution processes are very computationally expensive for any reasonably sized image, so we combine the Gaussian and Laplacian and use the resultant hybrid filter to convolve the image. We can thus smooth the image and calculate the second derivative in one step, requiring far fewer arithmetic operations. Formula 4.1 gives the form [7] of the two-dimensional *LoG* function centred on zero and with Gaussian standard deviation σ, from which we can derive a *LoG* kernel.

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.1}$$

The current implementation uses a *LoG* kernel of size 9×9 with Gaussian $\sigma = 1.4$ (as pictured in Figure 3.1). The convolution stage of the algorithm had been implemented by the end of the first term, however, it had been tested only on grey-scale images (Figure 3.2(a), for example). Several stereograms have now been converted into *HIPS* format; a convolved version of the goblet stereogram can be seen in Figure 4.2.

Had it been decided to implement the multi-level matching procedure as described by Grim-

0	1	0
1	-4	1
0	1	0

Figure 4.1: **Laplacian Kernel.** One commonly used discrete approximation to the Laplacian filter.

son, it would have been necessary to produce a convolution kernel for each of the four stages of the process. Haralick [9] gives some guidelines for sizing of convolution kernels. The variable w is related to the constant σ of formula 4.1 by the following relation, from [8]:

$$\sigma = \frac{w}{2\sqrt{2}} \tag{4.2}$$

The central region of the *LoG* kernel is a disk of radius $\sqrt{2}\sigma$. The domain of the *LoG* kernel must be at least as large as a disk of radius $3\sqrt{2}\sigma$ (and hence of diameter $6\sqrt{2}\sigma$).

In practice, as we are only looking for a zero-crossing and do not, for example, wish to examine the exact values on either side of it, the *LoG* kernel values are multiplied by some constant and the resulting values quantised to integers. Care must be taken during quantisation so that the sum of the positive entries equals the absolute value of the sum of the negative entries. To ensure this, we can use:

$$new\ kernel\ value = truncate[LoG(x, y)] \tag{4.3}$$

The kernel size can be calculated as $6\sqrt{2}\sigma$ and the sizes which would have been used are as follows; the values of w are as derived by Grimson from the data of Wilson and Bergen [26].

w	σ	$6\sqrt{2}\sigma$	Kernel Size (in pixels)
4	1.4	11.9	13×13
9	3.2	27.2	29×29
18	6.7	56.9	57×57
36	12.7	107.8	109×109

In the above table, increasing values of w correspond to increasing coarseness of the resulting filter. The final kernel size was chosen to be the odd floor of $6\sqrt{2}\sigma$ as this allows the kernel to be symmetrical about zero (the *LoG* graph is centred on zero). These are ideal sizes, however, and would probably need to be reduced to allow a reasonable runtime. A program was written (kernel_gen.c) to derive values for the different sizes of kernel. Although this is capable of generating kernels quite effectively, elusive problems were encountered in the implementation of quantisation – even though the values were quantised using the method suggested above, the values of the mask would not sum to zero.

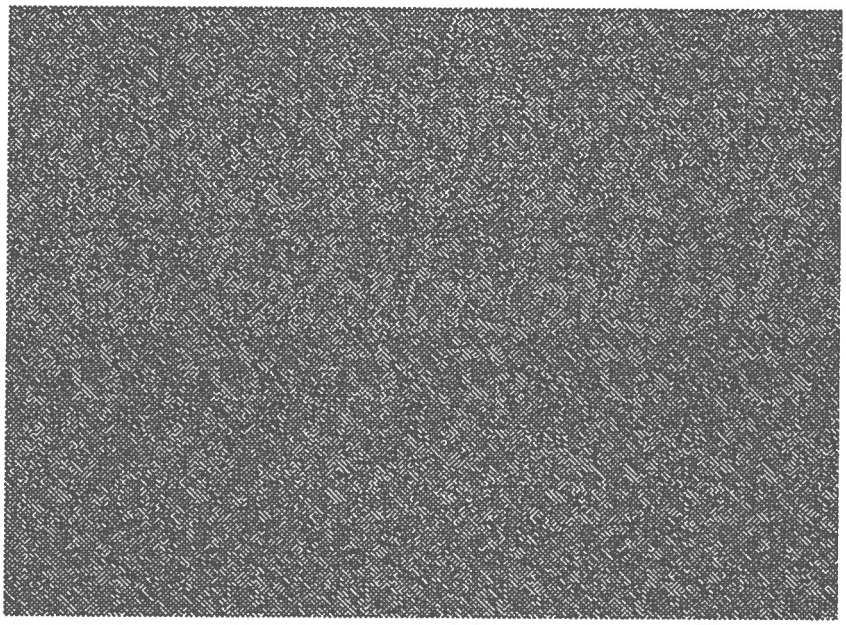


Figure 4.2: **Example of *LoG* Convolution.** Convolved version of the goblet stereogram.

4.2 Detection and Description of the Zero-Crossings

Next, we must search for zero-crossings in the convolved image. This is done by scanning each line of the image horizontally for adjacent pixels whose intensity values are of opposing sign, or for three adjacent elements where the middle pixel has intensity value 0 and the pixels on either side are of opposing sign. When such a combination of pixels is found, their location is recorded, together with their sign and magnitude.

Sign is an indication of whether the intensity values being considered exhibit a change from positive to negative or vice versa. Grimson [8] suggests that the two-dimensional orientation of a zero-crossing be recorded. However, since, as we shall see in the next section, matching is reduced to a one-dimensional problem, orientation may be ignored and instead, a simpler measure – magnitude is recorded. Magnitude is simply a measure of the degree of change of intensity across the zero-crossing. The zero-crossings obtained from the convolved image of the goblet are shown in Figure 4.3. The reader may also like to note that the goblet can still be seen in the convolution and the zero-crossings of the stereogram.

4.3 Matching Strategy

In Grimson's implementation, the left image of the stereogram pair was matched with the right. In the case of single image stereograms, we must match the input image with itself. This adds the complication that a zero-crossing could easily find a match with itself, however, as we shall see below, this behaviour has been suppressed.

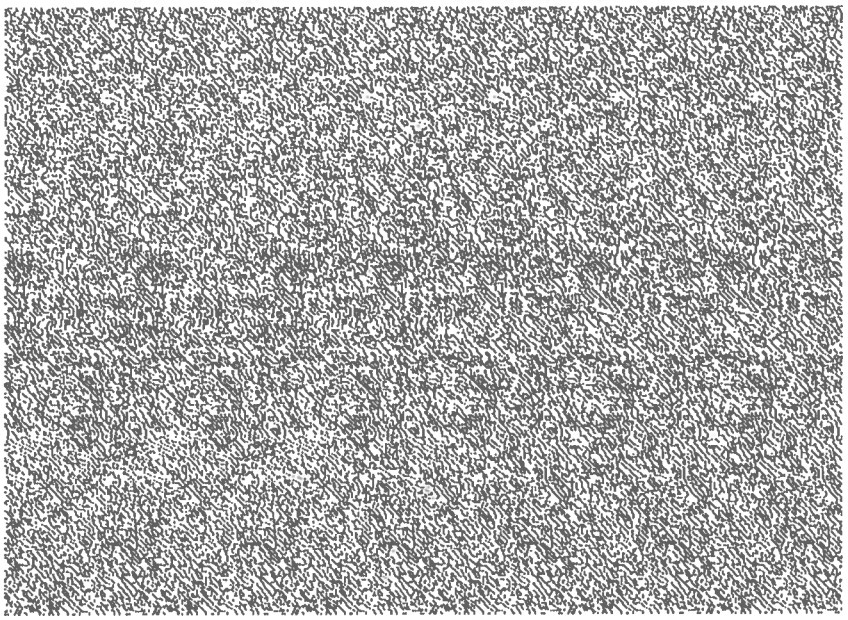


Figure 4.3: **Example of Zero-Crossings.** Zero-crossings detected in the goblet stereogram.

Although we are matching two identical images, we shall refer to them as the right and left images. Essentially what the matching procedure entails is: for any zero-crossing at position (x, y) in the left image we must try to find a match at $(x + d, y)$ in the right image, where d is the disparity associated with the match. The matching criteria are as follows; for a zero-crossing a to match with a zero-crossing b, a and b must satisfy:

1. a and b are not the same zero-crossing (*i.e.* $(x_a, y_a) \neq (x_b, y_b)$),

2. a and b occupy the same row in the image (*i.e.* $x_a = x_b$),

3. the *signs* of a and b are identical,

4. the *magnitudes* of a and b are equal to within a certain threshold.

In Grimson's implementation, matching proceeds as follows: for each zero-crossing in a given row r in the right image, try to match it with every zero-crossing in the corresponding row in the left image within a $\pm w$ radius, where $w \simeq 3\sigma$ [9]. Thus, for the current implementation, $w \simeq 4$. It is unclear how Grimson used this figure, but in this case, where the majority of disparity values are in the region of 40 — 110 pixels, this is clearly inappropriate. Therefore, a larger value is used; this must be set fairly high *i.e.* around 100, as disparities vary widely between stereograms. This means that we will now search for matches within a ± 100 pixel radius of each zero-crossing.

An added heuristic proposed is to disregard any matches found whose disparity is less than 30. This is an arbitrary figure but experimentation has shown that disparities of much less than

30 do not make for good stereograms nor easy viewing. All of the stereograms used for testing and experimentation purposes were therefore of disparity greater than 30.

The matching module of the program also houses a function which is used later to compile statistics on the performance of the matcher.

4.4 Computing Disparity

This function is also fulfilled by the matching module. Disparity is computed by calculating the horizontal displacement between a zero-crossing and its match. This figure can be positive – denoting *convergent* disparity *i.e.* the match is to the right of the zero-crossing, or negative – denoting *divergent* disparity *i.e.* the match is to the left.

For the purposes of plotting the positions of the matches found and eventually fitting a surface to them, it is necessary to convert disparity into a representation of depth. As depth is proportional to disparity, we can simply take the absolute value of the disparity of a point to be its depth.

Thus, the output of the matching module is a set of three-dimensional coordinates of the form (x, y, d), which denotes a match at (x, y) with associated depth d. We can then plot this set of points using *MATLAB* as in Figure 4.4 which shows the matches extracted from the goblet stereogram (Figure 1.1) and those extracted from a stereogram of a bowl (Figure A.2).

In Figure 4.4 and throughout this report, in cases where this data is plotted in three dimensions, the vertical axis denotes the depth (*i.e.* absolute disparity) of a match. Depth increases from the top of this axis and so the points are plotted from the perspective of someone looking down on them from above – the deepest points are at the bottom of the plot. The other two axes in such a plot denote position of the match in the image in numbers of pixels. In the cases where the data is plotted in *two* dimensions, the axes denote position of the match in the image in numbers of pixels – the depth of each match is not represented.

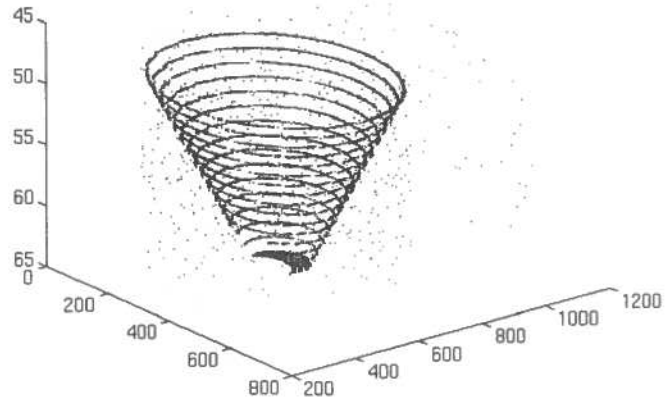
4.5 File Operations

The code for reading and writing *HIPS* files and for converting between arrays of characters and integers has been imported from [5].

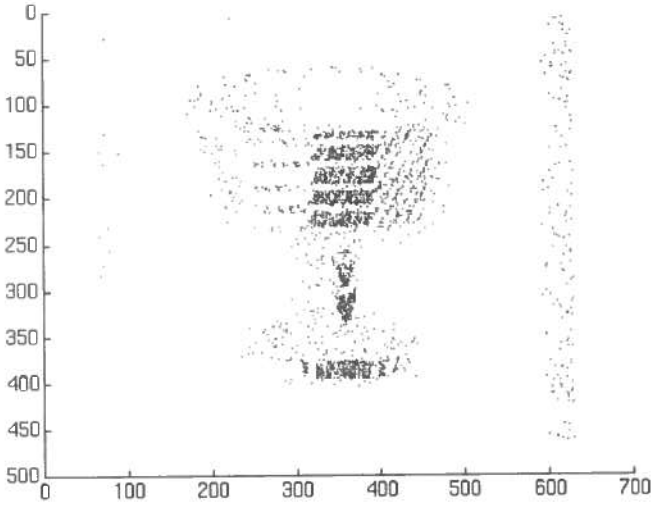
4.6 Interpolation/Approximation of a 3–D Surface

When Grimson constructs his surface, he fits it to the hidden object and down across the backdrop of the image. Such an approach is inappropriate in the case of single image stereograms for the following reason.

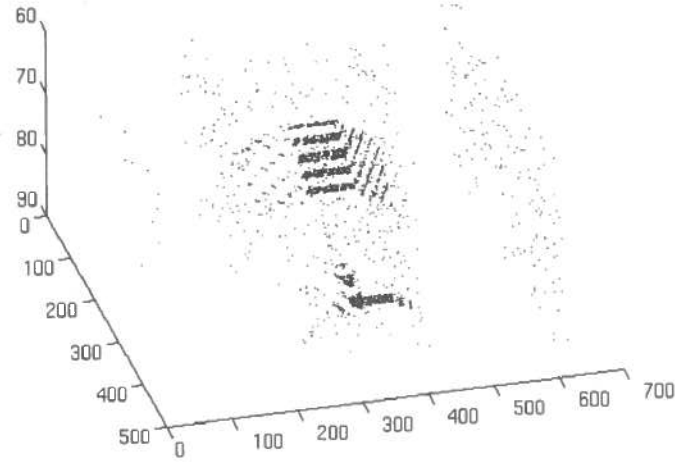
The backdrop of a double image stereogram is intrinsically more complex than that of a single



(a)



(b)



(c)

Figure 4.4: **Examples of Matches.** Figure (a) shows the matches (of negative disparity only) extracted from the bowl stereogram. Figures (b) and (c) show two different views of the matches (of negative disparity only) from the goblet stereogram. In Figures (a) and (c), the vertical axis denotes the depth (absolute value of disparity) of a point and the other two denote position in the image in numbers of pixels. In Figure (b), the axes denote position in the image in numbers of pixels.

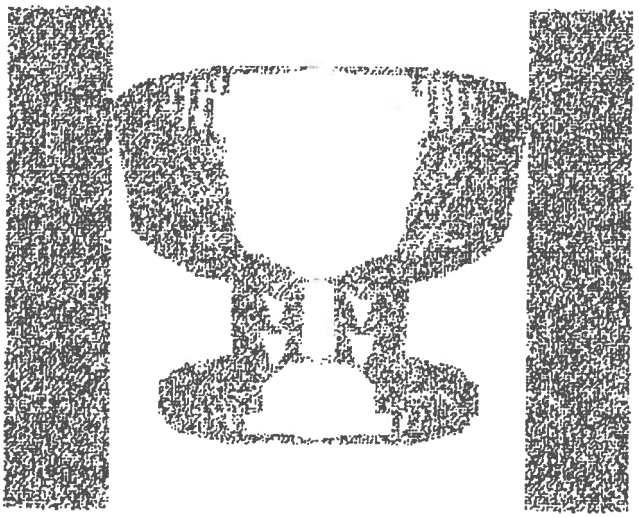


Figure 4.5: **Matches forming the backdrop of the goblet stereogram.**

image stereogram. This is because in a single image stereogram, both the left and right parts of the hidden object (we shall henceforth refer to these parts as *fields*) reside in the same image. If the perceived object has constant disparity, then these fields will be identical and horizontally displaced by this disparity. This is a simplified case, however, as objects in stereograms are more often composed of more than one disparity – for example curved objects, where disparity may vary across the surface of the object in order to create the impression of some parts of the object being further away than others. In such cases, each point in the fields will be offset by a different amount which has the effect of making one or both of the fields appear "stretched".

As disparity is usually less than the width of the object itself, the two fields of the object are actually partially intermeshed to some extent. This results in a shadow incongruent with the object itself; this effect can be seen in the case of the goblet, in Figure 4.5. Another complication which becomes apparent in this figure is that unlike in Grimson's case, where matches are detected evenly over the backdrop, here we have two bands of concentrated matches, one down each side of the image (the right hand one associated with positive disparity and the left with negative). Such bands of matches are a feature inherent to all the stereograms tested.

The features described above, although not usually visible to a viewer of the stereogram (due to their high disparity), make fitting a surface difficult as they do not constitute a regular backdrop. One solution to this problem would be to eliminate the banding at the sides somehow – probably easiest done by disregarding any matches less than some arbitrary distance from the edges of the stereogram. This should not affect the important zero-crossings in the picture as these are invariably confined to a fairly central region, to promote ease of viewing. This would not be a foolproof measure as the width of banding varies among instances of stereograms but would be appropriate in most cases. Having done this, however, we are still left with the problem

of the shadowing behind the object itself.

