An Algorithm and Architecture for Computing Curvature Maps from Range Images

Morie Eve Margret Malowany

B Eng. (Electrical, McGill University) 1987

Department of Electrical Engineering

McGill University

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Engineering

March 1989

AN ALGORITHM AND ARCHITECTURE FOR COMPUTING CURVATURE FROM RANGE

AN ALGORITHM AND ARCHITECTURE FOR COMPUTING CURVATURE FROM RANGE

Abstract

This thesis examines a method for computing maps of mean and Gaussian curvature from range images. The treatment is in two parts.

The first part of the thesis presents an experimental study of an algorithm for computing the curvatures. This study demonstrates the performance of the algorithm on artificial and on real range images. The mean and Gaussian curvatures which the algorithm extracts are second-order surface characteristics from the theory of differential geometry which have desirable properties for range-image understanding.

The second part of the thesis presents a hardware architecture for implementing the algorithm. The novel aspect of this architecture is a new floating-point VLSI processor for fit-error computations. The expected performance of the architecture is evaluated. Computing the curvatures using special-purpose hardware is aimed at lessening the low-level processing bottleneck that is often a problem in image-understanding schemes.

Résumé

Cette thèse présente une méthodologie pour extraire les courbures des surfaces dans les images télémétriques. Le traitement est en deux parties.

La première partie de la thèse présente une étude experimentale d'un algorithme pour extraire les courbures. Cette étude démontre la performance de l'algorithme sur des données de télémétrie artificiels et réelles. La courbure moyenne et la courbure gaussienne que l'algorithme produit viennent de la théorie de géométrie différentielle; ces courbures possèdent des attributs désirables pour l'analyse des images télémétriques.

La deuxième partie de la thèse présente l'architecture d'un système réalisant l'algorithme. L'aspect spécial de cette architecture est un élément TGEI nouveau point-flottant pour calculer les erreurs d'approximations. Une évaluation est faite des performances prévues de l'architecture. L'utilisation d'une architecture spécialisée pour extraire les courbures devrait diminuer un embouteillage de calcul préliminaire qui se produit fréquemment dans l'analyse des images.

Acknowledgements

I wish to thank all those who helped make this work meaningful I thank my supervisor, Dr. A.S. Malowany, for the guidance and inspiration he provided over the course of my studies. I thank the professors, professional staff, and graduate students of the McRCIM research community who provided support, encouragement, constructive criticism, and motivation toward excellence. I thank NRC, CMC, NSERC, and FCAR for their co-operative efforts and financial support which have facilitated the growth of research in Electrical Engineering here at McGill. Finally, I thank my family for their understanding, patience, and support during my studies.

Contents

List of Figures

Chapter 1 Introduction

1.1 Objective

The objective of this thesis is to demonstrate the robustness and viability of a method for computing Gaussian and mean curvature maps from range data [Malo88c] and to propose a sound and efficient hardware architecture to implement this method [Malo89a][Malo89b]. Complementing this objective, efforts are made to document the usefulness of curvature quantities in surface characterization as well as the advantage of VLSI implementations in machine vision using examples from the literature. We aim to build a case for implementing a modified version of the curvature-extraction method using VLSI elements.

1.2 How Does The Objective Fit Into the Larger Scheme of Things?

In recent years there have been dramatic advancements in the field of range-finding sensor technology. The resulting cheaper, more-accurate, and lighter-weight sensors [Riou86] have been attracting increasing interest in application areas where machine vision is involved including robotics [Sica87], computer-aided design [Naik88], and even dentistry [Sams87].

Sensors such as laser range-finders provide valuable information about the environment. This information can be used to guide and validate activities of robots or inform humans about how to manipulate remote-controlled equipment in hostile environments. Range image understanding [Besl85] attempts to derive the identities and positions of objects in the environment from the range-finder input. This is the type of high-level description needed for guidance and validation decisions. Deriving this information is a complex, multi-step process.

Range images have the advantage over intensity images of containing explicit depth information. The effects of the generally-unknown lighting sources in the environment do not have to be removed from the image to deduce depth. Hence, the problem of understanding range images is said to be "well-posed" in contrast to the related problem for intensity images which is "ill-posed" [Besl85].

Range image understanding can be seen to require three major steps. These include surface characterization, refinement and organization of the resulting surface information into an object description, followed by object recognition. Surface characterization aims to extract quantities from the data which will uniquely identify, or at least narrow the set of possibilities for, the constituent surfaces in the image. Image segmentation and creation of an object description can then be performed. Subsequently, object recognition can match the object descriptions to object models stored in a database and determine the transformation relating the model to the instance in the range image. Model formation [Aubr89] is an important pre-requisite for object recognition, though here it is implicitly assumed to have been done in advance. If done using the above three steps, range image understanding becomes a bottom-up hierarchical process as shown in Figure 1.1. It is important for a general system to be data-driven by starting at the bottom with low-level operations on the range data rather than assuming a pre-determined object form at the higher level. An intelligent system may determine that additional views are required to understand the image and request such additional views.

Expert systems are attracting increasing attention as a paradigm for higher-level reasoning, for example in the repair of circuit boards in a robotic workcell [Malo88a] [Malo88b] [Malo88c]. The information supplied by the system developed in this thesis will

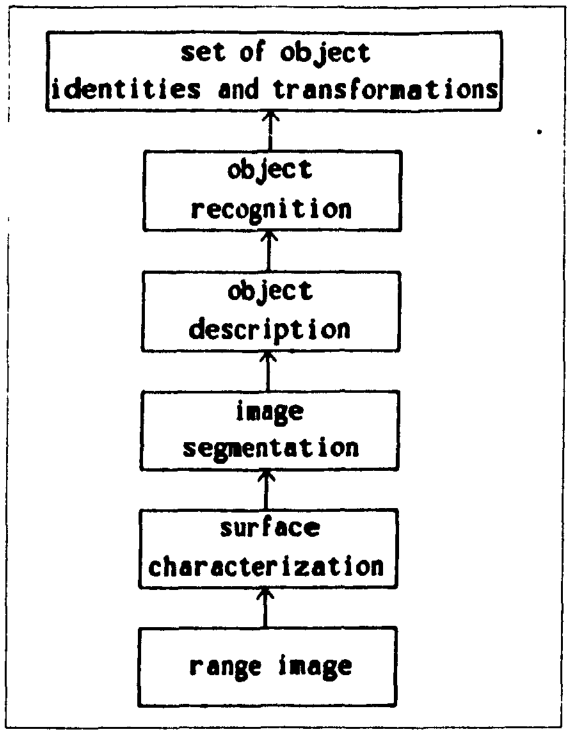


Figure 1.1 Bottom-up Hierarchical Processing for Range Image Understanding

pass its results up the hierarchy of Figure 1.1. Since expert systems have already been successfully applied in machine vision [Nazi84][Ferr81], this methodology is envisaged for realizing the higher levels of the hierarchy of Figure 1.1. The inclusion of sensor fusion, i.e. the synthesis of sensor data obtained from multiple sources, should greatly enhance the effectiveness of such expert systems in understanding the environment. For example, color video images [Delc88] could supplement the information obtained from the range images.

In pursuit of range image understanding, most efforts to date have extracted information for surface characterization from range images in software. This has resulted in implementations with long processing times. Hence, efforts are underway to implement some of the necessary low-level image-processing operations as VLSI circuits in order to improve processing speeds. In industrial contexts, such as inspection of manufactured parts, low-cost solutions are required. A solution featuring special-purpose VLSI elements must be high-volume to make economic sense due to the high initial cost of producing a new chip. However, the cost per unit falls rapidly as more are produced and sold. Market-place issues such as market share and product distribution are important in determining whether or not a chip should be produced. Since there is a potentially large market for sensor-assisted robots in manufacturing, a VLSI solution for the robotics market is promising. This contrasts with more research-oriented applications (e.g. processing of topographic map data or biomedical data) where very powerful, general-purpose computers are a viable alternative.

Curvature information such as the mean and Gaussian curvatures (or the related principal curvatures) of a surface proves to be useful in surface characterization. There is even evidence that curvature quantities play a part in biological vision systems [Dobb87]. From differential geometry, it is known that curvature quantities can be used to classify surface regions as one of eight primitive types, including the flat surface type which is a special (degenerate) case of a curved surface. Furthermore the curvature values for a surface are invariant with respect to how the surface is embedded in space and with respect to the position of the observer as long as the object is visible. This invariance can be exploited to simplify the matching processes of object recognition.

The curvature computations are based on theory from differential geometry which is valid only for continuous surfaces. A typical application environment, however, be it a man-made bin of parts or a natural outdoor scene, features many discontinuities in addition to continuous surface segments Hence, discontinuity information, such as maps of depth discontinuities and surface-orientation discontinuities, must be considered along with curvature information to get meaningful results in later processing stages such as image segmentation.

1.3 Overview of Thesis

This thesis describes a method for computing Gaussian and mean curvature maps from range data and a hardware architecture for efficient realization of this method featuring a new systolic floating-point VLSI fit-error cell.

In this chapter, the objective and context of the work have been described. The role that curvature quantities can play in surface characterization and range image understanding has also been outlined.

Chapter 2 presents the curvature extraction algorithm including some background on curvatures in differential geometry. Approaches to surface characterization in the literature involving curvatures and alternate methods are discussed The original and modified versions of the algorithm for :tracting maps of mean and Gaussian curvature from a range image are detailed. The motivation for the introduction of the modified algorithm is to facilitate a VLSI circuit implementation Results of experiments are presented including studies with artificial range data showing the response of the original and modified algorithms to noise and to the window operator size. The performance of the modified algorithm on real range data is also presented.

Chapter 3 deals with the hardware architecture proposed for computing the curvatures. Some terminology related to the use of VLSI processors in machine vision is introduced and examples from the literature are given. The architecture of a dedicated hardware environment proposed for the curvature extraction is outlined and each of the component subsystems is treated emphasizing the expected performance. In particular, the new floating-point VLSI processor for fit-error computation, the most novel aspect of the hardware development, is presented and evaluated The overall performance estimate for the dedicated hardware environment is compared to that obtained on a Sun workstation in a networked, multi-user environment.

Chapter 4 presents a discussion of the results obtained in the algorithm and architecture studies of Chapters 2 and 3. The relationship between the two sets of results and their implications for future work are stressed.

Chapter 5 summarizes the conclusions from the results presented on the two major foci of this thesis: the algorithm and the architecture.

Chapter 2 Curvature Extraction Algorithm

This chapter begins with a brief treatment of curvatures in differential geometry, followed by treatments of previous work, the curvature extraction algorithm, and the experiments. The differential geometry is discussed first as the terminology will be helpful in understanding several of the the other approaches in the literature as well as our own approach.

2.1 Curvatures in Differential Geometry

The field of *differential geometry* studies the behavior of curves and surfaces[†]. This field can be divided into *classical differential geometry*, which studies local properties of surfaces determined only by the behavior of the surface in the neighborhood of a point, and *global differential geometry* which concerns the behavior of an entire surface. The classical differential geometry, or that involving local properties of surfaces, is of primary interest for surface characterization because it can handle the case of surfaces which are partly occluded.

Differential geometry uses the formalisms of calculus and so assumes that all surfaces under consideration are *continuous* and *differentiable* (at least up to the second

† As in [DoCa76] p. 1 While DoCarmo's book gives a formal treatment, Chapter IV of [Hilb52] gives a nice intuitive treatment of the subject.

order.) When analyzing range data, the surface information exists as a set of *samples* rather than as a known continuous function. It will be assumed that the samples are taken from some continuous and differentiable function. This assumption is valid locally except at *discontinuities* where different surfaces meet. At such points, higher-level processing must rely on edge information obtained from the depth map to assist the continuity-dependent curvature information in interpreting the image.

Typically, a general *surface* in 3D space is expressed parametrically:

$$S = \left\{(x(u,v), y(u,v), z(u,v)) : (u,v) \in D \subseteq R^2\right\} \tag{2.1}$$

However, the range images to be analyzed in this study are available as samples of depth, z, taken at regularly spaced points on a grid in x and y. Hence the surfaces to be analyzed can be expressed in the simplified form of a so-called *graph surface* or *Monge patch surface*. Such a surface has the representation:

$$S = \left\{(x, y, z(x,y)) : (x,y) \in D \subseteq R^2\right\} \tag{2.2}$$

It is known in differential geometry that a surface, S, is uniquely determined by its *first* and *second fundamental forms* at each point on the surface. The fundamental forms are scalar quantities that can be expressed in terms of the first and second partial derivatives of the surface. Although not sufficient to recover the fundamental forms, some of the important information contained in these fundamental forms can be re-expressed as the *maximum* and *minimum principal curvatures* (κ_1, κ_2) at each point on the surface, or the *mean* and *Gaussian curvatures* (H, K) at each point on the surface. Reasons for preferring the latter of these representations shall be discussed shortly. The mean and Gaussian curvatures are given in terms of the partial derivatives of a surface by the following equations:

$$H = \frac{\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} + \frac{\partial^2 z}{\partial x^2} \cdot \left(\frac{\partial z}{\partial y}\right)^2 + \frac{\partial^2 z}{\partial y^2} \cdot \left(\frac{\partial z}{\partial x}\right)^2 - 2 \cdot \frac{\partial z}{\partial x} \cdot \frac{\partial z}{\partial y} \cdot \frac{\partial^2 z}{\partial x \partial y}}{2\left[1 + \left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2\right]^{3/2}} \tag{2.3}$$

$$K = \frac{\frac{\partial^2 z}{\partial x^2} \cdot \frac{\partial^2 z}{\partial y^2} - \left(\frac{\partial^2 z}{\partial x \partial y}\right)^2}{\left[1 + \left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2\right]^2} \tag{2.4}$$

There are many ways of expressing the relationships between the minimum and maximum principal curvatures, the two fundamental forms, and the mean and Gaussian curvatures. The principal curvatures are associated with specific directions called the *principal directions*. At each point on a surface, a direction of *maximum normal curvature* and a direction of *minimum normal curvature* exist. These are the principal directions, with associated maximum and minimum principal curvatures (κ_1, κ_2). It is known that the principal directions are orthogonal. The principal directions together with the principal curvatures are sufficient to recover the first and second fundamental forms. The Gaussian curvature is the *product* and the mean curvature the *average* of the two principal curvatures.

Surface properties such as the curvatures can be classified as *intrinsic* or *extrinsic* surface properties. Intrinsic surface properties are also called *metric properties* or *isometric invariants*. Intrinsic surface properties are only changed by distortions of the surface which alter the distance between two points on the surface, while extrinsic properties change their values in response to less severe distortions of the surface such as reversing the direction of the surface normal. Surface area is an example of an intrinsic surface property. The Gaussian curvature and the first fundamental form are also intrinsic surface properties while the mean and principal curvatures as well as the second fundamental form are extrinsic surface properties.

Extrinsic surface properties can be said to "care" how a surface is embedded in space, while intrinsic ones do not. There is a subtle distinction to be made here about what is meant by "caring". The mean curvature will be affected by how the surface is embedded in space if this embedding includes redirection of the normal with respect to the surface (by reversing it to become inward rather than outward, for example.) However, both the Gaussian and mean curvatures of a surface region possess desirable *visible-invariance* properties, meaning they are invariant under certain changes to the surface so long as these changes do not affect the visibility of the surface region under study. For brevity, we use the term *invariance*, where *visible* is implicit. The mean and Gaussian curvatures are characterized by:

1) Invariance to arbitrary transformations of the surface parameterization, which in the Monge patch case reduces to the co-ordinate system reference, providing the Jacobian of the transformation is non-zero.

2) Invariance to arbitrary translations and rotations of the surface.

3) Invariance to partial occlusion. This is because the Gaussian and mean curvatures are local surface properties, hence the whole surface need not be visible to extract them.

These properties are very useful in designing a view-independent scheme for surface characterization. Such invariance properties are typical of approaches based on local differential geometry and constitute a prime advantage of such approaches. This is in contrast to another class of approaches based on classical variational calculus, where a global functional form is employed and dependence on the co-ordinate system chosen is strong. This alternate methodology is pursued in [Blak87]. In fact the two approaches have been described† as proceeding in opposite directions, i.e. from local to global properties in differential geometry and from global to local properties in the calculus of variations.

† In [Hilb52] p. 190.

Some additional properties of the curvatures that are worthy of consideration in designing a surface characterization framework for range images were cited by Besl and Jain in [Besl86] They suggest that since the mean curvature is the average of the principal curvatures, its sensitivity to noise in numerical computation is slightly less than those of the principal curvatures. The Gaussian curvature, being the product of the principal curvatures, is slightly more sensitive to noise. Besl and Jain also state that a few additional numerical computations appear to be needed to compute the principal curvatures as opposed to the mean and Gaussian curvatures.

Having examined the theoretical merits of the Gaussian and mean curvature compared to other surface measures, let us consider in more detail how the mean and Gaussian curvature can be used to determine surface type.

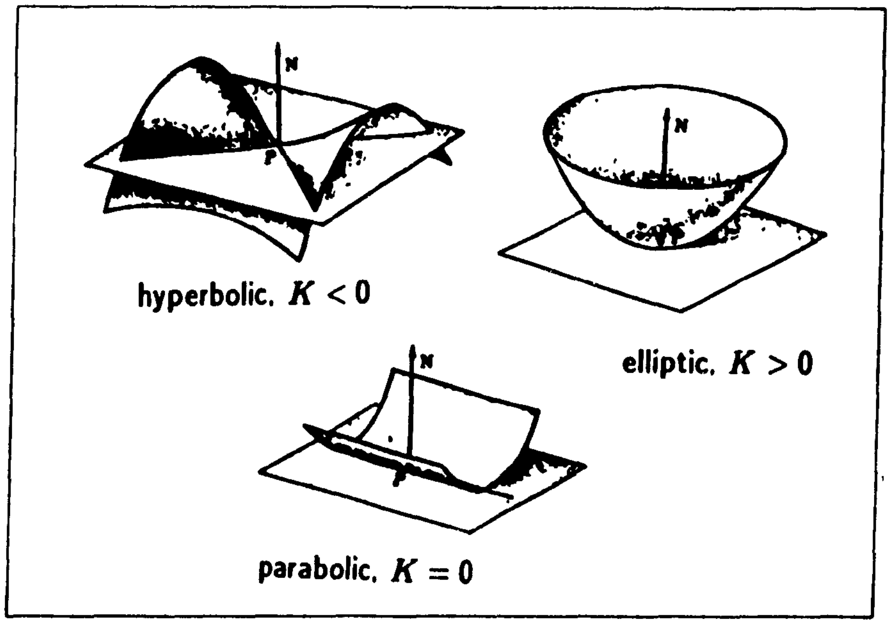


Figure 2.1 Types of Surfaces Distinguishable by the Sign of the Gaussian Curvature

The sign of the Gaussian curvature can be seen to relate to whether the surface is bending with the same concavity along its two principal directions. If there is some bending along both directions, bending with the same concavity gives $K > 0$, or an elliptic point which is cup- or cap-like, and bending with opposite concavity gives $K < 0$, or a hyperbolic point which is saddle-like. If there is no bending along one principal direction, we have $K = 0$ and have a parabolic point. These are illustrated in Figure 2.1[†]. No bending along both principal directions gives a flat surface for which K is also zero. The flat case is one of two surface types for which K is constant. These are called *umbilic surfaces* The other umbilic case is a spherical surface, where K is a non-zero constant. If $K = 0$ at every point on a surface, the surface is called a *developable surface*.

The mean curvature can be seen as giving an indication of the predominant concavity of a surface. If $H > 0$, the surface is predominantly concave and if $H < 0$ the surface is predominantly convex. If $H = 0$ at every point on a surface, it is called a *minimal surface*. Flat surfaces fall into this category. They also have $K = 0$. If $K < 0$, we have a saddle-shaped minimal surface. The case of $K > 0$ and $H = 0$ is unrealizable. The classification of surfaces resulting from taking the signs of the mean and Gaussian curvatures together is shown in Figure 2.2[‡].

2.2 Previous Work on Range Image Understanding

An excellent survey paper on the state of research in range image understanding up to 1985 was written by Besl and Jain [Besl85]. This paper defines the range-image understanding problem as a well-posed inverse-mapping problem, outlines desirable characteristics for a system to solve this problem, treats object and surface representations, discusses surface-rendering in the context of validating solutions to the problem, considers how range images are formed, and outlines approaches in the literature for range-image processing, surface characterization, and object recognition. All of these topics are important in designing a general system capable of understanding range images. Much work is left to be done, as range images have not been as popular as intensity images in machine vision historically. Recent improvements in range-sensing technology are changing this however,

[†] As in [Yoko87] p.10 and [Lips69] p.177.

[‡] As in [Besl86] p.49.

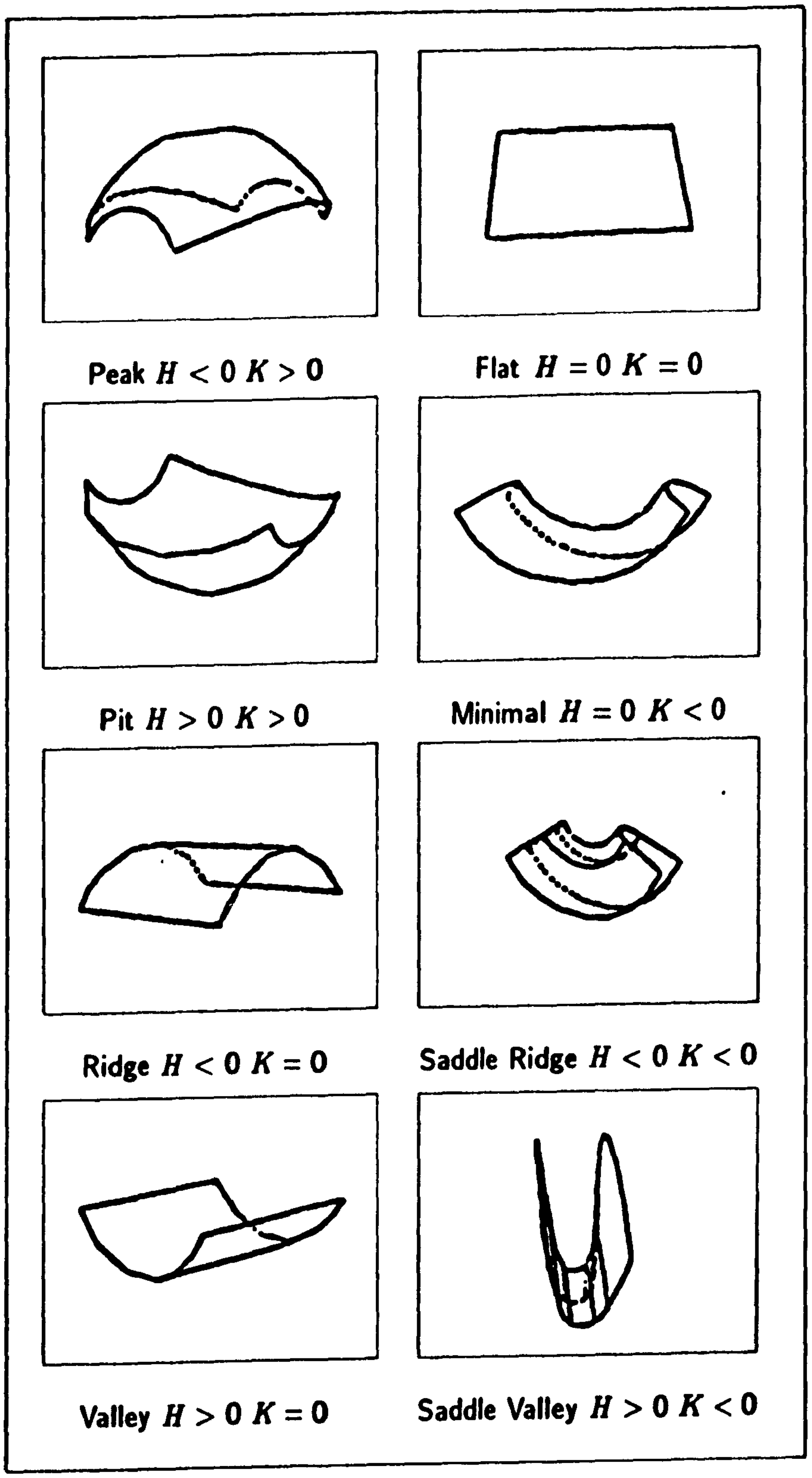


Figure 2.2 Eight Surface Types Distinguishable By Signs of Mean and Gaussian Curvature

and it is expected in the future that the explicitness of depth information in range images will make them the preferred form for use in machine vision.

In the following two sections, representative examples from previous work in range image understanding will be reviewed. The emphasis will be placed on approaches involving curvature-based and alternate methods for surface characterization. Emphasis on these topics was motivated by the objectives of this thesis and in no way implies that other sub-problems in range image understanding are any less important.

2.2.1 Curvature-Based Methods

This discussion of curvature-based methods begins by considering approaches whose methods are most similar to those proposed in this thesis and proceeds to the more dissimilar approaches. Naturally, the work most similar is that of Yokoya and Levine on which the current study was based. Their method is described in detail in [Yoko87] but was also presented in [Yoko88] and will soon appear in [Yoko89]. The curvature-extraction method of Yokoya and Levine is identical to the one described in section 2.3 of this chapter, excluding the parallel analysis of the "modified" algorithm. However, their analysis does not stop at curvature extraction, but develops a range image segmentation scheme.

This range image segmentation scheme relies on the curvature maps as well as maps of jump- and roof-edge magnitude. The map of jump-edge magnitude is computed using the following criterion:

$$M_{jump}(x,y) = Max\left\{|z(x,y) - z(x+k,y+l)| : -1 \leq k,l \leq 1\right\} \tag{2.5}$$

and the roof-edge map according to the criterion:

$$\begin{aligned} M_{roof}(x,y) &= Max\left\{Cos^{-1}\left(\frac{n(x,y)\cdot n(x+k,y+l)}{|n(x,y)||n(x+k,y+l)|}\right) : -1 \leq k,l \leq 1\right\} \\ &= Max\left\{Cos^{-1}\left(n(x,y)\cdot n(x+k,y+l)\right) : -1 \leq k,l \leq 1\right\} \end{aligned} \tag{2.6}$$

where $n(x,y)$ is a surface normal estimate (computed from partial-derivative estimates as are the curvature estimates). The curvature maps are thresholded and combined to produce a KH-sign map, where K is the Gaussian and H the mean curvature respectively. The edge-type maps are thresholded as well Two new maps are produced from the thresholded maps by 1) superimposing the edge maps onto the KH-sign map, 2) component-labelling of surface regions, 3) expansion of surface regions. These two new maps are a surface-edge map and a region map. Care is taken in the labelling and expansion of surface regions not to merge regions across edge boundaries and not to create isolated regions corresponding to discontinuities. The region map tags regions and associates each tag with one of the eight surface types distinguishable by the signs of the mean and Gaussian curvatures (as in Figure 2.2). More than one tag may be associated with any given surface type as there may be more than one instance of this surface type within the image. The surface-edge map identifies jump-edge pixels with a value of -1 and roof-edge pixels with a value of -2. This map can be used to identify occluding contours associated with depth discontinuities. Hence, the resulting segmentation as given by these two maps divides the image into regions of constant curvature which do not overlap any discontinuities. The description of the scene represented by the segmented regions is given by a region adjacency graph. Results of this segmentation scheme applied to real and artificial range images are shown pictorially in [Yoko87].

Another approach to range image segmentation which has much in common with that of Yokoya and Levine, and hence also with the approach of this thesis, has been explored by Roth and Levine [Roth89]. The method of Roth and Levine employs the quadric fit and refinement process from [Yoko87] to extract maps of mean and Gaussian curvature. However, the refinement process is modified somewhat.

In the original method of Yokoya and Levine and in this thesis as well, the fit is refined by selecting a best window position for each range-image point corresponding to a local minimum in an error map (section 2.3 gives the details.) This can be described as implementing a "full shift" away from discontinuities. In Roth and Levine's method, however, the shift is weighted using the error magnitudes, resulting in a "partial shift". Instead of implementing the full shift (u_*, v_*) away from the central pixel (x, y), the partial shift (u_p, v_p) is found by additional processing as follows:

1) Compute the error difference $\triangle E$ for each pixel, where $\triangle E = |E(x,y) - E_*(x,y)|$, and $E_*(x,y) = E(x - u_*, y - v_*)$ is the local minimum of the error map occurring at the full-shift best window position of (u_*, v_*).

2) Set a threshold $\triangle E_{max}$ for which the full shift is allowed.

3) Recompute the shift, obtaining the partial shift, as $u_p = min(u, round(u \cdot \triangle E/\triangle E_{max}))$ and $v_p = min(v, round(v \cdot \triangle E/\triangle E_{max}))$.

Grey-scaled pictorial results for fitted depth in [Roth89] indicate that the partial shift produces better results as it avoids shifting away from insignificant discontinuities. This improvement comes at the price of additional complexity in processing and a need to establish a threshold, $\triangle E_{max}$.

The treatment and analysis of the error map in 3-D curvature computations is of particular interest for the purposes of this thesis. The error map is central to the algorithm of Yokoya and Levine on which our modified algorithm is based. The computation of the error map is also the most time-consuming part of the algorithm and hence became the prime target for special-purpose hardware development resulting in our new VLSI fit-error processor cell which is presented in Chapter 3. In addition to the error treatments in [Yoko87] and [Roth89], some work by Abdelmalek and Boulanger [Abde89] on algebraic error analysis for surface curvatures of 3-D range images is soon to be presented. This increasing interest in the error map may prove valuable in developing new reliable methods for surface characterization.

Roth and Levine's curvature extraction method also includes higher-level processing of the curvatures to segment the range image. An iterative relaxation labelling procedure is applied to establish regions in the range image. Each region belongs to one of the eight primitive surface types distinguished by the signs of the mean and Gaussian curvature (Figure 2.2) Results obtained using two different relaxation schemes are compared in [Roth89].

Another method which incorporates the mean and Gaussian curvature was described by Besl and Jain in [Besl86]. Many interesting properties of the curvatures are discussed by Besl and Jain in the context of differential geometry theory. They cover points similar to those made in section 2.1 of this thesis but in more mathematical detail. The method described by Besl and Jain for computing curvatures involves performing a local surface fit on the data over a window and using this fit with related analytical expressions for the partial derivatives of the surface. The fitting of the depth map and estimation of partial derivatives is achieved with convolution window operators. This is similar to the approach of this thesis. However, Besl and Jain perform the quadric fit using a set of three orthogonal polynomial functions:

$$\phi_0(u) = 1, \quad \phi_1(u) = u, \quad \phi_2(u) = \left(u^2 - \frac{m(m+1)}{3}\right) \tag{2.7}$$

where $m = (n-1)/2$. is the half-width of the window and n X n is the (odd) window size. Normalized versions of these functions from [Besl86] are as follows:

$$\begin{aligned} b_0(u) &= 1/n, \quad b_1(u) = \left(\frac{3}{m(m+1)(2m+1)}\right) u \\ b_2(u) &= \frac{1}{P(m)}\left(u^2 - \frac{m(m+1)}{3}\right) \end{aligned} \tag{2.8}$$

with $P(m)$ defined as the fifth-order polynomial in m:

$$P(m) = \frac{8}{45}m^5 + \frac{4}{9}m^4 + \frac{2}{9}m^3 - \frac{1}{9}m^2 - \frac{1}{15}m \tag{2.9}$$

A fit of the data to the following functional form is performed (where $\hat{z}(u,v)$ is the fitted depth value at the point (u,v) in the fitting window about the central pixel).

$$\hat{z}(u,v) = \sum_{i=0}^{2}\sum_{j=0}^{2} a_{ij}\phi_i(u)\phi_j(v) \tag{2.10}$$

The fit is in the least-square sense, minimizing the error term:

$$\epsilon = \sum_{u=-m}^{m} \sum_{v=-m}^{m} \{z(u,v) - \hat{z}(u,v)\}^2 \tag{2.11}$$

where $z(u,v)$ are the depth samples in the fitting window about the central pixel. The solution for the coefficients is given by:

$$a_{ij} = \sum_{u=-m}^{m} \sum_{v=-m}^{m} z(u,v)b_i(u)b_j(v) \tag{2.12}$$

The partial derivatives are then given by:

$$z_u = a_{10}, z_v = a_{01}, z_{uv} = a_{11}, z_{uu} = 2a_{20}, z_{vv} = 2a_{02} \tag{2.13}$$

The solution for the a_{ij} is easily decomposed into a convolution-window-operator form so that the partial-derivative estimates can be computed directly from the depth map via convolutions. Then, the mean and Gaussian curvature can be computed from the partial derivatives (according to equations 2.3 and 2.4 given in section 2.2.1).

Besl and Jain's method differs from ours in that no refinement of the fit to select the "best window position" is done in their approach. Instead, they perform a preliminary smoothing of the range image before computing curvatures plus an additional smoothing of the curvature maps after they are extracted. This method is thought to have the disadvantage of smoothing over discontinuities in the image. Discontinuities are valuable information to be preserved if at all possible. Besl and Jain also compute several other surface quantities from the partial-derivative estimates, including the metric determinant, the quadratic variation, the co-ordinate angle, and the principal direction angle. Maps of fitted depth and fit error are also produced (as they are in the experiments of section 2.4 of this thesis). Results are presented in the form of grey-scaled pictures of these maps for both real and artificial range data, including some examples where noise was added to the artificial range images. Window sizes used ranged from 5 X 5 through 13 X 13. No scheme is proposed for using the resulting maps in higher-level processing in the paper [Besl86]. Yang and Kak outline an approach similar to that of Besl and Jain for characterizing segmented surface regions in [Yang86], except they use a B-spline function for fitting the surface and deriving partial derivative estimates.

Vemuri, Mitiche, and Aggarwal [Vemu86] propose a scheme which uses the two principal curvatures (κ_1, κ_2) and their product, the Gaussian curvature K, as estimated from a local surface fit to classify surface patches as one of five primitive surface types. These surface types are determined as follows:

1) elliptic, $K > 0$

2) hyperbolic, $K < 0$

3) parabolic, $K = 0$

4) umbilic, $K = constant, \kappa_1 = \kappa_2 \neq 0$

5) planar umbilic, $\kappa_1 = \kappa_2 = 0$

The range data which this scheme processes is somewhat unusual in that it is not restricted to the Monge patch form, where depth is a function of x and y (eq. 2.2).

Rather, it assumes the more general form where all three space co-ordinates have equal freedom to vary (eq 2.1). Such a surface is not restricted to be single-valued in depth, z. This data was collected with the so-called "White" scanner which uses triangulation on a projected plane of laser light to compute depth At any rate, the local fitting process for such data is more complex than for the schemes discussed thus far. The usual square window operators are replaced by "roughly rectangular grids" which are obtained from rectangular meshes deformed to fit the surface After determining these grids parametrically, three standard surface-fitting problems must be solved over them. The fitting uses tensor products of splines under tension and the fit is in the least-mean-square-error sense. The resulting parametric form for the fit is·

$$\begin{aligned} x(s,t) &= \sum_i \sum_j \alpha_{ij}\varphi_i(s)\psi_j(t) \\ y(s,t) &= \sum_i \sum_j \beta_{ij}\theta_i(s)\nu_j(t) \\ z(s,t) &= \sum_i \sum_j \gamma_{ij}\rho_i(s)\kappa_j(t) \end{aligned} \tag{2.14}$$

where $(\varphi, \psi, \theta, \nu, \rho, \kappa)$ are tension splines, (α, β, γ) are coefficients of the tensor products, and the neighborhood in space is defined by $1 \leq i \leq m$, and $1 \leq j \leq n$. The partial-derivative information required to compute curvatures from this fit is said to be provided by a public-domain surface-fitting software package.

The overall strategy of the algorithm of Vemuri *et al* is as follows.

1) The range image is divided into overlapping windows. This is so that all edges will occur internal to some window.

2) Windows containing jump boundaries are detected by thresholding the standard deviation of the Euclidean distance between points in the window. Windows not containing jump boundaries are fitted to surface patches.

3) Curvatures are computed and edge points are extracted by means of a threshold on the curvatures, then non-maximal suppression is applied perpendicular to the direction of maximum absolute principal curvature (the presumed edge direction).

4) Each non-edge point is classified as one of the five primitive surface types listed earlier on the basis of the Gaussian and the two principal curvatures. Neighboring points of the same surface type are merged into regions. Mode filtering, which replaces the label of a point within a region by the most dominant label within the region, is applied to the merged surface-type map to produce maximal regions.

The output is thus a segmentation of the range image into regions according to the curvature signs.

In the paper [Vemu86], Vemuri *et al* do not describe how their method handles adjacent regions of the same curvature sign. However, one of their sample images suggests that such regions are merged. This is in contrast to the method of Yokoya and Levine, which makes efforts to keep such regions disjoint. Experimental results from real range images of objects including a wedge, a cylinder, a balloon, and a light-bulb are supplied in [Vemu86] as color images, where each of the five primitive surface types is associated with a different color. The light-bulb example shows favorable results on a complex image. (It contains four distinct surface-type regions.) An example of the results using "direct" computation of curvatures, presumably with numerical differentiation, rather than computation of curvatures from fitted surface patches indicates that the direct method is inadequate. These researchers have recently extended their work to address localization of objects in range images [Vemu88].

Sander and Zucker [Sand86] [Sand88] describe a method for locating bounding surfaces within data where some variable (call it intensity) is measured as a function of the position in 3D space.

$$I = I(x, y, z) \tag{2.15}$$

Such data is common in the biomedical field; examples are positron emission tomography (PET) scans and magnetic resonance images (MRI's). The problem can be viewed as a 3D equivalent of edge detection. Although this problem is different (and more complex due to the additional dimension involved) compared to our stated problem of range image understanding, some of the methods involved are similar. The Gaussian and mean curvatures are used to form a segmentation of the proposed surface points into homogeneous regions belonging to one of four basic types (elliptic, parabolic, hyperbolic, and planar) according to the sign of the Gaussian curvature and whether or not the mean curvature is zero. The processing steps involved are roughly described as follows:

1) A 3D gradient operator is convolved with the data, $I(x, y, z)$, thresholded, and passed through local-maxima selection to give an initial guess at the surface points. These points will be triples (x, y, z) similar to the range-image format of Vemuri *et al* in that z is not a function of x and y. An estimate of the surface normal at these points also results from convolution.

2) A local surface fit to a non-central quadric form in a tangent-plane co-ordinate system is performed on the points from step 1 in a least-square sense. Curvatures then follow from the fit according to analytically stated relationships in terms of the fit coefficients.

3) An iterative, relaxation-based procedure is carried out to refine the surface description. Constraints between surface points over neighborhoods include continuity of surface normal and the fact that regions of elliptic and hyperbolic points must be separated by transitional zones of parabolic points.

The use of the local, tangent-plane co-ordinate system is related to the theory of Darboux frames. The positioning of the co-ordinate system is such as to align with the principal directions of the surface, the sense being chosen to give a right-handed orthogonal system.

Results are given in [Sand88] in the form of grey-scaled pictures of curvature-type region maps and surface normal maps. These images point out the improvement produced by the iterative relaxation technique, even in the presence of substantial added noise. Data for these results included artificial images with added noise and real magnetic resonance images of a biomedical nature (heart, skull).

The above-described method of Sander and Zucker has recently been applied to range data in an effort to achieve sensor-derived models of objects by Ferrie *et al* [Ferr89]. The aim is to extract coarse 3D models from sensor data for robotic collision avoidance and grasping purposes. Ferrie *et al* stress the importance of the *stability* of the extracted model with respect to noise perturbations in the input range image. The approach incorporates the so-called *Curvature Consistency Algorithm*, or CCA, which is the iterative relaxation-based procedure of Sander and Zucker discussed above and includes criteria based on differential-geometric curvatures.

Hoffman and Jain [Hoff87] describe a method which employs a somewhat more heuristic notion of curvature rather than the classical definitions of differential geometry. Their curvature measure involves the inter-pixel change in surface normal. Surface normals are computed by fitting the best plane to the data over n X n neighborhoods in the least-square sense. Hoffman and Jain prefer a 5 X 5 neighborhood, stating that it gave the best trade-off between noise suppression and loss of fine detail in their experiments with various window sizes. They also state that they have compared their curvature criterion with other, more "sophisticated" ones, presumably those of differential geometry, and found little difference in the results.

The approach of Hoffman and Jain [Hoff87] can be summarized as follows:

1) Perform a fit to the best plane on the data and extract the surface normals.

2) Segment the image using a clustering algorithm, where criteria for clustering include a similarity in location (x, y), depth z, and surface-normal orientation. Some refinement is done on the output of the clustering algorithm which includes eroding of the regions.

3) Classify the regions from step 2 as either planar, convex, or concave using a non-parametric trend test for planarity. This test requires sizeable regions, so some regions prove too small. In addition, the test may be inconclusive. In such cases, the method falls back on curvature-based methods and lastly on eigenvalue analysis.

4) Merge adjacent regions having the same classification (planar, convex, concave) providing a test for crease-edge character along the boundary fails.

The clustering algorithm is applied to a sub-sampled version of the image and subsequently the remaining pixels are associated with the cluster whose center is closest. The non-parametric trend-test for planarity is expressed in terms of Spearman's statistic, for which tabulated values are available. The curvature method for classifying regions as planar or non-planar is based on the cumulative distribution function (CDF) of the curvature measure over the region. The amount of noise present in the image should be known, since an ideal CDF of a step edge plus noise is required for the criterion. No pictorial results were supplied for this algorithm in [Hoff87].

An interesting method for computing Gaussian curvature without using explicit derivative estimates is described in [Lin82]. This method comes from the Regge Calculus of general relativity where geometry is analyzed without co-ordinates. The method involves discrete triangularization of the surface. Unfortunately, mean curvature can not be computed in such a fashion owing to the extrinsic nature of mean curvature.

Many other examples of the use of differential geometry for surface characterization exist in the literature. Those described above show that in general some sort of fit is performed on the data allowing extraction of partial derivative estimates and computation of surface properties such as the mean and Gaussian curvatures. Regions can then be formed as areas where these properties are homogeneous. Some kind of refinement is performed on the regions to give a segmentation of the image.

2.2.2 Non-Curvature-Based Methods

Alternate methods include extracting a polyhedral model by assuming a planar form for the fit and working with the normals. Assuming pre-determined forms for the surfaces, such as generalized cylinders and cones, is another alternate method. The Hough transform is sometimes used to map regions onto the Gaussian sphere and detect the footprints of the assumed functional forms These approaches are quite dissimilar to that of the current study and so will not be discussed in detail. Only a few interesting examples from among the many alternate approaches are cited in this section.

Oshima and Shirai employ planar surface primitives to compose a face-edge-vertex description of objects in range images [Oshi83]. In a subsequent refinement stage, quadrics are fitted to curved regions.

Grimson has developed a system for recognizing objects composed of planar elements in noisy, occluded data [Grim84] [Grim87]. A recent modification which will enable this system to handle curved objects is described for the case of 2D intensity images but can be generalized to the case of 3D range data [Grim88]. This method uses the Hough transform as a pre-processor to limit the search space for recognition

Muller and Mohr first determine a general quadric surface fit for range data and transform it using the Hough transform to detect planar and quadric surfaces [Mull84].

Faugeras and Hebert [Faug83] fit planar faces on objects in their approach. Global approximation of object surfaces is then performed. Later processing utilizes quaternions in solving the matching problem of object recognition.

Cohen and Rimey [Cohe88] present a maximum-likelihood approach to segmenting range data. Their method considers planes, cylinders, and spheres as the only types of surfaces present in the range image. A Taylor series approximation is used to simplify some of the analysis in this approach.

Acharya and Henderson [Acha88] explore mathematical methods for range data analysis assuming planar surface fits are used to extract edges A weighting scheme is introduced for the least-squares fitting of data

Godin and Levine [Godi89a][Godi89b] present an edge-based approach which involves the construction of an "edge junction graph" for objects within range images. The junctions, or nodes, where edges meet are classified according to a basic vocabulary of possible 3-D configurations. This is preceded by a standard edge-detection step to obtain the edges. The novelty of the approach lies in the interpretation of the edge maps. The approach detailed in [Godi89a] is applied to curved objects in [Godi89b].

Aubry and Hayward employ level curves [Aubr87] for range image analysis. Rioux *et al* use sine wave coding and Fourier transformation methods to segment range images [Riou87]. Many more examples exist. Range imaging is currently a topic of great interest in the literature and this trend looks as though it might continue for quite some time.

Having looked at a selection of range-image processing approaches from the literature, the next section will detail the curvature extraction method of this thesis and the theory on which it rests.

2.3 Extraction of the Mean and Gaussian Curvature from a Range Image

2.3 1 Co-ordinate System

It is useful to present first the co-ordinate system which will be used in the treatment of the algorithm and the experiments. Readers will then be able to orient them-

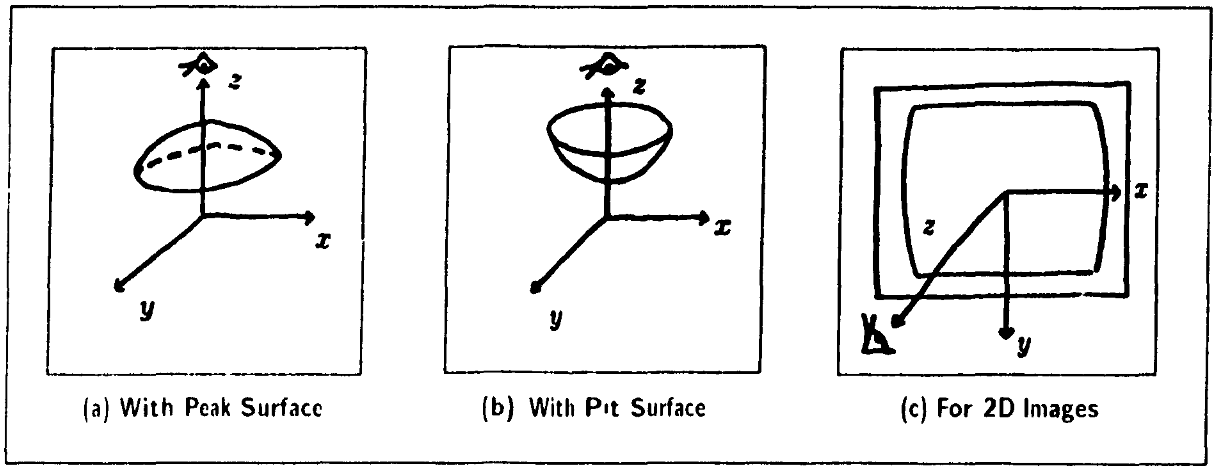


Figure 2.3 The Co-ordinate System

selves. The co-ordinate system is a right-handed system. Three views of this system are shown in Figure 2.3.

The first two views show peak and pit surfaces embedded in the co-ordinate system and are useful in visualizing how the artificial range data for the experiments was created. The eye represents the viewing direction of a hypothetical range-finder which might have produced the artificial data.

The third view represents the aspect of the co-ordinate system with respect to the grey-scaled pictures of the results that appear in the experiments section of this chapter. The figure shows such a result-image as if it were on a computer screen. This is to emphasize the fact that the system was chosen for its convenience in computer processing The eye in the figure once again represents the assumed viewing direction of the range-finder.

2.3.2 Overview of the Algorithm

A block diagram of the algorithm is shown in Figure 2.4. This processing will be briefly outlined here.

The curvature extraction algorithm begins by performing a local fitting of a surface form to the range data. The theory of fitting samples of one-dimensional data to

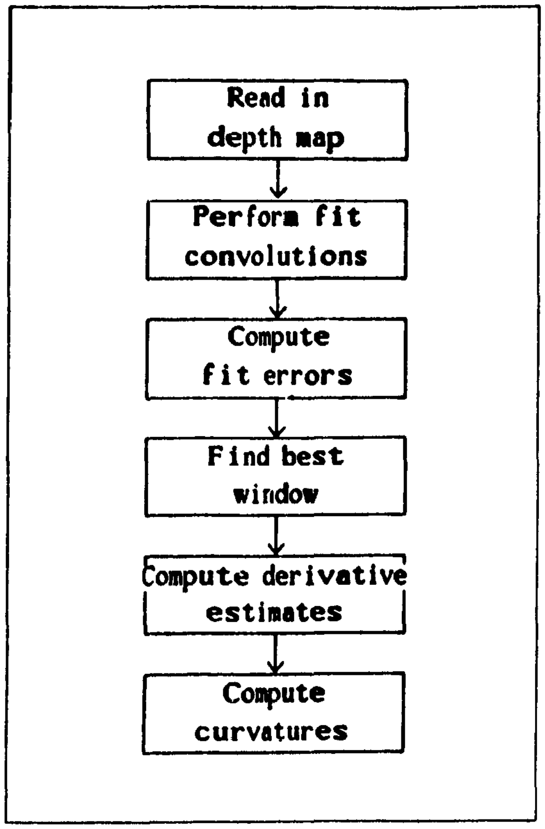


Figure 2.4 Block Diagram of Curvature Extraction Algorithm

a functional form is a well-explored area also known as curve fitting, or regression analysis in the case of a polynomial form. Surface fitting is really very similar; there is simply one more dimension involved.

Quadric surfaces are the simplest polynomial surfaces which can model curvature. This is why the quadric form was chosen to locally model the depth information. Other choices popular for locally modelling range data are planar patches and cubic spline patches.

The general quadric has nine coefficients, allowing the surface complete rotational and translational freedom [Trim83]. However, for the algorithm described in this thesis, a simpler form was assumed featuring only six coefficients. This form allows the depth, z, to be expressed as a quadric in x and y as given by the following equation:

$$z(x,y) = ax^2 + by^2 + cxy + dx + ey + f \tag{2.16}$$

This form allows for rotations and translations of quadric surface patches in the x-y plane. Some quadrics, such as the sphere, cannot be expressed exactly in this form, and, of course, neither can higher-order polynomial surfaces or non-polynomial surfaces. Since this fit has only to suffice locally, the form is thought to be sufficiently general.

The range data will be fitted to the above quadric locally. Hence, a local parameterization can be made where the origin is at the central sample (or pixel) within a square neighborhood. This will be referred to as the (u,v) parameterization, where u is the local equivalent of x and v the local equivalent of y The local variables u and v will index the square neighborhood about the central pixel (x_0, y_0) which is under the window operator. Hence u and v will range between $-m$ and m, where $(2m + 1) = n$, n X n being the window-operator size. The local surface fit under the window operator about the central pixel will then be given by the equation below:

$$z(x_0 + u, y_0 + v) = au^2 + bv^2 + cuv + du + ev + f \tag{2.17}$$

The data will be fitted to this equation on the basis of the sample points in the square neighborhood about the central pixel (x_0, y_0) to give an initial estimate of the coefficients (a, b, c, d, e, f). This estimate is associated with the point $(u, v) = (0,0)$ in the local parameterization.

A refinement of this estimate will be derived by finding the "best window position" (BWP). This will be regarded as the best estimate of the surface. The reason for this refinement is to avoid poor fitting and smoothing effects near discontinuities. This occurs when the central point (x_0, y_0) is near enough to a discontinuity to allow points on both sides of it into the fitting window. In such cases, the best estimate of the surface at (x_0, y_0) will not correspond to the fit centered on (x_0, y_0). In general, the optimum fit will appear at a neighboring point $(u = u_*, v = v_*)$ for which the fitting window contains points from one side of the discontinuity only. Hence the initial fit estimate coefficients $(a_*, b_*, c_*, d_*, e_*, f_*)$ at $(x, y) = (x_0 - u_*, y_0 - v_*)$ will be used to give the best estimate of the surface (or fitted depth) at (x_0, y_0) according to the equation:

$$z_*(x_0, y_0) = a_* {u_*}^2 + b_* {v_*}^2 + c_* u_* v_* + d_* u_* + e_* v_* + f_* \tag{2.18}$$

Let us restate for emphasis that both the coefficients $(a_*, b_*, c_*, d_*, e_*, f_*)$ as well as u_* and v_* in this equation are functions of the central pixel location, (x, y), and will be determined by the refinement process of selecting the best window position. A particular central pixel we have been concentrating on for the sake of discussion has been denoted by (x_0, y_0). Yokoya and Levine [Yoko87] point out that this method of finding the best fit is based on the facet model smoothing of intensity images [Hara80] [Pong81] and is similar to the methods of selective averaging [Yoko78], edge-preserving smoothing [Naga79], and computational molecules [Terz83].

The difference between the (u, v) space and the (x, y) space is simply the choice of origin, since here it is assumed that the sense of the (u, v) axes is the same as for the (x, y) axes. Hence, the partial derivatives with respect to x and y are the same as those with respect to u and v. This will allow the partial derivative estimates to be computed with the following set of equations once the best window position has been found:

$$\frac{\partial z}{\partial x}(x, y) = 2a_* u_* + c_* v_* + d_* \tag{2.19}$$

$$\frac{\partial z}{\partial y}(x, y) = 2b_* v_* + c_* u_* + e_* \tag{2.20}$$

$$\frac{\partial^2 z}{\partial x^2}(x, y) = 2a_* \tag{2.21}$$

$$\frac{\partial^2 z}{\partial y^2}(x, y) = 2b_* \tag{2.22}$$

$$\frac{\partial^2 z}{\partial x \partial y}(x,y) = c_* \tag{2.23}$$

The estimates of mean and Gaussian curvature then follow using these derivative estimates according to equations (2.3) and (2.4) of this chapter.

2.3.3 Least-Squares Analysis: Obtaining the Initial Fit

The data points in a square window about each pixel are to be fitted to equation (2.17) above. The window size is n X n, where n is an odd integer. The half-width of this window is defined as m such that $m = (n-1)/2$. According to the standard least-squares-fit technique[†] this involves minimizing the sum of the squares of the fit errors. These fit errors are the differences between the depth samples at each (u,v) and the value of the expression (2.17) to be fitted there. In general, the samples can be assigned unequal weighting factors according to their perceived importance in the fitting. The squares of the fit-errors would then be multiplied by their associated weights and summed. A case could be made for assigning greater weights to samples nearer the central pixel. However, we choose for simplicity to assign all the samples equal weight so that the sum to be minimized is given by:

$$E^2(a,b,c,d,e,f) = \sum_{u=-m}^{m} \sum_{v=-m}^{m} \left[(au^2 + bv^2 + cuv + du + ev + f) - z_{uv}\right]^2 \tag{2.24}$$

where the dependence of the expression on x and y is implicit in the fact that u and v are local offsets with respect to the central pixel (x,y). The samples about the central pixel, $z(x+u, y+v)$, are denoted by z_{uv} for the sake of compactness in the coming analysis. (Note that these are *depth samples*, not mixed partials, despite the notation.)

[†] As described in many standard texts, for example, in [Rals65] chapter 6.

The minimum value of the error expression to be determined for the central pixel will be important later on for refining the fit.

The set of coefficients (a, b, c, d, e, f) that minimize the above error expression can be determined exactly. The procedure is to write down the six expressions for the partial derivatives of the error with respect to each coefficient, set them equal to zero, and solve the resulting six equations simultaneously. The solution of this linear system is the desired set of coefficients.

The goal of this section is to obtain the above solution in the form of six convolution window operators, one for each coefficient, such that the convolution product of each operator with the window of depth samples yields the corresponding fit coefficient. The convolution product form sought is a two-dimensional digital convolution, such as:

$$p(x,y) = (P \otimes z)(x,y) = \sum_{u=-m}^{m} \sum_{v=-m}^{m} P(u,v) z_{uv} \qquad (2.25)$$

where $p(x, y)$ is one of the six fit coefficients (a, b, c, d, e, f), the symbol $\otimes$ stands for the convolution operation, P is the convolution window operator sought, and z is the window of depth samples. There are six operators P to be determined, one for each fit coefficient. For each window size, corresponding to the choice of n and therefore of m in equation (2.25), a different set of such convolution window operators results. The convolution operators can be expressed as matrices with constant elements

To see how the operators are derived, the operator $P = A$, associated with the fit coefficient "a" will be highlighted in the following discussion. The expressions for the other five operators $P = B, P = C, P = D, P = E, P = F$ associated with the fit coefficients (b, c, d, e, f) are defined in a similar manner. First consider the partial derivative of the error expression with respect to the coefficient "a".

$$\frac{\partial E}{\partial a} = 2 \cdot \sum_{u} \sum_{v} \left[(au^2 + bv^2 + cuv + du + ev + f) - z_{uv} \right] u^2 = 0 \qquad (2.26)$$

which simplifies to:

$$\begin{aligned}
&a\left(\sum_u\sum_v u^4\right)+b\left(\sum_u\sum_v u^2v^2\right)+c\left(\sum_u\sum_v u^3v\right)\\
&+d\left(\sum_u\sum_v u^3\right)+e\left(\sum_u\sum_v u^2v\right)+f\left(\sum_u\sum_v u^2\right)\\
&=\left(\sum_u\sum_v u^2 z_{uv}\right)
\end{aligned}\tag{2.27}$$

The five other partial-derivative expressions can be put in this form as well. The following linear system results, where $\sum$ implies a double summation with respect to (u,v) over the range $-m \le u \le m$, $-m \le v \le m$:

$$\begin{bmatrix}
\sum u^4 & \sum u^2v^2 & \sum u^3v & \sum u^3 & \sum u^2v & \sum u^2\\
\sum u^2v^2 & \sum v^4 & \sum uv^3 & \sum uv^2 & \sum v^3 & \sum v^2\\
\sum u^3v & \sum uv^3 & \sum u^2v^2 & \sum u^2v & \sum uv^2 & \sum uv\\
\sum u^3 & \sum uv^2 & \sum u^2v & \sum u^2 & \sum uv & \sum u\\
\sum u^2v & \sum v^3 & \sum uv^2 & \sum uv & \sum v^2 & \sum v\\
\sum u^2 & \sum v^2 & \sum uv & \sum u & \sum v & \sum 1
\end{bmatrix}
\begin{bmatrix} a\\ b\\ c\\ d\\ e\\ f\end{bmatrix}
=
\begin{bmatrix}
\sum u^2 z_{uv}\\ \sum v^2 z_{uv}\\ \sum uvz_{uv}\\ \sum uz_{uv}\\ \sum vz_{uv}\\ \sum z_{uv}
\end{bmatrix}\tag{2.28}$$

The elements of the matrix on the left all evaluate to constants for a given window size, $n = 2m + 1$. These constants can be computed and the matrix inverted to give:

$$\begin{bmatrix} a\\ b\\ c\\ d\\ e\\ f\end{bmatrix}
=
\begin{bmatrix}
w_{a1} & w_{a2} & w_{a3} & w_{a4} & w_{a5} & w_{a6}\\
w_{b1} & w_{b2} & w_{b3} & w_{b4} & w_{b5} & w_{b6}\\
w_{c1} & w_{c2} & w_{c3} & w_{c4} & w_{c5} & w_{c6}\\
w_{d1} & w_{d2} & w_{d3} & w_{d4} & w_{d5} & w_{d6}\\
w_{e1} & w_{e2} & w_{e3} & w_{e4} & w_{e5} & w_{e6}\\
w_{f1} & w_{f2} & w_{f3} & w_{f4} & w_{f5} & w_{f6}
\end{bmatrix}
\begin{bmatrix}
\sum u^2 z_{uv}\\ \sum v^2 z_{uv}\\ \sum uvz_{uv}\\ \sum uz_{uv}\\ \sum vz_{uv}\\ \sum z_{uv}
\end{bmatrix}\tag{2.29}$$

From here, we can show how to obtain the convolution window operators. For the coefficient "a", consider multiplying out the first row of the matrix on the right-hand side of equation (2.29) with the vector containing the sums of depth samples, z_{uv}. This gives:

$$a = w_{a1} \sum_u \sum_v u^2 z_{uv} + w_{a2} \sum_u \sum_v v^2 z_{uv} + w_{a3} \sum_u \sum_v uv z_{uv} + w_{a4} \sum_u \sum_v u z_{uv} + w_{a5} \sum_u \sum_v v z_{uv} + w_{a6} \sum_u \sum_v z_{uv} \tag{2.30}$$

The double sums in (u, v) in the above expression can be combined so that it becomes:

$$a = \sum_u \sum_v \alpha_{uv} z_{uv} \tag{2.31}$$

The multipliers, α_{uv} are the desired elements $A(u, v)$ of the convolution window operator A for the fit coefficient "a" and are given by:

$$\alpha_{uv} = w_{a1} u^2 + w_{a2} v^2 + w_{a3} uv + w_{a4} u + w_{a5} v + w_{a6} \tag{2.32}$$

The other five convolution window operators can be synthesized similarly. The tedious part of the above process is the computation of the w's. In fact, it is known that the matrix on the left-hand side of equation (2.28) is ill-conditioned, i.e. its determinant is very small. Fitting data by least-squares to a polynomial form (or regression) results in a linear system with an ill-conditioned matrix and the smallness of the determinant becomes a worse problem as the order of the system is increased[†]. Hence, the numerical solution of the system (2.28) can be a cause for concern in terms of lost significant figures and lost accuracy. For the experimental study in this thesis, we rely on convolution coefficients computed by retaining the fractional form in the intermediate calculations, thus avoiding underflow as was done in [Yoko87]. Since the operator sizes used here are not too large, this method is adequate. However, if very large operators become necessary, there is another wa, of solving the problem. It involves the use of an orthogonal basis of polynomials[‡].

[†] As described in [Rals65] p. 233.

[‡] Chapter 6 of [Rals65] contains a description of such an orthogonalization method.

Such a method was used by Besl and Jain in [Besl86] where quite large window sizes are featured.

The convolution operators of this thesis are given in Appendix A in the exact fractional form resulting from the derivation as well as in the approximate decimal form actually used for the experiments. These include operator sets of three sizes: 3 X 3, 5 X 5, and 7 X 7

Given such a set of operators, the initial estimate of the local surface fit can be made. The process involves convolving the depth map samples, z, with each of the six window operators to yield six maps of fit coefficients. Hence, each point (x, y) in the range image will have its own fit coefficients, $(a(x,y), b(x,y), c(x,y), d(x,y), e(x,y), f(x,y))$ spread across six new maps computed by convolution.

2.3.4 Analysis of Fit-Errors: Refining the Fit

Recall that in the preceding discussion the minimum mean-square error associated with the least-squares fit was mentioned in passing. This measure forms the basis for selecting the best window position (i e. refining the fit) in the original algorithm of Yokoya and Levine [Yoko87] The value of this error measure was not actually computed in doing the fit, although it was used as a starting point for deriving the convolution operators. After the convolutions have been done, a second pass of processing on the range data is needed to compute the error measure at every point in the range image according to the equation:

$$E_0^2(x,y) = \left(\sum_{u=-m}^{m} \sum_{v=-m}^{m} [z_0(x+u, y+v) - z(x+u, y+v)]^2 \right) / n^2 \tag{2.33}$$

where:

$$z_0(x+u, y+v) = a(x,y)u^2 + b(x,y)v^2 + c(x,y)uv + d(x,y)u + e(x,y)v + f(x,y) \tag{2.34}$$

Equation (2.33) is an aggregate measure of how well the fit coefficients at the central pixel (x, y) represent the entire surface patch (or square neighborhood) about that central pixel This is what will be called the "goodness" of the fit The equation above will sometimes be referred to as the original or square error criterion in the remainder of the thesis.

An alternate expression to rate the "goodness" of the initial fit is proposed in equation (2 35) below It aggregates the absolute magnitudes of the fit errors rather than their squares. This alternate form becomes the basis for the "modified" algorithm whose performance this thesis studies compared to that of the original algorithm which uses equation (2.33) as its measure of the "goodness" of the initial fit. Apart from this difference in "goodness" criteria, the two algorithms are identical Equation (2.35) will sometimes be referred to as the modified or absolute error criterion in the remainder of the thesis. The expression $z_0(x+u, y+v)$ is as shown above in equation (2.34).

$$|E_0(x,y)| = \left(\sum_{u=-m}^{m} \sum_{v=-m}^{m} |z_0(x+u, y+v) - z(x+u, y+v)| \right) / n^2 \tag{2.35}$$

Sometimes a smoothing criterion, S, is also considered in assessing the "goodness" of a fit. Such a function would introduce some kind of a continuity criterion between the fitted patches centered on neighboring pixels, perhaps in a weighted fashion A somewhat complex example is the weak continuity constraints of Blake and Zisserman (p. 40, [Blak87]). There a total-energy expression which is the sum of discontinuity penalties P, faithfulness to data D, and stretchability S, is minimized As a simpler example, a modified "goodness" criterion W might be taken as the sum of the fit-error criterion E and the smoothness criterion S:

$$W = E + S \tag{2.36}$$

No additional smoothness criterion is featured by either the original or the modified algorithms as implemented for this thesis.

After computing a map of fit errors using either equation (2.33) or (2.35), the surface fit is refined by selecting the "best window position". This procedure is motivated by the fact that some windows will overlap discontinuities and hence the pixels at the centers of these windows would be better represented by the fit at one of their neighbors. The fit-error maps form the basis for selecting the best window position, since the fit error will be large in the neighborhood of a discontinuity The method of selecting the best window position described here is said to perform "discontinuity-preserving smoothing" according to Yokoya and Levine [Yoko87].

At each point, (x, y), in the range image, the best window position is determined to be the set of coefficients $(a_*, b_*, c_*, d_*, e_*, f_*)$ and the offset (u_*, v_*) within the square neighborhood about (x, y) where the fit error is a minimum. This is a search problem, which can be stated more formally for the original algorithm as the location of E_*^2 for each point (x, y) such that the following holds:

$$E_*^2(x,y) = E_0^2(x-u_*, y-v_*) = Min\left\{E_0^2(x-u, y-v) : -m \leq u \leq m, -m \leq v \leq m\right\} \tag{2.37}$$

For the modified algorithm, a similar search problem is solved over the square neighborhood about each point (x, y) to locate $|E_*|$ for that point such that we have instead:

$$|E_*(x,y)| = |E_0(x-u_*, y-v_*)| = Min\left\{|E_0(x-u, y-v)| : -m \leq u \leq m, -m \leq v \leq m\right\} \tag{2.38}$$

The same operation is actually being performed in both cases: a constrained search for the minimum of the fit-error map. In the actual implementation, the differences between the algorithms were incorporated entirely in the generation of the fit-error map.

Once the above-described minimum is found for each point (x,y) of the range image, the best window position for each such point is known. The derivatives and curvature estimates then follow in a straight-forward manner as described in the overview (Section 2.3.2). The fitted depth map is also created using the best window position according to equation (2.18) of the overview section.

2.3.5 Algorithm Complexity

A data-flow diagram for the curvature extraction algorithm is given in Figure 2.5 where the labelled arrows represent data streams and the boxes represent computational steps. This algorithm was implemented as a C program. The initial input, shown at the top of the diagram, is the range image z plus the convolution window operators indicated as $(a_0, b_0, c_0, d_0, e_0, f_0)$ in the figure. The outputs that will be examined in this thesis include the maps of fit error E_0, best-window-position offset (u_*, v_*), fitted depth z_*, and of course curvature (mean H and Gaussian K). A slightly extended version of the program was used for calculation of the fitted depth map z_* and storage of the extra outputs E_0, u_*, v_*, and z_* for analysis in the experiments section. The basic curvature extraction program does not require calculation of the fitted depth and only stores the curvatures (H,K) to files. It is the basic version of the program that was benchmarked for execution-time comparisons.