A broader solution would therefore be to suppress all matches deeper (of greater disparity) than those constituting the object and thus eliminate all shadowing *and* banding; both of which always appear entirely to the rear of the object. This seems to be an ideal solution, however in practice we find that, as the depths of objects vary between stereograms, so do the depths of their shadows. This means that it is not possible to set a constant depth which guarantees precise separation of an object from its background. So the problem remains; although it may be solvable by analytical techniques outwith the scope of this project. Thus the only solution available to us is to use the following crude and time-consuming method:

- run the program on the stereogram using a high upper bound on disparity, say 100,
- manually examine the plotted data, observing the construction of the image and choose an appropriate cut-off depth,
- run the program again using this figure as the upper bound, to obtain a set of matches which excludes any shadowing/banding visible in the original.

Having eliminated these troublesome features, it will be possible to fit a surface to the data, however, as all matches have necessarily been removed from the backdrop, the surface will cover only the extent of the matches found on the object. Before we do this, we must tackle the problem of noise occurring in the form of isolated spurious matches or groups of matches.

4.7 Noise Removal

Figure 4.7 shows the still noisy set of matches obtained from the stepped stereogram shown in Figure 4.6. Our aim is to reduce or if possible, exclude all noise from such output sets of three-dimensional match locations. One could attempt to fit a surface over the noisy points by lowering a "penetrable membrane" over the points from above, where the membrane would yield to single points or small groups of noise, letting them pass through it and eventually settle on the comparatively dense population of points constituting the object, thus forming a surface. The only drawback with this method is that it takes hours, on even the fastest machines.

A more realistic approach is to scan the data, discarding a point if it has any less than a certain number of neighbouring points within a certain radius of itself. Optimum values for the number of neighbours and radius constants would have to be discovered by experimentation. This method would probably not be overly reliable as it would miss any sizeable knots of noise present.

Another potential approach would be to smooth the data by averaging each point with several of its nearest neighbours. This method would perhaps fall down if we had noise points at any significant distance from the true data.

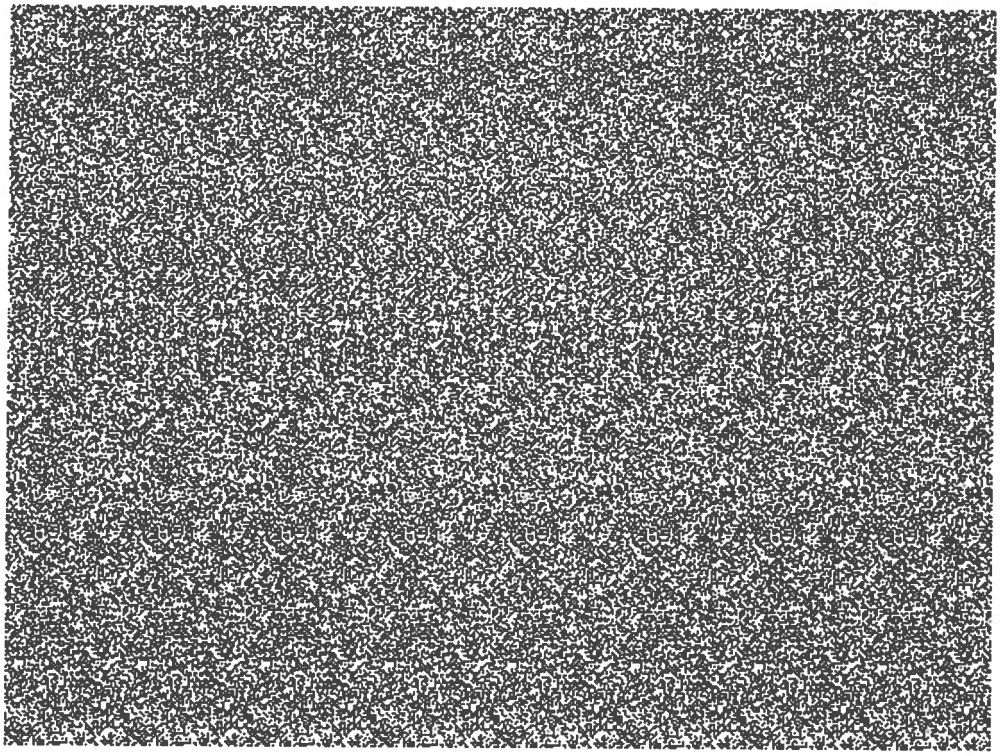
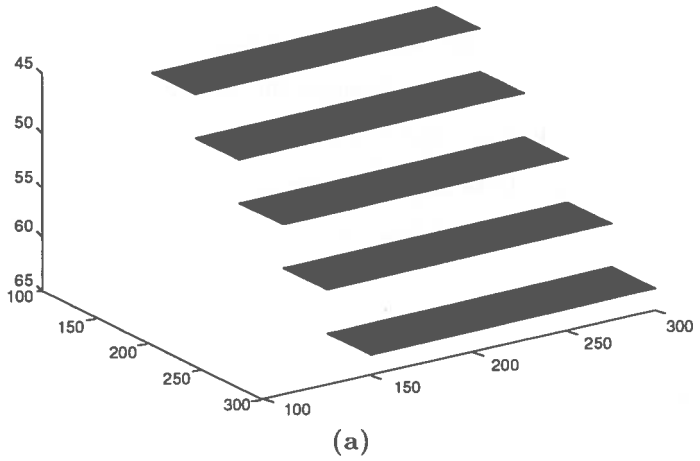


Figure 4.6: **A stepped stereogram.** This stereogram, generated by *3D Library* [10], depicts a series of five horizontally aligned steps; see Figure A.11 for an idea of what should be seen.

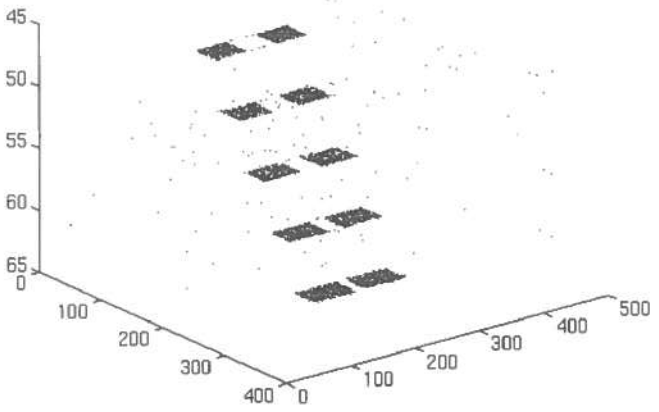
Maybe a combination of the latter two methods would be the best approach, although it is possible that timing constraints would rule this option out.

One method already explored is to run the system twice on the same image, each time using a different Gaussian kernel and then compare the two output sets, retaining only the matches they have in common, in the hope that these would be the true matches. This was tried, comparing Gaussians: $\sigma = 1.3$ (size 12) with $\sigma = 3$ (size 26) and $\sigma = 1.3$ (size 12) with $\sigma = 6$ (size 24). This was unsuccessful – in each case there were no matches in common at all. In retrospect, it is understandable that no matches were found in common as different sized filters with different σ values have the effect of quantising zero-crossings to slightly different positions in an image.

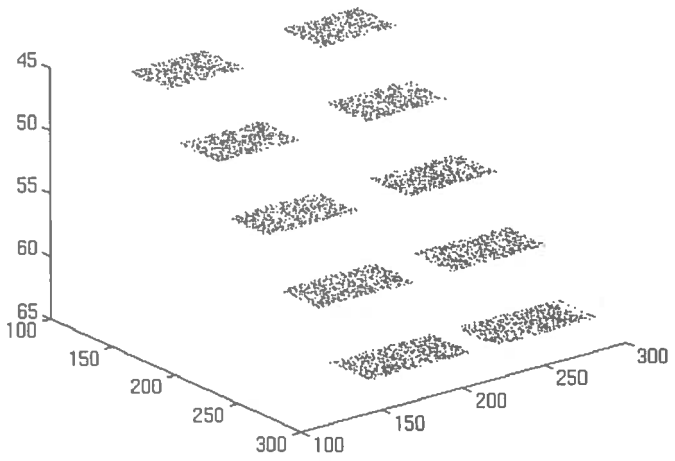
The method used in the final implementation is as follows. Noise is removed by examining each dimension of the data independently. For example, in the case of the x dimension, for each x coordinate in the input image (*i.e.* from 0 to the number of columns in the input) the point list is scanned and a count maintained of how many points have that x value. Then if the final count is lower than some threshold (ten was found to be the optimum threshold in most cases), discard all the points which had that x value. This process is repeated for the y and z dimensions. This has the effect of filtering out isolated points in the data, while leaving the important clusters of points associated with objects largely untouched. The data in Figure 4.7(b) is the result



(a)



(b)



(c)

Figure 4.7: **Example of Noise Removal.** Figure (a) shows how the steps in Figure 4.6 would look before the random dots were overlayed when generating the stereogram. Figure (b) shows the raw set of matches obtained from the stereogram and Figure (c) shows this same data set, after all noise has been removed.

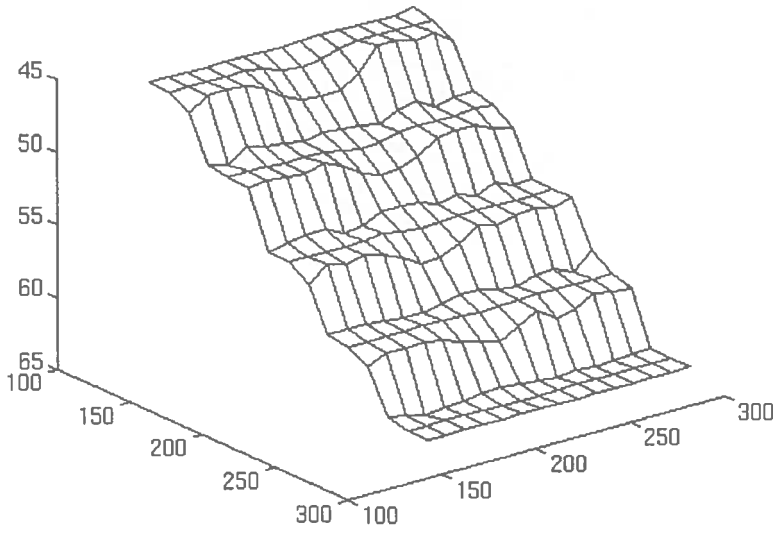
of running the system (without noise removal) on the stepped stereogram in Figure 4.6[1]. The reader will observe that this data is not entirely true to what can be seen in the stereogram from which it came; there is a gap in the centre of each step. The reasons behind the occurrence of such gaps are explained in Section 6.3; we shall ignore them for the present. If we now apply our noise removal strategy to this data, from Figure 4.7(c) it can be seen that it successfully removes all noise. To be precise, of the 4447 points in the initial set (Figure 4.7(b)), the procedure removes 316 (about 14%) leaving 4131 in Figure 4.7(c). We can now reconsider the possibility of fitting a surface to our data.

4.8 Surface Fitting Revisited

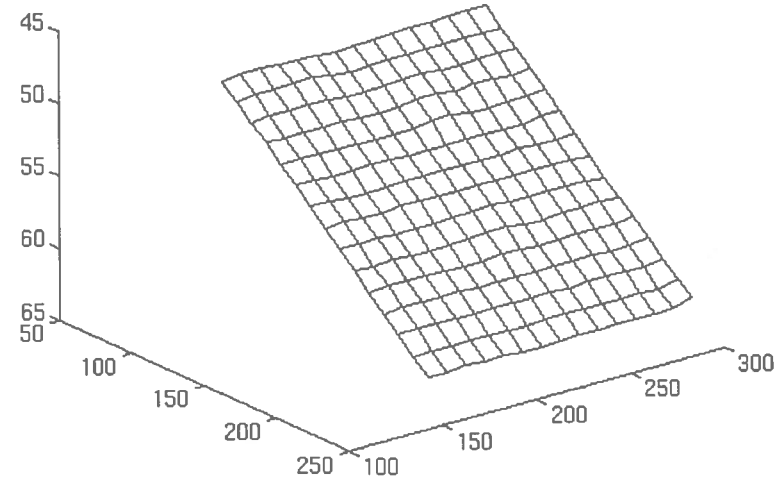
Having eliminated noise using the method described above, it is now possible to fit a surface to the available data. There is a simple *MATLAB* procedure available for fitting a surface to regularly spaced data. However, as our data is very *irregular*, it is necessary to follow a more complicated routine for fitting a surface to irregular data. First, a new evenly spaced grid is generated from the x and y values of the input data by noting the maximum and minimum value of each. Two m by n matrices defining the new grid are then created; one – X_i – contains the new x values and the other – Y_i – contains the new y values – m and n specify the number of rows and columns in the new grid respectively. X_i and Y_i contain values evenly spaced between the minimum and maximum on the x and y axes respectively, such that all the rows of X_i are the same and all the columns of Y_i are the same. Next, the z values in the input are mapped to the evenly spaced grid as defined by X_i and Y_i by using interpolation on the original x, y and z values. For a more detailed description of this routine see [16].

The surface fitted to the noise-filtered data of Figure 4.7(c), using this method, is shown in Figure 4.8(a) – the result is quite satisfactory. As expected, the surface covers the data only and does not extend down to the backdrop of the stereogram, as it was necessary to remove this. Disturbances can be observed all the way down the centre of the steps; these are accounted for by the gaps in the data visible in Figures 4.7(b) and (c). Figure 4.9(c) shows the noise-filtered matches extracted from the stereogram of a set of narrower steps (Figure A.5). The surface fitted to this data can be seen in Figure 4.8(b). Disregarding the gaps down the centre of the steps, the match set obtained for each step is not as thick as might be expected after looking at the original steps in Figure 4.9(a). Consequently, the surface constructed – a steady slope as opposed to a stepped surface – is not accurate. This thinning of the detected matches is an artifact of convolution and is explained fully in Section 6.3.

[1] When viewing the stereograms such as this, generated by *3D Library*, one will sometimes see a sort of double image – in this case it may appear that there is a narrower set of steps in front of the main set. This is caused by one's eyes converging at a too distant point, resulting in points in the picture matching not with the next available point but the one after that. Another possibility is that the image obtained is "inverted" or "sunk-in", this is caused by using the wrong viewing method – these stereograms require the use of the wide-eyed method (see Figure 2.2).

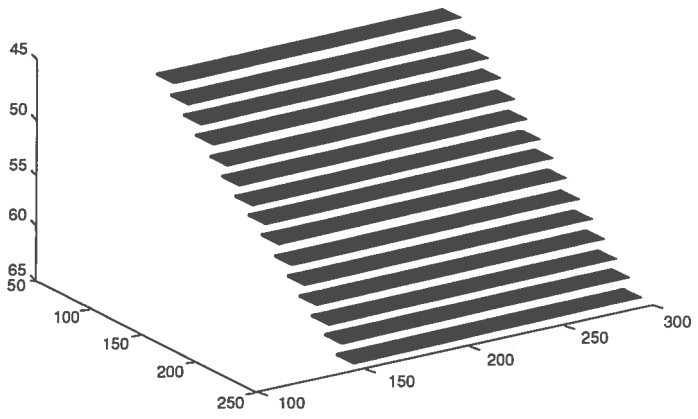


(a)

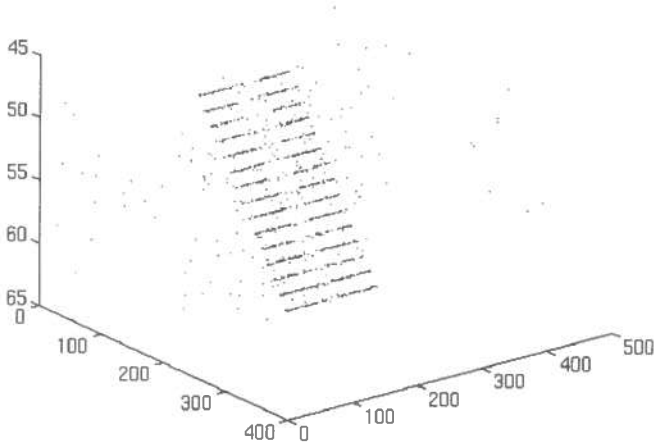


(b)

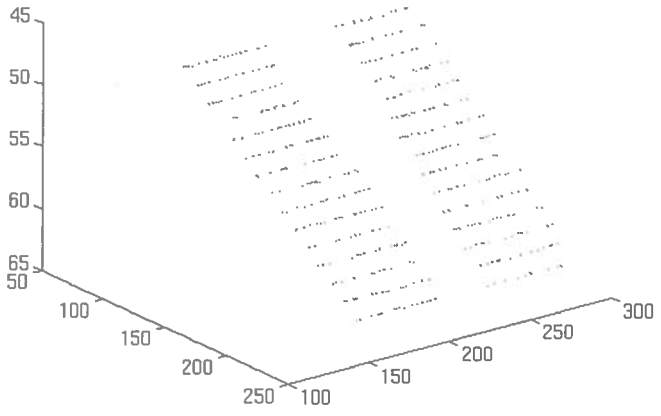
Figure 4.8: **Fitted Surfaces.** The figures show surfaces fitted to match data after noise removal. Figure (a) shows the surface reconstructed from the steps stereogram (Figure 4.6); the disturbance visible down the centre is due to the gaps in the data (see Figure 4.7). Figure (b) shows the surface reconstructed from the stereogram of 15 narrow steps shown in Figure A.5.