Figure 2.5 indicates the main parts of the algorithm as labels at the right side of the figure. Three steps are indicated: fit, errors, and curvatures. This division is based on the actual C-program code in which the fit convolutions are done in six loops, the computation of the fit errors in one big loop, and the curvature calculation in one big loop as well. The curvature loop contains three parts: 1) a neighborhood search about the central pixel location (x,y) to find the best window position, 2) use of this information to calculate the partial derivatives at (x,y), and 3) combination of the partial derivatives to form the curvatures The big loop for curvatures then loops back to find the best window position at the next central pixel location, and so on. The data-flow diagram of Figure 2.5 also shows what inputs are required and what outputs are produced at each step in the

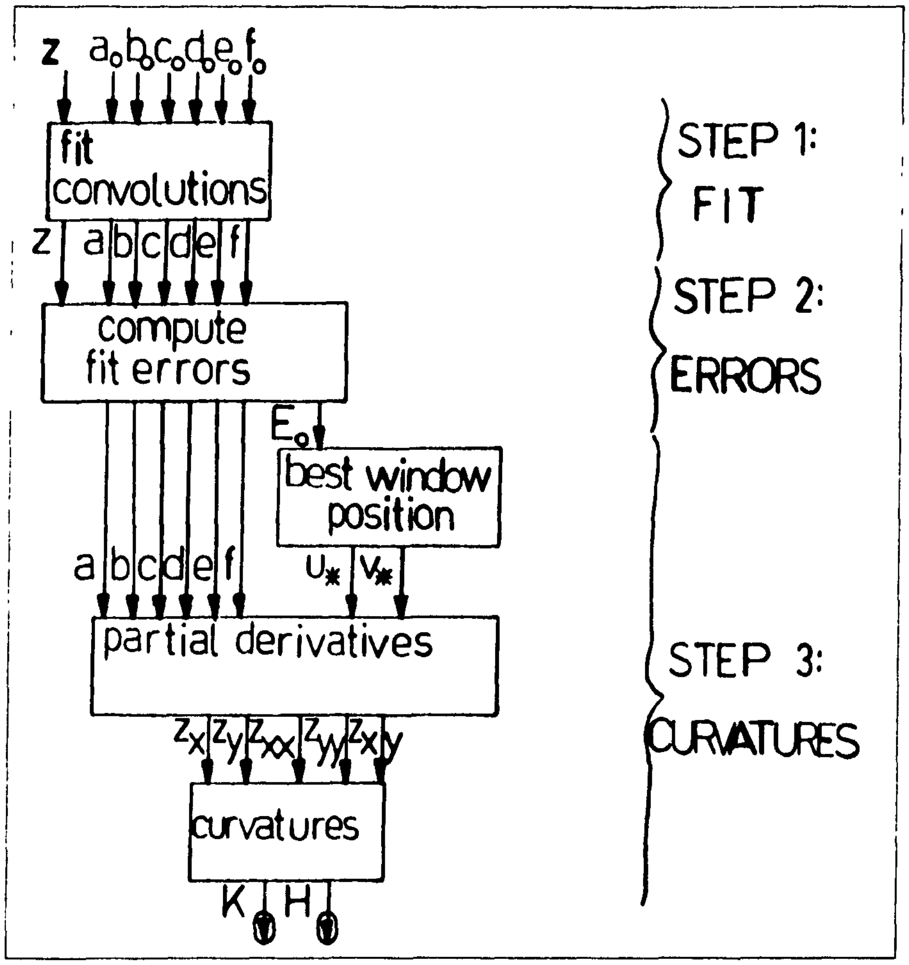


Figure 2.5 Data-Flow Diagram of the Curvature Extraction Algorithm

algorithm.

The total number of operations involved in the above algorithm goes up as the square of the size of the range image on a side (N) and also as the square of the window-operator size on a side (n). Here, a square range image of size N X N and a square window-operator of size n X n are assumed. In particular, each of the three main steps of the algorithm requires the operations listed in Figure 2.6. A further breakdown of the third "curvatures" step into its three components is also shown. Only the operations necessary for curvature extraction are included in the totals of Figure 2.6. Operations to compute the fitted depth are not included

The first two algorithm steps (the fitting and errors steps) as well as the first

ALGORITHM STEP	ADDS +SUBT.	MULT.	DIV.	SQUARE or MAGN.	COM-PARES	POWER (3/2)
1)FITTING	$6N^2(n^2-1)$	$6N^2n^2$	-	-	-	-
2)ERRORS	$N^2(6n^2-1)$	$8N^2n^2$	N^2	N^2n^2	-	-
3)CURVATURES						
a)Best window	-	-	-	-	$N^2(n^2-1)$	-
b)Derivatives	$4N^2$	$8N^2$	-	-	-	-
c)Curvatures	$9N^2$	$15N^2$	$2N^2$	-	-	N^2

Figure 2.6 Number and Type of Operations in the Curvature Extraction Algorithm

part of the curvatures step (the best window position search) are neighborhood-oriented; the number of operations for these goes up as the product of the image size N^2 times the window size n^2. The last two parts of the third (curvatures) step are point-oriented; the number of operations involved in these parts goes up in proportion to the image size alone. The large load of additions, subtractions, and multiplications in the first two algorithm steps (fitting, errors) forms the bulk of the computational burden in the algorithm (as verified by execution time benchmarks presented in the next section), despite the fact that the most complex operation, the power 3/2, occurs in the third (curvatures) step. In fact, the errors step turns out to be the most time consuming, indicating that the computation of the fit error is a good target for special-purpose hardware. The fit-error computation is simpler to implement as a VLSI circuit using equation (2.35) which features the absolute magnitude rather than equation (2.33) which features the more-complex square operation. The effect of this change will be studied with practical experiments in the next section.

2.4 Experiments

Development of software and processing of results for the experiments were carried out in a networked, multi-user environment of Sun 3/60 workstations under UNIX using the C programming language as well as functions from the Pixrect low-level graphics library and the HIPS image-processing library. Experiments were performed with four sets of data, two artificial and two real data sets. Both the original algorithm, featuring a sum-of-squares criterion on the fit errors, and the modified algorithm, featuring a sum-of-magnitudes criterion, were used to process the artificial data while only the modified algorithm was used on the real data. The real range data was obtained from the National Research Council in Ottawa.

2.4.1 Execution Time

Execution time results collected for the experiments reinforce the idea that the algorithms under study are quite slow when implemented using a high-level language (such as C) in a general computing environment (such as the multi-user, networked UNIX environment of Sun workstations) Figure 2.7 shows typical execution times for the experiments. The Sun 3/160 workstations on which the experiments were conducted feature Motorola 68881 floating-point units (FPUs) and 32-bit Motorola 68020 central processing units (CPUs). All computations used 64-bit double-precision floating-point format.

Image	Window	Time (sec)	% Fitting	% Errors	% Curvatures
64X64	3X3	12.2	36	38	20
64X64	5X5	23 6	35	48	14
64X64	7X7	41 9	37	50	11
256X256	5X5	388 5	34	48	15

Figure 2.7 Typical Execution Times for Experiments

The artificial range images in the experiments were deliberately kept small (64 X 64 pixels) to keep execution times down while the real range images were significantly larger (256 X 256 pixels). In Figure 2.7, the percentage of time spent in fitting the surface, computing the fit errors, and calculating curvatures are given These three steps (those of Figure 2 5) account for nearly all of the processing time The remainder is a small percentage required for overhead (space allocation, type conversions, and file access).

The main factors that differentiated the time required for various experiments were image size and window-operator size. Although the original algorithm with its sum-of-squares criterion took slightly longer to run than the modified algorithm in some of the cases, the difference was not significant. In all cases, the fit-error computation required the greatest proportion of the execution time, this proportion being near 50 percent.

The effect of the window size on the execution time is shown by the different times for the 64 X 64 image in Figure 2.7. The increase in execution time is somewhat less than proportional to the increase in operator size for the constant image size. This is as expected from the complexity analysis, since there are parts of the algorithm that are point-oriented and hence do not require additional operations as the window size is increased. The point-oriented parts of the algorithm are in the third (curvatures) step and hence th.s step accounts for a decreasing proportion of the execution time as the window size is increased.

The time required to process a 256 X 256 image with 5 X 5 window operators is just over 16 times that required for a 64 X 64 image with the same operators. This is in agreement with the complexity analysis, since virtually all parts of the algorithm involve computation loads proportional to the image size and there are 16 times as many pixels in the 256 X 256 image as there are in the 64 X 64 image.

2.4.2 Artificial Range Images

2.4.2.1 Artificial Range Image 1: The Parabolic "Cap"

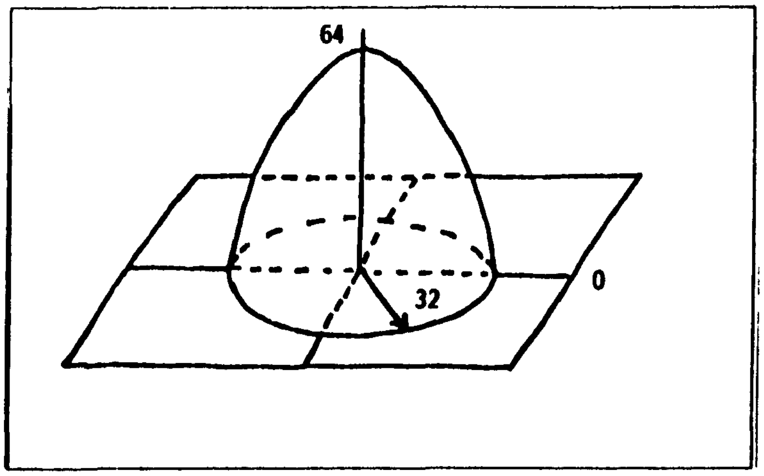


Figure 2.8 Parabolic Surface Segment for the "Cap" Image

The top section resulting from truncation of a paraboloid was chosen as the first artificial range image. Since it looks like a hat or cap, it is called the "Cap" image for brevity. A diagram showing this parabolic "Cap" suggestive of its aspect in 3D is given in Figure 2.8. No noise was added for the experiments with this image. The window operator size used on this image was 5 X 5.

The equation of the surface [Thom79] sampled to give the depth map of the cap is shown below, where the parameters (k, b) had values (64, 2).

$$z = k - b(x^2 + y^2) \tag{2.39}$$

Formulas were established to produce maps of ideal mean (H) and Gaussian (K) curvature. These are given below.

$$H = \frac{-(k/r^2)\left[1 + \left[1 + (4k^2/r^4)(x^2 + y^2)\right]\right]}{\left[1 + (4k^2/r^4)(x^2 + y^2)\right]^{3/2}} \tag{2.40}$$

$$K = \frac{4k^2/r^4}{\left[1 + (4k^2/r^4)(x^2 + y^2)\right]^2} \tag{2.41}$$

As shown in Figure 2.8, the surface is a peak surface. This implies its curvatures are such that:

$$H < 0, K > 0$$

which was indeed found to be the case for the extracted curvatures.

Figure 2.9 shows grey-scaled pictures representing the original and fitted depth values for the "Cap" image. There is little visible difference between the pictures for the fitted data from the two algorithms. In the continuous area within the "Cap", the three images match up nicely[†]. However, at the edge the fitted data shows a sharper transition

[†] A good match for images means equivalent visual (perceptual) quality

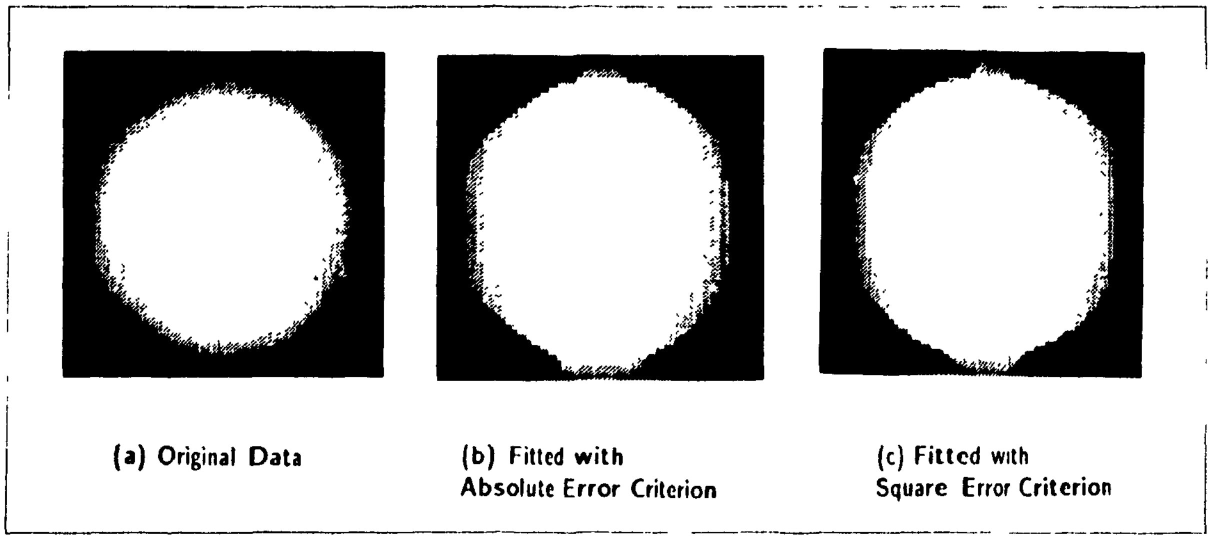

(a) Original Data
(b) Fitted with Absolute Error Criterion
(c) Fitted with Square Error Criterion

Figure 2.9 Depth for the "Cap" Image

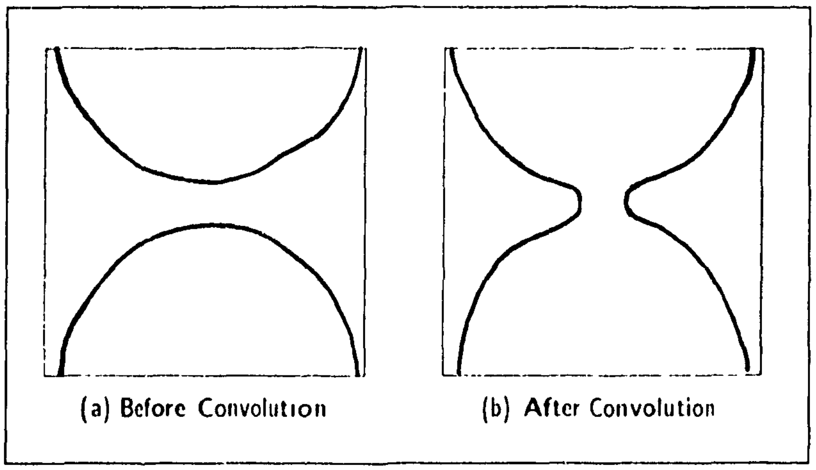
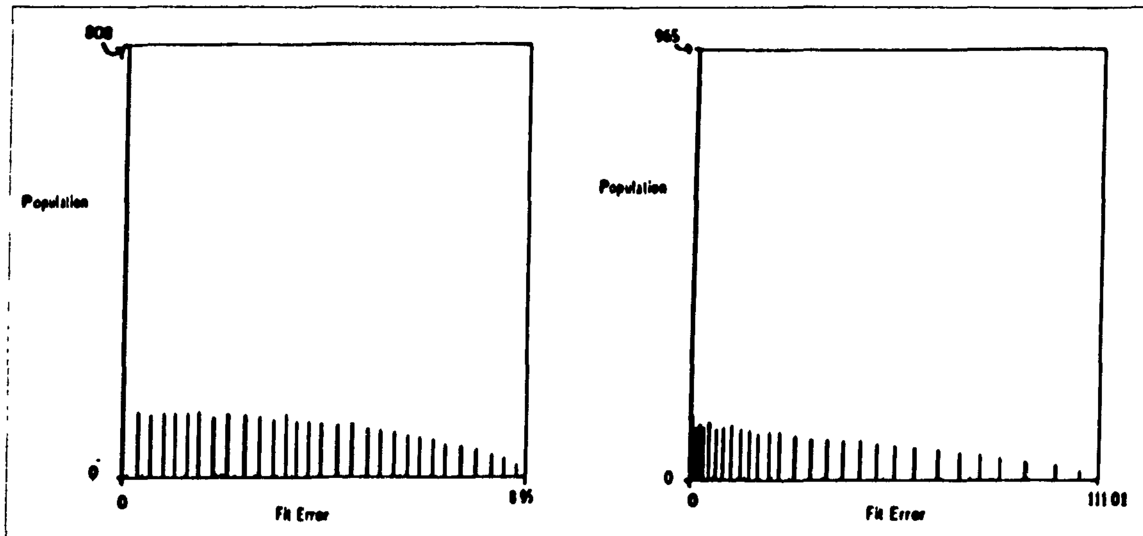


Figure 2.10 Merging of Real and Ghost "Caps" to Produce an Artifact

than the original data This is not very surprising, since the continuity assumption implicit in the fitting is violated there. Another interesting artifact in the fitted images occurs at the vertical and horizontal extremes of the "Cap" boundary. There has been a "pulling out" of the circular boundary. This is due to the way the algorithm extends the images To get a value of the convolution product for the fit in the first and last two rows and columns, the image was extended by reflection of the data. For example, to compute values in row 0 of the image, a row -1 and row -2 were needed by the 5 X 5 window convolution. Row 0 was reflected to row -1 and row 1 to row -2. This creates a "ghost" cap above the real cap. These two caps are merged slightly under the smoothing influence of the window operations. This effect is illustrated in Figure 2.10. With more foresight, this artifact could have been avoided by leaving a larger black border around the cap. However, since it does not interfere too much with the study, time was not invested in redoing these experiments



(a) With Absolute Error Criterion

(b) With Square Error Criterion

Figure 2.11 Histograms of Fit Error for the "Cap" Image

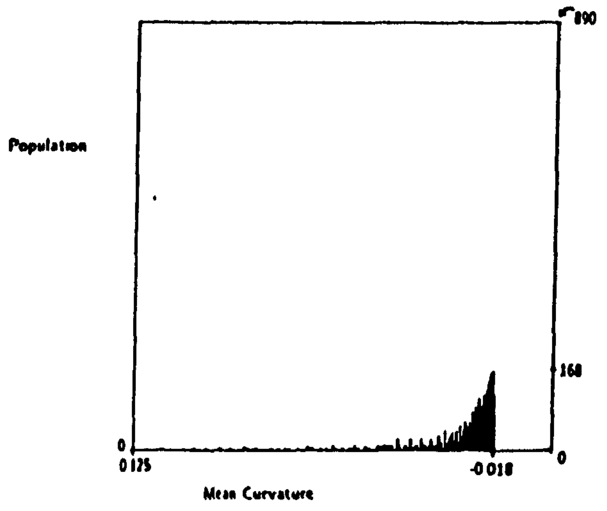
The maps of fit error studied in all the experiments of this chapter correspond to $|E_0|$ of equation (2.35) for the modified algorithm and E_0^2 of equation (2.33) for the original algorithm. Figure 2.11 shows histograms of the fit error obtained for the "Cap" image by the modified and original algorithms. Between the two algorithms, the modified (absolute criterion) algorithm shows more population in smaller fit-error-value bins for the "Cap" image.

The small arrows in Figure 2.11 represent a convention used in all the histogram plots. The plots are scaled such that the vertical height represents the population of the most-populous bin. The arrow points at this most-populous bin which frequently falls at the left or right extreme of the histogram and hence would not be noticeable without the small arrow. In the case of Figure 2.11, the most-populous bin is the 0 bin, indicating that a large number of points were fitted "perfectly" and therefore had zero fit errors. For many of the histograms shown later, the most-populous bin corresponds to the background points, but this is not the case with the fit error.

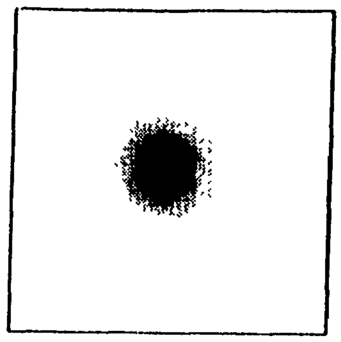
The curvature values for the "Cap" are not constant. This can be seen for the mean curvature in Figure 2.12 where a grey-scaled picture (a) and a histogram (b) for the ideal values are shown. These ideal values were computed by a simulation program according to the equations cited earlier as (2.40) and (2.41). The histogram height is determined by the 0 bin, corresponding to the many points in the flat ($H = 0$) background; the flat background points appear white in the picture (a) because they correspond to the maximum value of 0. The "Cap" itself appears increasingly dark toward the center, indicating negative mean-curvature values of increasing magnitude. The most negatively-curved area is the peak of the paraboloid located at the center of the picture (a).

Corresponding histograms and pictures for the extracted mean curvature maps also appear in Figure 2.12. Results from the modified algorithm (with the absolute error criterion) are shown in (c) and (d) Results from the original algorithm (with the square error criterion) are shown in (e) and (f). Several interesting features are apparent in comparing the ideal and extracted mean curvatures.

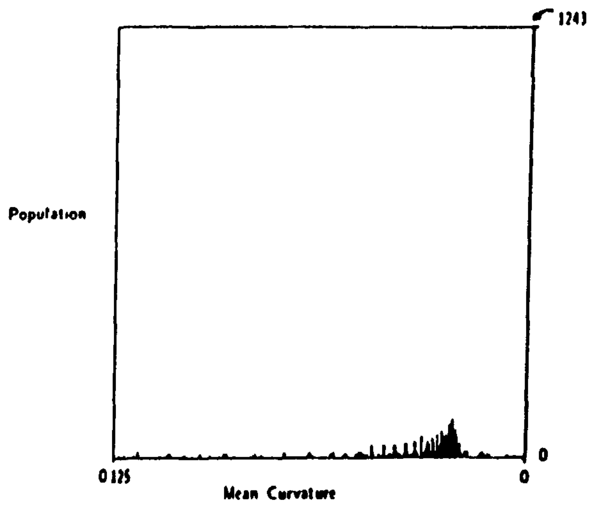
First, there is once again little observable difference between the results of the two algorithms. This is true of the histograms as well as the pictures Both algorithms have spread out the sharp discontinuity in the original histogram. This is most likely due to a smoothing-out of the discontinuity in angle where the paraboloid is truncated Note that the vertical scales of the histograms for the extracted curvatures are larger than for the ideal, with the total height (determined by the 0 bin) corresponding to populations near 1200, rather than 800 as in the ideal data's histogram. More points have been mapped into the 0 bin. This is also best explained by a smoothing effect near the rim of the "Cap". In the pictures this effect shows up as an apparent shrinkage of the grey area around the periphery. The "pulling-out" artifact described earlier is also visible.



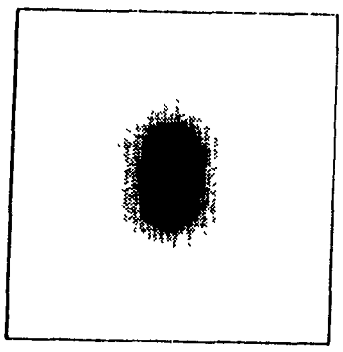
(a) Ideal Histogram



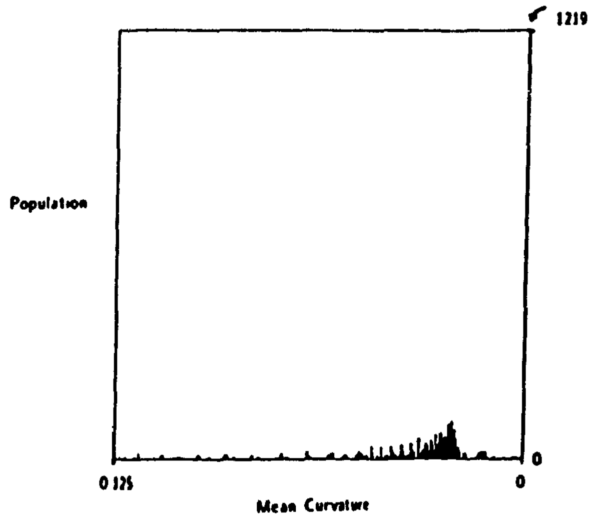
(b) Ideal Picture



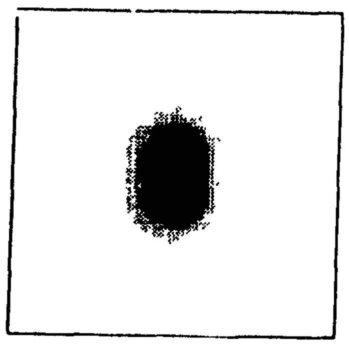
(c) Extracted Histogram. Absolute



(d) Extracted Picture. Absolute



(e) Extracted Histogram. Square



(f) Extracted Picture. Square

Figure 2.12 Mean Curvature for the "Cap" Image

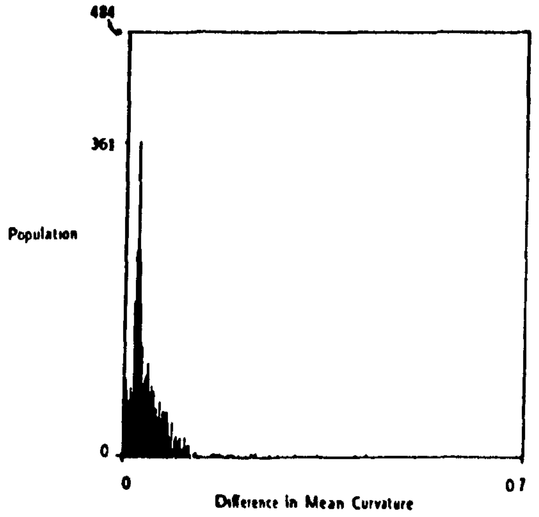
An unexpected effect appears at the peak of the cap. It appears to have been stretched along the y-axis and compressed slightly along the x-axis. This effect is believed to be related to a bias in the selection of the best window position (u_*, v_*); the existence of such a bias is confirmed in a later experiment (Figure 2.16). Since the bias is present equally for both algorithms, it does not seriously affect our comparative study of the two algorithms.

In order to examine the discrepancies between the ideal and extracted mean curvatures more closely, the absolute differences between these were computed. Figure 2.13 shows the absolute-difference maps in both picture and histogram form for the modified (or absolute) and original (or square) algorithms respectively. Some further statistics are shown in the table of Figure 2.14.

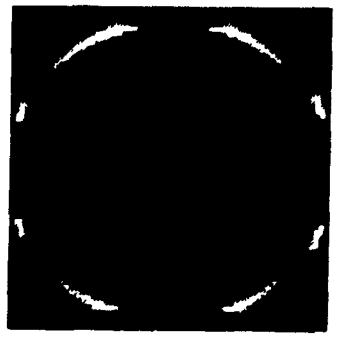
Once again, there is little observable difference in the histograms or the pictures corresponding to the absolute differences in mean curvature for the two algorithms. The pictures show an expected effect in that the maximum errors are found around the periphery where continuity is violated These maximum errors show up in white The disturbance of this pattern at the vertical and horizontal extremes can be explained by the way the image is extended as was the case for the "pulling-out" artifact. However, unexpected effects in the pictures include two internal circular contours in the upper part of the image

The statistics of the absolute differences in mean curvature are quite similar as well according to Figure 2.14. The original algorithm appears to have slightly smaller errors (differences), as shown by the mean (average) values. However, the margin separating the two is very narrow. As a basis for comparison, the mean curvature values (which were subtracted to give the differences of Figure 2.14) ranged between -0.125 and 0 for the "Cap". The differences in curvature have means near 0.05, which constitutes a sizeable fraction. The raw numbers showed, however, that the periphery (corresponding to the white areas in the pictures of Figure 2.13) accounted for virtually all of the differences.

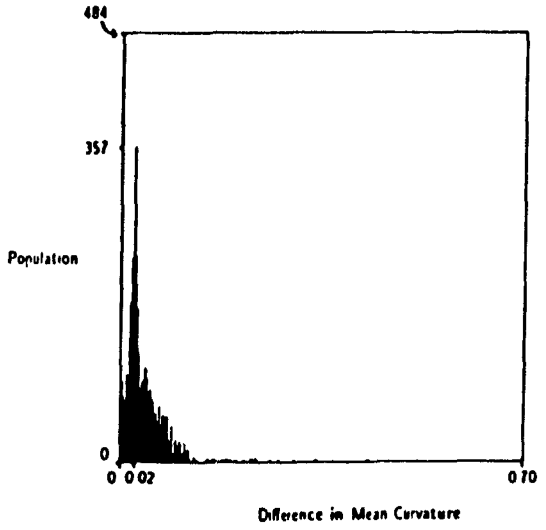
Figure 2.15 shows the ideal and extracted values of Gaussian curvature for



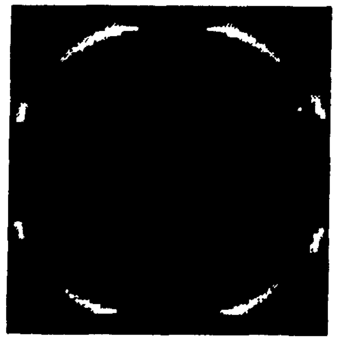
(a) Histogram.
|Ideal - Absolute|



(b) Picture.
|Ideal - Absolute|



(c) Histogram.
|Ideal - Square|



(d) Picture.
|Ideal - Square|

Figure 2.13 Difference of Ideal and Extracted Mean Curvatures for the "Cap" Image

Algorithm	Mean	Variance	Max
Absolute	0.050660	0.005151	0.693747
Square	0 049412	0.004785	0.693743

Figure 2.14 Statistics of Absolute Differences Between Ideal and Extracted Mean Curvature for the "Cap" Image

the "Cap" image. The fact that the Gaussian curvature is positive is visible in that the background is black (corresponding to the minimum of zero for the flat background points) and the cap itself is shown becoming progressively lighter (more positive) towards the center (corresponding to the most positively curved area at the peak) Artifacts visible for the Gaussian curvature are similar to those seen in the mean curvature The ideal Gaussian curvatures ranged from 0 to 0.0156

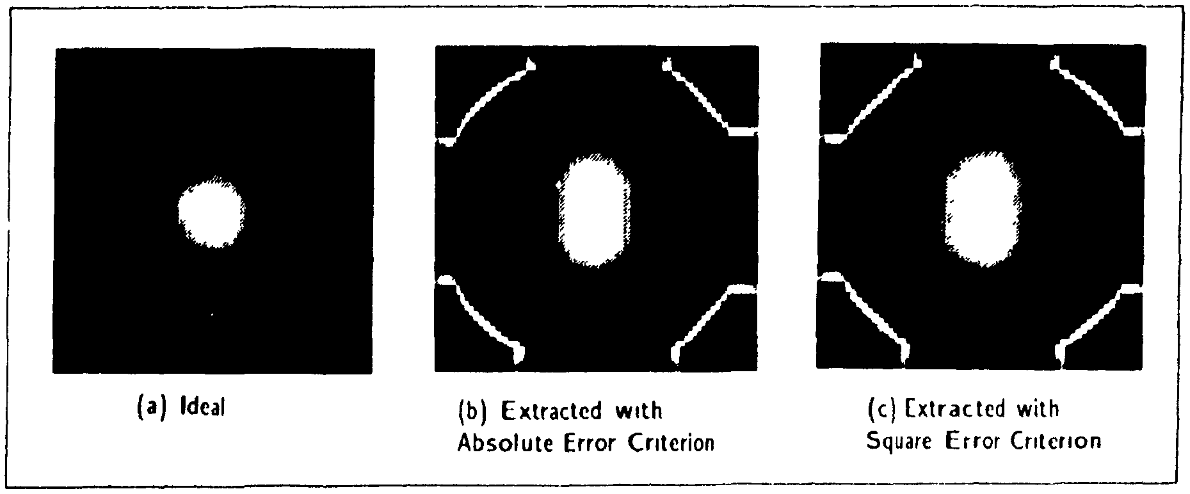


Figure 2.15 Gaussian Curvature for the "Cap" Image

As was described earlier in this chapter, a search is made by both algorithms to find the best window position at each pixel in the range image, where the search space about each pixel is equal to the window-operator size The choice of the best window position is determined using a criterion based on the fit errors which are aggregates of the differences between the original and fitted depth values The original algorithm minimizes the sum of the squares of the fit errors, while the modified algorithm minimizes the sum of the absolute magnitudes of the fit errors within the square neighborhood surrounding the pixel under consideration The result of this search is an offset (u_*, v_*) where u_* and v_* can be either positive or negative and have a maximum value of half the window size The asterisks, which indicate the optimum choice of offset (u, v) where the minimum error is found, are dropped at this point for simplicity. Let the asterisks be implicit in the following discussion of the offset results.