(a)



(b)



(c)

Figure 4.9: **Example of Noise Removal.** Figure (a) shows how the steps in Figure A.5 would look before the random dots were overlayed when generating the stereogram. Figure (b) shows the raw set of matches obtained from the stereogram and Figure (c) shows this same data set, after all noise has been removed.

4.9 Summary

In this chapter, we looked at the theories and mathematics behind the process. We began by explaining the calculation of the values for various sizes of *LoG* convolution kernels. We discussed the workings of the zero-crossing detection and matching modules and expounded the adjustments made from Grimson's algorithm which allow the system to work on *SIRDS*. The implications of fitting a surface to the data extracted from *SIRDS* were then considered, which led to a discussion of noise removal and finally to a solution of the surface fitting problem.

In the next chapter, we shall describe the methodology used in testing the system at its various stages of development.

Chapter 5

Testing

It was suggested in the proposal that the final program could be run on stereograms captured externally by video camera. However, during development, it was more convenient to test the program on computer-generated stereograms. To this end, several simple stereograms were constructed using *XFig* and *3D Library* [10]; one such stereogram can be seen in Figure 4.6. These stereograms were particularly useful for testing because the disparities of their hidden objects were known.

Originally, *3D Library* produced stereograms of size 770 × 1026 pixels (*rows* × *columns*). Testing the system on inputs of this size proved to be inconvenient due to long runtimes, so *3D Library* was converted so that it produced stereograms 75% smaller (385 × 513 pixels); these resulted in more manageable runtimes. We take a closer look at runtime efficiency in Section 6.2.

The testing of the system was staged at two levels. Preliminary tests were done running the program with a grey-scale image as input. For this purpose, several *HIPS* images such as the one of the keys in Figure 3.2(a) were grabbed in the vision lab. When satisfactory results were observed in these tests, the program was run with a stereogram image as input. As will be seen, intermediate images and data were output for verification at each stage.

5.1 Testing the Convolution Module

First, a grey-scale image was fed into the convolution module and a convolved version of the image output – an example of a convolved grey-scale image is shown in Figure 3.2(b). As can be seen from this example, there was a problem with noise at the extreme right and left sides of the image, which obviously persisted when applied to stereograms. It was suggested that the reason for this could be that the convolution process was overshooting the side of the image and that this could be solved by stopping the process several pixels before the edges. This solution is put into practice in the final implementation and as can be seen by the absence of such noise in Figure 4.2, this was successful. However, it was noted that the noise occurs in proportion to the

size of the input image, so as it stands this is a band-aid solution and is therefore unreliable. A more permanent fix would be to stop the convolution short of the edge by a distance proportional to the size of the input.

5.2 Testing Zero-Crossing Detection

Once the convolution module had been tested satisfactorily, it was integrated with the zero-crossing detection module and the combined unit tested. As before, this was run on a grey-scale image initially, which was first convolved and the resulting image searched for zero-crossings using the strategy described in Section 4.2. The descriptions of the zero-crossings found in the image were then converted into a visual representation for verification. A visual representation of the zero-crossings found in Figure 3.2(b) is shown in Figure 3.2(c). Next, the convolution/zero-crossing detection unit was run on a stereogram image and again a visual representation of the zero-crossings was constructed; the zero-crossings obtained from the goblet stereogram (Figure 1.1) can be seen in Figure 4.3.

5.3 Testing the Matcher

The matcher was then integrated with the convolution/zero-crossing detection unit and the following strategy used to test the combined entity. Firstly, a grey-scale image was input and this was matched with itself. This, as expected, produced a series of matches of points in the "left" image with points at the same place in the "right" image (the condition suppressing points matching with the same point in the other image was disabled for the purpose of this test). The next logical step from this would have been to match two identical, but offset, grey-scale images, to see if a set of matches with the expected offset (disparity) were output. However, the system is written with the matching of two identical images – *i.e.* matching an image with a copy of itself – specifically in mind and thus it would have been necessary to alter it significantly, merely to perform this test. Therefore, the third stage of testing was applied next; had this not worked, it would have been possible to reconsider the second stage.

The third stage of testing was to perform matching on a simple stereogram. This stereogram was created by myself and so the disparity of the hidden object was known, thus making this just as effective a test as the second one would have been. This test was run with the matching criteria as described in Section 4.3, using a threshold of 50 for magnitude and accepting multiple matches. This produced a majority of matches with exactly the correct disparity and also many spurious matches which appeared in the final plot as noise (see for example Figure 4.4).

It was noted here that the majority of the matches were exact in terms of magnitude, so it was decided to adjust the threshold to 0 so that now only zero-crossings of *equal* magnitude would be allowed to match. It was also noted that approximately 90% of the matches found

Table 5.1: **Results from searching various disparity ranges.**

Input Stereogram	Disparity Range Searched	Disparities Plotted	Observations
goblet.hip	$60<d<90$	positive and negative	stretched cup with 2 stems, strips of noise at both sides (Figure B.1(a)).
	$60<d<90$	positive only	cup slightly stretched, left slope clearer, band of noise on left hand side (Figure B.1(b)).
	$60<d<90$	negative only	more natural view, band of noise on right hand side (Figure B.1(c)).
bowl.hip	$45<d<63$	positive and negative	circles making up 2 bowls inter-mesh, offset by disparity (Figure B.2(a)).
	$45<d<65$	negative only	get most of bowl, erosion near rim, crescent shadow at bottom left and band to right ($d = 64$) (Figure B.2(b)).
	$45<d<63$	negative only	no erosion at rim, circular bite from base, some noise at far right (Figure B.2(c)).
	$45<d<63$	positive only	almost identical to above, except reversed (Figure B.2(d)).
square.hip	$40<d<60$	positive and negative	two vertical rectangles joined at top and bottom by thin bands, enclosing a rectangular gap (Figure B.3(a)).
	$40<d<60$	positive only	a rectangle to the left (Figure B.3(b)).
	$40<d<65$	negative only	some matches around sides of square, large band of noise on far right ($d = -64$) (Figure B.3(c)).

were unambiguous. Remembering that our end goal is to fit a surface to the data, this number of matches is quite sufficient for this purpose – indeed, disambiguating the remaining matches could conceivably detriment the process by adding more spurious points (noise). The unambiguous matches were then plotted using *MATLAB* and a good density of matches was observed.

If it were decided at a later date to subject multiple matches to disambiguation, there is code written for this purpose (the functions DisambMatches and DomSign in the matching module) which could be brought in. Note that where Grimson describes dividing the matches into three distinct pools (Section 2.3) – one for those having *convergent* disparity, one for *divergent* disparity and one for zero – we would only require two as there will be no matches having zero disparity. The reason for this is because, as explained in Section 4.3, a zero-crossing may not match with itself and this is the only case from which a disparity of zero could arise.

Table 5.1 shows results observed when different disparity ranges were searched, for a variety of stereograms. The input stereograms: goblet.hip, bowl.hip and square.hip can be seen in Figures 1.1, A.2 and A.1 respectively. The plotted sets of matches from which these observations were taken are shown in Appendix B. Of course, in the cases of bowl.hip and square.hip it was known in advance which disparity ranges to search as these stereograms were generated by the

author, using *3D Library*.

One thing to note from the table is the "stretching" observed in the case of the goblet – this demonstrates the phenomenon described in Section 4.6 whereby an object which varies in depth will have the points in its two fields offset by varying disparities. Another point to raise is that these results demonstrate that if we search even a little too far out of the known disparity range of an object, we risk introducing erroneous features into the reconstructed image. This is especially evident in Figure B.2(b) where relaxing the ceiling of the disparity limit by just two (from 63 to 65) introduces a large crescent and a band at the base of the bowl, which would present problems with surface fitting. This obviously has undesirable implications for situations where the disparities of objects in the input are not known *a priori*.

5.4 Summary

In this chapter, we looked at how a grey-scale image and then an image of a stereogram were used in turn to test the capabilities of the system.

In the next chapter, we give an account of the various experiments which were carried out in order to gauge the performance of the completed system. We also account for the appearance of the output seen so far.

Chapter 6

Experimentation

In previous chapters, we have seen the results produced when the system runs on various input stereograms. From this we have an indication that the system is functioning reasonably reliably and generating the "right sort" of output but this alone is not sufficient; we must know exactly how "right" this output is. In this chapter we will attempt to quantify the performance of the system in terms of accuracy and efficiency by performing a number of experiments and interpreting their results. As a by-product of these experiments, we also gain an insight to the exact form of the output of the system.

6.1 Matching Statistics

Table 6.1 shows matching statistics for two of the test stereograms. The table has entries for dot density of a stereogram, the total number of (unambiguous) matches found, the number of these which were exactly correct, the number out by one pixel, the number which were wrong – *i.e.* the remaining matches – and this figure as a percentage of the total. Dot density refers to the ratio of black dots to white space in the stereogram. The input stereograms are square.hip (Figure A.1) and wedding.hip (Figure A.3). These were generated by *3D Library* and are sized 385 rows by 513 columns.

The comparatively large amount of wrong matches found in experiments 1 and 3 is explained by the large disparity range ($0<d<100$) searched in each case, compared to the small range in which the objects lie ($50<d<60$). In each of the experiments, the wrong matches found are those points which appear in the final plots as noise.

Table 6.1: **Matching statistics.**

	Input Stereogram	Density	Total	Exact	One Pixel	Wrong	% Wrong
1	square.hip ($0<d<100$)	66%	3573	2716	15	842	24%
2	square.hip ($45<d<65$)	66%	2972	2716	15	241	8%
3	wedding.hip ($0<d<100$)	66%	10410	9292	37	1081	10%
4	wedding.hip ($45<d<65$)	66%	9660	9292	37	331	3%

6.2 Runtime Performance

The runtime statistics of the program running on a *SUN SPARCclassic* workstation under *UNIX*, taking a variety of stereograms as input, are shown in Table 6.2. These figures are based on the program searching for matches with disparity $30<d<60$. From the table we can see that, on average, the runtimes range from 1.5 minutes for average-sized stereograms up to around 7 minutes for fairly large stereograms.

Table 6.2: **Runtime statistics.**

Input Stereogram	Size (*rows* × *cols*)	Runtime
square.hip	385 × 513 pixels	94.91 seconds
circle.hip	385 × 513 pixels	94.85 seconds
wedding.hip	385 × 513 pixels	95.55 seconds
bowl.hip	770 × 1026 pixels	399.38 seconds
face.hip	770 × 1026 pixels	395.2 seconds

6.3 Accuracy of the System

When the system is run on a stereogram and we observe the final surface constructed, it is not sufficient to say that this surface *looks like* what we see when we view the stereogram. It is important to know to exactly what extent the surface looks like what we see in the stereogram, *i.e.* how accurate is our reconstruction?

In order to be able to gauge accuracy, we must have at our disposal such information as the location and size of the objects in the stereogram, the disparity (disparities) of these objects and the properties of the convolution mask used. We know the exact size and location of the objects because we have access to the source code of the program which generated the stereograms. We know the disparities for this same reason and we also know about the convolution mask.

Look again at Figure 4.7(c). The reader will already have noticed that, where the stepped stereogram (Figure 4.6) depicts a set of regular steps, this plot shows a set of steps with a gap in the middle of each step. The reason for this is as follows. As explained in Section 4.6, any object seen in a stereogram is the sum of two disjoint fields – offset horizontally by the disparity of that object – which fuse together when viewed correctly. Now, what this usually means is that there is a central portion where these two fields overlap and a portion on either side where the side of each field protrudes by an amount equal to the disparity. These protruding edges are what is seen in Figures 4.7(b) and (c) and Figure 6.1(a). The latter depicts the matches extracted from the stereogram in Figure A.1 of a square (side 150 pixels) with disparity 50, hence the 50-wide protruding edges. So, from this we can deduce that the gap between the two protruding edges of the fields of any object will be inversely proportional to the disparity of that object. So the greater the disparity, the greater the offset of the two fields, the smaller the overlap, the smaller the gap observed in the final data. The only situation where the two fields will not overlap

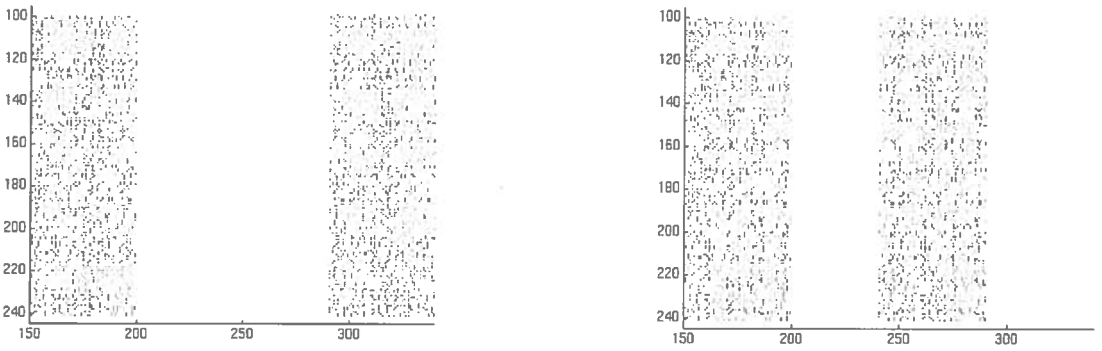
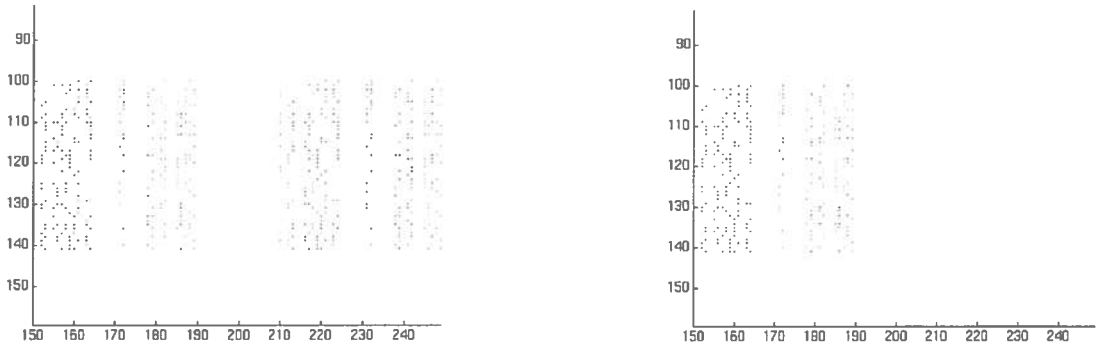


Figure 6.1: Figure (a) shows the matches obtained from a stereogram (Figure A.1) of a square with side 150 and disparity 50 – the left and right fields are incomplete, hence the gap between them. Figure (b) shows the matches in (a) after the right field has been shifted to the left by an amount equal to the disparity (50). The axes denote position in the image in numbers of pixels, with the origin at the top left of the image.

is when the disparity, d, is greater than the width, w, of the object – here the fields will be completely disjoint and separated horizontally by a gap of width equal to $d - w$. This situation arises in the stereogram in Figure A.4 of a square of side 50 with disparity 60. Figure 6.2(a) depicts the matches extracted from this stereogram and one can see two small square fields of side 50 offset by the disparity of 60, resulting in a gap of $60 - 50 = 10$ separating the fields. However, close examination of the plot will show that the gap is not 10 but nearer 20. This is accounted for by the fact that the convolution mask (of width 9 pixels), once applied to the stereogram has the effect of shrinking the spread of matches found by about 9 pixels, in both x and y dimensions, from the size of the original object. This also explains why the square has side 41 in the plot and not 50 as it should be. The same has happened to the square in Figure 6.1 – in the plot it has side 141 instead of 150.

Having accounted for the width of the gap observed in the set of matches found, we must ask the question: why are no matches found in this gap? This is down to the fact that, for example, if a zero-crossing, a, situated on the left "protruding" side of the object finds a match it will find one to its right (in the right field), whereas the point that it matches with will find at least two matches – a and another on its right hand side. This will be the case for any point situated in the area delineated by the gap. These represent multiple matches which are obviously vital to the true reconstruction of the scene but which are filtered out by the condition in the system which only accepts unambiguous matches. This suggested (contrary to the assertion made in Section 5.3) that it may be necessary to implement a disambiguation routine for multiple matches. Accordingly, the disambiguation functions already in place were debugged and integrated to the system. This, surprisingly, did not make much difference at all in fact as was also predicted in Section 5.3, all it served to do was introduce more noise. In the case of the square stereogram, we obtained precisely five extra match points all of which



(a) (b)

Figure 6.2: Figure (a) shows the matches obtained from a stereogram (Figure A.4) of a square with side $w = 50$ and disparity $d = 60$. Unlike in Figure 6.1, the fields *are* complete and the gap between them is explained by the fact that the disparity is greater than the width of the square, preventing them from interfering with each other. The width of the gap should be $d - w = 10$ but is 19 due to the convolution effect. Figure (b) shows the matches in (a) after the right field has been shifted to its left by an amount equal to the disparity (60), with the effect of overlaying the two fields exactly to obtain a single square.

appeared as noise around the perimeter of the square.

Recall that Grimson's disambiguation routine only disambiguates a point if it has exactly one match which has the same sign as the dominant neighbourhood sign. This would fail if there were two matches of the dominant neighbourhood sign, one of which was the true match. This could be what is happening here, in which case, this sort of disambiguation will not work. We shall discuss a possible solution to this problem in Section 7.2.1.

Note that this interference, manifested by gaps in the data, was also present in the case of the goblet – Figure B.1 shows the two fields of the goblet plotted together and separately. In the right field of the goblet (Figure B.1(c)), the right slope of the cup is better defined than left, where interference has occured due to the overlap of the left field (Figure B.1(b)); the extent of this overlap is apparent in Figure B.1(a).

Interfering fields would obviously not be a problem when decoding *double* image stereograms as the two fields, although still offset by the required disparity in relation to each other, do not *physically* overlap (as is the case with *SIRDS*) because each inhabits its own *independent* picture.

Let us assume that a method exists which would allow us to obtain most of the matches making up each field of the object and thus fill the gap. The two fields would then still be offset by disparity, so before finally outputting the coordinates of the matches, we could simply subtract the z (disparity) coordinate from the x coordinate of any matches with negative disparity (all matches in the right field have negative disparity). This would have the effect of shifting the right field by d pixels to the left, overlaying the two fields – this would work because all matches have negative disparity in the right field and positive disparity in the left. The results of shifting the right fields of Figures 6.1(a) and 6.2(a) can be seen in Figures 6.1(b) and 6.2(b) respectively.

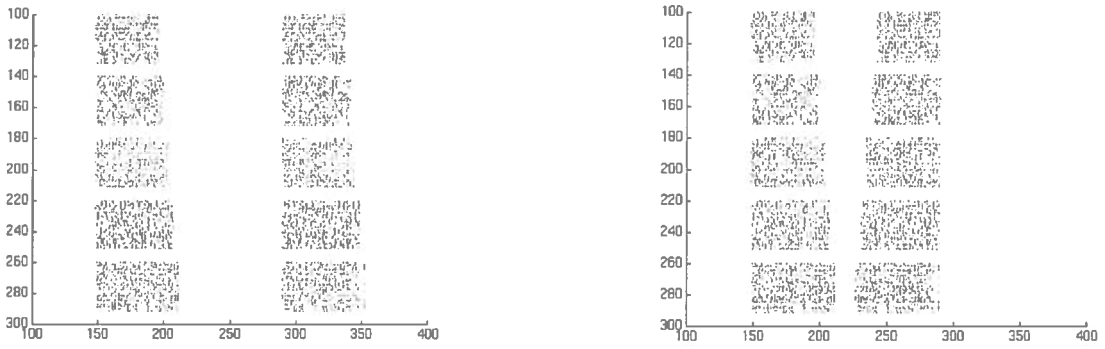


Figure 6.3: Figure (a) shows the matches obtained from the stepped stereogram (Figure 4.6). The disparities of the steps range from 47 at the top to 63 at the bottom. Again the fields are incomplete and have gaps between them. Figure (b) shows the matches in (a) after each point in the right field has been shifted to the left by an amount equal to its disparity. The axes denote position in the image in numbers of pixels, with the origin at the top left of the image.

In the case of the small squares in Figure 6.2, this renders a single complete square but in the case of Figure 6.1 it only serves to reduce the gap between the two partial fields.

After applying this horizontal shift, although there is still a gap in Figure 6.1(b), the edges of the object are mostly intact so we *can* get a precise idea of the accuracy of the system. We can do this by performing a comparison of the location of an object in the input stereogram and its location in the output. For example, Table 6.3 shows the difference between the positions of the vertices of a square of side 150 pixels in the matches extracted (plotted in Figure 6.1) and the positions at which they were placed when generating the stereogram (the stereogram is shown in Figure A.1). Note that the only discrepancy here is a nine pixel difference in height and width. This is attributable to the effect of the convolution mask – which is of size 9×9 pixels – explained above.

Accuracy is thus inversely proportional to the size of the convolution mask used. A possible solution to this problem would be to grow the object by the width of the convolution mask before surface fitting. This phenomenon also explains why, in Figure 4.8(b), the surface reconstructed from the stereogram of a set of 15 steps (Figure A.5) produces an even slope; the already narrow steps are further eroded by the convolution kernel so as to make them too thin to produce a

Table 6.3: **Accuracy statistics for the square stereogram.** (All distances are in pixels.)

Vertex	Position of vertex $(rows, cols)$		Difference
	in input	in output	
Top left	$(150, 100)$	$(150, 100)$	$(0, 0)$
Top right	$(300, 100)$	$(291, 100)$	$(-9, 0)$
Bottom left	$(150, 250)$	$(150, 241)$	$(0, -9)$
Bottom right	$(300, 250)$	$(291, 241)$	$(-9, -9)$

stepped surface.

Let us examine the slightly more complex situation of the stepped stereogram. This depicts a set of five steps, each of width 150 pixels and breadth 40 pixels with the disparity of each step increasing from the top of the picture downwards, that is, the steps appear increasingly further away towards the bottom. Figure 6.3(a) shows the matches extracted from this stereogram. The reader will observe that the gap apparent in each step lessens from top to bottom, *i.e.* as the the disparity of each individual step increases. This supports our earlier inference that the gap between an object's fields is inversely proportional to the disparity of that object.

Figure 6.3(b) shows this same data after each point in the right hand field has been shifted to the left by its disparity. In contrast with Figure 6.1 for example, the effect of this shift is much more noticable because instead of having the effect of moving the entire right hand field by the same amount, it moves the right hand field of each step by a different amount (this amount is of course the disparity of a given step).

Table 6.4 shows accuracy statistics for the data extracted from the stepped stereogram in Figure 4.6. It compares the positions of the vertices of each of the steps as plotted in Figure 4.7(c) to the positions at which they were placed when generating the stereogram. Again, the shrinking effect of the convolution mask is very apparent from the differences in verex positions. Although, from this analysis the data for the steps is less accurate than that obtained for the square above, if we again disregard the convolution effect, the largest discrepency recorded is of only three pixels – still very accurate. The statistics shown for the square and stepped stereograms are typical of all those examined.

So far in this discussion, we have not mentioned disparity. This is for the reason that in every stereogram tested, once noise had been removed, 100% of the disparity values extraced were exactly true to the values for each point in the input. So, even subject to the unavoidable effects of the convolution mask, we can conclude that the system is extremely accurate positionally. If we were to to discount the effects of the mask we could claim almost 100% accuracy. Obviously this claim can only extend to the stereograms generated by the author. Although it is difficult to test the accuracy of other stereograms for which we do not have disparity information for the original objects, one could reasonably expect the same sort of results. Accuracy is anticipated to significantly poorer on externally captured images – of posters for example – due to noise introduced by the various transformation processes involved in making the poster and capturing the image.

6.4 Summary

In this chapter, we began by examining the efficacy of the matching procedure and by giving an idea of the typical runtimes that can be expected of the system. We went on to consider the accuracy of the system's output and consequentially accounted for all aspects of the appearance

Table 6.4: **Accuracy statistics for the stepped stereogram.**

Step	Vertex	Position of vertex $(rows, cols)$		Difference
		in input	in output	
Top	Top left	$(150, 100)$	$(150, 100)$	$(0, 0)$
	Top right	$(300, 100)$	$(290, 100)$	$(-10, 0)$
	Bottom left	$(150, 140)$	$(148, 131)$	$(-2, -9)$
	Bottom right	$(300, 140)$	$(290, 130)$	$(-10, -10)$
Second	Top left	$(150, 140)$	$(150, 140)$	$(0, 0)$
	Top right	$(300, 140)$	$(290, 140)$	$(-10, 0)$
	Bottom left	$(150, 180)$	$(150, 171)$	$(0, -9)$
	Bottom right	$(300, 180)$	$(290, 171)$	$(-10, -9)$
Third	Top left	$(150, 180)$	$(149, 180)$	$(-1, 0)$
	Top right	$(300, 180)$	$(288, 182)$	$(-12, 2)$
	Bottom left	$(150, 220)$	$(150, 210)$	$(0, -10)$
	Bottom right	$(300, 220)$	$(291, 212)$	$(-9, -8)$
Fourth	Top left	$(150, 220)$	$(151, 221)$	$(1, 1)$
	Top right	$(300, 220)$	$(289, 218)$	$(-11, -2)$
	Bottom left	$(150, 260)$	$(150, 251)$	$(0, -9)$
	Bottom right	$(300, 260)$	$(289, 250)$	$(-11, -10)$
Bottom	Top left	$(150, 260)$	$(150, 260)$	$(0, 0)$
	Top right	$(300, 260)$	$(291, 260)$	$(-9, 0)$
	Bottom left	$(150, 300)$	$(150, 292)$	$(0, -8)$
	Bottom right	$(300, 300)$	$(290, 291)$	$(-10, -9)$

of this output. It was concluded that a disambiguation routine would be necessary in order to obtain a complete set of matches but, this aside, the positional accuracy of the system was proved to be very good. In the final chapter, we shall examine to what extent the final system meets the initial requirements of the project and explore the possibilities for extensions to the system.

Chapter 7

Conclusions

Having explained fully all aspects of the design, implementation and testing of the system and having performed thorough experimentation, interpreting the results as appropriate, we are now in a position to evaluate the project as a whole. In this chapter we shall assess how well the original aims of the project were fulfilled and consider what refinements and extensions may be appropriate in order to improve the performance of the process. In particular we shall investigate the implications of introducing colour/grey-scale into the input stereograms.

7.1 Critical Summary of Project

It is very difficult to decide for a system of this nature, whether it "works" or not. We attempted to gauge how well it works in Chapter 6. Here we shall attempt to determine how well the final system meets the specifications set out in the original project proposal.

I believe that all the main objectives of the project were realised. A system was implemented that successfully:

- takes as input a single image random dot stereogram,
- extracts zero-crossings from a convolved version of the input,
- matches these zero-crossings and computes depth information from the matches found,
- fits a surface to the data using the positions of the matches combined with their disparities.

Grimson's algorithm has been re-implemented and extended such that it works on *SIRDS* . The output produced is in some cases incomplete, but this was accounted for and we shall discuss a solution to this problem in the next section.

The surface fitting section of Grimson's algorithm was not implemented. This function was considered secondary to the main goal of the project, namely the decoding of stereograms. Thus it was decided to concentrate effort on the solution to the latter problem, leaving the implementation of surface fitting as a task to be performed last if time permitted. As it transpired, the

decoding problem was rather more complex than first anticipated and did not leave sufficient time. It was decided instead to avoid "reinventing the wheel" and to use *MATLAB* to perform surface fitting. This proved to be a success in the case of the simpler stereograms but unfortunately *MATLAB* could not cope with the more complex surfaces described by the data obtained from the bowl and goblet stereograms. However, as it is easier to gauge the accuracy of surfaces fitted to simpler data, *MATLAB* sufficed for our purposes.

As mentioned before, it was suggested in the proposal that the finished system could be tested on some images of stereograms acquired externally, for example on the posters in the department. It was decided that this capability should be first tested on my own computer generated stereograms on which it is known that the system performs well. Therefore, hard copies of several of these stereograms were obtained and images of these grabbed in the vision lab. This sounds like a rather pointless process, however it gives us a benchmark with which to gauge the effectiveness of the system on images obtained in this way. It transpired that the process of grabbing a bi-level stereogram introduced grey-levels into a previously black and white image.

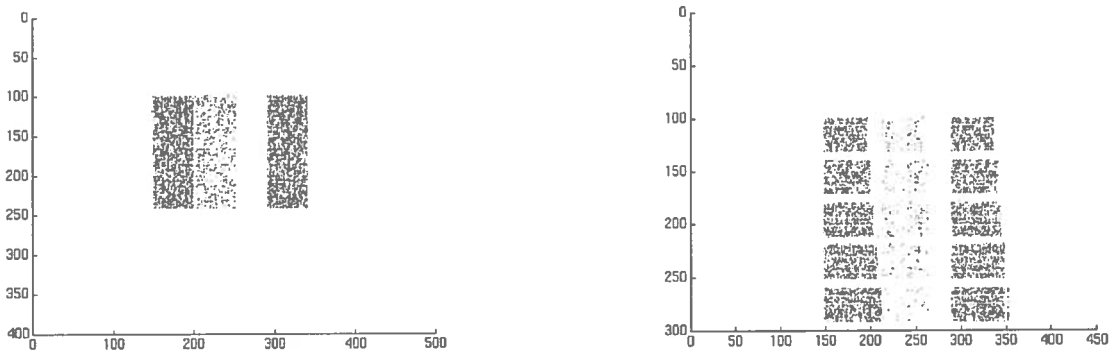
This, as we shall discuss in the next section, is unacceptable as the system can only cope with bi-level images. The camera also served to distort the image by expanding it vertically, which would have made it very difficult to compare results if any had been obtained. The problem with grey-levels could probably be remedied using thresholding and the distortion by compressing the image (see [6] for information on these operations). These, perhaps combined with a more faithful method of image capture – such as a scanner – would improve the quality of external images, however time was not available for further experimentation in this area.

If capturing images by camera had proven to be feasible, it would have been appropriate to compile matching statistics (as in Section 6.1) by running the system on the images grabbed and compare these with the statistics gathered for their unadulterated counterparts.

7.2 Possible Developments

7.2.1 Filling the Gaps

As was discussed in Section 6.3, we still do not have a method which finds the matches in the gaps observed in Figures 4.7 and 6.1. The author has recently observed a fundamental fact that may lead to a solution. Each of the points observed in any of the plots of match data seen in this report represent only one side of a match; at the position (x, y) of one zero-crossing, a match was found for it at $(x + d, y)$ and a match was therefore recorded at (x, y). However it seems that, when the matcher came to the zero-crossing at $(x + d, y)$, where it should have recorded a match at $(x + d, y)$ with the zero-crossing at (x, y), it did not. The reason for this is that there will have been more than one possible match – a situation which the disambiguation procedure



(a) (b)

Figure 7.1: The figures show the results obtained using the improved matching routine described in Section 7.2.1, *before* shifting the right field to the left. Figure (a) shows the matches extracted from the stereogram (Figure A.1) of a square with side 150 and disparity 50. Figure (b) shows the matches from the stepped stereogram (Figure 4.6). In both (a) and (b) the fields are still incomplete but a more sparse band of matches is visible between the original two. This band is the left hand edge of the right field.

failed to resolve because there was more than one possible match with the dominant sign of neighbouring matches.

So we have established that our disambiguation routine is inadequate and there seem to be no immediately obvious ways to improve it; what can be done? One improvement to the main matching routine which may go some or all of the way to fill the gaps is as follows. We can exploit the symmetry of each match found, *i.e.* if we have a match at a with b then there must be a match a b with a. So, in the situation described above where a match is found at (x, y) with a zero-crossing at $(x + d, y)$ the matcher will also record a match at $(x + d, y)$ with the zero-crossing at (x, y). Note that this contradicts the uniqueness constraint used by Grimson, described in Section 2.3.

Preliminary experimentation has been done using this technique and the results of running it on the square and stepped stereograms of Figures A.1 and 4.6 are shown in Figure 7.1. In each case, approximately 30% additional matches were found; these appear in Figure 7.1 as a slightly more sparse strip in between the original strips from the left and right fields. These matches, having negative disparity, account for the leftmost side of the right hand field. As part of the right hand field, however, these points when shifted to the left superimpose themselves almost exactly onto the leftmost side of the *left* field, rendering images similar to those in Figures 6.1(b) and 6.3 in the cases of the square and the steps respectively. So, although we have found a good amount of extra points, in the end we still have the same size of gap to fill.

It remains to be seen whether further experimentation in this area will produce a solution; this is perhaps a good starting point for any future research on the subject.

7.2.2 Stereograms and Colour

The system described in this report cannot deal very effectively with coloured stereograms. However, this was never an aim of the project – we set out with the intention of decoding simple (by today's standards) bi-level random dot stereograms. We have succeeded in this aim but the stereograms being produced today present us with more problems, exhibiting increasingly complicated patterning – rippling, stars and flowers for example – and more importantly, colour. In this section we discuss how colour may be introduced to stereograms and the implications for decoding such pictures.

Let us first consider how colour is incorporated into a stereogram picture [12] when it is being generated. Figure 7.2 depicts a stereogram with three hidden objects 1,2 and 3, at different perceived depths. When generating this stereogram in colour, one must first consider which points on the perceived objects project back to share a common pixel. This can be done by projecting rays from each eye to points on the objects and noting where the rays intersect the picture.