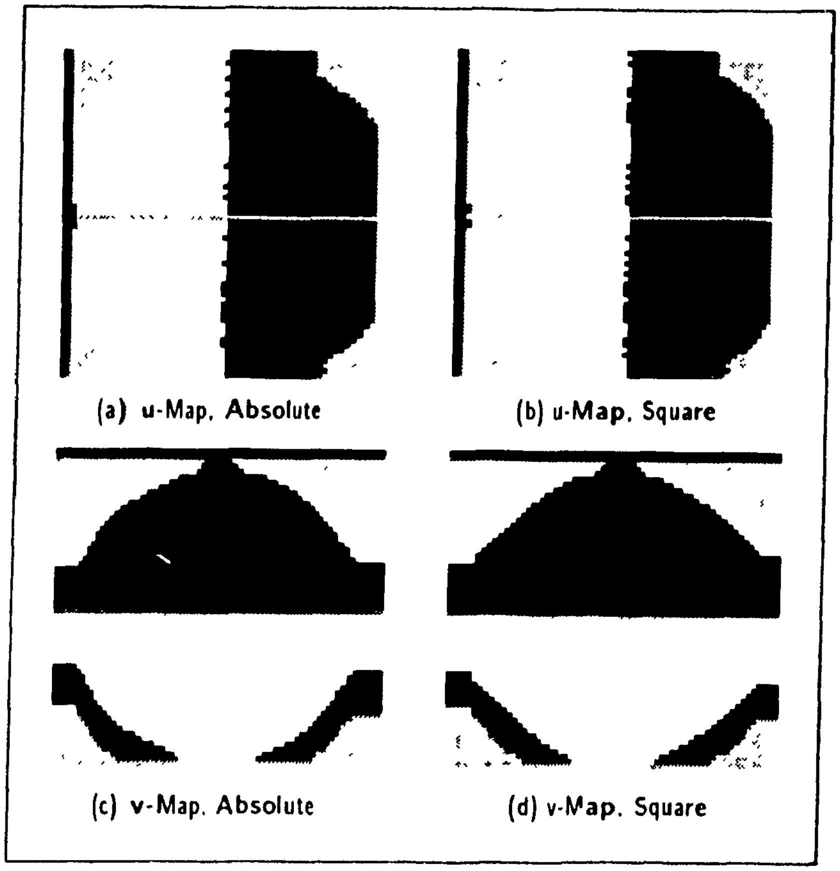


Figure 2.16 Choice of the Best Window Position Offset (u,v) for the "Cap" Image Using Absolute and Square Error Criteria

In the case of the "Cap" image, a window size of 5 X 5 was used. Hence, u and v range between -2 and +2 Figure 2.16 shows the maps of u and v produced by the two algorithms Black represents -2 and white +2 in these images, while medium-grey is zero, light grey +1, and dark grey -1 Although some minor differences can be observed between the results of the two algorithms (for example in the u-maps along the vertical in the center), the trend is the same for both. There appear to be consistent biases in the choice of the best window position

The co-ordinate axis definition is such that the positive x direction is toward the right and the positive y direction is downward, the origin (0,0) is in the center. On the u-maps, we have white (+2) on the left halves (negative x values) indicating a shift toward zero; the right halves (positive x values) appear black (-2) which is again a shift towards zero. Hence, along the x direction, there is an attraction towards the origin (where the peak of the "Cap" is). The situation is reversed for the v-maps. The top halves (negative y values) are black (-2) indicating a negative (upward) shift away from zero; the bottom halves (positive y values) appear white (+2) indicating a positive (downward) shift which is again away from zero. Hence, along y, there is repulsion away from the origin (i.e. away from the "Cap" peak.) This anisotropic, or biased, behavior (attraction along x versus repulsion along y) shows a similar trend to that of the distortion of the peak observed in the curvature images (compression along x and elongation along y).

One would expect that inside a smooth region such as the "Cap" interior, the choice of (u, v) would be (0,0) everywhere. This was not the case in Figure 2.16, as was noted above The best-window-position mechanism was designed to shift away from discontinuities and the discovery that it is active even in smooth regions indicates that it does not operate exactly as it is supposed to. Since both algorithms exhibited this biased behavior, the bias did not affect our conclusion that the two algorithms have a very similar performance. Hence, further investigation into the cause of the bias was not carried out.

2.4.2.2 Artificial Range Image 2: The Spherical "Bowl"

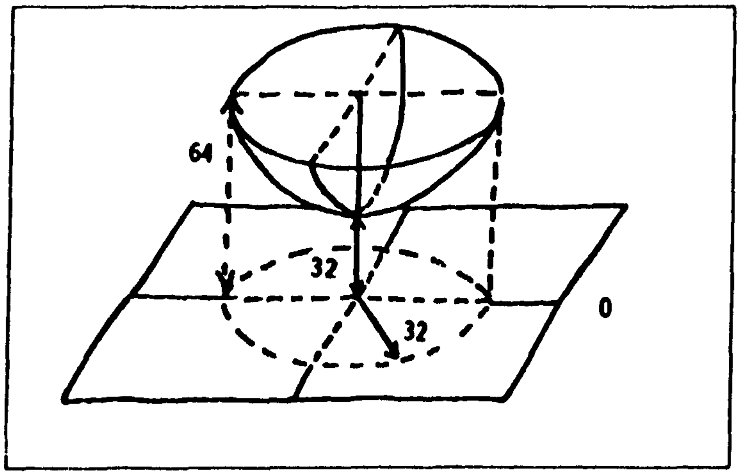


Figure 2.17 Spherical Surface Segment for the "Bowl" Image

For the second artificial range image, a hemispherical surface segment was chosen. Since this resembles a bowl, it will be called the "Bowl" image for brevity. Figure

2.17 shows a diagram suggesting the aspect in 3d of this "Bowl". It is oriented concavely with respect to the hypothetical range-finder. Hence it is a pit surface and its curvatures are such that

$$H > 0, K > 0$$

In fact, the curvatures are constant for a sphere, which is why it was chosen for the experiment. The curvatures of the "Bowl" are given by:

$$H = -1/r \tag{2.42}$$

$$K = 1/r^2 \tag{2.43}$$

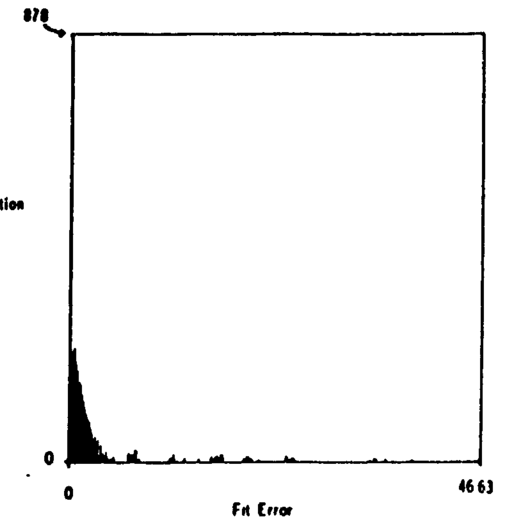
where "r" is the radius of the sphere[†]. For the "Bowl" image, the radius was 32.

The "Bowl" image contains not just a slope discontinuity as was the case for the "Cap" but a discontinuity in depth as well at its periphery. The rim of the bowl is at a depth of 64 and just adjacent to it are background pixels at a depth of 0. Analytically, the equation of the surface segment for the "Bowl" image is as follows:

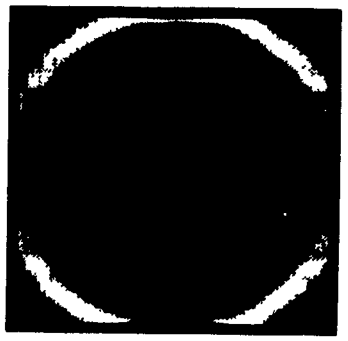
$$z = k - \sqrt{r^2 - (x^2 + y^2)} \tag{2.44}$$

The "k" in the above equation was 64 for the "Bowl" image and the radius r was 32 as mentioned above.

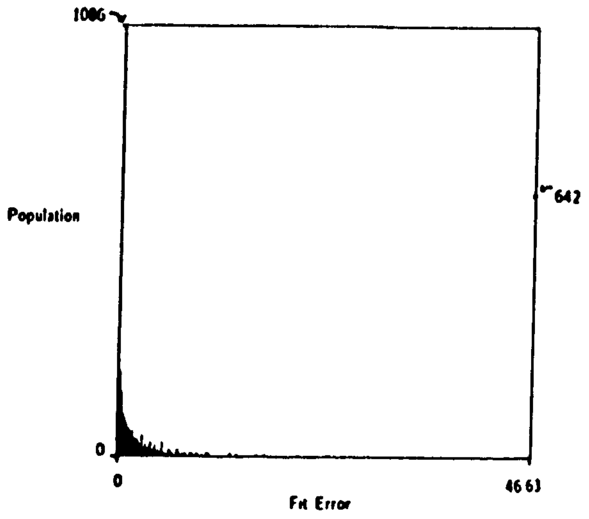
[†] As in [Besl86], pp. 51-52.



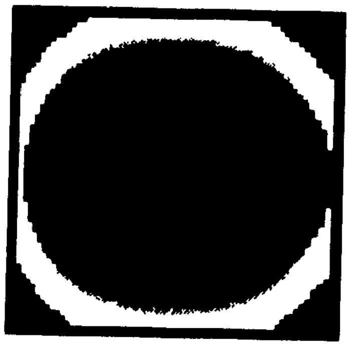
(a) Histogram, Absolute



(b) Picture, Absolute



(c) Histogram, Square



(d) Picture, Square

Figure 2.18 Fit Error for the "Bowl" Image Obtained Using Absolute and Square Error Criteria with 5X5 Window

Figure 2.18 shows histograms and grey-scaled pictures representing the fit errors obtained with the modified (absolute) and original (square) algorithms A window size of 5 X 5 was used with no added noise in both cases illustrated in Figure 2 18 The white rings visible in both pictures indicate that the error is concentrated around the periphery where the depth discontinuity exists. This is as expected. The original algorithm produces larger fit error values, as the histograms and pictures demonstrate. The original algorithm's fit-error measure is a squared quantity, however, while the modified algorithm's fit error is a linear measure (the absolute value). Hence, it is reasonable that the original algorithm's fit error be larger in magnitude.

Since the original algorithm's fit-error values were much larger (the maximum error was in fact 2360.42) than those of the modified algorithm, the histogram and picture of fit error for the original algorithm were scaled to the fit-error range of the modified algorithm. This is why the population of the last bin in the original algor.:hm's histogram is so large. This bin contains fit errors ranging all the way from 46.63 (the modified algorithm's maximum fit error) to 2360.42. The thicker white ring in the original algorithm's picture also illustrates the larger fit-error values produced by the original algorithm.

Experiment	Mean	Variance	Max
"Bowl", Absolute	0.632389	0.169580	1.515084
"Bowl", Square	0 791196	0.634932	3.191427
"Cap", Absolute	2.400006	1.934995	4.800005
"Cap", Square	10.687500	91.910156	32.000000

Figure 2.19 Statistics for Fit Errors in 1024-Pixel Regions of the "Bowl" and "Cap" Images

Having looked at the trend in the whole fit error image (which is dominated by the edge effects), more insight can be gained by studying the fit-error values produced in the interior of the surface. Let us consider these fit errors and compare them to those for the "Cap" image. The table of Figure 2.19 shows the fit error statistics obtained from 1024-pixel regions within each surface including no boundary points. The fit errors for both algorithms are substantially larger for the "Cap" image. This is probably because the "Cap" has a rapidly-changing curvature to its surface while the "Bowl" is more gently curved. The fit-error means of Figure 2.19 also indicate that the original algorithm produced larger fit errors for both images. This mirrors the trend of the edge-effect fit errors.

Since the "Bowl" has constant curvature values, it was selected mainly to observe how the addition of noise affects the extraction of curvatures. Before adding any

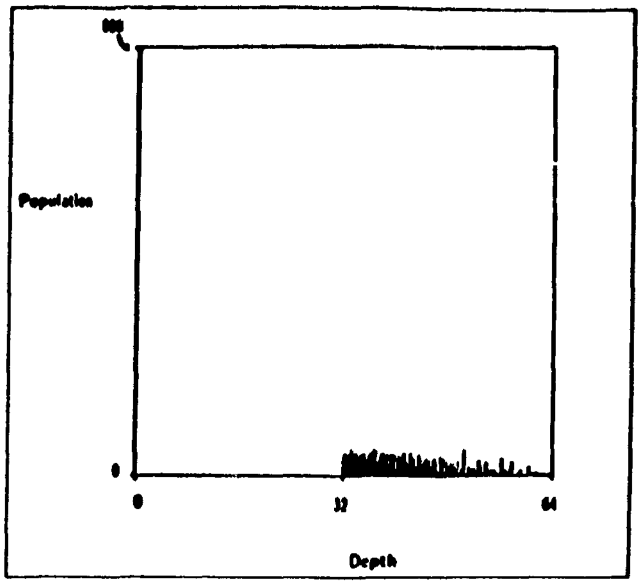


Figure 2.20 Histogram of Original Depth Values for the "Bowl" Image

noise, the "Bowl" image had a histogram as shown in Figure 2.20.

Figure 2.21 shows grey-scaled pictures representing the original and fitted depth maps as various amounts of noise were added. The noise was synthesized using a pseudo-gaussian noise process having zero mean and different standard deviations as shown by the symbol σ in the figure. Processing for these pictures was done with a 5 X 5 window-operator size.

The first row of pictures in Figure 2.21 illustrates the case of no noise. The second row shows results when a noise process with a standard deviation of 0.5 was added to the depth map before curvature extraction. Since the depth values are mostly around 32 for the "Bowl", this corresponds to between 1 and 2 percent noise. This amount of noise is not visible in the pictures. However the curvature statistics, which will be considered next, show that this noise level is enough to affect the curvature computations. In the third row of Figure 2.21, the added noise was increased to have a standard deviation of 5.0, which corresponds to about 15 percent noise. This noise is plainly apparent in the pictures.

Figures 2.22 and 2.23 show statistics for the Gaussian and mean curvatures respectively for different window sizes as noise was added. These statistics were obtained

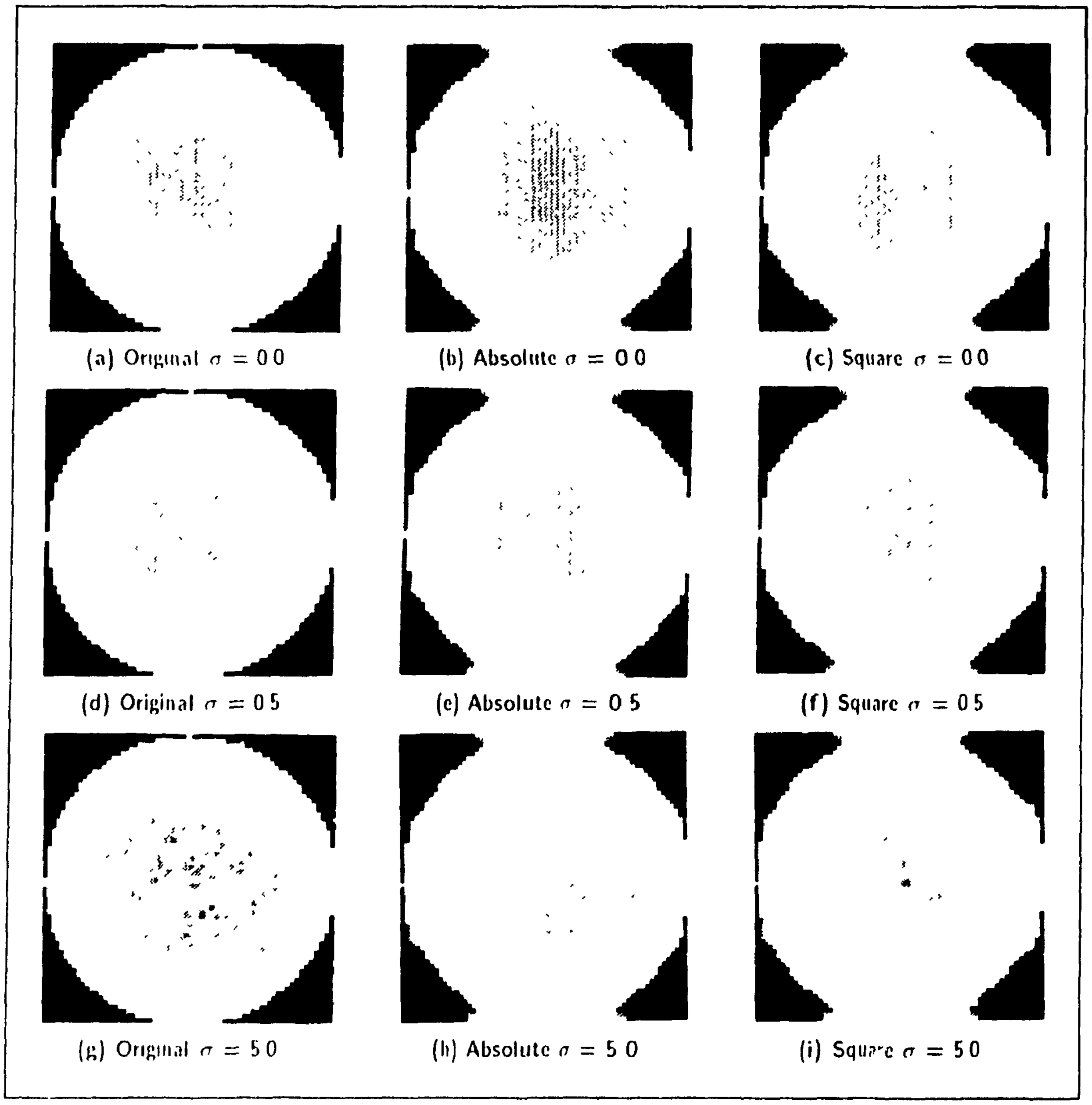

(a) Original $\sigma = 0.0$
(b) Absolute $\sigma = 0.0$
(c) Square $\sigma = 0.0$
(d) Original $\sigma = 0.5$
(e) Absolute $\sigma = 0.5$
(f) Square $\sigma = 0.5$
(g) Original $\sigma = 5.0$
(h) Absolute $\sigma = 5.0$
(i) Square $\sigma = 5.0$

Figure 2.21 Original and Fitted Depth for the "Bowl" Image versus Noise Deviation for 5X5 Window

for regions containing 1024 pixels in the interior of the "Bowl" excluding any edge pixels. In Figures 2.22 and 2.23, the symbol σ represents the standard deviation of the noise added to the image and the symbol $\triangle$ % represents the signed percent difference between the extracted and ideal curvature values. In general, the algorithms did a better job extracting the mean curvature than the Gaussian curvature. The fact that the Gaussian curvature is by nature more noise-sensitive than the mean curvature has been noted by other researchers

Experiment	Mean	Δ%	Variance
Ideal K	0.0009765625	0.0	0.0
$\sigma = 0.0$			
3X3 absolute *	0.000912	-6.6	< 0.000001
3X3 square *	0.000912	-6.6	< 0.000001
5X5 absolute *	0.000971	-0.6	< 0.000001
5X5 square *	0.000971	-0.6	< 0.000001
7X7 absolute *	0.000922	-5.6	< 0.000001
7X7 square *	0.000990	+1.4	< 0.000001
$\sigma = 0.5$			
3X3 absolute	-0.009665	-1089.7	0.068325
3X3 square	-0.063501	-6602.5	0.041784
5X5 absolute	-0.001073	-209.9	0.000133
5X5 square	-0.000416	-142.6	0.000116
7X7 absolute *	0.000455	-53.4	0.000006
7X7 square *	0.000308	-68.5	0.000005
$\sigma = 5.0$			
3X3 absolute	-0.642017	-65842.5	36.226621
3X3 square	-0.636277	-65254.8	36.212760
5X5 absolute	-0.010311	-1155.9	0.068384
5X5 square	-0.063501	-6602.5	0.041784
7X7 absolute	-0.012378	-1367.5	0.002732
7X7 square	-0.010582	-1183.6	0.001675

Figure 2.22 Statistics of Gaussian Curvatures in 1024-Pixel Constant-Curvature Regions versus Noise Deviation and Window Size

[Besl86]. The fact that the Gaussian curvature value is such a small number (1/1024 or 0.0009765625 compared to 1/32 or 0.03125 for the mean curvature) may also be a factor here. This second concern must be addressed by providing proper scaling to handle large dynamic ranges. The use of floating-point representation is one approach to this issue which is used in this thesis.

In order to perform segmentation using the curvatures, the algorithms must at least get the signs correct. Such results are marked with an asterisk in Figures 2.22 and 2.23 and can be deemed acceptable results. The successful cases are summarized in a yes/no form in Figure 2.24, where the appearance of a K or H implies successful extraction of the Gaussian or mean curvature signs respectively and subscripts A and S refer to the modified (absolute) and original (square) algorithms respectively.

Experiment	Mean	Δ%	Variance
Ideal H	0.03125	0.0	0.0
$\sigma = 0.0$			
3X3 absolute *	0.030197	-3.4	< 0.000001
3X3 square *	0.030197	-3.4	< 0.000001
5X5 absolute *	0.031158	-0.3	0.000001
5X5 square *	0.031158	-0.3	0.000001
7X7 absolute *	0.030319	-3.0	0.000004
7X7 square *	0.031464	+0.7	< 0.000001
$\sigma = 0.5$			
3X3 absolute	-0.063501	-303.2	0.041784
3X3 square	-0.063501	-303.2	0.041784
5X5 absolute *	0.018474	-40.9	0.005186
5X5 square *	0.025255	-19.2	0.004769
7X7 absolute *	0.031407	+0.5	0.000645
7X7 square *	0.026117	-16.4	0.000709
$\sigma = 5.0$			
3X3 absolute	-0.642017	-2154.5	36.226621
3X3 square	-0.636277	-2136.1	36.212760
5X5 absolute *	0.004570	-85.4	0.069974
5X5 square *	0.009381	-70.00	0.068711
7X7 absolute *	0.021356	-31.7	0.014696
7X7 square	-0.000699	-102.2	0.017766

Figure 2.23 Statistics of Mean Curvatures in 1024-Pixel Constant-Curvature Regions versus Noise Deviation and Window Size

Surprisingly, almost all the errors are negative for both mean and Gaussian curvature in Figures 2.22 and 2.23. Recall that the "Bowl" is a pit surface, both of its curvatures are positive but small. Hence, the acceptable cases can also be described as those that possess signed percentage differences relative to the ideal of greater than -100 percent. For example, -85 percent is acceptable, as is +0.5 percent. No large positive percent differences were found.

All three window sizes gave beautifully accurate results for both algorithms for the noise-free case. However, once a small amount of noise is added, the results for the 3 X 3 window are meaningless for both mean and Gaussian curvature. A larger window size is needed to average out the effect of the noise. In the case of the Gaussian curvature, even the 5 X 5 window cannot overcome the 0.5 noise level. For mean curvature, results are

	$\sigma = 0$	$\sigma = 0.5$	$\sigma = 5.0$
3X3	H_A, H_S K_A, K_S		
5X5	H_A, H_S K_A, K_S	H_A, H_S	H_A, H_S
7X7	H_A, H_S K_A, K_S	H_A, H_S K_A, K_S	H_A

Figure 2.24 Successful Cases of Curvature-Sign Extraction

acceptable with the 5 X 5 window The 7 X 7 window manages to extract both curvatures acceptably well in the presence of the 0 5 level noise

At the larger noise level of 5 0, the 3 X 3 window is hopelessly inadequate For Gaussian curvature, even the two larger window sizes are insufficient Perhaps a 9 X 9 would be able to handle this case For mean curvature, both algorithms succeed with the 5 X 5 window. Although the modified (absolute) algorithm gives a better result with the 7 X 7, strangely, the original algorithm fails with the 7 X 7 after succeeding with the 5 X 5. This last result is suspect Due to limited time, it was not investigated further since the body of other results show clear evidence of very similar performance for the two algorithms

In general, these results confirm the fact that larger window operators are needed to handle images containing more noise There is a tradeoff, however, because larger operators obscure fine detail along with settling the effects of the noise. This experiment was not designed to illustrate the tradeoff, since no fine detail exists in the homogeneously-curved regions studied.

2.4.3 Real Range Images

In this section, results of applying the modified algorithm with 5 X 5 window operators to real range images are presented. Other research efforts have indicated that the real images, obtained from NRC in Ottawa, are of excellent quality and possess less than 1 or 2 percent noise.

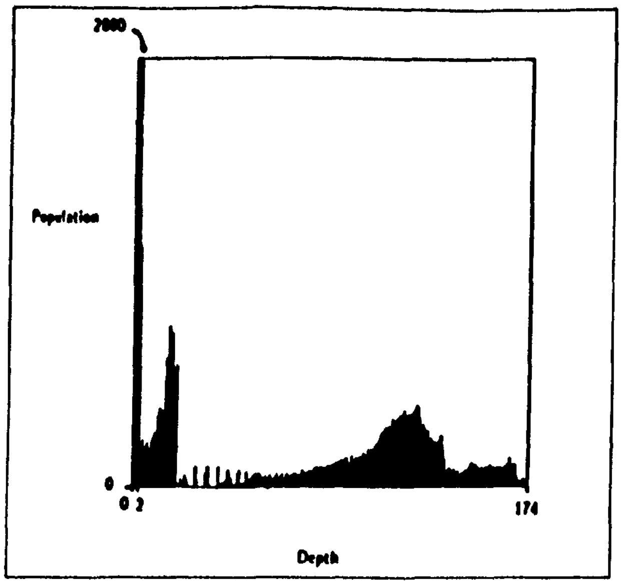


Figure 2.25 Histogram of Original Depth Values for Real Range Image "Mas5"

2.4.3.1 Real Range Image 1: The Carved Mask of a Face

The first real range image that was chosen features a mask of a face. It is one of a series of images of this mask and hence is called "Mas5". This image has many curved surfaces, featuring peaks, pits, and ridges. It was chosen to observe the algorithm working on a diverse, curved image. It is also reasonably smooth except around the edges where there is a depth discontinuity. Figure 2.25 shows a histogram of depth for the original "Mas5" image. Figure 2.26 shows results in the form of grey-scaled pictures produced by the modified algorithm applied to "Mas5" using 5 X 5 window operators.

Parts (a) and (b) of Figure 2.26 show grey-scaled pictures of the original and fitted depth. Note that the two images agree quite nicely. The fitted image does not appear to contain any artifacts of the type seen in the artificial data. The features of the face in "Mas5" appear to have been enhanced slightly by the fitting operation.

Part (c) of the figure shows a grey-scaled picture representing the fit error. The error is concentrated around the periphery of the mask, with a barely-perceptible amount around the nose as well. These areas feature sharp transitions or discontinuities, so some error is expected there. The errors ranged between 0 and 80.60, with a mean of 2.51 and

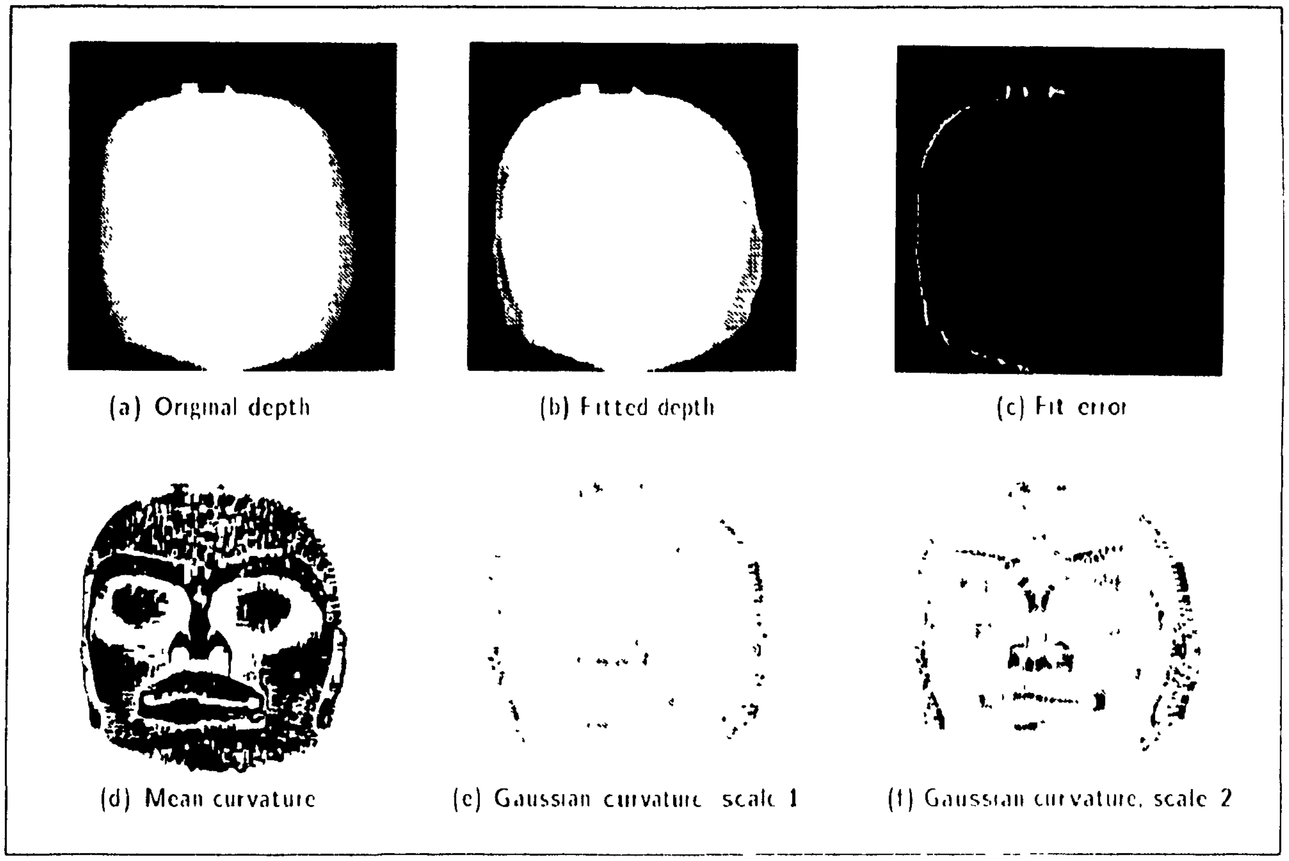


Figure 2.26 Results from Applying the Modified Algorithm to Real Range Image "Mas5

a variance of 18 9.

Part (d) of Figure 2.26 shows a grey-scaled picture representing the mean curvature map produced by the algorithm In this picture, the mean curvature appears positive (white) for pits and valleys such as the eye sockets, nostrils, and mouth crease Negative mean-curvature areas appear black in the picture. These are peaks and ridges such as the bridge of the nose, the eyes, the cheekbones, and the lips The forehead is more gradually curved, being almost flat in places, both black and white areas are found there This map of mean curvature is quite satisfactory The mean curvature values ranged between -1.88 and 8.02 for this image.

Grey-scaled pictures representing Gaussian curvature for "Mas5" are given in parts (e) and (f). The Gaussian curvature results are shown scaled to two different ranges to try and bring out the trend of the data. Part (e) is scaled to the range $(-0.06, 0.06)$ and part (f) to the range $(-0.006, 0.006)$. The differences between these pictures highlights the important problem of selecting threshold levels for the curvature maps once they have been extracted. The threshold-level-selection problem has not been addressed here. Although outside the scope of the thesis study, this issue must be dealt with if the curvature information is to be useful to higher levels of range-image processing such as segmentation.

2.4.3.2 Real Range Image 2: Blocks

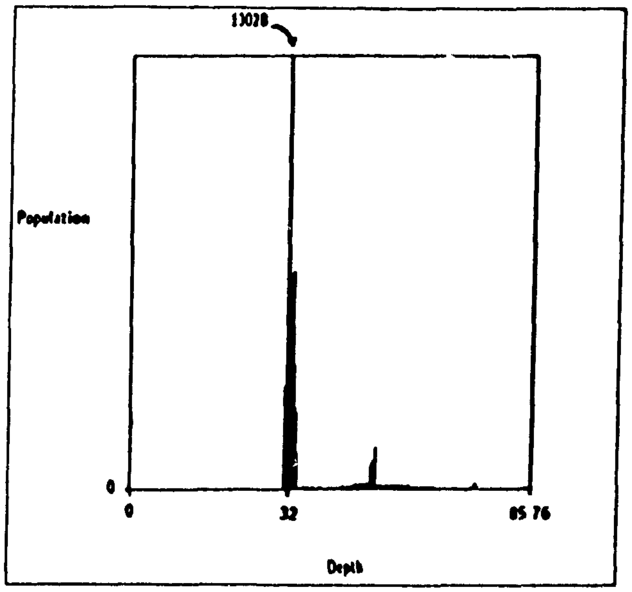


Figure 2.27 Histogram of Original Depth Values for Real Range Image "Dypro6"

The second real range image studied contained several blocks. It was one among a series of range images containing assortments of objects and is named, somewhat enigmatically, "Dypro6". It was chosen to show the results from an image with planes and angles. Figure 2.27 shows a histogram of the original depth values for this image. The "Dypro6" image is different from previous images studied here in that the background depth is not zero but rather 32. This is the tallest peak in the histogram. The smaller peak corresponds to the objects (blocks).