To begin with, let us project a ray from a point in object 1 to the left eye; this crosses the stereogram picture at point A. We can choose any colour we like for point A – for example, the desired colour of the point, as taken from the original object 1 or any other colour, green for example. Once this colour is chosen it must be assigned to parts of other objects in the stereogram scene as well; point B on the screen must also be green because this is the point at which the right eye sees the green point in object 1. Obviously, the left and right eyes must see the same thing when they are looking at the same point, so there is no choice but to colour point B green.

Point B is also seen by the left eye – this happens when it is looking at a point in object 2. The right eye sees this point in the object at position C on the screen, and so this too must be drawn in green. Colouring point C green has a knock on effect further down the line and by continuing this line of reasoning we end up with a row of four points labelled A to D, all coloured green. So, essentially this is process is only serving to make the stereogram a pretty colour and not give the objects their true colours – they just inherit whatever pattern of colours is making up the stereogram.

To complete the stereogram for the scene in Figure 7.2, all the remaining points on the screen would have to be plotted, under the same principles. Finally, a backdrop should be added, some distance behind all the objects. So, if the plotting algorithm scans the screen from left to right, then objects on the right will inherit whatever was chosen for the left hand side. In Figure 7.2, all the points lie in a horizontal line, the simple reason for this being that the human eyes are aligned horizontally. This demonstrates that – for the same reason as we were able to introduce the epipolar constraint in Section 2.3 – the stereogram colouring method only operates horizontally and there is no complication of interaction between adjacent rows. This same procedure would

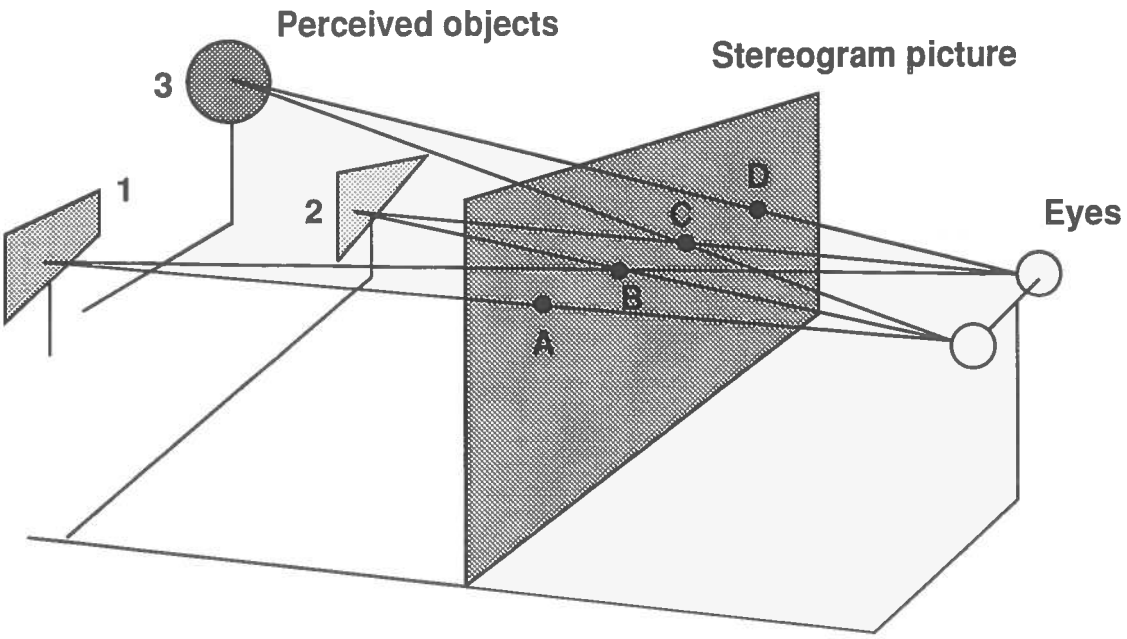


Figure 7.2: **How patterning relates to objects in stereograms.** In order for stereopsis to be attained effectively (stereopsis may be possible if only some of the matches are correctly coloured but the image obtained will not be as clear), the points labelled A,B,C and D in the above stereogram must all be printed in the same colour.

have to be followed when creating a stereogram in grey-scale.

It is possible that decoding coloured (or grey-scale for that matter) stereograms could be very hard [4] due to the fact that the smoothing applied by the *LoG* filter will unavoidably cause information to be lost. This is the information supplied by the fact that matches may only occur between like-coloured pixels and this will be diluted to a certain extent when each pixel is averaged with its neighbours during convolution. This would suggest that an alternative method which exploits this correlation between colours may be more appropriate for dealing with coloured stereograms. It will be important in doing this to consider how the colour is encoded in the stereogram. If it is encoded by means of interpolated RGB values, then it may be possible to match these separately or in parallel; some experimentation would be necessary to establish an optimum method.

7.3 Summary

In conclusion, I believe that this project can be considered a success. Original observations have been made about the nature of single image random dot stereograms and how these may be decoded. The end result is a system which can reconstruct three-dimensional surfaces from single image random dot stereograms – something which has, to the author's knowledge, never been achieved before.

Bibliography

[1] **Baker, H.H.** *Depth from Edge and Intensity Based Stereo.* AI Memo 347, Stanford University, Stanford, CA, USA, 1982, cited in [18].

[2] **Burt, P. and Julesz, B.** *Modifications of the Classical Notion of Panum's Fusional Area.* Perception 9, 671–682, 1980, cited in [18].

[3] **Chang, P.** *Stereography and Autostereograms.* <http://h2.ph.man.ac.uk/gareth/ss/ss.html>, 1994

[4] **Chang, P.** Personal communication with Peter Chang <peterc@man.ph.a3>, Feb 1995

[5] **Christophers, R.** *M.Sc. Dissertation "Precategorical Segmentation Using Disparity".* Department of Artificial Intelligence, The University of Edinburgh, 1992

[6] **Croft, D. et al.** *Manuals for the Undergraduate Vision Lab.* Department of Artificial Intelligence, The University of Edinburgh, 1986

[7] **Fisher R., Perkins S., Walker A. and Wolfart E.** *Hypermedia Image Processing Reference.* <file://localhost/hame/simonpe/ip/PROJECT/html/hipr_top.html>, Department of Artificial Intelligence, The University of Edinburgh, 1994

[8] **Grimson, W. Eric L.** *From Images to Surfaces — a Computational Study of the Human Early Visual System.* MIT Press, Cambridge, Massachusetts, 1981

[9] **Haralick, R.M. and Shapiro, L.G.** *Computer and Robot Vision Vol. 1.* Addison–Wesley Co., Inc., Reading, MA, 1992

[10] **Jones, R. and Short, J.** *3D Library – stereogram generator.* Public domain, e-mail Justin Short <js6@doc.ic.ac.uk>, 1994

[11] **Julesz, B.** *Binocular Depth Perception of Computer-generated Patterns.* Bell Systems Technical Journal 39, 1125-1162, 1960

[12] **Liardet, M.** *Article "Seeing is Believing".* Personal Computer World, December 1994

[13] **Marr, D.** *A Note on the Computation of Binocular Disparity in a Symbolic, Low-level Visual Processor.* MIT Artificial Intelligence Laboratory, Memo 327, 1974, cited in [8].

[14] **Marr, D. and Hildreth, E.** *Theory of Edge Detection.* Proceedings of the Royal Society of London, Series B, 207, 187–217, 1980, cited in [18].

[15] **Marr, D. and Poggio, T.** *A Theory of Human Stereo Vision.* Proceedings of the Royal Society of London, Series B, 204, 301–328, 1979, cited in [8].

[16] *MATLAB* **Technical Note.** *Contouring Unevenly Spaced Data.* <http://www.mathworks.com/tec2.14.html>, 1994

[17] **N.E. Thing Enterprises** *Magic Eye II.* Michael Joseph Ltd., 1994

[18] **Pollard, S.B., Mayhew, J.E.W., Frisby, J.P.** *PMF: a stereo correspondence algorithm using a disparity gradient limit.* Perception Vol. 14: 449-470, 1985

[19] **Raymond, R.** *Stereogram FAQ, Stereogram History.* <http://www.cs.waikato.ac.nz/~singlis/sirds/general.html#Subject10>, 1994

[20] **Richards, G. and Chang, P.** *SIRDS Gallery.* <http://h2.ph.man.ac.uk/gareth/gif.html>, 1995

[21] **Rosenfeld, A., Hummel, R. and Zucker, S.W.** *Scene Labelling by Relaxation Operations.* IEEE Transactions on Systems, Man and Cybernetics, 6: 420–433, 1976, cited in [18].

[22] **Schalkoff, R.J.** *Digital Image Processing and Computer Vision.* Wiley & Sons Inc., Dept. of Electrical and Computer Engineering, Clemson University, 1989

[23] **Sewell, P.** *CS3 Programming Methodology Notes.* Department of Computer Science, The University of Edinburgh, 1994

[24] **Tyler, C.W. and Clarke, M.B.** *The Autostereogram.* SPIE Stereoscopic Displays and Applications 1258: 182-196, 1990, cited in [19].

[25] **Ullman, S.** *The Interpretation of Visual Motion.* MIT Press, Cambridge, Massachusetts, 1979, cited in [8].

[26] **Wilson, H.R. and Bergen, J.R.** *A Four Mechanism Model for Threshold Spatial Vision.* Vision Research 19, 19–32, 1979, cited in [8].

Appendix A

Stereograms

This appendix contains some stereogram images referenced at different points in the text. Following these are the "answers" to these stereograms, that is, what you should see when you view them. The answers are of course two-dimensional whereas the image seen should appear three-dimensional.

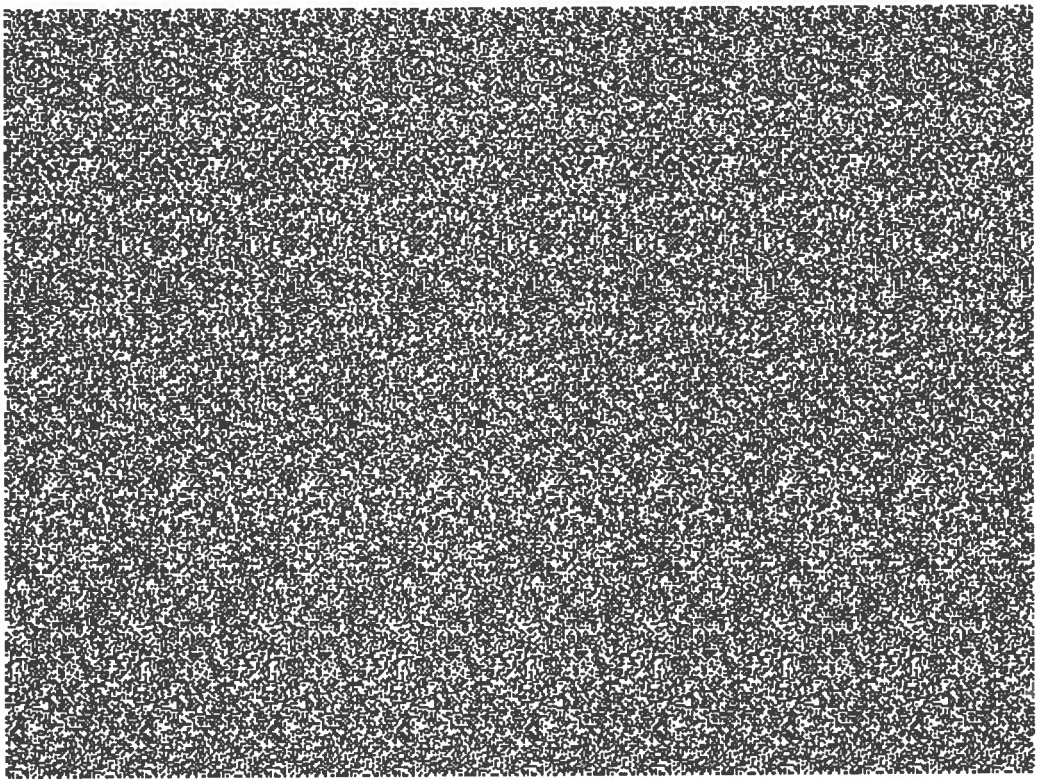


Figure A.1: **square.hip** – a stereogram of a simple square.

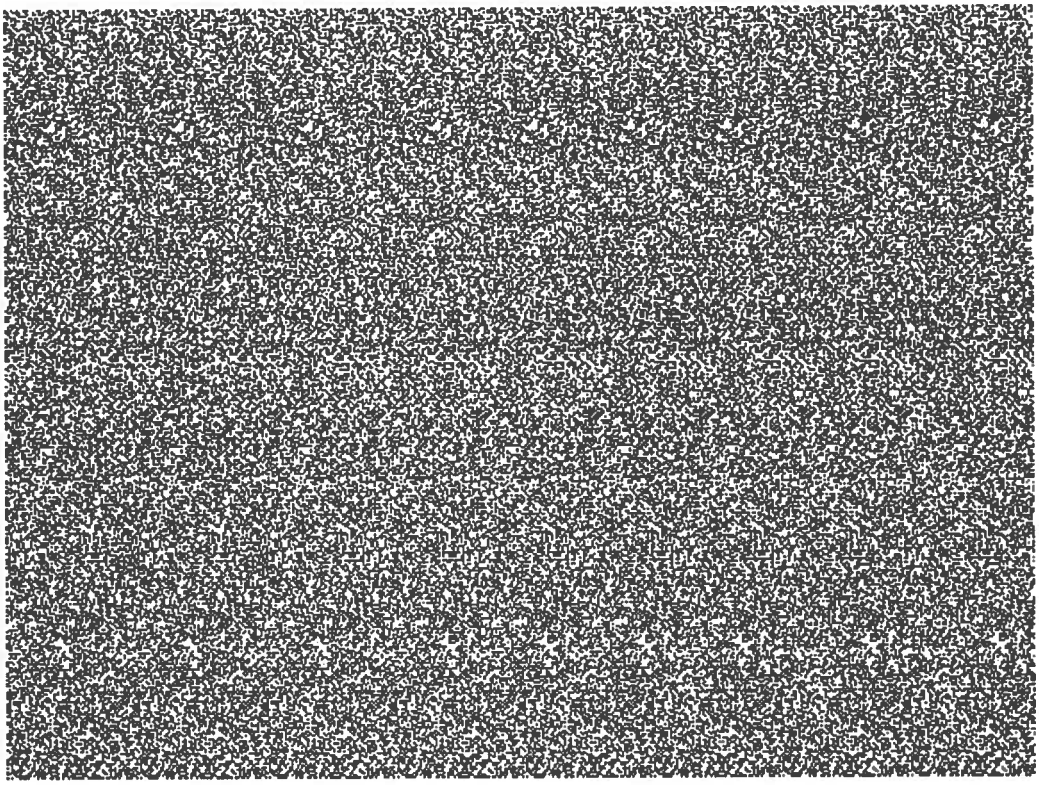


Figure A.2: **bowl.hip** – a stereogram of a bowl viewed from above.

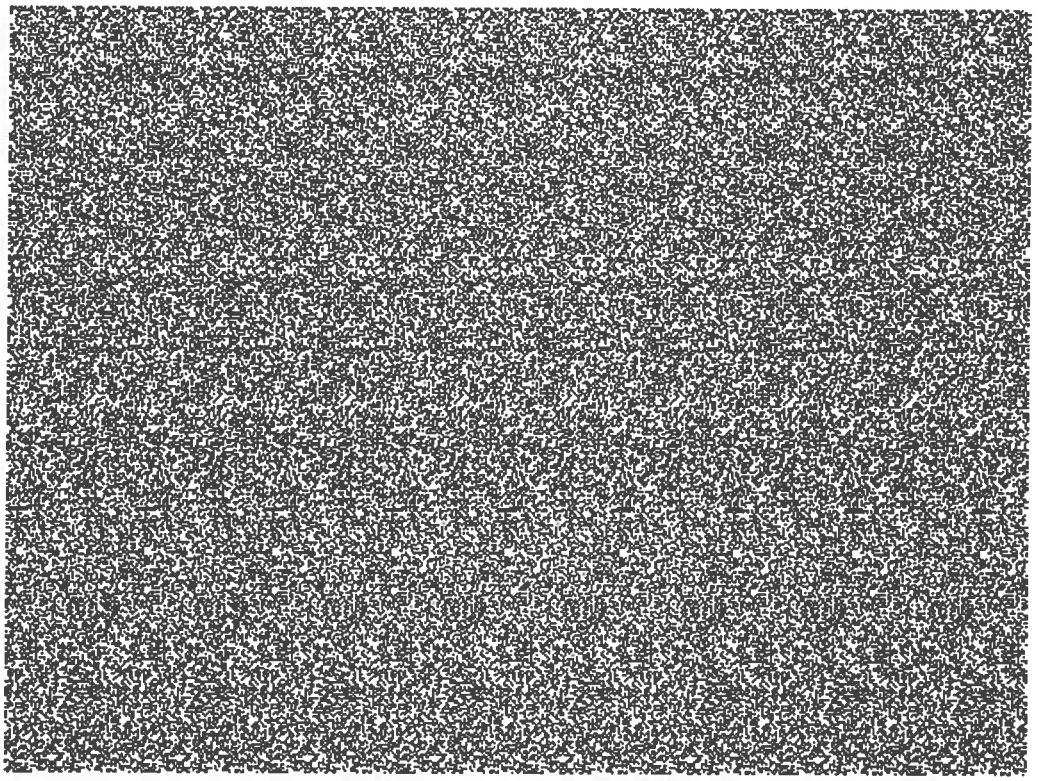


Figure A.3: **wedding.hip** – a stereogram of a layered wedding cake shape viewed from above.

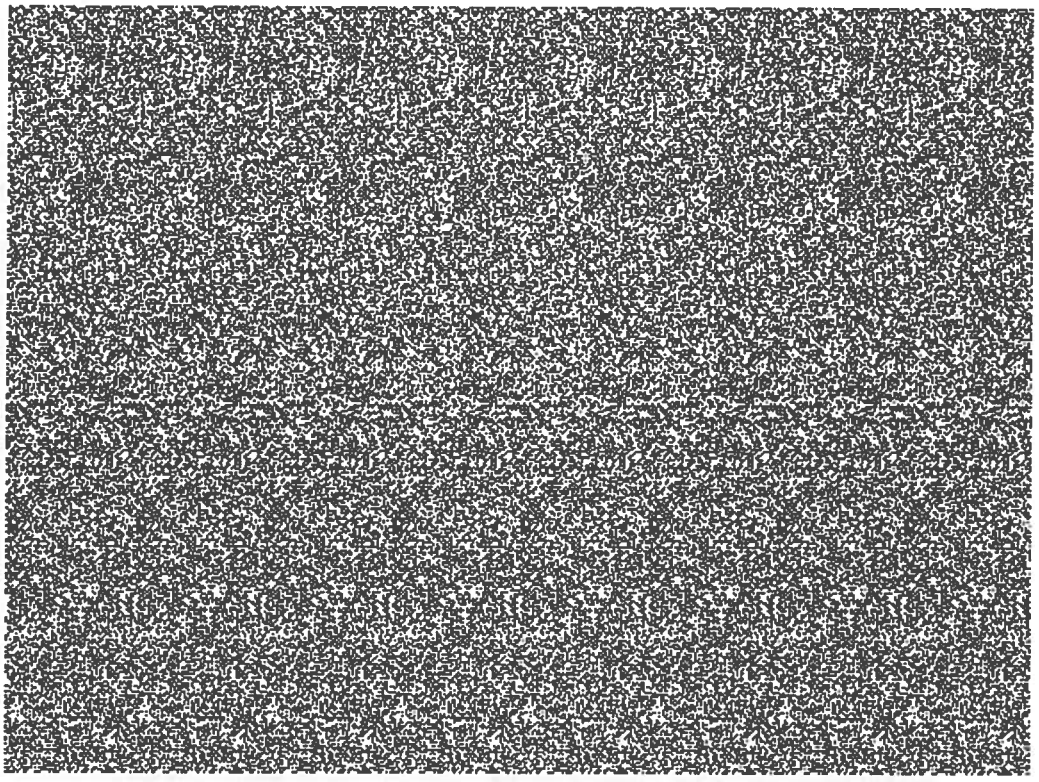


Figure A.4: **smallsq.hip** – a stereogram of a small square.

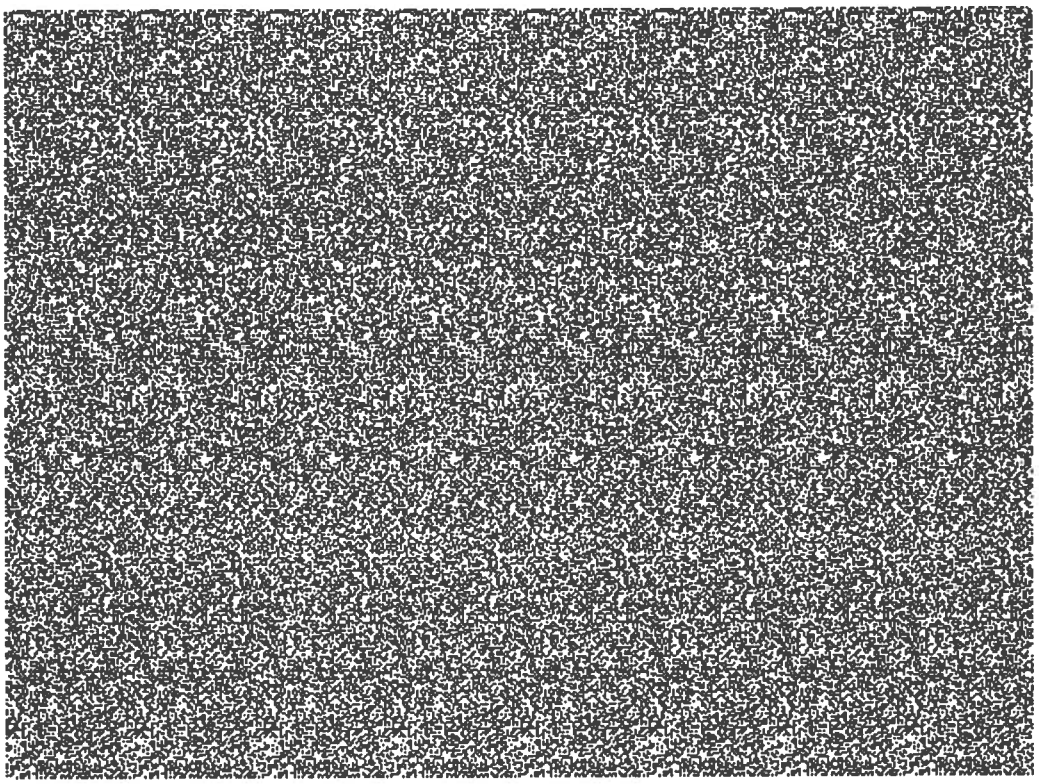


Figure A.5: **steps15.hip** – a stereogram of a set of 15 narrow steps.

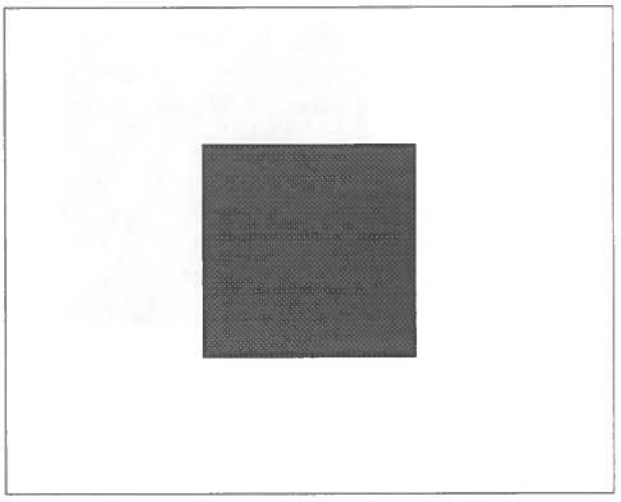


Figure A.6: **Answer to square.hip (Figure A.1).**

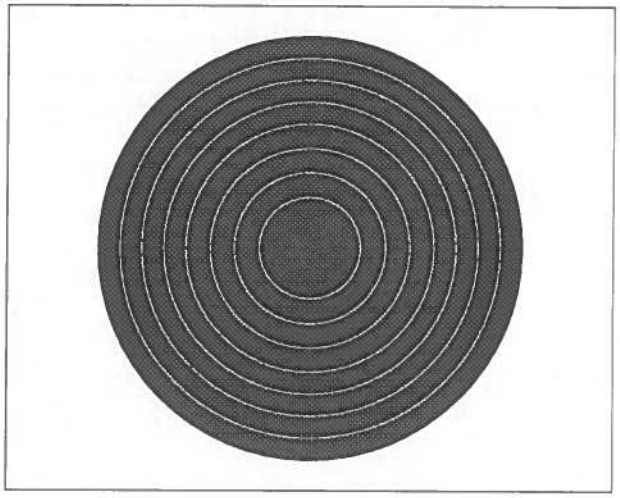


Figure A.7: **Answer to bowl.hip (Figure A.2).**

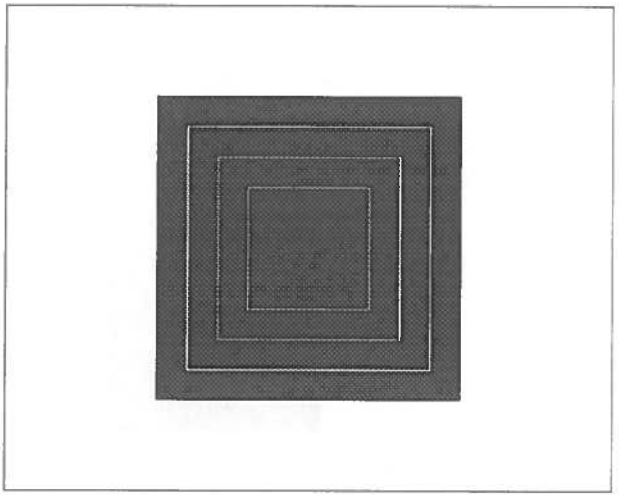


Figure A.8: **Answer to wedding.hip (Figure A.3).**

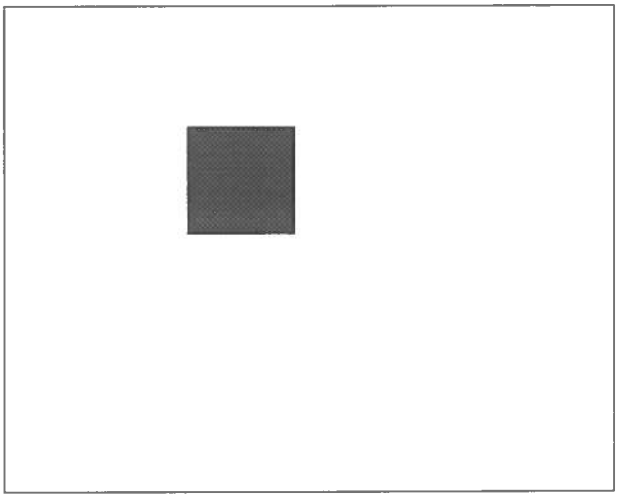


Figure A.9: **Answer to smallsq.hip (Figure A.4).**

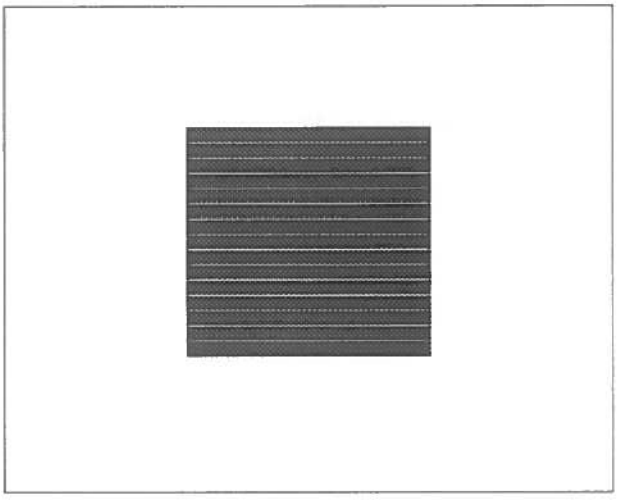


Figure A.10: **Answer to steps15.hip (Figure A.5).** The top step appears closest to the viewer.

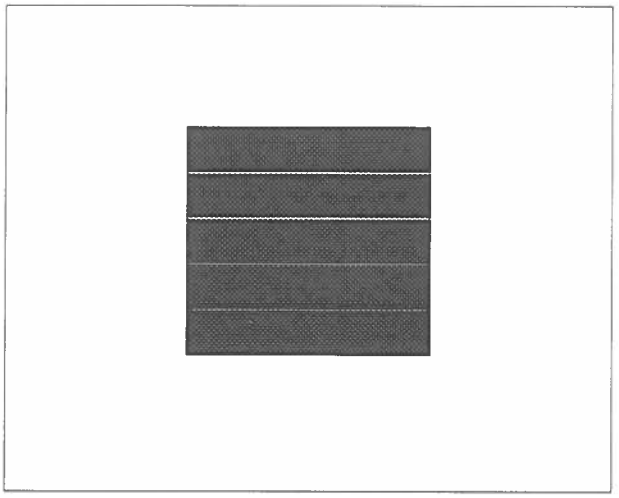
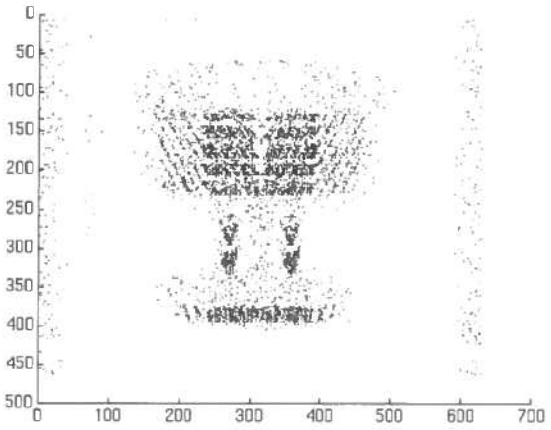


Figure A.11: **Answer to steps.hip (Figure 4.6).** The top step appears closest to the viewer.

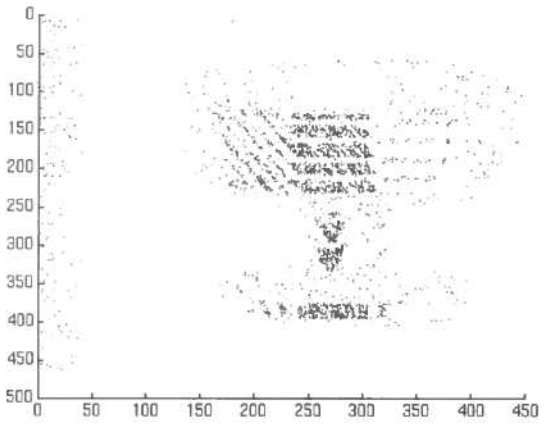
Appendix B

Additional Images

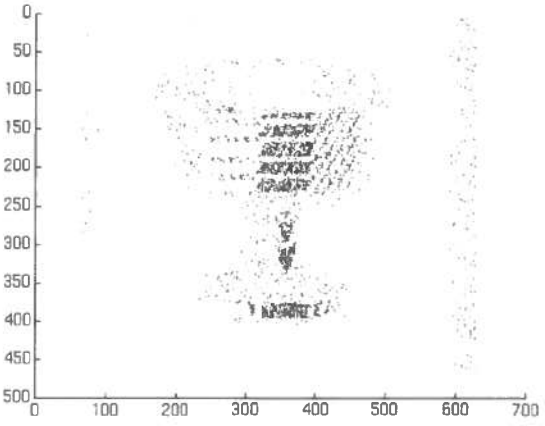
This appendix contains plots of the data sets referenced in Chapter 5, Table 5.1.



(a)



(b)



(c)

Figure B.1: **Matches From goblet.hip**. Figure (a) shows positive and negative matches of disparity $60<d<90$, Figure (b): positive matches only, $60<d<90$ and Figure (c): negative matches only, $60<d<90$. In each case, the axes denote two-dimensional position within the image.