Figure 2.28 shows grey-scaled pictures of the results of processing the "Dypro6" image with the modified algorithm and 5 X 5 window operators. The original and fitted

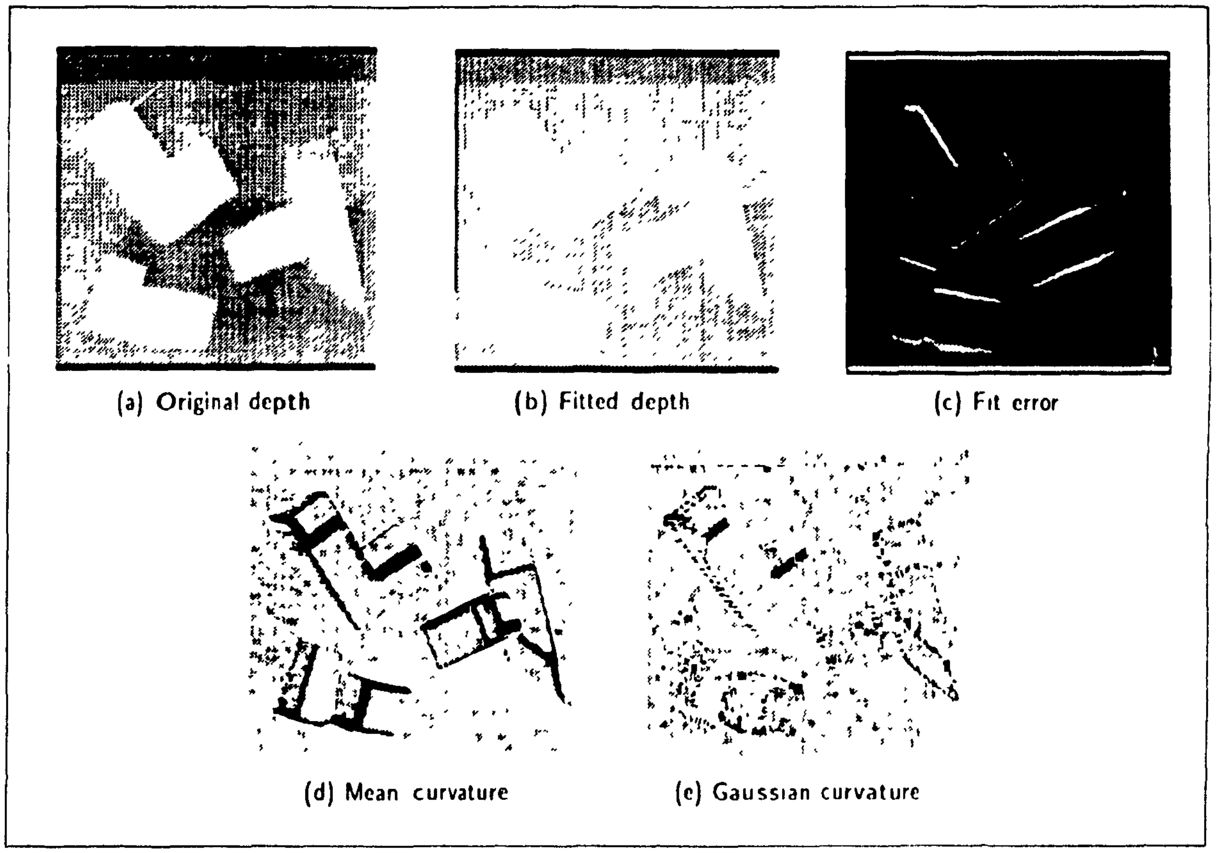


Figure 2.28 Results from Applying the Modified Algorithm to Real Range Image "Dypro6"

depth maps are observed to be very similar There is an absence of artifacts The 5 X 5 window does not appear to have removed any detail from the objects, even though these objects are smaller compared to objects previously studied The picture representing fit error shows that error is concentrated around the peripheries of the blocks, as expected since depth discontinuities exist there The discontinuities in surface orientation which can be inferred in the blocks do not seem to have generated significant fit errors

The mean and Gaussian curvature maps of Figure 2 28 show a predominance of grey. Black in these pictures represents negative values and white positive values Hence the grey represents values near zero. The surfaces of the blocks are planar, and for flat surfaces we expect both the mean and Gaussian curvature to be zero This is largely what was obtained. Only the edges of the blocks produced significant curvature values. In fact, angle edges (or roof edges) are lines of high-curvature points They resemble ridges. These are the black areas in the mean curvature image. In fact, the curvature maps have brought out some edges not visible in the original depth map but which a human would probably have added if asked to trace out the block edges. The purpose of the curvature extraction is, of course, not to detect edges, but it is a potentially useful result nonetheless.

Having explored the background, the methodology, and the performance of the curvature extraction algorithm, let us now consider its implementation in hardware. This is the subject of the next chapter.

Chapter 3 Hardware Architecture

After studying the curvature extraction algorithm, efforts to develop special-purpose hardware were directed towards the fit-error computation step. The fit-error computation is not only the most execution-time intensive step in the algorithm, but it is also one of the algorithm's most novel aspects and, being neighborhood-oriented, it is well-suited for a VLSI array implementation. In addition, the introduction of the magnitude error criterion yielding the modified version of the algorithm in Chapter 2 was motivated by expected simplifications in a circuit for computing the fit error It is for these reasons that the fit-error computation was selected as the primary focus of the hardware design effort. The result of this work is an architecture for a new floating-point VLSI systolic-array processing element to perform the fit-error computation. An architecture for a dedicated hardware environment is also proposed to assist the new VLSI processor so that the entire curvature extraction can be carried out more efficiently. Hence there are really two architectures proposed in this chapter, one for a VLSI cell and one for a dedicated computing environment.

An overview of this chapter is as follows. First, some general background on technology, trends, and terminology in VLSI will be presented as an introduction. Some application examples from the literature will then be considered as a motivation for the new VLSI cell development. The dedicated hardware design featuring the new floating-point VLSI fit-error processor as well as an earlier floating-point VLSI convolution processor is described next with an emphasis on the estimated performance of each component subsystem. The chapter concludes with an overall performance estimate for the curvature computation in this environment.

3.1 Background on Technology, Trends, and Terminology in VLSI

There are several logic families currently used for the design of integrated circuits and each has its own characteristic advantages and disadvantages. The *TTL* (or *transistor-transistor logic*) family enjoyed wide popularity in the 1970's and is still frequently used. The TTL components have the advantage that their functional characteristics, such as their transition times, are not strongly influenced by loading so long as their rated currents are observed. The ECL (or emitter-coupled logic) family provides the advantage of high speeds. Both of these families are based on the *bipolar junction transistor*, or *BJT*.

Families based on the *field-effect transistor*, or *FET*, typically use the *MOS* technology (or metal-oxide-semiconductor) in which a "sandwich" of these three materials forms the transistor. The principle behind the MOS transistor was proposed by J. Lilienfeld as early as 1925 while a structure similar to that used today was proposed by O. Heil in 1935; commercial use of the MOS transistor did not ensue until the late 1960's[†] Although it has since been prevalent in analog applications such as the design of high-quality amplifiers, the MOS transistor has also found increasing use as a switch for logic operations in digital circuits.

The *nMOS* family and, to a lesser extent, the *pMOS* family, enjoyed initial popularity since they each employ only one polarity of transistor and thus have simple fabrication processes. However, the *CMOS* (or *complementary-symmetry MOS*) family which uses transistors of both polarities is now the preferred MOS family. It has the advantage of symmetric logic levels and very low power consumption. Its higher silicon area requirements and more complex fabrication process which initially made it unpopular

[†] As in [West85], p.4

have been addressed by technology improvements such as better lithographic and process-control techniques. The result of these developments is that CMOS is now widely used for high-complexity digital micro-electronics such as semi-custom gate arrays and custom commodity parts[†].

Very large scale integration (or *VLSI*) technology has surpassed the limits of the earlier small-scale (SSI), medium-scale (MSI) and large-scale (LSI) integration technologies to produce circuits containing thousands[‡] of transistors on a single chip. The embodiment of detailed process knowledge into technology-specific design rules, pioneered by Mead and Conway [Mead80], has resulted in the development of VLSI designs by a broader base of engineers. In both industrial and university environments, VLSI design is becoming increasingly synonymous with CMOS design. One prime reason for this is the fact that the CMOS design rules scale down with few changes as technology improvements continue to shrink circuit dimensions[††].

Array processor architectures [Kung88] have been made popular by the growth of VLSI technology In contrast to the traditional Von Neumann architecture, array processor architectures stress parallelism, local interconnection of simple processing elements, and decreased generality. They are often algorithm-specific. Their regularity and modularity facilitate efficient VLSI implementations. They offer promising solutions to meet real-time processing requirements on huge amounts of data for specific tasks; more traditional, general-purpose architectures do not.

Array processor architectures can be classified into several types. The types discussed here are as described by Kung in [Kung87] and in Chapter 1 of [Kung88]. The

---

† As in [West85], p. ix.

‡ As in [Mukh86], p 1

†† CMOS fabrication processes with minimum feature sizes of 1 to 3 microns are currently in use. Sub-micron technologies are under development.

more general types of array processor are *single-instruction, multiple-data* (*SIMD*) structures and *multiple-instruction, multiple-data* (*MIMD*) structures.

The SIMD structure is one in which the processors possess local connectivity and each has its own local memory. A host computer broadcasts each instruction which is carried out by all processors in the array simultaneously; each processor operates on its own distinct data stream. Typically, the instruction set of the processors is quite limited, but there is still the flexibility inherent in how these instructions are put together to form a program.

The MIMD structure is one in which the processors each possess their own control unit, program, and data. The overall task is distributed among the processors in an effort to maximize parallelism. The efficiency of this structure can be diminished by contention among *processing elements* (*PE's*) for access to shared resources. Inter-processor synchronization is another problem frequently responsible for decreasing efficiency. The MIMD structure, although more versatile, has not been as popular as the SIMD for these reasons of efficiency.

Systolic arrays and *wavefront arrays* are popular for special-purpose VLSI designs. They are often algorithm-specific.

In a systolic array, computation is pipelined so that data is rhythmically processed and then passed on to the next processor. When the pipeline is fully filled, maximum parallelism is realized as each processor regularly pumps data in and out under the control of global timing "beats" and performs some computation on the data within the interval. Hence a systolic array is synchronous, or "clocked". Because of the global synchronization required, problems with clock skew and peak-power demands ensue as the systolic array becomes large.

In a wavefront array, pipelining and parallelism are also important. However, these arrays utilize the principle of *dataflow computing*. This approach is asynchronous.

requiring no global synchronization from outside. The array synchronizes its own computation according to the arrival of data from neighboring processors. This is interpreted as a change of state and invokes action. Since global synchronization and control are not required, clock skew and peak power levels are not as serious a problem as they are for the systolic array. However, correct sequencing for the wavefront array can be complex. An analysis based on timed petri nets is suggested for such sequencing in Chapter 5 of [Kung88].

Systematic methods for implementing algorithms with array processor architectures are being explored. This work is likely to popularize the use of such structures if their inherent complexities can be made more manageable. Examples of this work include an approach based on the *dependence graph* (DG) and the *dataflow graph* (DFG) described by Kung *et al* in [Kung87] and the *HIFI* methodology of Annevelink [Anne88].

Systematic design verification and simulation methods are also important in the development of large systems. During the design process, accurate simulation of the electrical characteristics of a circuit is achieved with circuit simulators; a very widely used example is the SPICE simulator. Simulators such as SPICE place large demands on computer memory and processing power, however, so their use is limited to smaller circuits or parts of larger ones. Timing simulation [Malo89c][Malo89d] is a promising option for the simulation of large VLSI systems. Typically, an approximation in the circuit solution is employed which allows significant reductions in processing time and memory requirements at the cost of a modest loss in accuracy. Verification of the layout for modules and for the final chip is performed by design rule checker programs which are specific to the fabrication process to be used. Specially designed simulators can also be useful in validating the correct operation of chip-level [Côté89] and board-level [Coll89] architectures.

Complex VLSI circuits already pose challenges for designers with regard to yield. As the complexity of circuits increases, the silicon area covered by a single design also increases and the probability of obtaining a complete circuit without a fault in it decreases. This yield problem becomes critical with the advent of *WSI*, or *wafer scale integration*, where a single design can contain up to half a million[†] transistors and occupies an entire silicon wafer. Research efforts aimed at solving this problem stress fault-tolerant architectures with the ability to reconfigure around defects [Cox87]. Built-in self test features and design for testability [Rajs87] are also central in making wafer-scale designs feasible

3.2 Previous Work on VLSI Implementations in Machine Vision

The use of special-purpose hardware to perform specific, complex, repetitive computations can significantly increase efficiency. As outlined in the previous section, the current revolution in VLSI techniques has allowed faster, denser, and more complex circuits to become feasible. Current research into fault-tolerant architectures, built-in-self-test capabilities, and massively-parallel computing holds great promise for the future. Designs to use this new technology are becoming popular in many applications areas such as robotics [Chan88] [Ling88].

Vision processing typically involves large quantities of data. Processing in real time at video frame rates is often desirable. These factors make vision computing a prime candidate for VLSI circuit implementations. Several examples have appeared recently in the literature. Some of these are cited here as a motivation for the development and inclusion of VLSI elements in special-purpose hardware for curvature extraction.

Sawh *et al* [Sawh86] report two custom CMOS chip designs for edge extraction. The first is a numerical differentiator which operates on a byte-wide stream of data. The second is an edge-extraction circuit which performs logical, rather than arithmetic, computations on picture elements. The first design has been reported to work at clock rates in excess of 3 MHz.

Blanchet and Poussart [Blan87] have designed arithmetic cells for a high-speed 3D orientation processor. The cells estimate local surface orientation based on the fitting

[†] According to [Fort87], p.13.

of a plane to a square neighborhood of data points.

The Hough transform has been the subject of several recent VLSI implementations. Rhodes *et al* propose a monolithic Hough transform processor based on restructurable VLSI [Rhod88]. This processor groups pixels in order to extract linear features. Hanahara, Maruyama, and Uchiyama have developed a real-time Hough transform processor [Hana88]. This implementation also concentrates on straight lines. Examples where processing took less than one second to generate the output parameters are discussed in the paper [Hana88] An improved systolic implementation of the Hough transform is described by Li, Pao, and Jayakumar in [Li88a]. A modification which performs a contiguity check on line pixels is featured in their design to address problems associated with line length.

A VLSI architecture for extracting shape features of handwritten characters is discussed by Li, Youssef, and Jayakumar in [Li88b]. Detection of endpoints of character boundaries is followed by formation of edge chains for character characterization. The algorithms have been simulated and a high accuracy of recognition has been found even for partly-rotated characters.

Sanz describes two real-time architectures for image processing and computer vision conceived at the IBM Almaden Research Center [Sanz88]. The first architecture is a Radon and projection transform-based processor. It permits the Hough transform of non-linear features to be computed, in contrast to the above linear Hough-transform-specific approaches. The second architecture is designed to segment images based on polynomial classification. The approach is described as model-based. A VLSI implementation of the first architecture is under development; the second architecture is also reported to be suitable for a VLSI realization.

An approach for implementing the connected components algorithm is presented by Yang in [Yang88a]. A prototype board has been built as a first design and the extension for a VLSI chip is described. The chip is expected to perform the connected-components computation at video frame rates.

Abdelguerfi *et al* have developed a VLSI processing element (PE) to perform the histogramming operation [Abde88]. The PEs operate bit-serially and are interconnected into an odd-even network topology to realize the histogramming function.

Most of the VLSI implementations in machine vision are designed for algorithms with inherent regularity. this is not surprising since many algorithms in vision processing are highly regular and the regularity makes them simpler to implement as special-purpose designs. A VLSI array capable of realizing arbitrary algorithms with no internal regularity is presented by Koren *et al* in [Kore88] This array structure implements the dataflow computing principle in a more general way than do wavefront arrays, which are also data-driven but feature fixed computational wavefronts.

The photosensitive qualities of CMOS are exploited to advantage in two recent analog designs, one by Allen *et al* [Alle88] and one by Dubois and Poussart [Dubo89]. These designs are unusual in that CMOS VLSI circuits in vision applications tend to be digital and the photosensitivity of the devices is typically minimized by shielding the circuits from light. The devices in [Alle88] and [Dubo89] utilize the photosensitive property and can be described as intelligent sensors. The design of Allen *et al* is called an orientation-selective VLSI retina; this device features hexagonal pixel lattices which, through a combination of on-site and array-bordering computational circuitry, produce an edge-enhanced image and a map of edge orientation. The design of Dubois and Poussart does real-time image contour extraction.

Another intelligent sensor implementation is presented by Ginosar and Zeevi [Gino88]. Their imaging chips feature adaptive sensitivity to light intensity and more flexible scanning patterns in place of the traditional rectangular grid.

Vainio *et al* present a variable-window-size edge-preserving filter suitable for real-time video signal processing [Vain88]. This design is a fast, pipelined CMOS implementation of the FIR Median Hybrid filter. The circuit is said to operate at clock rates of up to 80 MHz, fast enough for real-time processing of High Definition Television (HDTV) signals. The window size of the filter can be adjusted to contain between 3 and 257 samples.

In the real-time video communications domain, motion compensated video [Fort86] and motion estimation for teleconferencing [Pirs88][Yang88b][Dian88] have been the focus of some special-purpose VLSI designs. In particular, the block-matching algorithm (BMA) for motion estimation is the focus of the last three papers listed above. The BMA algorithm has much in common with the determination of the best window position (BWP) which was described in Chapter 2 of this thesis Both algorithms feature a search of error terms to find a minimum. In the BMA, the errors are differences between the pixel intensities in a current versus a previous video image. The displacement vector for a certain block in the image is determined as the offset from the block in the previous frame to the one in the current frame where the error is a minimum. This offset is like the (u_*, v_*) offset to the best window position

Chapter 2 of this thesis studied the performance of the curvature extraction algorithm using two different criteria to compute the error map for the BWP search. The two error criteria featured were the square (original) and absolute (modified) error criteria. The square criterion can be termed a mean-squared-error (MSE) measure and the absolute criterion a mean-absolute-difference (MAD) measure Similarly, in the BMA algorithm different criteria can be used to generate the error map. In [Pers88], Pirsch and Komarek cite the cross-correlation function, the mean squared error, the mean of the absolute differences (MAD), and an infinite norm as possible candidates. They favor the mean of the absolute differences, or MAD, which is the same criterion used in our modified algorithm. Both this thesis work and the work of Pirsch and Komarek attempt to formulate their algorithms (BWP and BMA respectively) to favor a VLSI implementation and the choice of the MAD simplifies the VLSI realization for computing the error maps.

In [Boud86], Boudreault and Malowany present a design for a systolic, serial convolution processor cell which operates on integer data. Signed (2's complement) coefficients and unsigned (0 to 255) data, each 8 bits in width, are supported. All other signals, including the output, are 16-bit signed (two's complement) integers. The cells may be chained together to implement various sizes of convolutions. This integer convolution cell was the first of several related special-purpose hardware designs for image processing Successor designs include a convolution board [Côté88], an architectural simulator for the convolution board [Coll89], a floating-point convolution cell [Laro89], and a simulator to validate the floating-point convolution cell architecture [Côté89]. The fit-error cell design of this thesis is a consequence of the above line of research. These predecessor designs, or "sister" designs, influenced the development of the hardware presented in this chapter. We return to describe these designs later in the chapter as they relate to the current design.

Having looked at some examples from the literature showing VLSI implementations in vision processing, we proceed now to describe the hardware for curvature extraction which includes a special-purpose VLSI design for fit-error computations.

3.3 A Dedicated Hardware Environment for Curvature Extraction

3.3.1 Overview of Hardware Architecture

A dedicated hardware environment is proposed for the efficient extraction of the curvatures using the modified algorithm of Chapter 2. This environment has the architecture shown in Figure 3.1 and consists of four major components. These components include three computational subsystems plus a supporting memory subsystem. Each of the three algorithm steps (fit, errors, and curvatures) shown in Figure 2.5 of Chapter 2 is performed by one of the three computational subsystems. The common bus is the Intel Multibus II which features a 32-bit data path.

The first two computational subsystems in the architecture, the convolution processor and the fit-error computation processor, are special-purpose processors based on systolic VLSI cells. These two processors perform the first two steps in the algorithm,

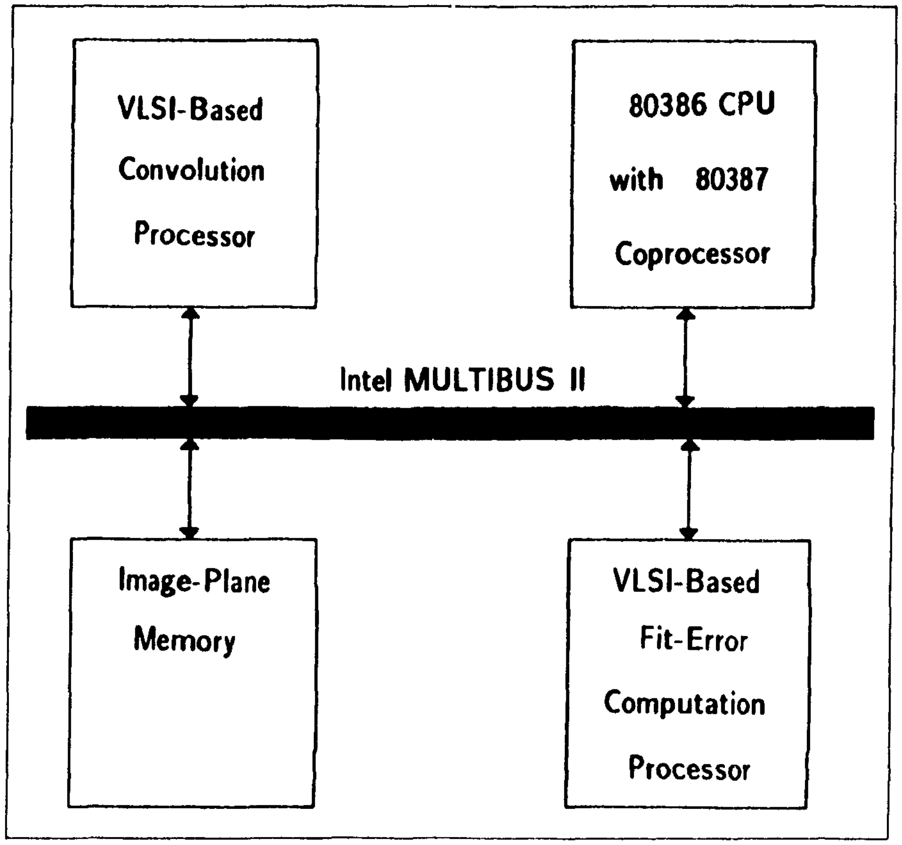

VLSI-Based Convolution Processor
80386 CPU with 80387 Coprocessor
Intel MULTIBUS II
Image-Plane Memory
VLSI-Based Fit-Error Computation Processor

Figure 3.1 Architecture of the Dedicated Environment for Curvature Extraction

the fit convolution and the fit-error computation respectively The third computational subsystem is a commercial product, the Intel iSBC 386/120 Single Card Computer[†], which features a 32-bit CPU (the Intel 80386) and a floating-point math co-processor unit (the Intel 80387). This single card computer serves as the host computer in the architecture. In addition to providing the control signals for the two special-purpose subsystems, this host performs the third step in the algorithm which finds the best window position and computes the curvatures.

All computations by the hardware on the range data input and on subsequent extracted quantities are to be performed in double-precision floating-point format. Double precision was also used in the software versions of the algorithm executed on the Suns. This choice was motivated by the earlier work of Yokoya and Levine [Yoko87] which indicated

[†] As in [Inte88] p 4-19 and 4-20:

that double precision is required in the processing. Use of floating-point representation also simplifies the problem of handling large dynamic ranges in the data; this is of particular importance for the Gaussian curvature as was stressed in the analysis of the experiments of Chapter 2

The architecture is based on a 32-bit word. The hardware respects the IEEE Standard 754 for Binary Floating-Point Arithmetic. Figure 3.2 shows the bit assignments for single (32-bit) and double (64-bit) precision floating-point numbers in the IEEE standard format[‡]. The mantissa has a positive binary representation, where an implicit "1" and an implicit binary point are interpreted as preceding the stored mantissa bits so that these stored bits represent the fractional part The exponent bits represent the power of "2" multiplying the mantissa. The exponent is formed from the appropriate signed, two's complement integer with a bias au [illegible] Addition of the bias facilitates comparison of numbers. For single precision, the bias is +127 and for double precision it is +1023.

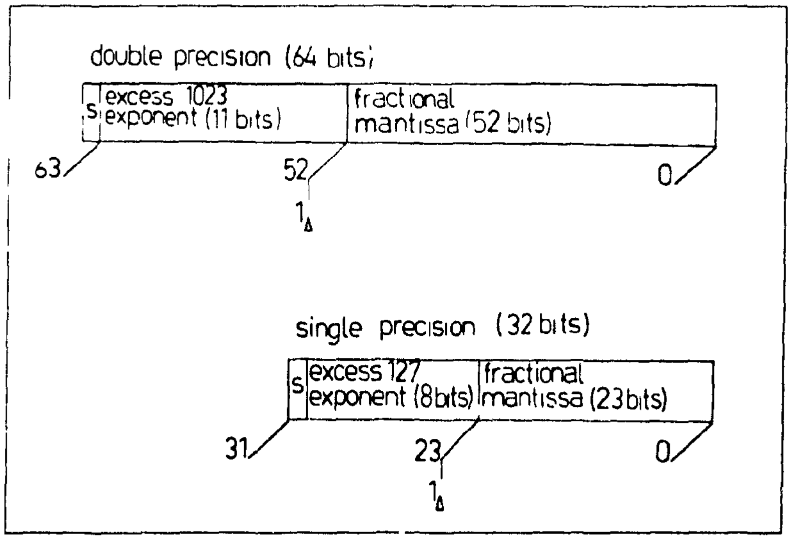

double precision (64 bits)
S
excess 1023 exponent (11 bits)
fractional mantissa (52 bits)
63
52
0
1
single precision (32 bits)
S
excess 127 exponent (8bits)
fractional mantissa (23bits)
31
23
0
1

Figure 3.2 Bit Assignments for Single and Double Precision in the IEEE Standard for Floating-Point Representation

Memory requirements for the three-step curvature extraction algorithm running in the new hardware environment on a 256 X 256 image are of the order of 6 megabytes.

‡ As in [Inte87], p. 2-12

Just over half a megabyte (525 Kbytes) is required to store a map of 256 X 256 double-precision numbers. Ten such maps are required for simultaneous storage of partial results during the processing envisaged A small amount of additional storage is required for smaller entities such as the convolution window operators. This memory requirement is within the memory capacity of the environment.

In the following three sections, each computational subsystem will be treated separately. Emphasis is placed upon the expected performance of each subsystem.

3.3.2 Convolution Processor

A floating-point systolic cell for convolution has been proposed by Larochelle *et al* [Laro89]. The cell uses the double-precision floating-point format of the IEEE standard for input, internal computations, and output This cell is being implemented as a VLSI circuit and a board design to support it is envisaged.

The floating-point convolution board design will be similar in principle to that of Côté *et al* [Côté88] which supports integer arithmetic and is based on the Boudreault convolution cell described earlier [Boud86] The integer board design implements a 9 X 9 convolution in four passes on an array-slice of 27 cells rather than the full 81 cells. Three rows of nine cells each are featured The board is configured as a DMA machine with four DMA channels supporting data input to each of the three rows of cells plus storage of the output. Use of DMA channels instead of long shift registers between rows allows a programmable image row length and also reduces the pipeline-fill time. An 8-input PLA is used for generation of control signals for the convolution cells. The board respects the Intel MULTIBUS I bus conventions. Logic on the board allows either 8 bits or the full 16 bits of convolved output to be stored. It is estimated that this integer convolution board can convolve a 512 X 512 image in less than a second The floating-point version of the convolution board will be based on the Larochelle convolution cell using double-precision floating-point arithmetic and will operate in the Intel MULTIBUS II environment.

This floating-point board will constitute the first subsystem in the architecture of the new hardware environment· the convolution processor.

The floating-point systolic convolution cell itself features a pipelined architecture with four stages. These are 1) coefficient times data multiplication, 2) alignment and addition of the product with the partial-sum input, 3) normalization of the new partial sum result, and 4) shifting of the partial-sum pipeline. The partial-sum and pixel-input data are shifted into and out of the cell four bits at a time, resulting in a requirement of 16 clock cycles to load a 64-bit operand The four stages of the computation pipeline within each cell are well matched so that each completes its job in 16 clock cycles. Hence processing takes place at the same rate as data is loaded. The convolution coefficients are loaded serially in an initial coefficient loading phase After initial coefficient loading and pipeline-fill periods, a valid result emerges from an array of such cells every 16 clock-cycles. An n X n window-size convolution can be realized with an n X n array of cells in one pass or with a smaller array slice in multiple passes as was the case for the Côté integer convolution board

The pipeline-fill period, or latency, for the convolution processor will be dependent upon the size of the cell array and hence upon the size of the convolution window implemented For example, a 7 X 7 window would require a longer fill period than a 5 X 5 window. A coefficient loading period also must precede the convolution in the case where successive convolutions use different window operators, this is the case for curvature extraction. However, these times are only a few percent of the total convolution time. Neglecting these pipeline-fill and coefficient-loading periods, we can obtain an order-of-magnitude estimate for the convolution time independent of the window size. Preliminary estimates indicate a clock rate of 16 MHz (i.e. a clock period of 62.5 nanoseconds) will be feasible for the new convolution cells which are to be implemented in 3-micron, double-metal CMOS There are sixteen clock cycles between outputs of successive results from the array. Hence, the time required for the convolution of a 256 X 256 image, assuming a single-pass implementation, is:

$$(62.5X10^{-9}sec)(16)(256X256) = 0.066sec \tag{3.1}$$

Addition of the pipeline-fill and coefficient loading times should still result in under 0.1 second for such a convolution. Hence the six convolutions required for the curvature extraction should be possible in less than a second (0.6 second).

3.3.3 The New Fit-Error Computation Processor

A new floating-point systolic VLSI cell architecture for computing the fit-error map is proposed in this thesis; a short paper [Malo89a] and a more detailed treatment [Malo89b] summarize the development and results for the fit-error cell presented here. This new floating-point fit-error cell follows the principle of the Larochelle cell [Laro89] for floating-point convolution described in the previous section. A new DMA board design featuring an array of the systolic fit-error cells will form the second subsystem in the dedicated hardware environment: the fit-error computation processor. The new fit-error cell, like the Larochelle convolution cell, works with double-precision floating-point numbers according to the IEEE standard.

3.3.3.1 Fit-Error Cell Architecture Overview

A block diagram of the new VLSI fit-error cell is shown in Figure 3.3. The cell features four main functional blocks. These are 1) a storage and access module for pre-computed UV products (block $B0$ in the figure), 2) a floating-point multiplier module (block $B1$ in the figure), 3) a floating-point align-and-add module (block $B2$ in the figure), and 4) a floating-point normalization module (block $B3$ in the figure). For the interested reader, details on the structure and operation of these modules are supplied in Appendix B. Sixteen full-length registers for 64-bit operands are featured, six of which are contained in the UV storage and access module, and additional special-purpose registers of varying lengths are also present within the other three modules (the multiplier module, the align-and-add module, and normalization module).

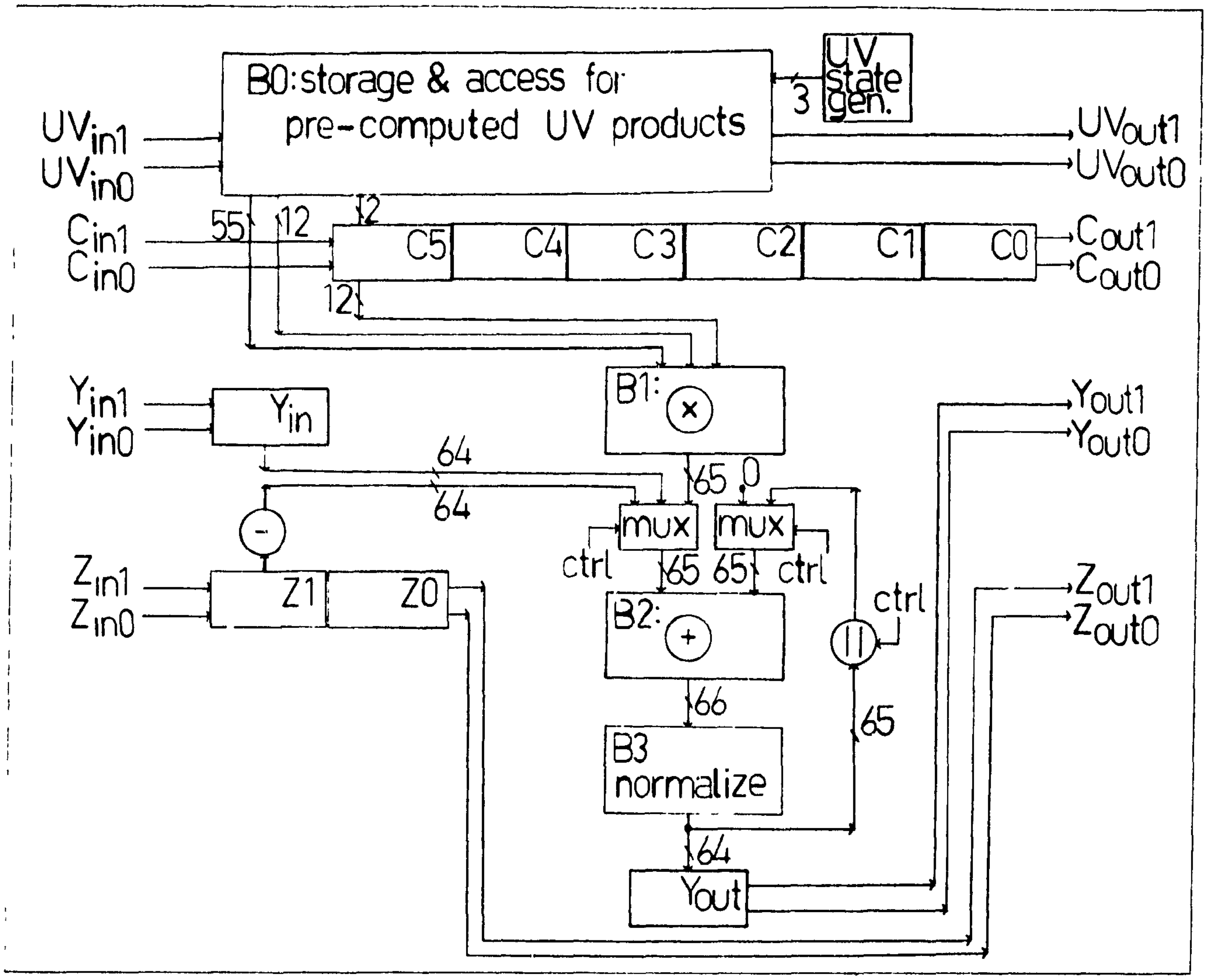


Figure 3.3 Block Diagram of the New VLSI Fit-Error Cell

Four data inputs and four data outputs, each two bits wide, are shown in the block diagram of Figure 3.3, requiring 16 data pins on the chip. The signals (UV, C, Y, and Z) and the operation of the fit-error cell will be discussed in the following section Additional pins are required for power, ground, and control inputs. The 64-bit operands are shifted in and out of the cell 2 bits at a time[1], bit serially, on the eight pairs of data pins. Hence, it takes 32 clock pulses to load an operand. Unlike the Larochelle cell design, the fit-error cell is not a true pipeline There is feedback between the modules and, most

[1] The tradeoffs considered regarding operand-handling in two-bit slices are described in Appendix B, section B 6, for the interested reader

of the time, all modules are computing parts of the same single output. (In the Larochelle cell, the four stages work in parallel on four successive distinct outputs.) The fit-error cell requires 256 (= 8 X 32) clock pulses to compute each output To see why this is so, let us consider the computation that the cell array must perform

3.3.3.2 Computation Overview for Fit-Error Systolic Array

The fit-error cells can be arranged in arrays of varying sizes to realize fitting windows of various sizes For the sake of discussion, a square fitting window size of n X n will be assumed here.

Four sets of data are handled by each cell The first is a set of six constant parameters, $UV = (u^2, v^2, uv, u, v, 1)$, associating each cell with its position in the array, these are pre-loaded before processing begins. The second data set is the fit coefficients, $C = (a, b, c, d, e, f)$, which are pre-computed from the range image by convolution There are six fit coefficients for each point in the range image The constant parameters UV multiply the fit coefficient inputs C and accumulate within each cell to produce the fitted depth according to the local quadric fit:

$$\hat{z}(u, v, x, y) = a(x, y)u^2 + b(x, y)v^2 + c(x, y)uv + d(x, y)u + e(x, y)v + f(x, y) \quad (3.2)$$

The third input data stream is the actual range-image values, $Z = z(x, y)$ These are subtracted from the fitted depths computed by the cells and the magnitude is taken to produce the fit-error contribution for the particular location within the array:

$$\epsilon(u, v, x, y) = |\hat{z}(u, v, x, y) - z(x + u, y + v)| \quad (3.3)$$

This error contribution computed by the particular cell is added to the currently-held element of the fourth input data stream: the partial sum input Y from the neighbor cell The result is then passed along as the partial sum output to the next cell

$$Y_{out} = Y_{in} + \epsilon \tag{3.4}$$

Hence, the total aggregated fit error, $E(x,y)$ emerges as the partial sum output of the lower-right corner cell n^2 compute-intervals[†] after the incidence of the coefficients C for (x,y) at the top-left corner of the array. The fit coefficients, C, and the partial sum, Y, travel together through the array from top-left to bottom-right, while the Z stream travels differently and the UV parameters are stationary within the cells. While there is a time skew between the incidence of coefficients for (x,y) and the output of the result at (x,y), the data flow is arranged so that all successive outputs from the lower-right corner cell (one per compute-interval) are complete error values for successive locations (x,y) after an initial pipeline-fill period. The resulting computation realized by the array of cells is:

$$E(x,y) = \sum_{u=-m}^{m} \sum_{v=-m}^{m} |\hat{z}(u,v,x,y) - z(x+u, y+v)| \tag{3.5}$$

where $m = (n-1)/2$ is the half-width of the fitting window and $\hat{z}(u,v,x,y)$ is the fitted depth as in equation (3.2) above. Note that in earlier definitions of the fit error, the quantity was scaled by dividing by the number of points in the window, n^2. This scaling is not really necessary for the purposes of curvature extraction since only the locations of local error minima are actually used. Since double-precision floating-point format is maintained within the fit-error cells, the problem of overflow should not arise from the omission of the division by n^2. The advantage of omitting this division, however, is a simpler cell architecture.

Next, the mapping of the above computation onto the hardware within each error cell is considered.

[†] One such interval requires 8 X 32 = 256 clock pulses, as mentioned earlier.

3.3.3.3 Computation Sequence within a Fit-Error Cell

The table of Figure 3.4 shows the computations occurring inside three adjacent cells over one compute-interval of 256 clock pulses which is divided into 8 sub-intervals of 32 clock pulses each. The first three sub-intervals of the following compute-interval are also included to show the data movement between cells.

The multiplier module (represented by the "mult" rows in Figure 3.4) requires 32 clock pulses to compute a product; the product is computed as a fit coefficient C is being shifted in. Six of the eight sub-intervals feature a product computation, the multiplier module being idle for the remaining two sub-intervals. Hence, six operands from the fit-coefficient data stream C are handled in each compute-interval. This necessitates a C input stream in which the six fit coefficients (a, b, c, d, e, f) are interleaved. The interleaving of the fit coefficients can be achieved with appropriate control circuitry on the output DMA channel of the convolution processor.

The align-and-add module and the normalization module each require 16 clock pulses to fulfill their functions but they must operate sequentially as a single unit (denoted by the "a/n" rows in Figure 3.4) due to the requirement for feedback of results from the normalization unit's output to the align-and-add unit's input. This feedback path enables the same two units (align-and-add, normalization) to be used for the fitted-depth accumulation as well as for the subtraction of the depth value and the addition of the difference magnitude to the partial-error-sum input from the neighbor cell. Using the same two units for all three operations avoids inclusion of multiple pairs of units and thus significantly saves on silicon area. So speed has been traded off to save area. The result is that, together, the align-and-add and the normalization modules require 32 clock pulses to complete their joint task.

This pair of modules forms the bottleneck in the cell; the align-and-add unit is active in the first half of each sub-interval and the normalization unit is active in the second half of each sub-interval. The pair serves in different modes as shown by the multiplexers

Sub-Interval	Cell(k-1)	Cell(k)	Cell(k+1)
1 X 32 mult: a/n: Y_{out}, Z_{out}:	 $a(i+1) * u^2$ $Y_{in}(i) + \vert fbk \vert$ idle	 $a(i) * u^2$ $Y_{in}(i-1) + \vert fbk \vert$ idle	 $a(i-1) * u^2$ $Y_{in}(i-2) + \vert fbk \vert$ idle
2 X 32 mult: a/n: Y_{out}, Z_{out}:	 $b(i+1) * v^2$ $prod + 0$ latch $Y_{out}(i)$	 $b(i) * v^2$ $prod + 0$ latch $Y_{out}(i-1)$	 $b(i-1) * v^2$ $prod + 0$ latch $Y_{out}(i-2)$
3 X 32 mult: a/n: Y_{out}, Z_{out}:	 $c(i+1) * uv$ $prod + fbk$ shift Y and Z	 $c(i) * uv$ $prod + fbk$ shift Y and Z	 $c(i-1) * uv$ $prod + fbk$ shift Y and Z
4 X 32 mult: a/n: Y_{out}, Z_{out}:	 $d(i+1) * u$ $prod + fbk$ idle	 $d(i) * u$ $prod + fbk$ idle	 $d(i-1) * u$ $prod + fbk$ idle
5 X 32 mult: a/n: Y_{out}, Z_{out}:	 $e(i+1) * v$ $prod + fbk$ idle	 $e(i) * v$ $prod + fbk$ idle	 $e(i-1) * v$ $prod + fbk$ idle
6 X 32 mult: a/n: Y_{out}, Z_{out}:	 $f(i+1) * 1$ $prod + fbk$ idle	 $f(i) * 1$ $prod + fbk$ idle	 $f(i-1) * 1$ $prod + fbk$ idle
7 X 32 mult: a/n: Y_{out}, Z_{out}	 idle $prod + fbk$ idle	 idle $prod + fbk$ idle	 idle $prod + fbk$ idle
8 X 32 mult. a/n: Y_{out}, Z_{out}	 idle $-Z(j+2) + fbk$ idle	 idle $-Z(j) + fbk$ idle	 idle $-Z(j-2) + fbk$ idle
1' X 32 mult a/n: Y_{out}, Z_{out}:	 $a(i+2) * u^2$ $Y_{in}(i+1) + \vert fbk \vert$ idle	 $a(i+1) * u^2$ $Y_{in}(i) + \vert fbk \vert$ idle	 $a(i) * u^2$ $Y_{in}(i-1) + \vert fbk \vert$ idle
2' X 32 mult: a/n: Y_{out}, Z_{out}:	 $b(i+2) * v^2$ $prod + 0$ latch $Y_{out}(i+1)$	 $b(i+1) * v^2$ $prod + 0$ latch $Y_{out}(i)$	 $b(i) * v^2$ $prod + 0$ latch $Y_{out}(i-1)$
3' X 32 mult: a/n: Y_{out}, Z_{out}:	 $c(i+2) * uv$ $prod + fbk$ shift Y and Z	 $c(i+1) * uv$ $prod + fbk$ shift Y and Z	 $c(i) * uv$ $prod + fbk$ shift Y and Z

Figure 3.4 Sequence of Operations Inside Three Adjacent Fit-Error Cells Over One Compute-Interval

of the earlier Figure 3.3. The first operand can be either $-Z$, Y_{in}, or the product from the multiplier and the second operand can be either zero, the true feedback (denoted fbk in Figure 3.4), or the absolute value of the feedback (denoted $|fbk|$ in Figure 3.4). At this point, the simplification resulting from the use of the modified algorithm of Chapter 2 can truly be appreciated. The absolute value operation (represented by the circle with the symbol "||" in Figure 3.3) can be easily implemented with a single logic gate on the sign bit of the floating-point number, but the square operation of the original algorithm would require circuit complexity equivalent to that for a general multiply. The multiplier unit currently included in the cell could not be used for the squaring unmodified, since this unit multiplies an input by a pre-computed UV product and so can handle only one variable input. Realization of the negative of Z (represented by the circle with the symbol "-" in Figure 3.3) is similarly simple logic on the sign bit.

Latching of a new result from the output of the normalization unit into the Y output register (denoted in the "Y_{out}, Z_{out}" rows in Figure 3.4) occurs at the start of only one sub-interval (sub-interval 2 in Figure 3.4). Similarly, shifting of the Y and Z data streams occurs only during one sub-interval (sub-interval 3 in Figure 3.4). An extra Z register is introduced into the Z pipeline in each cell to delay the flow of Z's so that the required skew is realized with respect to the C and Y data streams. For example, in sub-interval 8 of Figure 3.4, the three adjacent cells are shown processing $Z(j+2)$, $Z(j)$, and $Z(j-2)$; sub-interval 8' of the next compute-interval, if it had been shown in Figure 3.4, would have shown the cells processing $Z(j+3)$, $Z(j+1)$, and $Z(j-1)$ respectively.

From the above description, it is apparent that only one operand from each of the Y and Z data streams is handled during each compute-interval while six operands are handled from the fit-coefficient data stream C. This is as required since there are six fit coefficients (a, b, c, d, e, f) for each point in the range image $z(x, y)$ and only one error result Y (which becomes the complete $E(x, y)$ after the last cell) for that point $z(x, y)$. However, the n^2 range-image values in the square neighborhood about $z(x, y)$ contribute to the aggregate fit-error measure $E(x, y)$. Hence, while the same six operands C follow the partial result Y through the array, different Z's contribute in each cell and different UV parameters are used in each cell as well. Note that the capital Z above refers to the depth-value data stream in the fit-error processor cell while the small $z(x, y)$ refers to a generic point in the range image.

3.3.3.4 Feeding and Interconnection of the Systolic Array

To appreciate how the data will be processed by the systolic array, it is helpful to be able to visualize and orient oneself. Figure 3.5 is intended to help in this visualization. The figure shows two views of a square neighborhood for fit-error computation; this square neighborhood is to be mapped onto the systolic array of fit-error processor cells.

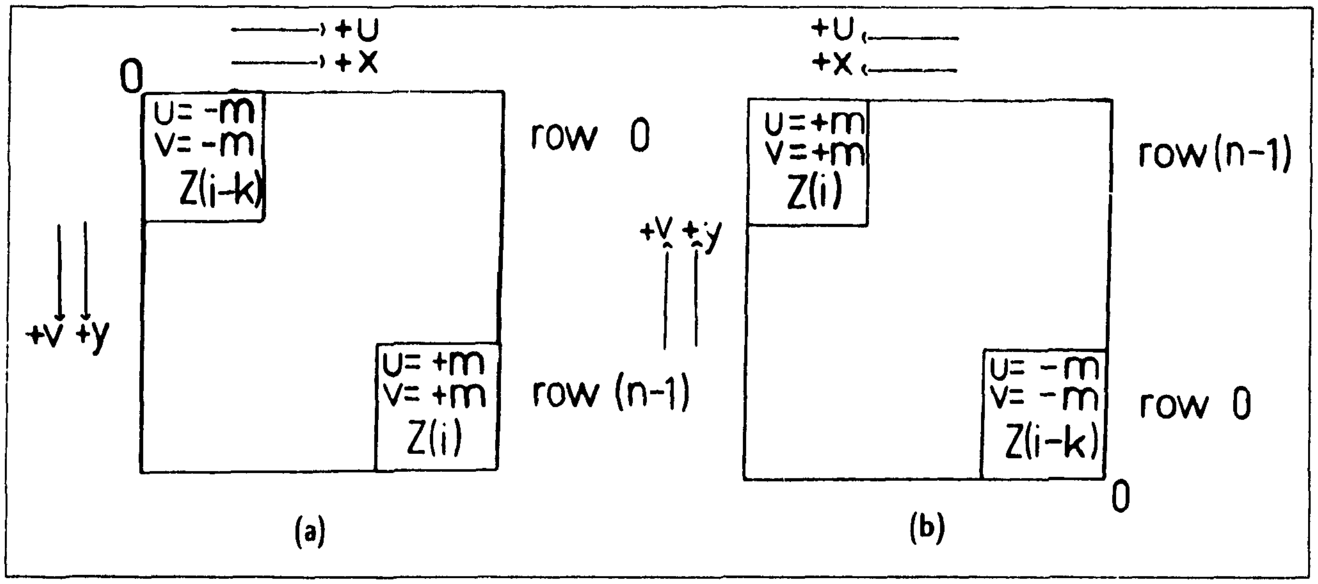


Figure 3.5 The Neighborhood for Fit-Error Computation: (a) Conventional Viewpoint (b) Systolic-Array Viewpoint

Part (a) of Figure 3.5 shows the square neighborhood the way it is usually pictured: the upper-left corner corresponds to the most negative extreme of the indices defining the neighborhood ($u = -m, v = -m$) and the indices increase to the right and down along the page. Accompanying this convention of part (a), we usually think of the image data as a large stationary array over which the neighborhood operator for computing the fit error is passed to the right along each successive row and down within the stationary image. When regarding the range image as a linear data stream, we start numbering from element 0 in the upper-left corner and increase along the first row, then the second, and so on until we reach the lower-right corner, which is element number $(N^2 - 1)$ assuming an N X N square range image. Hence, when the neighborhood operator is passed over the range image, the data point under the upper-left corner of the neighborhood operator has a smaller linear index value ($Z(i-k)$ in part (a) of Figure 3.5) compared to that under the lower-right corner ($Z(i)$).

Part (b) of Figure 3.5 shows the way to picture the fit neighborhood and image data when considering the systolic array of cells for fit-error computation. Part (b) is simply part (a) rotated 180 degrees about the secondary diagonal (i.e. that diagonal from the lower-left to the upper-right corner of the neighborhood). This system is more convenient for consideration of the hardware. Instead of having a stationary image and a moving neighborhood, in the hardware case we have a moving image which flows through a stationary neighborhood. The stationary neighborhood is realized by the systolic array of fit-error computation cells. Conceptually, we have the image data entering the systolic array at the upper-left corner and traversing the array in a pipeline which flows to the right and down along the page. The data point in the upper-left corner cell of the array now has a larger index value, $Z(i)$ in part (b) of Figure 3.5, compared to that in the lower-right corner of the array, $Z(i-k)$. This is because the newer data points just entering the array at the upper-left corner have higher index values while the older data points about to leave the array at the lower-right corner have lower index values. To make this scheme realize the same computation as in part (a), we need to make the upper-left most cell contain the most-positive extreme of the neighborhood-indexing parameters, $(u = m, v = m)$, and have the (u, v) indices decrease towards the right and down along the array. This determines the ordering of the UV parameters that are loaded into the systolic array during the initial coefficient-loading period.

Having seen how the UV parameters are set up in the systolic array relative to the range data Z that flows through the array, let us consider now how the fit-coefficients

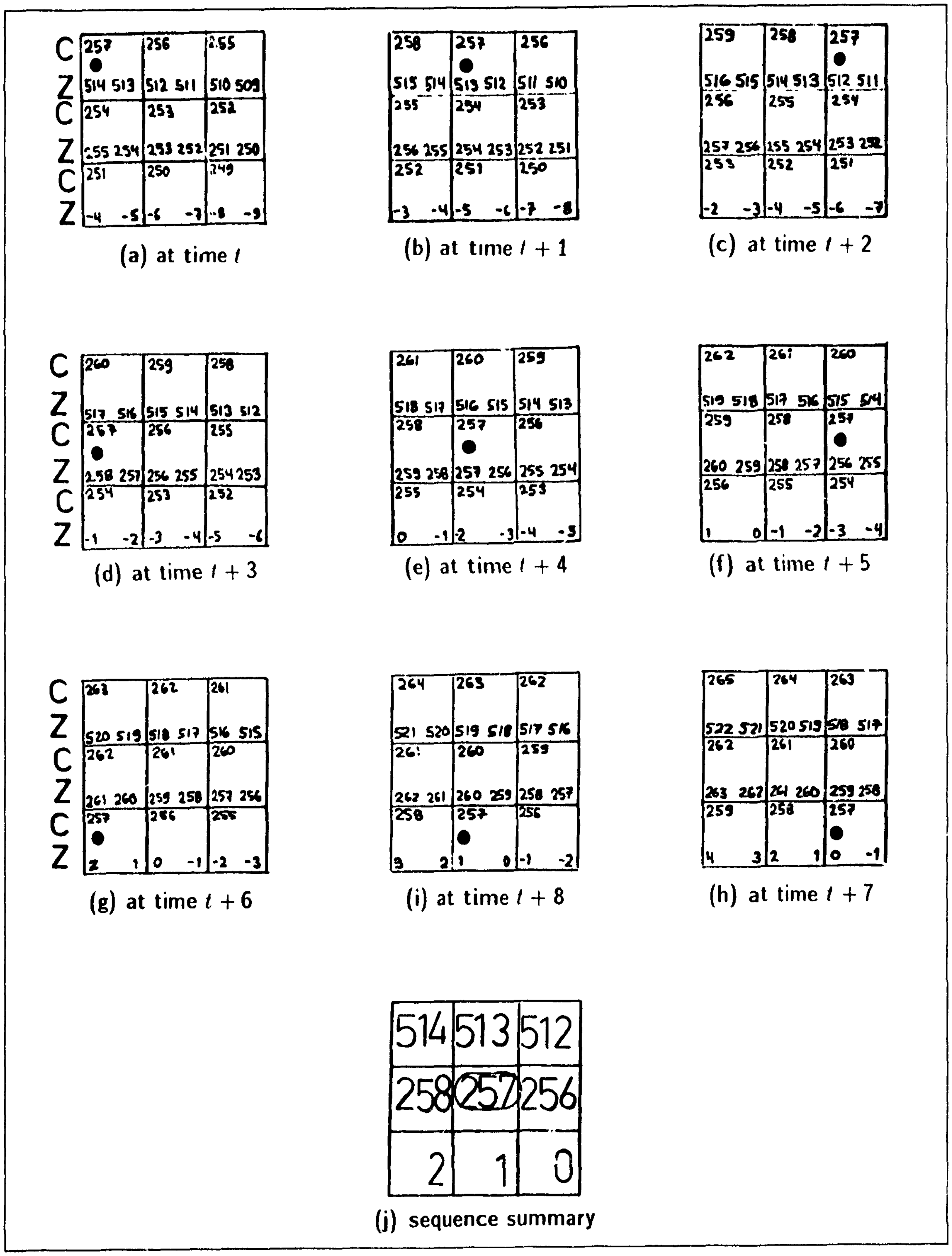


Figure 3.6 An Example Illustrating the Systolic Fit-Error Computation Highlighting the Pipeline Flow

$C = (a, b, c, d, e, f)$ flow through the array and how the result is composed by the array. This is perhaps best illustrated using an example. Figure 3.6 shows successive steps (a) through (i) to realize the fit-error computation summarized in (j). For simplicity, a 3 X 3 fitting neighborhood was chosen for this example. The range image size is assumed to be 256 X 256. In order to generate the fit error $E(x,y)$ of equation (3.5), nine error contributions $\epsilon(u, v, x, y)$ as given in equation (3.3) must be summed in nine successive steps.

The fit coefficients C and the partial result Y (the latter of which is not shown in Figure 3.6) travel together through the systolic array in a "wavefront". At any one time, there are 9 such wavefronts in the array at various stages of completion. The black dot in Figure 3.6 shows the progress of the $E(257)$ wavefront through the array. Each step (such as going from (a) to (b)) requires one compute-interval of 256 clock pulses. Each of the smallest squares in Figure 3.6 represents a fit-error cell; the three numbers shown inside each cell represent the index of the fit coefficients (top number), and the indices of the two depth values Z (bottom two numbers) within that cell over that compute-interval. This corresponds to the C and Z labels that appear at the left of each row of cells. Hence, the indices within the cells show the shifting of the C and Z pipelines.

The sequence (a) through (i) of Figure 3.6 is "summarized" in part (j) showing the nine Z values that are sequentially used in computing $E(257)$. The correspondence between each Z and its UV parameter set is determined by its relative position in this "summary" window. For example, $Z(258)$ corresponds to the UV parameter set for $u = 1$, $v = 0$ according to the summary window in (j). In the detailed sequence, part (d) correspondingly illustrates the left-most cell of the middle row (with the black dot) "computing" the fourth error contribution according to the equation below:

$$\epsilon_4 = \left| \left[a(257)u^2 + b(257)v^2 + c(257)uv + d(257)u + e(257)v + f(257)\right] - Z(258)\right| \quad (3.6)$$

with $u = 1$ and $v = 0$. To evaluate the nine error contributions for $E(257)$, distinctive

(u, v) combinations are used corresponding to the locations of the black dot in Figure 3.6. A different Z is involved at each step. However, the same six fit coefficients, $C(257) = (a(257), b(257), \ldots f(257))$, participate in each of the nine steps (a) - (i) in the computation

Let us now follow the $E(257)$ computation wavefront. Although not shown in Figure 3.6, the partial result Y will be included in the discussion. The upper-left-most cell in the array is the "cell of interest" in part (a) since it has the black dot This cell has its partial result input, Y_{in}, tied to zero; all other cells take their Y_{in} from the Y_{out} of their left-hand neighbors. The left-hand neighbor of the left-most cell in a row is the right-most cell of the row above it. In the step shown in part (a), the first error contribution with the fit coefficients $C(257)$ and depth value $Z(514)$ is calculated by the "black dotted" cell and added to the Y_{in} (which is zero) to give the Y_{out} for that "first contribution" cell.

For the next step shown in part (b), notice that the C and Z pipelines have shifted right by one. Although not shown, the Y pipeline has also shifted right by one, and the Y_{out} computed by the upper-left-most cell in part (a) has been latched in as the Y_{in} of its right-hand neighbor for part (b). In part (b), the "black dotted" cell computes the second error contribution to the $E(257)$ wavefront, that with $C(257)$ and $Z(513)$, and then adds it to the partial result, Y_{in}, to generate Y_{out}. Meanwhile, the upper-left-most cell is computing the first error contribution in the next wavefront (i.e. the wavefront for $E(258)$). Hence, at every step, cell k in the array computes contribution k of a new wavefront.

The wavefront shown by the black dot continues through the array, accumulating a new error contribution into its associated partial result at each step, until it reaches part (i) in the figure. On the following step, the finished result for $E(257)$ is pushed out of the systolic array to be stored in the external memory. Note that $E(257)$ corresponds to (row 1, column 1) and is really the first valid point that can be computed in the fit-error map of this example. Hence, some of the Z pipeline is not properly filled during the $E(257)$ computation; this is indicated in Figure 3.6 using negative indices. Having seen how the fit-error output accumulates, let us now move on to consider some implementation details.

There are two options for implementing the Z pipeline. One is to feed each row of the systolic array separately. The other is to feed the whole array as one pipeline using long shift registers between the rows. Figure 3.7 shows the interconnection of a systolic array based on the first alternative to realize an n X n fitting neighborhood. Each cell within the array of Figure 3.7 shows its UV indexing parameters, which range from $(u = m, v = m)$ to $(u = -m, v = -m)$ where $m = (n - 1)/2$.

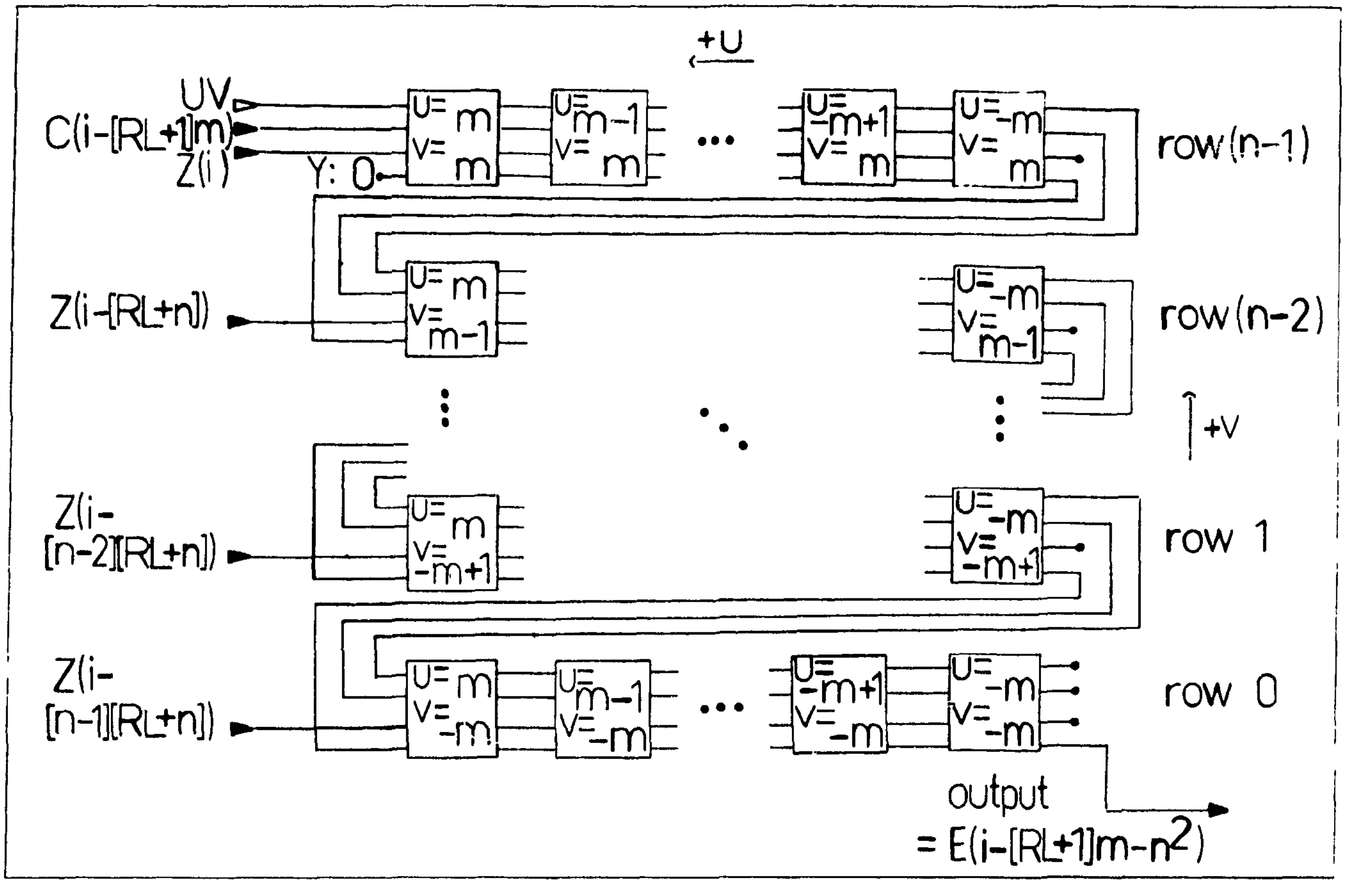


Figure 3.7 Systolic Array of Fit-Error Cells Showing Interconnections and End Conditions

To supply the data to the systolic array of Figure 3.7, a DMA machine is proposed with a separate DMA channel for each of the n rows of range data input Z, plus one input DMA channel to supply the fit coefficients C. One output DMA channel is also required for storing the output errors, E, which issue from the end of the partial-result data stream Y of the array. These DMA channels are represented by filled-in arrows in the diagram of Figure 3.7. The DMA capability is not required for the loading of the UV parameters, since these need only be loaded once as an initialization.

This DMA approach to a board design for the array resembles that of the convolution board of Côté et al [Côté88]. However, that implementation features 4 DMA channels and computes a 9 X 9 convolution in four passes on a 3 X 9 systolic array. The proposed approach for the fit-error systolic array shown in Figure 3.7 requires $(n + 2)$ DMA channels for a one-pass computation of the n X n neighborhood fit errors using an n X n cell array Similar reductions of the cell array size and number of DMA channels could be realized for the fit-error systolic array at the cost of requiring more than one pass of computation.

Unlike the UV, C, and Y data paths for the fit-error systolic array of Figure 3.7, the Z data path does not feature a direct connection of the last cell in each row to the first cell in the next row. Each row has its own input channel for Z If one long pipeline were to be implemented for the Z data, a long shift register equal in length to the image row length less $2n$ would have to be interposed between the last element of each row and the first element of the next row. Unless programmable-length shift registers were used, this method of a single Z pipeline would limit the flexibility of the design by requiring images to be of a certain fixed row length and would also require a longer pipeline-fill time

Since the computation of the fit error is a neighborhood operation, there are border effects to be considered. The left and right borders of the image are calculated by wrapping the window around due to the pipelined array structure; this effectively implies that the range image is treated as a cylinder where the end of one row is joined to the beginning of the next This treatment is necessary for efficient operation of the array However, the top and bottom of the array can be dealt with differently. The first and last m rows as well as the first m elements of the $(m + 1)$st row from the top and the last m elements of the $(m + 1)$st row from the bottom are points where not enough data exists to fill the array. The corresponding points in the output error map should be set by the host processor to a value of "infinity", i.e. the largest representable number. This is so that in the following step of the algorithm, when the error map is searched for local minima, these points will not be selected. Rather, the fits at neighbor points interior to the image will be obtained in the "best window position" search and these more-reliable fits will be used to compute the curvatures at the top and bottom of the image. This implies that the first valid fit error stored from the array will be at linear index $[m \cdot (N+1)]$ and the last valid fit error stored will be at index $[N^2 - 1 - m \cdot (N+1)]$ with respect to the first position in the output map. Hence, there are a total of $[N^2 - 2m \cdot (N+1)]$ valid fit errors computed by the array; the first $[m \cdot (N+1)]$ and the last $[m \cdot (N+1)]$ fit errors in the output map are contaminated by the border effect. Note that in Figure 3.7, RL is used to designate the image row length rather than the N (associated with an N X N square image) which we have used here. A square example is used for simplicity in the discussion.

The image to be processed with the systolic fit-error array must be arranged in a linear, sequential memory area to favor the serial access of the DMA channels feeding the array's data streams. The indices shown for the input and output data streams in Figure 3.7 indicate the relative offsets of the linear data streams at some general point "i" during the processing. These general relationships are used to set the DMA-channel pointers and are summarized in equations (3.7) through (3.10) as well as in Figure 3.7.

The index for the top (i.e. the $(n-1)$st) row of the Z data stream is chosen as the reference, and so this index is defined to be i as in equation (3.7). For the kth row of the Z data stream from the bottom, the index is given according to equation (3.8), where $k = 0$ for the bottom row, $k = 1$ for the next row up, and so on. There are n such Z input channels. There is only one channel C for input of the fit coefficients and its relative index is given by equation (3.9). The relative index of the output fit-error data stream E is then as shown by equation (3.10).

$$Index(Z_{n-1}) = i \tag{3.7}$$

$$Index(Z_k) = i - (n - 1 - k) \cdot (RL + n) \tag{3.8}$$

$$Index(C) = i - m \cdot (RL + 1) \tag{3.9}$$

$$Index(E) = i - m \cdot (RL + 1) - n^2 \tag{3.10}$$

Note that the fit-error output index is behind the fit-coefficient input index by n^2 "pixels" at any given time. This corresponds to the n^2 compute-intervals required to generate any result. The range for i is from 0 to $(RL \cdot CL - 1)$, the latter of these numbers being one less than the number of pixels in the image. Negative indices resulting for some of the channels at the start of the processing relate to the border effect discussed previously.

The systolic array of fit-error cells is to be supported by a DMA board design such as the one outlined here. The cells, supported by such a board, will serve as the fit-error computation processor subsystem in the dedicated hardware environment for curvature extraction. In this thesis, the fit-error computation processor constitutes the major contribution since this processor is based on a new floating-point VLSI cell architecture not previously proposed or built elsewhere.