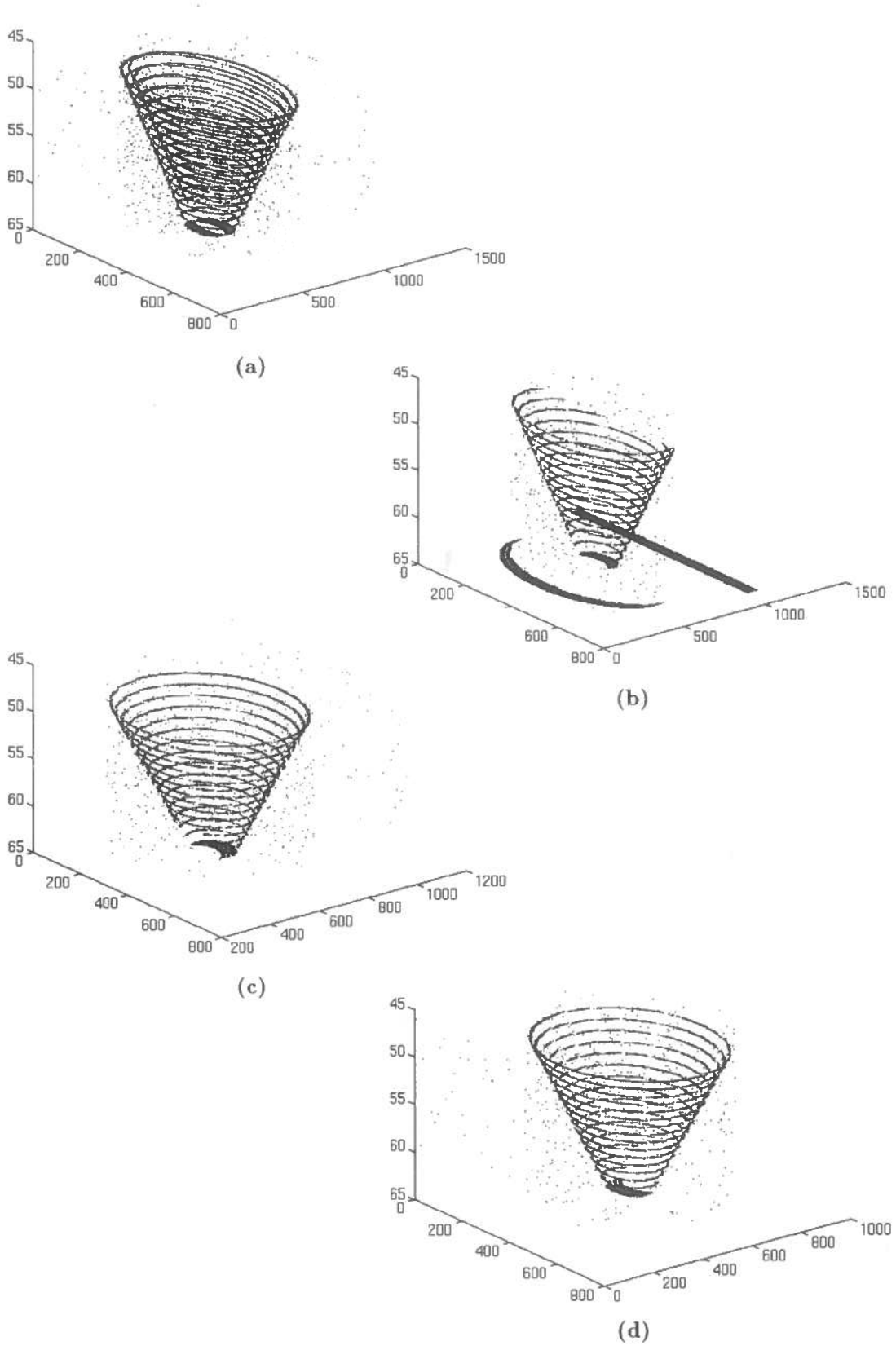
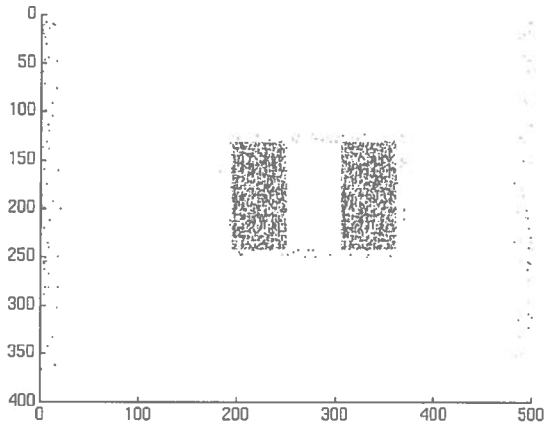
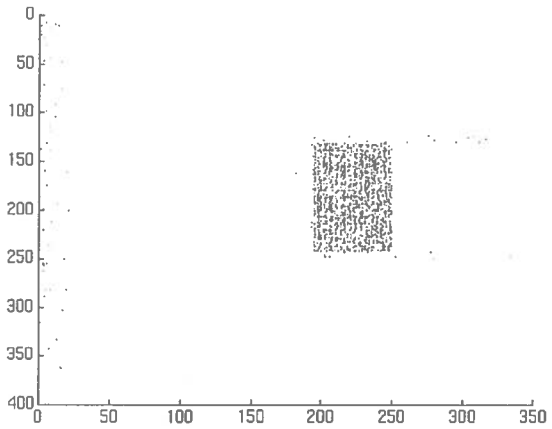


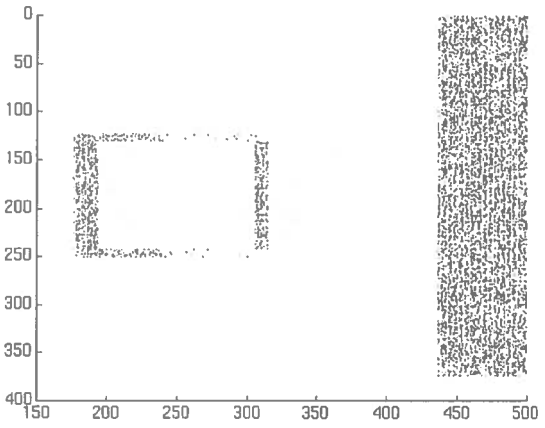
Figure B.2: **Matches From bowl.hip.** Figure (a) shows positive and negative matches of disparity $45<d<63$, Figure (b): negative matches only, $45<d<65$, Figure (c): negative matches only, $45<d<63$ and Figure (d): positive matches only, $45<d<63$. In each case, the vertical axis denotes disparity and the other two denote two-dimensional position within the image.



(a)



(b)



(c)

Figure B.3: **Matches From square.hip.** Figure (a) shows positive and negative matches of disparity $40<d<60$, Figure (b): positive matches only, $40<d<60$ and Figure (c): negative matches only, $40<d<65$. In each case, the axes denote two-dimensional position within the image.

Appendix C

Source Code

There follows listings of the source code for each of the modules constituting the system described in this dissertation.

```
/*
 *  main.c
 *
 *  John W.M. Wilson
 *  19th February 1995
 *
 *  This is the main function of the Grimson's algorithm code.
 *
 *  Run the program by typing: decster <hips_image>, where hips_image is
 *  a bi-level single image random dot stereogram in HIPS format.
 *
 */

#include <stdio.h>
#include <time.h>
#include "stereo.h"
#include "kernels.h"

void PrintPic();
void PrintZC();
void CalcDensity();
void PrintInfo();

main(argc,argv)
     int argc;
     char *argv[];
{
  int sqrdisp[1]={64};          /* disparity values for input stereograms, */
  int weddisp[4]={60,57,54,51};  /* used for stats purposes only. */

  int x, *d;
  short *convolved,             /* pointer to image after convolved */
        *unconvolved;           /* pointer to image after converted to shorts */

  short *convolved1,            /* pointer to images convolved by kernel1, */
        *convolved2,            /* kernel2,  */
        *convolved3,            /* kernel3,  */
        *convolved4;            /* kernel4.  */

  unsigned char *original,      /* pointers to input, */
                *cross,         /* zero-crossing,     */
                *dispmap,       /* disparity map      */
                *result;        /* and result images. */

  struct zc *zcmap1,            /* zero-crossing maps corresponding */
            *zcmap2,            /* to each of the sizes of filter.  */
            *zcmap3,
            *zcmap4;

  struct pt *ptlist;            /* zc and depth info from matcher */

  int k1size=9,                 /* side of each of the kernels in pixels */
      k2size=11,
      k3size=0,
      k4size=0;

  if(argc != 2)                 /* check the user has entered 2 words */
```

```
    {
      fprintf(stderr, "usage: decster <hips_image>\n");
      return(0);
    }

  ReadImage(&(original),argv[1],&no_rows,&no_cols);

  x=sizeof(short);
  unconvolved = (short *)malloc(x*picsize);      /* img converted to shorts */

  convolved = (short *)malloc(x*picsize);        /* image after convolution */

  result = (unsigned char *)malloc(picsize);     /* memory to store result  */

  cross = (unsigned char *)malloc(picsize);      /* image of zero-crossings */

  x=sizeof(struct zc);
  zcmap1 = (struct zc *)malloc(x*picsize);       /* zc map of convolved1    */

  CharToInt(original, unconvolved);
  Convolve(unconvolved, convolved, 9, &kernel1[0][0]);
  IntToChar(convolved, result);
  DetectZCs(convolved,zcmap1);
  WriteImage(result, "log.hip", argv, argc);

  MakeCross(zcmap1,cross);
  WriteImage(cross, "cross.hip", argv, argc);

  /*free(original);*/
  free(unconvolved);
  free(convolved);

  x=sizeof(struct pt);
  ptlist = (struct pt *)malloc(x*picsize);        /* store zc & depth info  */

  Match(zcmap1,63,ptlist);   /* second arg is upper bound on disp. searched */
  free(zcmap1);

  dispmap = (unsigned char *)malloc(picsize);

  DisparityMap(ptlist,dispmap);
  WriteImage(dispmap, "dispmap.hip", argv, argc);
  free(dispmap);

  d=&sqrdisp[0];               /* change this depending on input stereogram */
  /*PrintInfo(argv[1],original,ptlist,d);

  print_elapsed_time( ); */
}


/* CalcDensity: calculate the dot density of dots in the input stereogram.
 *
 */
void CalcDensity(pic)
     unsigned char *pic;
{
```

```
  int x,d,count=0;

  for (x=0; x<=picsize; x++) {
    if (*pic==0) {
      count++;
      pic++;
    }
    pic++;
  }
  d=(int) (100*(float)count/(float)picsize);
  printf("Dot density - %d percent\n(SIRDS only)\n\n", d);
}


/* PrintPic and PrintZC: utility functions for printing an image and a
 *                       zero-crossing map respectively.
 */
void PrintPic(pixel)
short *pixel;
{
  int x;

  for (x=0; x<picsize; x++) {
    printf("%d \n",*pixel);
    pixel++;
  }
}

void PrintZC(zcmap)
struct zc *zcmap;
{
  int x;

  for (x=0; x<10000; x++) {
    if (zcmap->mag != 0) {
      printf("%d %d %c %d \n", zcmap->row,zcmap->col,zcmap->sign,zcmap->mag);
      zcmap++;
    }
  }
}


/* PrintInfo: prints name, size, dot-density and matching statistics of the
 *            input image.
 */
void PrintInfo(imgname,original,ptlist,d)
     char imgname[];
     unsigned char *original;
     struct pt *ptlist;
     int *d;
{
  printf("%s\n\n", imgname);
  printf("Rows=%d\nCols=%d\nPicsize=%d\n\n", no_rows,no_cols,picsize);
  CalcDensity(original);
  MakeStats(ptlist,d,1);
}
```

```
/*
 *  LoG.c
 *
 *  John W.M. Wilson
 *  19th February 1995
 *
 *  This code forms the LoG convolution stage of the Stereo Algorithm
 *  [Marr & Poggio].
 *
 */

#include <stdio.h>
#include <math.h>
#include "stereo.h"


/* Convolve: used to convolve an image with a specified kernel.
 *
 */
void Convolve(data1,data2,kersize,kerptr)
     short *data1,*data2;
     int kersize;
     int *kerptr;

{
  int x, y;
  int row, col;
  float sum;
  int *kerval;

  kerval=kerptr;
/* convolve data1 with kernel to get data2 */
  for(col=0; col < no_cols-10; col++)
    for(row=0; row < no_rows-10; row++)
      {
        sum = 0.0;
        for(x=0; x<kersize; x++)
          for(y=0; y<kersize; y++)
            {
              sum += (*kerval)*
                      ((int)*INDEX(data1,no_cols,col+x,row+y));
              kerval++;
            }
        *INDEX(data2,no_cols,col,row) = (short)sum;
        kerval=kerptr;
      }
}
```

```
/*
 *  zero.c
 *
 *  John W.M. Wilson
 *  18th February 1995
 *
 *  This code forms the zero-crossing detection stage of the Stereo
 *  Algorithm [Marr & Poggio].
 *
 *  Given a pointer to a convolved image, this finds all the
 *  zero-crossings in the image.  It does this by testing for
 *  adjacent pixels of opposite sign OR three horizontally adjacent
 *  pixels where the middle one is zero and the first and third are
 *  of opposite sign.
 *
 *  When a zero-crossing is detected, its location, sign and
 *  magnitude are recorded.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "stereo.h"

void TestForZC();


/* DetectZCs: searches through the entire convolved image horizontally,
 *            looking for zero-crossings.
 */
void DetectZCs(convolved,zindex)
     short *convolved;
     struct zc *zindex;                  /* first elem of a zcmap */
{
  int count;
  short first,mid,last;                      /* pixels to compare */

  count=0;
  first=*convolved++;               /* set first, mid, last to   */
  mid  =*convolved++;               /* be the first three pixels */
  last =*convolved++;               /* of convolved image.       */

  while (count<picsize) {
    if (mid==(short)0)
      TestForZC(zindex,first,last,count);
    else
      TestForZC(zindex,first,mid,count);

    zindex++;
    first=mid;
    mid=last;
    last=*convolved++;
    count++;
  }
}
```

```
/* TestForZC: if zero-crossing, record:
 *             location - the row and column of 'first' in the array,
 *             sign - '+' if crossing is - to +, '-' if + to -,
 *             magnitude - in increments of 30 degrees.
 */
void TestForZC(zindex, pix1, pix2, count)
struct zc *zindex;
short pix1,pix2;
int count;
{
  if (pix1<(short)0 && pix2>(short)0) {
    zindex->row=div(count, no_cols).quot;
    zindex->col=div(count, no_cols).rem;
    zindex->sign='+';
    zindex->mag=abs((int)pix1)+abs((int)pix2);
  }
  else
    if (pix1>(short)0 && pix2<(short)0) {
      zindex->row=div(count, no_cols).quot;
      zindex->col=div(count, no_cols).rem;
      zindex->sign='-';
      zindex->mag=abs((int)pix1)+abs((int)pix2);
    }
}


/* MakeCross: constructs a visual representation of the zero-crossings
 *            from a zero-crossing map.
 */
void MakeCross(crossing, cross)
     struct zc *crossing;     /* ptr to start of a zcmap */
     unsigned char *cross;
{
  int c;

  for(c=0; c<picsize; c++) {
    if (crossing->mag > 0) {
      *cross=(unsigned char) 0;     /* black */
      cross++;
    }
    else {
      *cross=(unsigned char) 255;   /* white */
      cross++;
    }
    crossing++;
  }
}
```

```
/*
 *  matcher.c
 *
 *  John W.M. Wilson
 *  19th February 1995
 *
 *  This code performs the matching stage of the Stereo
 *  Algorithm [Marr & Poggio].
 *
 */

#include <stdio.h>
#include <math.h>
#include "stereo.h"

#define HOOD_SIZE   50

void FindUnambMatches();
void DisambMatches();
void MakeDepthInfo();
void PrintPts();
void XNoise();
void YNoise();
void ZNoise();
void Stats();
int DomSign(struct zc *z, struct zc *zcmap);


/* Match: for each convolved image, we must match it's set of
 *        zero-crossings with itself.  Given the size of w and a
 *        pointer to a zcmap, this procedure carries out this task.
 *        Given also a ptr to the point list, it fills this too.
 */
void Match(zcmap,w,ptlist)
     struct zc *zcmap;
     int w;
     struct pt *ptlist;
{
  FindUnambMatches(zcmap,w);
  /* DisambMatches(zcmap,w); */
  MakeDepthInfo(zcmap, ptlist);
  XNoise(ptlist);
  YNoise(ptlist);
  ZNoise(ptlist);
  PrintPts(ptlist);
}


/* FindUnambMatches: finds all zero-crossings which have exactly 1 match
 *                   (unambiguous matches), records the match and the
 *                   resulting disparity.  Ignores all ambiguous matches.
 */
void FindUnambMatches(zcmap,w)
     struct zc *zcmap;
     int w;
{
  int matches;                    /* count number of matches for a z-c */
```

```
  int d;                                        /* temp store for disparity */

  struct zc *zinfo, *match;
  struct zc *cinfo;             /* points to candidate matches for zinfo */

  for (zinfo=zcmap; zinfo<zcmap+picsize; zinfo++) {

    if (zinfo->mag != 0) {           /* it is a zc (some will be blank) */
      matches=0;

      for (cinfo=zinfo-w; cinfo<=zinfo+w; cinfo++) {

        if (cinfo->mag!=0 &&                    /* IF it is a z-c,        */
            zinfo->row==cinfo->row &&           /* its on the same row,   */
            zinfo->col!=cinfo->col &&           /* its not the same z-c,  */
            zinfo->sign==cinfo->sign &&         /* their signs match and  */
            zinfo->mag==cinfo->mag) {           /* their magnitudes match */
                                      /* THEN we have a match.  */
          matches++;
          if (matches==1) match=cinfo;
        }
        if (matches>1) break;
      }

      if (matches==1) {
        zinfo->match=match;
        cinfo->match=zinfo;  /* record symmetrically opposite match too */

        if (match->col < zinfo->col)          /* ie. divergent disparity */
          d = 0-(zinfo->col - match->col);
        else                                              /* ie. convergent */
          d = match->col - zinfo->col;

        if (d>46 || d<-46) {     /* ignore matches with small disparity */
          zinfo->disp=d;
          cinfo->disp=-d;     /*  opposite match has opposing disparity */
        }
      }
    }
  }
}


/* DisambMatches: the purpose of this procedure is to disambiguate any
 *                remaining zero-crossings which have multiple matches.
 *                All unambiguous matches have already been found by
 *                FindUnambMatches, so any matches remaining will be
 *                ambiguous.  We search through the zcmap and for each
 *                ambiguous zc, if it has exactly 1 match which has the
 *                same disparity sign as the dominant neighbourhood sign,
 *                then we accept that match, otherwise we leave it
 *                ambiguous.
 */
void DisambMatches(zcmap,w)
     struct zc *zcmap;
     int w;
{
```

```
  int dm,cm;                /* count no of divergent/convergent matches */

  struct zc *zinfo;
  struct zc *cinfo;            /* points to candidate matches for zinfo */
  struct zc *dmatch,*cmatch;       /* points to one match of each sign */

  printf("\n=====DISAMBIGUATING MATCHES=====\n");

  for (zinfo=zcmap; zinfo<zcmap+picsize; zinfo++) {

    if (zinfo->mag!=0 &&             /* it is a zc (some will be blank) */
        zinfo->disp==0) { /* it has not already been assigned disparity */
      dm=0; cm=0;

      for (cinfo=zinfo-w; cinfo<=zinfo+w; cinfo++) {

        if (cinfo->mag!=0 &&                  /* IF it is a z-c,        */
            zinfo->row==cinfo->row &&         /* its on the same row,   */
            zinfo->col!=cinfo->col &&         /* its not the same z-c,  */
            zinfo->sign==cinfo->sign &&       /* their signs match and  */
            zinfo->mag==cinfo->mag) {         /* their magnitudes match */
                                              /* THEN we have a match.  */

          if (cinfo->col > zinfo->col) {      /* ie. divergent */
            dmatch=cinfo;
            dm++;
          }
          if (cinfo->col < zinfo->col) {      /* ie. convergent */
            cmatch=cinfo;
            cm++;
          }
        }
        if (!(dm==1 && cm==1)) {
          if (dm==1 && DomSign(zinfo,zcmap)==0) {
            zinfo->match=dmatch;
            zinfo->disp = 0-(zinfo->col - dmatch->col);      /* divergent */
          }

          if (cm==1 && DomSign(zinfo,zcmap)==1) {
            zinfo->match=cmatch;
            zinfo->disp = cmatch->col - zinfo->col;         /* convergent */
          }
        }
      }
    }
  }
}


/* DomSign: function used to find the dominant sign of disparity in the
 *          neighbourhood (the extent of which is defined by HOOD_SIZE)
 *          of a zero-crossing in an attempt to assign it an unambiguous
 *          match.  DomSign examines sign of disparity only of zc's with
 *          unambiguous matches - only these zc's have non-zero disparity
 *          values.  Sign refers to whether disparity is convergent (con)
 *          or divergent (div).
 *          Returns 1 if domsign is convergent, 0 if divergent.
```

```
 */
int DomSign(z, zcmap)
     struct zc *z;          /* zc in question */
     struct zc *zcmap;      /* ptr to start of zcmap from which z comes */
{
  int x,y;
  int con,div;              /* scores for disparity sign */
  struct zc *nb;

  con=0; div=0;

printf("ds");
  for (x=0; x<=HOOD_SIZE; x++) {
    for (y=0; y<=HOOD_SIZE; y++) {
      nb = INDEX(zcmap,no_cols,z->col+x,z->row+y);
      if (nb->disp > 0) con++;
      if (nb->disp < 0) div++;
    }
  }

  if (con>div) return 1;
  if (con<div) return 0;
  else return 5;             /* ie. no unamb matches in hood or con=div */
}


/* MakeDepthInfo: given a map of all disambiguated matching zero-crossings,
 *                this puts the matches into an array, each element of the
 *                form (x,y,d) denoting a zero-crossing at position (x,y)
 *                in the image with associated disparity d.
 */
void MakeDepthInfo(z, pt)
     struct zc *z;
     struct pt *pt;
{
  int x;

  for (x=0; x<picsize; x++) {
    if (z->disp!=0) {
      pt->x = z->col;
      pt->y = z->row;
      pt->d = z->disp;

      pt++;
      z++;
    }
    else
      z++;
  }
}


/* XNoise: Filters noise from the image as follows - for each x coordinate in
 *         the input image (i.e. 0-no_cols) scan the point list counting how
 *         many points have that x value as their x coordinate.  Then if this
 *         count is lower than some threshold, go through the point list
 *         again, setting the disparity of all points with that x value to 0.
```

```
 *         (All points with 0 disparity (z coordinate) will be suppressed in
 *         the final list of points output.)
 */
void XNoise(pt)
     struct pt *pt;
{
  int i,j,count,threshold;
  struct pt *point;

  threshold = 10;

  for (i=0; i<no_cols; i++) {
    point=pt;
    count=0;
    for (j=0; j<picsize; j++) {
      if (point->x==i) count++;
      point++;
    }
    if (count<threshold) {
      point=pt;
      for (j=0; j<picsize; j++) {
        if (point->x==i) point->d = 0;
        point++;
      }
    }
  }
}


/* YNoise: Filters noise from the image as follows - for each y coordinate in
 *         the input image (i.e. 0-no_rows) scan the point list counting how
 *         many points have that y value as their y coordinate.  Then if this
 *         count is lower than some threshold, go through the point list
 *         again, setting the disparity of all points with that y value to 0.
 *         (All points with 0 disparity (z coordinate) will be suppressed in
 *         the final list of points output.)
 */
void YNoise(pt)
     struct pt *pt;
{
  int i,j,count,threshold;
  struct pt *point;

  threshold = 15;

  for (i=0; i<no_rows; i++) {
    point=pt;
    count=0;
    for (j=0; j<picsize; j++) {
      if (point->y==i) count++;
      point++;
    }
    if (count<threshold) {
      point=pt;
      for (j=0; j<picsize; j++) {
        if (point->y==i) point->d = 0;
        point++;
```

```
      }
    }
  }
}


/* ZNoise: Filters noise from the image as follows - for each z coordinate in
 *         the input image (i.e. 0-no_rows) scan the point list counting how
 *         many points have that z value as their y coordinate.  Then if this
 *         count is lower than some threshold, go through the point list
 *         again, setting the disparity of all points with that z value to 0.
 *         (All points with 0 disparity (z coordinate) will be suppressed in
 *         the final list of points output.)
 */
void ZNoise(pt)
     struct pt *pt;
{
  int i,j,count,threshold;
  struct pt *point;

  threshold = 10;

  for (i=-100; i<100; i++) {
    point=pt;
    count=0;
    for (j=0; j<picsize; j++) {
      if (point->d==i) count++;
      point++;
    }
    if (count<threshold) {
      point=pt;
      for (j=0; j<picsize; j++) {
        if (point->d==i) point->d = 0;
        point++;
      }
    }
  }
}


/* PrintPts: Prints each of the 3D match points on the screen.
 *
 */
void PrintPts(pt)
     struct pt *pt;
{
  int x;

  for (x=0; x<picsize; x++) {               /* +=10 for sparse output */
    /* if (pt->d<0) pt->x = pt->x + pt->d; */
                /* if match has -ve disparity, shift it d to the left */
    if (pt->d!=0) printf("%d\t%d\t%d\n",pt->x,pt->y,pt->d);
    pt++;                                   /* +=10 for sparse output */
  }
}
```

```
/* MakeStats: compute matching statistics for each disparity value in
 *            the input stereogram image.
 */
void MakeStats(ptlist, disp, n)
     struct pt *ptlist;
     int *disp;
     int n;
{
  int *d;
  int tl,wr,ex,op;
  int total,          /* total number of matches */
      wrong,        /* number matched wrongly */
      exact=0,        /* number exactly right  */
      onepix=0;       /* number one pixel out */
  float propwrong;    /* percentage of wrong matches */

  printf("\n====MATCHING STATISTICS====\n");
  for (d=disp; d<disp+n; d++) {
    Stats(ptlist, *d, &tl, &ex, &op);
    total=tl;
    exact+=ex;
    onepix+=op;
  }
  wrong=total-exact-onepix;
  propwrong=100*(float)wrong/(float)total;

  printf("\nTotal=%d\nExact=%d\nOne Pixel=%d\n", total,exact,onepix);
  printf("Wrong=%d\n\%Percentage Wrong=%.2f\n", wrong,propwrong);
}


/* Stats: compute matching statistics for one disparity value.
 *
 */
void Stats(ptlist, disp, tl, ex, op)
     struct pt *ptlist;
     int disp,*tl,*ex,*op;
{
  int x=0;
  int exact=0,        /* number exactly right for this disp  */
      onepix=0;       /* number one pixel out for this disp */

  struct pt *p;

  p=ptlist;
  while (p->d != 0) {
    if ((p->d==disp) || (p->d==0-disp)) exact++;
    else
      if ((p->d==disp+1) || (p->d==disp-1)) onepix++;
    p++;
  }
  *tl=p-ptlist;
  *ex=exact;
  *op=onepix;
}
```

```
/* DisparityMap: generate a disparity map from depth information in
 *               ptlist.  If the disparity of a match is negative, the
 *               corresponding pixel is given a slightly darker colour.
 */
void DisparityMap(ptlist, dispmap)
     struct pt *ptlist;
     unsigned char *dispmap;
{
  struct pt *p;
  int r,c;

  p=ptlist;
  for (r=0; r<no_rows; r++) {
    for (c=0; c<no_cols; c++) {
      if ((p->x==c) && (p->y==r)) {
        if (p->d > 0) {
          *dispmap=(unsigned char) 255;
          dispmap++;
          p++;
        }
        else {
  *dispmap=(unsigned char) 200;
  dispmap++;
  p++;
        }
      }
      else {
        *dispmap=(unsigned char) 0;
        dispmap++;
      }
    }
  }
}
```

```
/*
 *  fileops.c
 *
 *  John W.M. Wilson
 *  19th February 1994
 *
 *  This code reads and writes hips image files.
 *  When a file is read in, it is converted from char to int representation
 *  and vice versa, before it is saved.
 *
 */

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <hipl_format.h>
#include "stereo.h"


char Progname[]="segment";            /* to satisfy HIPS */


void ReadImage(ptrdata, filename, ptr_rows, ptr_cols)
char filename[];
unsigned char ** ptrdata;
int *ptr_rows, *ptr_cols;
{
    int fd;
    struct header hd;
    unsigned char * data;

    fd = open(filename, 0);
    if(fd == -1)
      {
        fprintf(stderr, "ReadImage: can't open file\n");
        exit(0);
      }
    fread_header(fd, &hd);              /* hips library function */
    picsize = hd.cols * hd.rows;
    data = (unsigned char *)malloc(picsize);

    if(data == (unsigned char *)0)
      {
        fprintf(stderr, "ReadImage: not enough memory\n");
        exit(0);
      }
    pread(fd, data, picsize);           /* hips library function */
    close(fd);
    *ptrdata = data;
    *ptr_rows = hd.rows;
    *ptr_cols = hd.cols;
}


void WriteImage(data, filename, argv, argc)
unsigned char *data;
char filename[];
```

```
char *argv[];
int argc;
{
    struct header hd;
    int fd;
    long now_time;
    char null ='\0';
    fd=creat(filename,0755);

    if(fd==-1)
      {
fprintf(stderr, "WriteImage: can't create file\n");
exit(0);
      }

/* generate hips header */
    now_time=time((long *)0);
    init_header(&hd,
                "isis",
                filename,
                1,
                ctime(&now_time),
                no_rows,
                no_cols,
                8,
                0,
                PFBYTE,     /* pixel format */
                &null,
                &null);
    update_header(&hd, argc, argv);      /* hips lib functions */
    update_desc(&hd, "Output of isis write command");
    fwrite_header(fd,&hd);

/* write pic to file */
    write(fd, data, picsize);
    free(data);
    free(data);
    close(fd);
}


void CharToInt(data1, data2)
     unsigned char *data1;
     short *data2;
{
  register int i;

  for(i=0; i<picsize; i++)
    data2[i]=(short)data1[i];
}

void IntToChar(data1, data2)
     short *data1;
     unsigned char *data2;
{
  register int i;
  short largest,smallest,factor;
```

```
  largest=0;
  smallest=0;
  for(i=0; i<picsize; i++) {
    if(data1[i] > largest)
      largest=data1[i];
    else if(data1[i] < smallest)
      smallest=data1[i];
  }

  factor=(largest-smallest)/(short)255;
  for(i=0; i<picsize; i++)
    data2[i] = (unsigned char)floor(abs(data1[i]/(short)factor));
}
```

```
/*
 *  stereo.h
 *
 *  John W.M. Wilson
 *  19th February 1995
 *
 *  Header file for Grimson's algorithm code.
 *
 */

#define INDEX(NAME, COLS, X, Y) ((NAME) + ((COLS) * (Y) + (X)))

/* Shared material from zero.c */
void DetectZCs();
void MakeCross();

/* structure to hold info on one zero-crossing */
struct zc {
  int row;
  int col;              /* location in image of z-c */
  char sign;            /* direction of change rep'd by z-c */
  int mag;              /* magnitude of z-c */
  struct zc *match;     /* unambiguous match (if found)   */
  int disp;             /* disparity if unamb match found */
};


/* Shared material from matcher.c */
void Match();
void MakeStats();
void DisparityMap();

struct pt {              /* holds depth info gathered */
  int x;                 /* during matching: (x,y,d)  */
  int y;                 /* denotes a z-c at position */
  int d;                 /* (x,y) with disparity d.   */
};


/* Shared material from LoG.c */
void Convolve();


/* Shared material from fileops.c */
void ReadImage();
void WriteImage();
void CharToInt();
void IntToChar();

int no_rows, no_cols;     /* number of rows and cols in input */
int picsize;              /* no_rows x no_cols */
```

```
/*
 * kernels.h
 *
 * The convolution kernels.
 * kernel1 is the one used in the current implementation, the others are
 * included for experimentation.
 *
 */

/* Kernel from HIPR WWW page, sigma=1.4 */
int kernel1[9][9] = { {0,0,3,2,2,2,3,0,0},
      {0,2,3,5,5,5,3,2,0},
      {3,3,5,3,0,3,5,3,3},
      {2,5,3,-12,-23,-12,3,5,2},
      {2,5,0,-23,-40,-23,0,5,2},
      {2,5,3,-12,-23,-12,3,5,2},
      {3,3,5,3,0,3,5,3,3},
      {0,2,3,5,5,5,3,2,0},
      {0,0,3,2,2,2,3,0,0} };

int kernel2[11][11]={ {-1,-8,-39,-116,-219,-270,-219,-116,-39,-8,-1},
      {-8,-61,-270,-736,-1286,-1533,-1286,-736,-270,-61,-8},
      {-39,-270,-1073,-2478,-3567,-3847,-3567,-2478,-1073,-270,-39},
      {-116,-736,-2478,-3988,-2276,-229,-2276,-3988,-2478,-736,-116},
      {-219,-1286,-3567,-2276,8643,16977,8643,-2276,-3567,-1286,-219},
      {-270,-1533,-3847,-229,16977,29426,16977,-229,-3847,-1533,-270},
      {-219,-1286,-3567,-2276,8643,16977,8643,-2276,-3567,-1286,-219},
      {-116,-736,-2478,-3988,-2276,-229,-2276,-3988,-2478,-736,-116},
      {-39,-270,-1073,-2478,-3567,-3847,-3567,-2478,-1073,-270,-39},
      {-8,-61,-270,-736,-1286,-1533,-1286,-736,-270,-61,-8},
      {-1,-8,-39,-116,-219,-270,-219,-116,-39,-8,-1} };

int kernel3[11][11]={ {0,0,0,0,0,0,0,0,0,0,0,},
      {0,0,0,-1,-1,-2,-1,-1,0,0,0,},
      {0,0,-1,-3,-4,-5,-4,-3,-1,0,0,},
      {0,-1,-3,-5,-3,0,-3,-5,-3,-1,0,},
      {0,-1,-4,-3,12,24,12,-3,-4,-1,0,},
      {0,-2,-5,0,24,41,24,0,-5,-2,0,},
      {0,-1,-4,-3,12,24,12,-3,-4,-1,0,},
      {0,-1,-3,-5,-3,0,-3,-5,-3,-1,0,},
      {0,0,-1,-3,-4,-5,-4,-3,-1,0,0,},
      {0,0,0,-1,-1,-2,-1,-1,0,0,0,},
      {0,0,0,0,0,0,0,0,0,0,0,} };

/* Haralick's kernel sigma=1.4 (p349) */
int kernel4[11][11]={ {0,0,0,1,1,2,1,1,0,0,0},
      {0,0,2,4,8,9,8,4,2,0,0},
      {0,2,7,15,22,23,22,15,7,2,0},
      {1,4,15,24,14,1,14,24,15,4,1},
      {1,8,22,14,-52,-103,-52,14,22,8,1},
      {2,9,23,1,-103,-178,-103,1,23,9,2},
      {1,8,22,14,-52,-103,-52,14,22,8,1},
      {1,4,15,24,14,1,14,24,15,4,1},
      {0,2,7,15,22,23,22,15,7,2,0},
      {0,0,2,4,8,9,8,4,2,0,0},
      {0,0,0,1,1,2,1,1,0,0,0} };
```