3.3.3.5 Execution Time Estimate for the New Fit-Error Computation Processor

An order-of-magnitude estimate for the execution speed of the fit-error computation processor can be done in the same manner as for the convolution processor in the previous section. The same estimated clock rate is assumed for the fit-error cell as for the convolution cell (16 MHz), since both are of similar design and are to be implemented using a 3-micron, double-metal CMOS process. Hence the clock period for the cells is 62.5 nanoseconds. There are 256 clock cycles between outputs of successive results from the array (i.e. 256 clock cycles form one compute-interval). We assume an image size of 256 X 256, which implies 65 536 pixels must be processed. This gives a fit-error computation time of:

$$(65536 pixels)\left(256\frac{clocks}{pixel}\right)\left(62.5X10^{-9}\frac{sec}{clock}\right) = 1.05 sec \qquad (3.11)$$

Parameter loading time for serially loading each of the systolic array's n^2 fit-error cells with its own set of six UV parameters involves loading $(6n^2)$ parameters in all. Consider a large-operator case, with a 15 X 15 array of cells forming the fit-error processor. For this case, we must load 1 350 parameters. Since each UV parameter is a double-precision floating-point number with 64 bits and is loaded two bits at a time, the parameter loading time becomes:

$$(1350 parameters)\left(\frac{64}{2}\frac{clocks}{parameter}\right)\left(62.5X10^{-9}\frac{sec}{clock}\right) = 0.0027 sec \qquad (3.12)$$

This parameter loading time corresponds to less than 0.3 % of the fit-error computation time from equation (3.11), and a large array size (15 X 15) was assumed to arrive at this percentage. A smaller array size, such as 5 X 5 would be an even smaller percentage, since only 150 instead of 1 350 parameters would have to be loaded. Therefore, parameter loading time is negligible and we have an estimate for the fit-error computation time that is independent of array size. This estimate is 1.05 seconds for a 256 X 256 image according to equation (3.11) for computation of the fit-error map in double-precision, floating-point format.

3.3.4 The 386/387 Host Computer

The third computational subsystem in the proposed hardware environment is the commercially-available Intel iSBC 386/120 Single Card Computer which is referred to here as the 386/387 host computer. This host computer executes the initialization control programs for the operation of the VLSI-based special-purpose convolution and fit-error processors. The 386/387 host also performs the third "curvatures" step in the algorithm. This third algorithm step is itself divided into three parts: 1) the search of the fit-error map to find the best window positions (BWPs), 2) the use of the BWPs to compute partial derivative estimates, and 3) the non-linear combination of the partial derivative estimates to produce the mean and Gaussian curvature maps

The 386/387 host computer features the 80387 math co-processor which operates in tandem with the 80386 microprocessor in a closely-coupled fashion. In fact, the 387 operates "in the shadow" of the 386 CPU which actually executes the code and passes instructions to the 387 when they are detected to be numeric instructions. Both the 386 and 387 are run at a clock rate of 20 MHz. The 386/387 host computer also features 64K bytes of static RAM cache memory and a full 32-bit MULTIBUS II Parallel System Bus interface.

The Intel iSBC 386/120 Single Card Computer was selected as one of the computational subsystems in the hardware environment because the 80387 math co-processor on this card is well-suited to satisfy the need, in the last step of the algorithm, for a variety of "point-oriented" (rather than neighborhood-oriented) numeric operations. The formation of partial derivatives and then curvatures for each point in the range image involves non-linear functions of the fit coefficients at and (u_*, v_*) offset to the best window position associated with that particular point in the range image:

$$H, K \text{ at } (x, y) = \text{Functions of } (u_*, v_*, a_*, b_*, c_*, d_*, e_*) \text{ at } (x, y)$$

Such a task is ideally suited to the rich floating-point, numeric instruction set of the 80387 math co-processor. Supported numeric instructions of particular interest in the curvature computation are comparison, addition, subtraction, multiplication, division, square root, and absolute value. In addition there are load and store operations required for memory access. Typical execution times required by the 387 for these instructions are given in

Instruction	# of Periods, T	Time (μsec)
Load	25	1.250
Store	45	2.250
Compare	31	1.550
Add	37	1.850
Subtract	36	1.800
Multiply	57	2.850
Divide	94	4.700
Square Root	129	6.450
Absolute Value	22	1.100
*Power(3/2)	243	12.150

Figure 3.8 Typical Instruction Times for 80387 Math Co-Processor

Figure 3.8.

The execution times of Figure 3.8 are given in terms of the number of clock periods, T, and in terms of microseconds. The times in microseconds are computed assuming a 20 megahertz clock; hence the clock period T, assumed is 50 nanoseconds. These execution times are based on the upper limits of the ranges cited for the double-precision (or 64-bit real) format of the 387[†]. The "power(3/2)" operation, i.e. computing $x^{3/2}$, is indicated in Figure 3.8 with an asterisk (*) because it is the only operation listed that does not correspond to a single 387 instruction. Here, it is assumed to be computed as $x^{1/2} \times x^{1/2} \times x^{1/2}$, using 1 square-root and 2 multiplication instructions.

ALGORITHM STEP	ADDS +SUBT.	MULT.	DIV.	SQUARE or MAGN.	COM-PARES	POWER (3/2)
1)FITTING	9 437 184	9 830 400	-	-	-	-
2)ERRORS	9 764 864	13 107 200	-	1 683 400	-	-
3)CURVATURES:						
a)Best window	-	-	-	-	1 572 864	-
b)Derivatives	262 144	524 288	-	-	-	-
c)Curvatures	589 824	983 040	131 072	-	-	65 536

Figure 3.9 Number and Type of Operations for Curvature Extraction Using 5 X 5 Operators on a 256 X 256 Image

[†] As given in Appendix E of [Inte87], pages E-34 and E-35 The times given in Figure 3.8 are associated with the integer/real memory option of the table in [Inte87].

Figure 3.9 gives the number of numeric operations of the various types required for the different steps in the curvature extraction algorithm for the case of a 256 X 256 range image with 5 X 5 window and neighborhood operations This is the test case being studied The first two steps, the "fitting" and "errors" steps in the figure, are to be performed by the VLSI-based special-purpose processors discussed previously while the third step, "curvatures", is to be performed by the 386/387 host. This is the proposed partitioning of the curvature-extraction task among the processors in the dedicated hardware environment. However, it is interesting to estimate the performance that would be obtained if the 387 were used in all three steps. Such an estimate is given in Figure 3.10.

ALGORITHM STEP	Numeric Op's Time (sec)	Memory Op's Time (sec)	Numeric + Memory Op's Time (sec)
1)FITTING	45.476	67.437	112.913
2)ERRORS	57.273	85.944	143.217
3)CURVATURES	9.593	14.451	24.044
a)Best window	2.438	5.505	7.943
b)Derivatives	1.979	2.753	4.732
c)Curvatures	5.176	6.193	11.369
TOTAL	112.342	167.832	280.174

Figure 3.10 Execution Time Estimates for the Three Algorithm Steps if Performed on the 80387

Figure 3.10 breaks down the execution time among the various steps of the algorithm; the times required at each step for numeric and memory-access operations are shown separately in addition to the total time for memory-access plus numeric operations. The time estimates for numeric operations were obtained by multiplying the numbers of operations of the various types in each step from Figure 3.9 by the corresponding instruction times per operation on the 387 as shown in Figure 3.8. The memory-access times are worst-case estimates obtained assuming that every numeric operation requires one load from memory and one store to memory. The estimate is a worst case since, in a well-written 387 program, a good proportion of the numeric operations will have internal 387 registers as the source and/or destination rather than memory and inter-register transfers require less time than memory-access instructions. Hence, the worst-case memory-access times were obtained by counting the total number of numeric operations in each step from Figure 3.9 and multiplying the total by 3.5 microseconds, the time for one load plus one store operation as listed in Figure 3.8

The simple estimates of Figure 3.10 indicate that the first two steps, the "fit" and "errors" steps, would be quite slow on the 387. In fact, the total execution time of 4.: minutes (280 seconds) and the relative times spent on each algorithm step shown in Figure 3.10 are quite similar to those of Figure 2.7 for the general computing environment of Sun workstations given in Chapter 2 The bulk of the time is spent on the first two steps, the "fitting" and "errors" steps, since these are neighborhood-oriented.

The neighborhood nature of the fit convolution and fit-error computation makes these steps well-suited to systolic VLSI implementations. Significant gains in processing speed result for these steps by utilizing the special-purpose VLSI processors instead of the networked Suns or even the dedicated 386/387 host. On the Suns, these steps required 132 and 188 seconds respectively; on the 387 the estimates are 113 and 143 seconds, while the VLSI processors reduce the times to 0.6 and 1.05 seconds respectively. Hence, the analysis of Figure 3 10 reinforces the soundness of selecting the "fitting" and "errors" steps for special-purpose VLSI designs.

The third "curvatures" step, however, remains allocated to the 386/387 host in the dedicated hardware architecture. Figure 3.10 indicates that 9.6 seconds of numeric operations are required which is less than the 14.5 seconds of memory-access time predicted using the worst-case method described above. The total time for the "curvatures" step is estimated at 24 seconds. This time estimate is very conservative, but still indicates a factor of 2 improvement over the time taken in the networked Sun environment (59 seconds). This result, as well as the fact that the more complex division and square root operations are required in the "curvatures" step, support the idea that this algorithm step is well-suited to the 387. It would be difficult to design a VLSI element that could out-perform the commercial 387 product for the third point-oriented "curvatures" step.

A truly accurate estimate of the execution time for the third "curvatures" step of the algorithm on the 386/387 host would require writing the actual program. It is recommended that this program be written in 386/387 macro assembler code to retain maximum control over how the program is translated into 387 instructions Even if a very good compiler is used on a program written in a high-level language such as C, coding the algorithm in assembler can result in improved performance if the program is carefully written by a programmer with a good knowledge of the hardware being used. In particular, the program should favor the cache memory. Special care should be taken in coding the search of the fit-error map for local minima and in choosing how the best-window-position parameters are arranged in memory since these parts of the algorithm step involve many memory accesses. Once written, the code should be tested on a system such as the IBM PS/2 model 80 which has a 386 CPU with a 387 math co-processor. Finally, the complete algorithm can be tested in a prototype of the proposed hardware environment including the Intel iSBC 386/120 Single Card Computer board once this prototype environment is assembled.

Having seen that the third "curvatures" step in the algorithm is well-suited to the 387 while the first two steps are not, the performance estimates presented thus far will be summarized in the next section: the overall performance evaluation of the dedicated hardware environment.

3.4 Overall Performance Evaluation of the Dedicated Hardware Environment

In the previous three sections, the performance of each of the three computational subsystems in the proposed hardware environment has been estimated for the test case of a 256 X 256 range image processed with 5 X 5 local operators. In this section, the individual estimates will be tied together as an overall estimate and compared to the performance using the C program in the general-purpose, multi-user, networked environment of Sun 3/60 workstations as implemented for the experiments of Chapter 2.

The workstations in the networked environment of Suns used for benchmarking each feature a 20 MHz Motorola 68020 16-bit CPU with a 68881 floating-point unit (FPU). The environment runs the UNIX BSD Version 4 3 operating system and the Ethernet network is used. Figure 3.11 shows how the execution time for the test case (256 X 256 image, 5 X 5 operators) using the C program in the Sun environment is distributed among various activities. These activities include space allocation, reading data in from files and writing data out to files, as well as the three computational steps in the algorithm. The three computational steps account for over 97 percent of the execution time, while the other activities require less than 3 percent of the total time. These other activities are analogous to coefficient loading time and pipeline-fill time in the case of the special-purpose VLSI processors. The two test runs summarized in Figure 3.11 show very similar statistics. Hence, the "other activities" can be neglected in the performance-estimate discussions for the C program running in the Sun environment

Activity	Run 1 (sec)	Run 1 (%)	Run 2 (sec)	Run 2 (%)
alloc. space for depth map	0.26	0.067	0.32	0.083
read in depth map	0.20	0.051	0.18	0.047
depth map type conv.	0.62	0.160	0.86	0.223
space alloc.	3.08	0.11	2.40	1.231
read in convol. masks	0.16	0.041	0.08	0.021
fit surface (convolve)	132.32	34.060	133.54	34.679
calc. fit err.s (abs. method)	188.00	48.391	184.96	48.032
compute curvatures	58.98	15.181	57.6	14.958
store curvatures	3.56	0.916	2.80	0.727
TOTAL	388.50 (6.5 min)	100.	385.08 (6.4 min)	100.

Figure 3.11 Breakdown of Execution Time Spent on the Various Parts of the C Program Running in the Sun Environment

Figure 3.12 compares the execution times for the curvature extraction algorithm 1) as estimated for the dedicated hardware environment and 2) as benchmarked using the C program in the general-purpose, networked, multi-user environment of Sun workstations. The times are listed to the nearest tenth of a second The most impressive speed improvements are realized with the floating-point VLSI arrays for the "fitting" and "errors" steps. For the test case studied, these VLSI implementations yield a speed increase on the order of 100 over the general-purpose environment. More precisely, considcration of the ratio of the general-purpose environment time over the dedicated environment time indicates that a factor 220.5 improvement ensues for the fit convolution and a factor 170.9 ensues for the fit-error computation. Conservative estimates for the "curvatures" step done on the 386/387 host indicate a factor 2.5 speed improvement over the general-purpose environment. A reasonable overall speed improvement factor of 14.8 results for the curvature extraction in the dedicated environment. In absolute terms, the user must wait just under half a minute for the curvatures in the dedicated environment as opposed to 6.3 minutes in the general-purpose environment.

ALGORITHM STEP	Time (sec) in Proposed Dedicated Environment	Time (sec) in General-Purpose Sun Environment
1) Fitting	0.6	132.3
2) Errors	1.1	188.0
3) Curvatures	24.0	59.0
TOTAL	25.7	379.3

Figure 3.12 Comparison of Execution Times for the Curvature Extraction Algorithm with 256 X 256 Image and 5 X 5 Operators Running in the Dedicated versus the General-Purpose Computing Environment

The estimated speed improvements are even more pronounced for larger operator sizes. This is especially true of the systolic-array processing times as they are virtually independent of operator size while the C-program execution time is approximately proportional to operator size.

An additional experiment was run to demonstrate the situation for the large operator size of 15 X 15. Since window operators for $n = 15$ were not developed, dummy operators with arbitrary coefficients were used. The fact that the operators are not correct does not affect the processing time since the algorithm is totally one-pass, with no iteration or convergence required The C program was run with the arbitrary 15 X 15 operators and benchmarked Execution-time estimates for the dedicated environment were calculated with the same methods applied to the previous test case of 5 X 5 operators and a 256 X 256 image. The comparison for the new test case of 15 X 15 operators and a 256 X 256 image is illustrated in Figure 3.13. The window operations for the new test case are 9 (= 225/25) times as large as in the previous test case.

ALGORITHM STEP	Time (sec) in Proposed Dedicated Environment	Time (sec) in General-Purpose Sun Environment
1) Fitting	0.6	1 186.6
2) Errors	1.1	1 914.1
3) Curvatures	90.2	350.0
TOTAL	91.9	3 450.7

Figure 3.13 Comparison of Execution Times for the Curvature Extraction Algorithm with 256 X 256 Image and 15 X 15 Operators Running in the Dedicated versus the General-Purpose Computing Environment

For the new test case, the VLSI implementations show an order of 1000 speed improvement over the general-purpose environment for their respective algorithm steps. Specifically, the fit convolution step shows a factor 1978 improvement and the fit-error computation a factor 1740 improvement. The host-processor shows a factor 3.9 improvement for the "curvatures" step and the overall improvement for the dedicated environment shows a factor 37.5 improvement. The absolute times for the entire curvature extraction are 1.5 minutes with the dedicated environment and nearly an hour with the general-purpose environment. Hence, for larger operators, the estimated speed improvement becomes very significant indeed.

Having looked at the case of a large window size, consider increasing the size of the input range image. Two factors become important. First, the total memory requirements for performing the curvature extraction become excessive in both environments.

Second, the computation load increses since this load is proportional to the number of pixels in the range image.

The dedicated hardware environment for curvature extraction featuring a new floating-point VLSI processor for fit-error computation have been presented and evaluated in this chapter. The next chapter discusses the results presented in this thesis, with an emphasis on their relationship to one and other and their implications for future work.

Chapter 4 Discussion

The algorithm and the architecture for curvature extraction were treated in Chapters 2 and 3 respectively. Each of these chapters followed the principle of presenting first some terminology and a review of the literature, then describing the design of the algorithm or the architecture, and finally presenting results of experiments or estimates of performance. In this chapter, the implications of the two sets of results are discussed in light of how they influenced each other and how they point out directions for future work.

4.1 Results of Algorithm Studies

In Chapter 2, two algorithms for computing curvature maps from range images were studied, the first was an algorithm developed by Yokoya and Levine [Yoko87], and the second was a modified version of the first algorithm. The modification simplified the design of a systolic VLSI cell for computing the fit-error map described in Chapter 3. Computation of the fit-error map is central to both algorithms and is the most execution-time intensive step in both algorithms when they are implemented in C in a networked environment of Sun workstations.

Experimental studies of the original and modified algorithms examined the magnitude and distribution of fit error, how the best window positions were selected, and how the extracted curvatures compared with analytically-computed curvatures for artificial range images. The robustness of the algorithms was investigated by computing statistics of the extracted curvatures in regions of known, constant curvature with different window sizes and with varying amounts of noise added Emphasis was placed upon the ability of the algorithms to correctly extract the signs of the mean and Gaussian curvature. The performance of the modified algorithm on real range images was also demonstrated Execution times for the algorithms with different image and window sizes were discussed using benchmarks obtained for a C program running in a networked environment of Sun workstations.

The behavior of the original and modified algorithms was found to be very similar in all of the aspects studied in Chapter 2. Hence, the results of Chapter 2 indicate that the modified algorithm is just as robust as the original algorithm for the cases studied. It is reasonable to assume that this similarity is representative of the relative performance of the two algorithms in general.

The studies of constant-curvature regions in artificial range images with various amounts of noise added indicate that larger window sizes are needed to extract the curvatures reliably as the noise level is increased The Gaussian curvature was confirmed to be more noise-sensitive in these studies than the mean curvature The algorithm proved able to correctly extract the curvature signs from noisy range data, provided the noise level is not too high and/or large enough window operators are used Segmentation of range images using the curvature maps is therefore feasible, but some form of post-processing on the curvatures is recommended to "clean them up" before use in higher-level processing Since they are second-order surface characteristics, the curvatures are inherently noise-sensitive. However, with proper choice of window size and some post-processing, the curvature maps obtained in this thesis should prove adequate. Hence, the modified algorithm's ability to extract the curvatures in the presence of noise was deemed favorable enough to warrant proceeding with the development of the VLSI fit-error cell embodying the modification.

Long execution times were observed for both algorithms running in a general-purpose, multi-user network of Sun workstations. This provided an incentive for developing a dedicated environment featuring special-purpose hardware.

The problem of threshold-level selection is important in using the curvature maps for further processing. In this thesis such further processing was not addressed. The final results herein are curvature maps, since extracting these was the goal of the thesis work However, in an object recognition system, such later processing stages as region growing, mode filtering, and matching against a database of models might be required. In any case, choosing the appropriate threshold levels is essential to preserve the trends in the curvature information and forms a problem requiring future study. The viewing of the Gaussian curvature map of "Mas5" under two different scale transformations in Chapter 2 (Figure 2.26 parts (e) and (f)) suggested the importance of this area of study since image features were virtually invisible for the wider scale; the wider scale corresponds to a higher positive and lower negative threshold bracketing zero.

The requirement for an adequate method of handling large dynamic ranges is another issue of vital importance, particularly for the intermediate processing stages. During the algorithm studies, large dynamic ranges were obtained in the maps of fit error. Two clusters of fit error values were found. The fit-error values for points near discontinuities tended to be very large while those values for points in continuous areas were quite small. In order for the "best window positions" computed from the fit-error map to be meaningful, the small differences between the fit errors at points within each cluster should be preserved. In this thesis, floating-point representations have been advocated for handling the dynamic-range problem at the hardware and software levels. Double-precision floating-point format is used by the C program of Chapter 2, by the two VLSI-based processors of Chapter 3, and by the 386/387 host computer of Chapter 3. A comparison between the results obtained with internal calculations done in single versus double precision floating point would be a worthwhile future study; it would help to determine exactly how sensitive the dynamic-range problem is.

4.2 Results of Architecture Studies

In Chapter 3, a hardware architecture for a dedicated computing environment was proposed. The architecture features three computational subsystems. a commercially-available host processor card based on the 80387 math co-processor with advanced. floating-point capabilities and two special-purpose processors based on systolic VLSI cell arrays configured as DMA machines. The design of one of these VLSI cell arrays. the one for fit-error computation [Malo89b]. is an original design produced as part of the work for this thesis. The other cell array is an original "sister" design produced by other researchers [Laro89] but is associated with a common effort towards special-purpose hardware development. The three computational subsystems in the dedicated environment use the Intel MULTIBUS II as a common bus to share a common memory.

Results for the hardware environment proposed in Chapter 3 appear in the form of execution-time estimates. Estimates of the execution time for each subsystem were computed and combined for comparison against the execution time required by the C program running in the general-purpose. networked environment of Sun workstations

The execution-time estimates also reinforce the design choice of implementing the fit-convolution and fit-error computation steps with systolic VLSI-based processors as opposed to using the dedicated 386/387 host to perform the entire curvature extraction algorithm. The first two algorithm steps. the fit convolutions and the fit-error computations. are neighborhood-oriented and feature less-complex floating-point operations (no divisions or square roots). These steps are seen to be well-suited for special-purpose VLSI arrays. The third "curvatures" step is dominated by point-wise operations and requires division and square-root operations. this step is better suited to the rich instruction set of the 387 math co-processor existing on the host computer card.

The performance evaluation suggests impressive speed improvements for the VLSI-based processors and a reasonable speed improvement for the dedicated hardware environment as a whole Speed increase factors of order 100 were estimated for the VLSI-based processors and an overall speed increase factor near 15 was predicted for the environment as a whole using 5 X 5 operators. The reduction in computing time becomes more pronounced as the operator size is increased since the processing time for the systolic

VLSI subsystems in the dedicated environment is virtually independent of operator size. With 15 X 15 operators, the speed increase factor for the VLSI-based processors rises to be of order 1000 while the speed increase factor for the overall environment approaches 40. These results are examples of the performance gains that can result when dedicated systems and VLSI implementations are used in machine vision applications.

Simulation and layout of the fit-error cell as well as construction of the dedicated environment for curvature extraction are planned in the near future. Fault-tolerance and testability concerns are to be addressed as part of the future work on the fit-error cell implementation Modifications to the architecture may be called for as a result of this future work in order for the cell to be acceptable in terms of its silicon area and testing strategy The cell size can then be determined and the 16 MHz clock rate confirmed or modified Simulations of the cell and board architectures for the fit-error processor are also planned, special simulators for this purpose will be developed in the near future. These will follow the methods of earlier simulators such as the convolution-board simulator by Collet *et al* [Coll89] and the floating-point convolution-cell simulator by Côté *et al* [Côté89]. The board simulator of Collet *et al* verifies such aspects as correct operation of the DMA channels and PLA control signal generator. The cell simulator of Côté *et al* is a register-level debugger, allowing examination of the cell's registers as the cell's clock is single-stepped Similar simulators are to be designed for the fit-error cell and board.

This chapter has reviewed the study of an algorithm and architecture for computing curvature maps from range images. The next chapter concludes with a brief summary of what has been accomplished.

Chapter 5 Conclusion

In this thesis, a modified algorithm for computing curvature maps from range images has been studied and a dedicated hardware architecture featuring a new floating-point systolic VLSI cell for fit-error computations has been proposed for efficient realization of the algorithm. A review of the literature has been presented illustrating the role of curvature quantities and alternative methods in range-image understanding. Examples from the literature featuring VLSI implementations in machine vision have also been considered. Results of applying the curvature extraction algorithm to real and artificial range images have been presented showing the effects of window-operator size and input-image noise level. Execution time estimates have been given for the curvature extraction algorithm using the proposed hardware architecture and these have been compared with the execution time required using a C program running in a networked environment of Sun workstations.

Studies of the algorithm indicated that the modified version, which favors a systolic VLSI implementation for the fit error, is just as robust as the original algorithm and that both algorithms can successfully extract the curvature signs in the presence of noise if sufficiently large operator sizes are used. Execution time estimates indicate very promising speed improvements on the order of 100 to 1000 for the fit-error computation using the new VLSI fit-error processor; a reasonable increase in speed over that of a general-purpose computing environment is also predicted for the complete curvature extraction algorithm as performed by the dedicated hardware architecture. Estimated speed improvements for the new hardware are most dramatic for cases where large operator sizes are used.

References

[Abde88] M. Abdelguerfi, A. K. Sood, S. Khalaf, "Parallel Bit-Level Pipelined VLSI Processing Unit for the Histogramming Operation", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Ann Arbor, Michigan, June 1988, pp. 945 – 950.

[Abde89] N. Abdelmalek, P. Boulanger, "Algebraic Error Analysis for Surface Curvatures of 3-D Range Images", accepted for presentation at the conference *Vision Interface '89*, London, Ontario, June 19-23, 1989.

[Acha88] P. K. Acharya, T. C. Henderson, "Parameter Estimation and Error Analysis of Range Data", *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 3, Philadelphia, Pennsylvania, 1988, pp. 1709 – 1714.

[Alle88] T. Allen, C. Mead, F. Faggin, G. Gribble, "Orientation-Selective VLSI Retina" (invited paper), *Proceedings of the Third SPIE Conference on Visual Communications and Image Processing*, Vol. 1001, Part 2, Cambridge, Massachusetts, November 9-11, 1988, pp. 1040 – 1046.

[Anne88] J. Annevelink, "HIFI: A Design Methodology for Implementing Signal Processing Algorithms on VLSI Processor Arrays", doctoral dissertation, Department of Electrical Engineering, Delft University of Technology, Delft, Netherlands, January 1988.

[Aubr87] S. Aubry, V. Hayward, "Range Image Analysis Using Level Curves", *Fifth Scandinavian Conference on Image Analysis*, Stockholm, Sweden, June 1987, pp. 661 – 668.

[Aubr89] S. Aubry, V. Hayward, "Building Hierarchical Solid Models from Sensor Data", to appear in *Advances on Spatial Reasoning*, Su-shing Chen (Ed.), ABLEX Publishing.

[Blak87] A. Blake, A. Zisserman, *Visual Reconstruction*, Cambridge: MIT Press, 1987.

[Besl85] P. Besl, R. Jain, "Range Image Understanding", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, California, 1985, pp. 430 – 449.

[Besl86] P. J. Besl, R. C. Jain, "Invariant Surface Characteristics for 3D Object Recognition in Range Images," *Computer Vision, Graphics, and Image Processing*, Vol. 33, January 1986, pp. 33 – 80.

[Blan87] M. Blanchet, D. Poussart, "High-Speed 3D Orientation Processor Based on

VLSI Arithmetic Cells", *Proceedings of the IEEE COMPINT '87 Conference*, Montreal, Quebec, November 1987, pp. 161 - 168.

[Boud86] Y. Boudreault, A. Malowany, "A VLSI Convolver for a Robot Vision System," *Proceedings of the Canadian Conference on Very Large Scale Integration*, Montreal, Quebec, October 1986, pp. 265 - 270

[Chan88] P. R. Chang, C. S. G. Lee, "Residue Arithmetic VLSI Array Architecture for Manipulator Pseudo-Inverse Jacobian Computation", *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 1, Philadelphia, Pennsylvania, 1988, pp. 297 - 302.

[Cohe88] F. S. Cohen, R. D. Rimey, "A Maximum Likelihood Approach to Segmenting Range Data", *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 3, Philadelphia, Pennsylvania, 1988, pp. 1696 – 1701.

[Coll89] C. Collet, J. F. Côté, D. D. Haule, A. S. Malowany, "Architectural Simulation in Digital Systems Design", accepted for presentation at *The Summer Computer Simulation Conference*, Austin, Texas, July 1989

[Côté88] J. F. Côté, C. Collet, D. D. Haule, A. S. Malowany, "A High Performance Convolution Processor", *Proceedings of the Third SPIE Conference on Visual Communications and Image Processing*, Vol 1001, Part 1, Cambridge, Massachusetts, November 9-11, 1988, pp 469 – 475.

[Côté89] J. F. Côté, F. Larochelle, A. S. Malowany, "Architectural Simulation of VLSI Design to Validate Algorithms", accepted for presentation at *The Summer Computer Simulation Conference*, Austin, Texas, July 1989.

[Cox87] H. Cox, K. Fadlallah, S. Gaiotti, A. Jain, M. Malowany, R. Tio, B. Mandava, J. Rajski, N. C., Rumin, "A Processing Element for a Reconfigurable Massively-Parallel Processor", *Proceedings of the 1987 Canadian Conference on Very Large Scale Integration*, Winnipeg, Manitoba, October 1987, pp. 241 – 246.

[Dian88] R. Dianysian, R. L. Baker, "Bit-Serial Architecture for Real Time Motion Compensation", *Proceedings of the Third SPIE Conference on Visual Communications and Image Processing*, Vol. 1001, Part 1, Cambridge, Massachusetts, November 1988, pp. 900 - 907.

[Dobb87] A. Dobbins, S. W. Zucker, M. Cynader, "Endstopped Neurons in the Visual Cortex as a Substrate for Calculating Curvature", *Nature*, Vol. 329 (6138), 1987, pp. 438 – 441.

[DoCa76] M. P. Do Carmo, *Differential Geometry of Curves and Surfaces*, Englewood Cliffs: Prentice-Hall, 1976, Ch. 1, p. 1.

[Delc88] C. J. Delcroix, M. A. Abidi, "Fusion of Edge Maps in Color Images", *Proceedings of the Third SPIE Conference on Visual Communications and Image Processing*, Vol. 1001, Part 1, Cambridge, Massachusetts, November 1988, pp. 545 - 554.

[Dubo89] D. Dubois, D. Poussart, "Real-Time Image Contour Extraction with Analog and Photosensitive CMOS Devices", accepted for presentation at the conference *Vision Interface '89*, London, Ontario, June 19-23, 1989.

[Faug83] O. D. Faugeras, M Hebert, "A 3-D Recognition and Positioning Algorithm Using Geometrical Matching Between Primitive Surfaces", *Proc. Eighth IJCAI*, 1983, pp 996 - 1002.

[Ferr81] F. P. Ferrie, M. D. Levine, "A Rule-Based Picture Interpretation System for Cell Tracking and Analysis", *Seventh Conf. Canadian Man-Computer Communications Society*, Waterloo, Ontario, June 10-12, 1981.

[Ferr89] F. P. Ferrie, P. Whaite, J. Lagarde, "Toward Sensor-Derived Models of Objects", accepted for presentation at the conference *Vision Interface '89*, London, Ontario, June 19-23, 1989.

[Fort86] M Fortier, S. Sabri, O Bahgat, "Architectures for VLSI Implementation of Movement Compensated Video Processors", Special Issue of *IEEE Transactions on Circuits and Systems*, Vol CAS-33, February 1986, pp. 250 - 259.

[Fort87] J. A. B. Fortes, B. W Wah, "Systolic Arrays - From Concept to Implementation", *Computer*, Vol. 20, No 7, July 1987, pp 12 - 17

[Gino88] R. Ginosar, Y Y Zeevi, "Adaptive Sensitivity / Intelligent Scan Imaging Sensor Chips", *Proceedings of the Third SPIE Conference on Visual Communications and Image Processing*, Vol 1001, Part 1, Cambridge, Massachusetts, November 1988, pp. 462 - 468.

[Godi89a] G. Godin, M. D. Levine, "Edge-Based Descriptions of Objects in Range Images", technical report TR-CIM-89-2, McGill Research Centre for Intelligent Machines, McGill University, January 1989

[Godi89b] G. Godin, M. D. Levine, "Building the Edge Junction Graph from Range Image of Curved Objects", accepted for presentation at the conference *Vision Interface '89*, London, Ontario, June 19-23, 1989.

[Grim84] W. E. L. Grimson, T. Lozano-Peréz, "Model-Based Recognition and Localization From Sparse Range or Tactile Data", *International Journal of Robotics Research*, Vol. 3, No. 3, 1984, pp. 3 - 35.

[Grim87] W. E. L. Grimson, T. Lozano-Peréz, "Localizing Overlapping Parts by Searching the Interpretation Tree", *IEEE Transactions on Pattern Analysis and Machine*

Intelligence. Vol. 9. No. 4. 1987. pp. 469 - 482.

[Grim88] W. E. L. Grimson. "On the Recognition of Curved Objects". *Proceedings of the IEEE International Conference on Robotics and Automation*. Vol. 3. Philadelphia. Pennsylvania. 1988. pp. 1414 - 1420

[Hana88] K. Hanahara. T. Maruyama. T. Uchiyama. "A Real-Time Processor for the Hough Transform". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 10. No. 1. January 1988. pp. 121 - 125

[Hara80] R. M. Haralick. "Edge and Region Analysis for Digital Image Data". *Computer Graphics and Image Processing*. Vol 12. No 1. January 1980. pp 60 - 73.

[Hilb52] D. Hilbert. S Cohn-Vossen. *Geometry and the Imagination*. New York: Chelsea Publishing Company. 1952. a translation into English by P. Nemenyi from the German *Anschauliche Geometrie*. Berlin: Julius Springer. 1932

[Hoff87] R H. Hoffman. A. K. Jain. "Segmentation and Classification of Range Images". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. PAMI-9. No 5. September 1987. pp. 608 - 620.

[Inte87] Intel Corporation. *80387 Programmer's Reference Manual*. Santa Clara: Intel Corporation. 1987.

[Inte88] Intel Corporation. *OEM Boards and Systems Handbook*. Santa Clara Intel Corporation. 1988.

[Kore88] I. Koren. B. Mendelson. I. Peled. G M Silberman "A Data-Driven VLSI Array for Arbitrary Algorithms". *Computer*. Vol 21. No 10 October 1988. pp. 30 - 43

[Kung87] S Y. Kung. S C Lo. S N Jean. J N Hwang. "Wavefront Array Processors - Concept to Implementation". *Computer*. Vol 20. No 7. July 1987. pp. 29 - 33.

[Kung88] S Y. Kung. *VLSI Array Processors*. Englewood Cliffs: Prentice-Hall. 1988

[Laro89] F Larochelle. J F. Côté. A. S. Malowany. "A Floating-Point Convolution Systolic Cell". accepted for presentation at the conference *Vision Interface '89*. London. Ontario. June 1989. 9 pages.

[Levi85] M. D. Levine. *Vision in Man and Machine*. New York: McGraw-Hill Book Company. 1985. ch. 6. p. 292.

[Li88a] H. F. Li. D. Pao. R. Jayakumar. "Improvement and Systolic Implementation of the Hough Transformation for Straight Line Detection". *Proceedings of the Vision Interface Conference*. Edmonton. Alberta. June 1988. pp. 86 - 89.

[Li88b] H. F. Li. M Youssef. R. Jayakumar. "Parallel Algorithms for Extracting Shape Features of Handwritten Characters". *Proceedings of the Vision Interface Conference*.

Edmonton, Alberta, June 1988, pp. 80 - 85.

[Lin82] C. Lin, M. J. Perry, "Shape Description Using Surface Triangularization", *Proc. IEEE Workshop on Computer Vision: Repres. and Control*, Rindge, New Hampshire, August 1982, pp 38 - 43.

[Ling88] Y L C Ling, P. Sadayappan, K. W. Olson, D E. Orin, "A VLSI Robotics Vector Processor for Real-Time Control", *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol 1, 1988, Philadelphia, Pennsylvania, pp. 303 - 309.

[Lips69] M M Lipschutz, *Schaum's Outline of Theory ad Problems of Differential Geometry*, New York McGraw-Hill Book Company, 1969.

[Malo88a] M. E. Malowany, A. S. Malowany, "A Rule-Based Framework for Controlling a Robotic Workcell", *Proceedings of the Seventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Edmonton, Alberta, June 1988, pp. 191 - 198.

[Malo88b] M E Malowany, A S Malowany, "A Rule-Based System for Automated Assembly and Repair of Printed-Circuit Boards in a Robotic Workcell", *Proceedings of The 1988 ASME Computers in Engineering Conference*, Vol 2, San Francisco, California, July 1988, pp 345 - 351

[Malo88c] M E Malowany, A S Malowany, "Making Curvature Estimates of Range Data Amenable to a VLSI Implementation", *Proceedings of the Third SPIE Conference on Visual Communications and Image Processing*, Vol 1001, Part 1, Cambridge, Massachusetts, November 1988, pp 345 - 353

[Malo89a] M E. Malowany, A S Malowany, "A Floating-Point Systolic Cell for Fit-Error Computations in Vision Processing", accepted for presentation at *The IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing*, Victoria, British Columbia, June 1989

[Malo89b] M. E Malowany, A. S. Malowany, "A Systolic Cell for Fit-Error Computations in Range-Image Processing", accepted for presentation at *The 1989 ASME International Computers in Engineering Conference*, Anaheim, California, July/August 1989

[Malo89c] M. E. Malowany, A. S. Malowany, "Timing Simulation of Digital CMOS Integrated Circuits Using ELsim", accepted for presentation at *The Summer Computer Simulation Conference*, Austin, Texas, July 1989.

[Malo89d] M. E. Malowany, A. S. Malowany, "ELsim: A Timing Simulator for Digital CMOS Integrated Circuits", accepted for presentation at *The 1989 IEEE Pacific Rim*

Conference on Communications, Computers, and Signal Processing, Victoria, British Columbia, June 1989

[Mead80] C. Mead, L. Conway, *Introduction to VLSI Systems*, Reading: Addison-Wesley, 1980

[Mukh86] A. Mukherjee, *nMOS and CMOS VLSI Systems Design*, Englewood Cliffs: Prentice-Hall, 1986

[Mull84] Y. Muller, P. Mohr, "Planes and Quadrics Detection Using Hough Transform", *Proceedings of the Seventh International Conference on Pattern Recognition*, August 1984, pp 1101 - 1103

[Naga79] M Nagao, T. Matsuyama, "Edge-Preserving Smoothing", *Computer Graphics and Image Processing*, Vol 9, No 4, April 1979, pp. 394 - 407.

[Naik88] S. M. Naik, R C Jain, "Spline-Based Surface Fitting on Range Images for CAD Applications", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Ann Arbor, Michigan, June 1988, pp 249 - 253

[Nazi84] A. Nazif, M. D. Levine, "Low-Level Image Segmentation. An Expert System", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-6, No 5, September 1984, pp 555 - 577

[Oshi83] M Oshima, Y. Shirai, "Object Recognition Using Three-Dimensional Information", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 5, No 4, July 1983, pp 353 - 361

[Pirs88] P. Pirsch, T Komarek, "VLSI Architectures for Block Matching Algorithms" *Proceedings of the Third SPIE Conference on Visual Communications and Image Processing*, Vol. 1001, Part 2, Cambridge, Massachusetts, November 1988 pp 882 - 891.

[Pong81] T C. Pong, L. G Shapiro, R. M Haralick, "A Facet Model Region Growing Algorithm", *Proceedings of the IEEE Conference on Pattern Recognition and Image Processing*, August 1981, pp. 279 - 284.

[Rajs87] J. Rajski, H. Cox, "A Method of Test Generation and Fault Diagnosis in Very Large Combinational Circuits", *Proc. Int Test Conf*, September 1987, pp. 932 - 943.

[Rals65] A. Ralston, *A First Course in Numerical Analysis*, New York: McGraw-Hill Book Company, 1965.

[Rhod88] F. M. Rhodes, J. J. Dituri, G. H. Chapman, B. E. Emerson, A. M. Soares, J. I. Raffel, "A Monolithic Hough Transform Processor Based on Restructurable VLSI", *IEEE Transactions on Pattern Analysis and Machine Intelligence.* Vol. 10. No. 1. January 1988. pp. 106 – 110.

[Riou86] M. Rioux. F. Blais. "Compact Three-Dimensional Camera for Robotic Applications". *SPIE Proceedings.* Vol. 728. 1986. pp. 235 – 242.

[Riou87] M. Rioux. P Boulanger. T. Kasvand. "Segmentation of Range Images Using Sine Wave Coding and Fourier Transformation". *Appl. Opt.*, Vol. 26. No. 2. 1987. pp. 287 – 293

[Roth89] G Roth. M D. Levine. "Range Image Segmentation Based on Differential Geometry and Refined by Relaxation Labelling". accepted for presentation at the conference *Vision Interface '89.* London. Ontario. June 19-23. 1989.

[Sams87] M. Samson. D. Poussart. D. Laurendeau. "3-D Range from Optical Absorbance - Application to Dental Imprint Measurements for Orthodontics Diagnosis". *Proceedings of the IEEE COMPINT '87 Conference.* Montreal. Quebec. November 1987. pp. 76 – 79

[Sand86] P. T. Sander. S. W. Zucker. "Stable Surface Estimation". *Proceedings of the Eighth International Conference on Pattern Recognition.* Paris. France. 1986. pp. 1165 – 1167.

[Sand88] P. T. Sander. "Tracing Surfaces for Surface Traces". technical report: CIM-88-2. McGill Research Centre for Intelligent Machines. McGill University. February 1988

[Sanz88] J. L. C Sanz. "Two Real-Time Architectures for Image Processing and Computer Vision". *Real-Time Object Measurement and Classification.* A. K Jain ed.. Berlin Springer-Verlag. 1988. pp. 1 – 23.

[Sawh86] D. Sawh. J Loewen. W. Lehn. H C. Card. M. Pawlak. D. M Burek. R D McLeod. "Edge Extraction Algorithms in Silicon". *Proceedings of the Canadian Conference on Very Large Scale Integration.* Montreal. Quebec. October 1986. pp 133 - 138.

[Sica87] P. Sicard. M. D. Levine. "Automatic Joint Recognition and Tracking for Robotic Arc Welding". *Proceedings of the IEEE COMPINT '87 Conference.* Montreal. Quebec. November 1987. pp. 290 – 293.

[Terz83] D. Terzopoulos. "The Role of Constraints and Discontinuities in Visible-Surface Reconstruction". *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* August 1983. pp 1073 – 1077.

[Thom79] G. B. Thomas. R. L. Finney. *Calculus and Analytic Geometry. Part II.* Fifth Edition. Reading: Addison-Wesley Publishing Company. 1979. ch. 11. pp. 526 – 531.

[Trim83] D. W. Trim, *Calculus and Analytic Geometry*, Reading: Addison-Wesley, 1983, ch. 12, p.512.

[Vain88] O. Vainio, H. Tenhunen, T. Korpiharju, J. Tomberg, Y. Neuvo, "An Edge Preserving Filter with Variable Window Size for Real Time Video Signal Processing", *Proceedings of the Third SPIE Conference on Visual Communications and Image Processing*, Vol 1001, Part 1, Cambridge, Massachusetts, November 1988, pp. 442 – 449.

[Vemu86] B. C. Vemuri, A Mitiche, J K Aggarwal, "Curvature-Based Representation of Objects from Range Data", *Image and Vision Computing*, Vol. 4, No 2, May 1986, pp. 107 - 114.

[Vemu88] B. C. Vemuri, J K Aggarwal, "Localization of Objects from Range Data", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Ann Arbor, Michigan, June 1988, pp. 893 – 898.

[West85] N. Weste, K. Eshraghian, *Principles of CMOS VLSI Design*, Reading: Addison-Wesley Publishing Company, 1985.

[Yang86] H. S. Yang, A. C. Kak, "Determination of the Identity, Position, and Orientation of the Topmost Object in a Pile", *Computer Vision, Graphics, and Image Processing*, Vol. 36, No. 2/3, November/December 1986, pp. 229 - 255

[Yang88a] X D. Yang, "Design of Fast Connected Components Hardware", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Ann Arbor, Michigan, June 1988, pp. 937 – 944.

[Yang88b] K. M Yang, L. Wu, H. Chong, M T Sun, "VLSI Implementation of Motion Compensation Full-Search Block-Matching Algorithm", *Proceedings of the Third SPIE Conference on Visual Communications and Image Processing*, Vol 1001, Part 2, Cambridge, Massachusetts, November 1988, pp. 892 – 899

[Yoko78] N Yokoya, T Kitahashi, K Tanaka, T. Asano, "Image Segmentation Scheme Based on a Concept of Relative Similarity", *Proceedings of the Fourth International Joint Conference on Pattern Recognition*, November 1978, pp. 645 - 647.

[Yoko87] N. Yokoya, M. D. Levine, "Range Image Segmentation Based on Differential Geometry: A Hybrid Approach", technical report: McRCIM-TR-CIM 87-167, McGill Research Center for Intelligent Machines, McGill University, September 1987.

[Yoko88] N. Yokoya, M. D. Levine, "A Hybrid Approach to Range Image Segmentation", *Proceedings of the 9th International Conference on Pattern Recognition*, November 1988, pp. 1 – 5.

[Yoko89] N. Yokoya, M. D. Levine, "Range Image Segmentation Based on Differential Geometry: A Hybrid Approach", accepted for publication in the journal *IEEE Transactions on Pattern Analysis and Machine Intelligence*

Appendix A. Convolution Window Operators for Local Quadric Surface Fit

Exact Fractional Form

3 X 3 Operators

$$a : \frac{1}{6}\begin{bmatrix} 1 & -2 & 1 \\ 1 & -2 & 1 \\ 1 & -2 & 1 \end{bmatrix} b : \frac{1}{6}\begin{bmatrix} 1 & 1 & 1 \\ -2 & -2 & -2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$c : \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} d : \frac{1}{6}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$e : \frac{1}{6}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} f : \frac{1}{9}\begin{bmatrix} -1 & 2 & -1 \\ 2 & 5 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

5 X 5 Operators

$$a : \frac{1}{70}\begin{bmatrix} 2 & -1 & -2 & -1 & 2 \\ 2 & -1 & -2 & -1 & 2 \\ 2 & -1 & -2 & -1 & 2 \\ 2 & -1 & -2 & -1 & 2 \\ 2 & -1 & -2 & -1 & 2 \end{bmatrix} b : \frac{1}{70}\begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ -1 & -1 & -1 & -1 & -1 \\ -2 & -2 & -2 & -2 & -2 \\ -1 & -1 & -1 & -1 & -1 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

$$c : \frac{1}{100}\begin{bmatrix} 4 & 2 & 0 & -2 & -4 \\ 2 & 1 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 1 & 2 \\ -4 & -2 & 0 & 2 & 4 \end{bmatrix} d : \frac{1}{50}\begin{bmatrix} -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \end{bmatrix}$$

$$e : \frac{1}{50}\begin{bmatrix} -2 & -2 & -2 & -2 & -2 \\ -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix} f : \frac{1}{175}\begin{bmatrix} -13 & 2 & 7 & 2 & -13 \\ 2 & 17 & 22 & 17 & 2 \\ 7 & 22 & 27 & 22 & 7 \\ 2 & 17 & 22 & 17 & 2 \\ -13 & 2 & 7 & 2 & -13 \end{bmatrix}$$

7 X 7 Operators

$$a : \frac{1}{588}\begin{bmatrix} 5 & 0 & -3 & -4 & -3 & 0 & 5 \\ 5 & 0 & -3 & -4 & -3 & 0 & 5 \\ 5 & 0 & -3 & -4 & -3 & 0 & 5 \\ 5 & 0 & -3 & 4 & -3 & 0 & 5 \\ 5 & 0 & -3 & -4 & -3 & 0 & 5 \\ 5 & 0 & -3 & -4 & -3 & 0 & 5 \\ 5 & 0 & -3 & -4 & -3 & 0 & 5 \end{bmatrix}$$

$$b : \frac{1}{588}\begin{bmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & -3 & -3 & -3 & -3 & -3 & -3 \\ -4 & -4 & -4 & -4 & -4 & -4 & -4 \\ -3 & -3 & -3 & -3 & -3 & -3 & -3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

$$c : \frac{1}{784}\begin{bmatrix} 9 & 6 & 3 & 0 & -3 & -6 & -9 \\ 6 & 4 & 2 & 0 & -2 & -4 & -6 \\ 3 & 2 & 1 & 0 & -1 & -2 & -3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -6 & -4 & -2 & 0 & 2 & 4 & 6 \\ -9 & -6 & -3 & 0 & 3 & 6 & 9 \end{bmatrix}$$

$$d : \frac{1}{196}\begin{bmatrix} -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \end{bmatrix}$$

$$e : \frac{1}{196}\begin{bmatrix} -3 & -3 & -3 & -3 & -3 & -3 & -3 \\ -2 & -2 & -2 & -2 & -2 & -2 & -2 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}$$

$$f : \frac{1}{147}\begin{bmatrix} -7 & -2 & 1 & 2 & 1 & -2 & -7 \\ -2 & 3 & 6 & 7 & 6 & 3 & -2 \\ 1 & 6 & 9 & 10 & 9 & 6 & 1 \\ 2 & 7 & 10 & 11 & 10 & 7 & 2 \\ 1 & 6 & 9 & 10 & 9 & 6 & 1 \\ -2 & 3 & 6 & 7 & 6 & 3 & -2 \\ -7 & -2 & 1 & 2 & 1 & -2 & -7 \end{bmatrix}$$

Actual Decimal Form Used

3 X 3 Operators

$$\begin{array}{cccc} & 0.1666667 & -0.3333333 & 0.16666667 \\ a & 0.1666667 & -0.3333333 & 0.16666667 \\ & 0.1666667 & -0.3333333 & 0.16666667 \end{array}$$

$$\begin{array}{cccc} & 0.1666667 & 0.1666667 & 0.16666667 \\ b & -0.3333333 & -0.3333333 & -0.33333333 \\ & 0.1666667 & 0.1666667 & 0.16666667 \end{array}$$

$$\begin{array}{cccc} & 0.25 & 0.0 & -0.25 \\ c & 0.0 & 0.0 & 0.0 \\ & -0.25 & 0.0 & 0.25 \end{array}$$

$$\begin{array}{cccc} & -0.1666667 & 0.0 & 0.16666667 \\ d & -0.1666667 & 0.0 & 0.16666667 \\ & -0.1666667 & 0.0 & 0.16666667 \end{array}$$

$$\begin{array}{cccc} & -0.1666667 & -0.1666667 & -0.16666667 \\ e & 0.0 & 0.0 & 0.0 \\ & 0.1666667 & 0.1666667 & 0.16666667 \end{array}$$

$$\begin{array}{cccc} & -0.1111111 & -0.2222222 & -0.11111111 \\ f & 0.2222222 & 0.5555556 & 0.22222222 \\ & -0.1111111 & -0.2222222 & -0.11111111 \end{array}$$

5 X 5 Operators

	0.0285714	-0.0142857	-0.0285714	-0.0142857	0.0285714
	0.0285714	-0.0142857	-0.0285714	-0.0142857	0.0285714
a	0.0285714	-0.0142857	-0.0285714	-0.0142857	0.0285714
	0.0285714	-0.0142857	-0.0285714	-0.0142857	0.0285714
	0.0285714	-0.0142857	-0.0285714	-0.0142857	0.0285714
	0.0285714	0.0285714	0.0285714	0.0285714	0.0285714
	-0.0142857	-0.0142857	-0.0142857	-0.0142857	-0.0142857
b	-0.0285714	-0.0285714	-0.0285714	-0.0285714	-0.0285714
	-0.0142857	-0.0142857	-0.0142857	-0.0142857	-0.0142857
	0.0285714	0.0285714	0.0285714	0.0285714	0.0285714
	-0.04	-0.02	0.0	0.02	0.04
	-0.02	-0.01	0.0	0.01	0.02
c	0.0	0.0	0.0	0.0	0.0
	0.02	0.01	0.0	-0.01	-0.02
	0.04	0.02	0.0	-0.02	-0.04
	-0.04	-0.02	0.0	0.02	0.04
	-0.04	-0.02	0.0	0.02	0.04
d	-0.04	-0.02	0.0	0.02	0.04
	-0.04	-0.02	0.0	0.02	0.04
	-0.04	-0.02	0.0	0.02	0.04
	0.04	0.04	0.04	0.04	0.04
	0.02	0.02	0.02	0.02	0.02
e	0.0	0.0	0.0	0.0	0.0
	-0.02	-0.02	-0.02	-0.02	-0.02
	-0.04	-0.04	-0.04	-0.04	-0.04
	-0.0742857	0.0114286	0.04	0.0114286	-0.0742857
	0.0114286	0.0971429	0.1257143	0.0971429	0.0114286
f	0.04	0.1257143	0.1542857	0.1257143	0.04
	0.0114286	0.0971429	0.1257143	0.0971429	0.0114286
	-0.0742857	0.0114286	0.04	0.0114286	-0.0742857

7 X 7 Operators

	0.0085034	0.0	-0.0051020	-0.0068027	-0.0051020	0.0	0.0085034
	0.0085034	0.0	-0.0051020	-0.0068027	-0.0051020	0.0	0.0085034
	0.0085034	0.0	-0.0051020	-0.0068027	-0.0051020	0.0	0.0085034
a	0.0085034	0.0	-0.0051020	-0.0068027	-0.0051020	0.0	0.0085034
	0.0085034	0.0	-0.0051020	-0.0068027	-0.0051020	0.0	0.0085034
	0.0085034	0.0	-0.0051020	-0.0068027	-0.0051020	0.0	0.0085034
	0.0085034	0.0	-0.0051020	-0.0068027	-0.0051020	0.0	0.0085034

	0.0085034	0.0085034	0.0085034	0.0085034	0.0085034	0.0085034	0.0085034
	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	-0.0051020	-0.0051020	-0.0051020	-0.0051020	-0.0051020	-0.0051020	-0.0051020
b	-0.0068027	-0.0068027	-0.0068027	-0.0068027	-0.0068027	-0.0068027	-0.0068027
	-0.0051020	-0.0051020	-0.0051020	-0.0051020	-0.0051020	-0.0051020	-0.0051020
	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0085034	0.0085034	0.0085034	0.0085034	0.0085034	0.0085034	0.0085034

	0.0114796	0.0076531	0.0038265	0.0	-0.0038265	-0.0076531	-0.0114796
	0.0076531	0.0051020	0.0025510	0.0	-0.0025510	-0.0051020	-0.0076531
	0.0038265	0.0025510	0.0012755	0.0	-0.0012755	-0.0025510	-0.0038265
c	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	-0.0038265	-0.0025510	-0.0012755	0.0	0.0012755	0.0025510	0.0038265
	-0.0076531	-0.0051020	-0.0025510	0.0	0.0025510	0.0051020	0.0076531
	-0.0114796	-0.0076531	-0.0038265	0.0	0.0038265	0.0076531	0.0114796

	-0.0153061	-0.0102041	-0.0051020	0.0	0.0051020	0.0102041	0.0153061
	-0.0153061	-0.0102041	-0.0051020	0.0	0.0051020	0.0102041	0.0153061
	-0.0153061	-0.0102041	-0.0051020	0.0	0.0051020	0.0102041	0.0153061
d	-0.0153061	-0.0102041	-0.0051020	0.0	0.0051020	0.0102041	0.0153061
	-0.0153061	-0.0102041	-0.0051020	0.0	0.0051020	0.0102041	0.0153061
	-0.0153061	-0.0102041	-0.0051020	0.0	0.0051020	0.0102041	0.0153061
	-0.0153061	-0.0102041	-0.0051020	0.0	0.0051020	0.0102041	0.0153061

	-0.0153061	-0.0153061	-0.0153061	-0.0153061	-0.0153061	-0.0153061	-0.0153061
	-0.0102041	-0.0102041	-0.0102041	-0.0102041	-0.0102041	-0.0102041	-0.0102041
	-0.0051020	-0.0051020	-0.0051020	-0.0051020	-0.0051020	-0.0051020	-0.0051020
e	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0051020	0.0051020	0.0051020	0.0051020	0.0051020	0.0051020	0.0051020
	0.0102041	0.0102041	0.0102041	0.0102041	0.0102041	0.0102041	0.0102041
	0.0153061	0.0153061	0.0153061	0.0153061	0.0153061	0.0153061	0.0153061

	-0.0476190	-0.0136054	0.0068027	0.0136054	0.0068027	-0.0136054	-0.0476190
	-0.0136054	0.0204081	0.0408163	0.0476190	0.0408163	0.0204081	-0.0136054
	0.0068027	0.0408163	0.0612245	0.0680272	0.0612245	0.0408163	0.0068027
f	0.0136054	0.0476190	0.0680272	0.0748299	0.0680272	0.0476190	0.0136054
	0.0068027	0.0408163	0.0612245	0.0680272	0.0612245	0.0408163	0.0068027
	-0.0136054	0.0204081	0.0408163	0.0476190	0.0408163	0.0204081	-0.0136054
	-0.0476190	-0.0136054	0.0068027	0.0136054	0.0068027	-0.0136054	-0.0476190

Appendix B. Internal Structure of the New VLSI Fit-Error Cell

This appendix describes the internal structure of the fit-error cell In particular, the four large modules within the cell will be treated individually emphasizing their operating principles. In addition, the treatment of overflows and underflows plus the motivation behind the cell's two-bit serial data paths will be discussed.

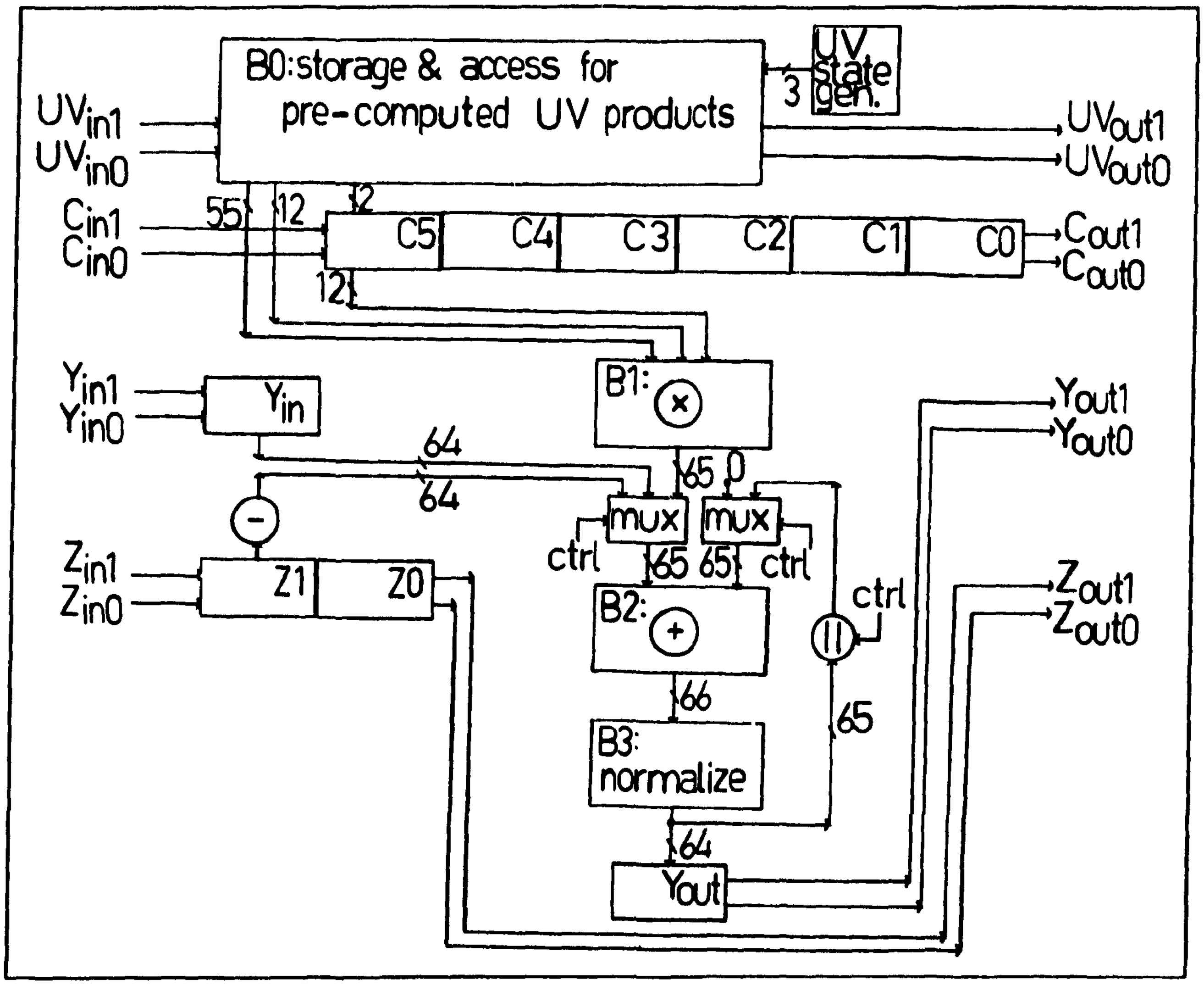


Figure B.1 The New VLSI Fit-Error Cell

The four modules to be considered are 1) the storage and access system for pre-computed *UV* products, 2) the floating-point multiplier module, 3) the floating-point align-and-add module, and 4) the floating-point normalization module. These four modules appear as blocks B0, B1, B2, and B3 respectively in the fit-error cell diagram of Figure B.1. This diagram also appears in Chapter 3 as Figure 3 3; it is reproduced here for ease of reference Figure B.1 illustrates the interconnection of the four modules It also suggests the "pin-out" of the eventual fit-error chip by displaying the I/O paths at the left and right sides of the diagram, to minimize the complexity, power, ground, clock, and control input signals have been omitted. Similar block diagrams depicting each module's internal structure are presented in this appendix. The diagrams are not exhaustive in the sense of showing every part of each circuit; rather they are presented as aids in illustrating the operating principles of the modules being described. The four main modules within the fit-error cell were patterned after the "stages" of the Larochelle floating-point convolution cell design [Laro89].

B.1 The Storage and Access System for Pre-Computed UV Products

The storage and access system for pre-computed UV products is shown as block B0 in Figure B.1. This system is closely linked to block B1, the multiplier module. To speed the mantissa multiplication in the multiplier module, an approach reminiscent of a look-up table is featured. The input fit-coefficient data C is loaded two bits at a time into the C register chain and each pair of bits is used to select a pre-computed product of one of the UV parameters stored in block B0. The fact that these two fit-coefficient bits are used for UV-parameter product selection in B0 is indicated in Figure B.1 by the routing of two bits from the register $C5$ to the block B0. Twenty-six pre-computed products are accessed in this manner using successive pairs of input C bits in the formation of a single product by the multiplier unit. A twenty-seventh access with a forced "1" is also required to realize the implicit "1" of the IEEE floating-point standard.

The UV state generator (shown at the upper-right of Figure B.1) is used to "enable" one of the six UV parameters for 27 successive accesses associated with a single product. The enabled UV parameter changes from one product to the next by means of the

UV state count, since the UV state generator is a 0-to-5 counter whose three output bits are used as control bits within block B0. The UV state count mechanizes the formation of the fitted depth value according to the equation:

$$\hat{z} = au^2 + bv^2 + cuv + du + ev + f \cdot 1 \tag{B.1}$$

by ensuring that the correct fit coefficient from the input data stream C gets multiplied by the correct UV parameter from the set $(u^2, v^2, uv, u, v, 1)$. The two mantissa bits of the input C determine which product of this UV parameter gets selected Figure B.2 shows the internal structure of the storage and access system for pre-computed UV products (which was B0 in the previous figure). The three bits of the state generator and the two bits of the input C constitute the five control bits of "*ctrl1*" in Figure B.2.

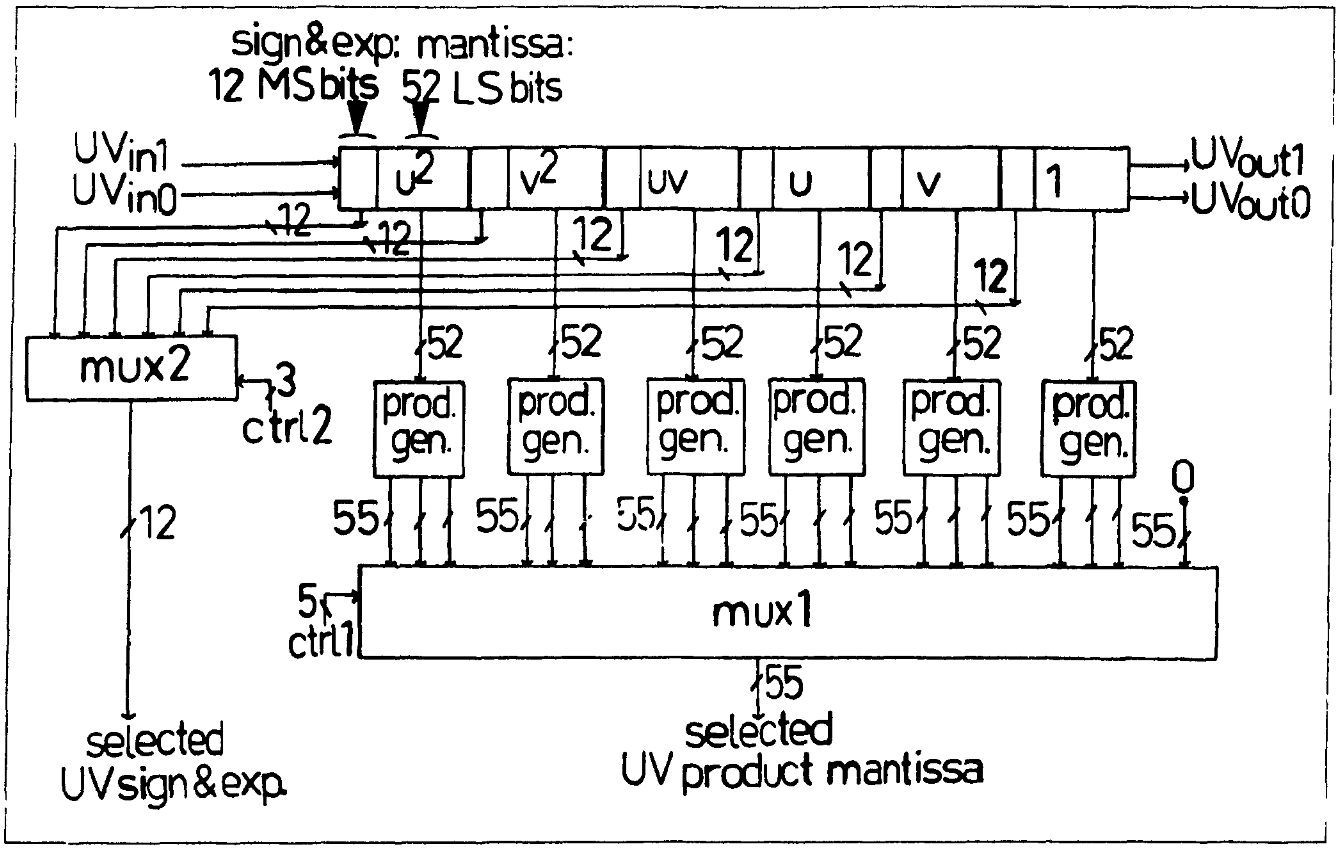


Figure B.2 The Storage and Access System for Pre-Computed UV Products

There are four possible products for each of the six *UV* parameters: the parameter times 0, 1, 2, or 3. All parameters times 0 give zero so a single hard-wired zero is provided for this case. The other three products are available at the outputs of product-generation circuits shown as "prod gen." blocks in Figure B 2 These product-generation circuits are realized by simply routing the data lines to perform a shift for the times 2 product and providing one adder for the times 3 product as shown in Figure B.3. There are 55 19-to-1 multiplexers within "mux1" of Figure B.2 which select the appropriate *UV*-parameter product according to "*ctrl*1". Note that only the mantissas, not the exponents, of the *UV* parameters are passed through the product-generation circuits and "mux1". The correct *UV* exponent is selected separately by "mux2" according to "*ctrl*2" of Figure B.2. The "*ctrl*2" signal corresponds to the three bits of the *UV* state generator.

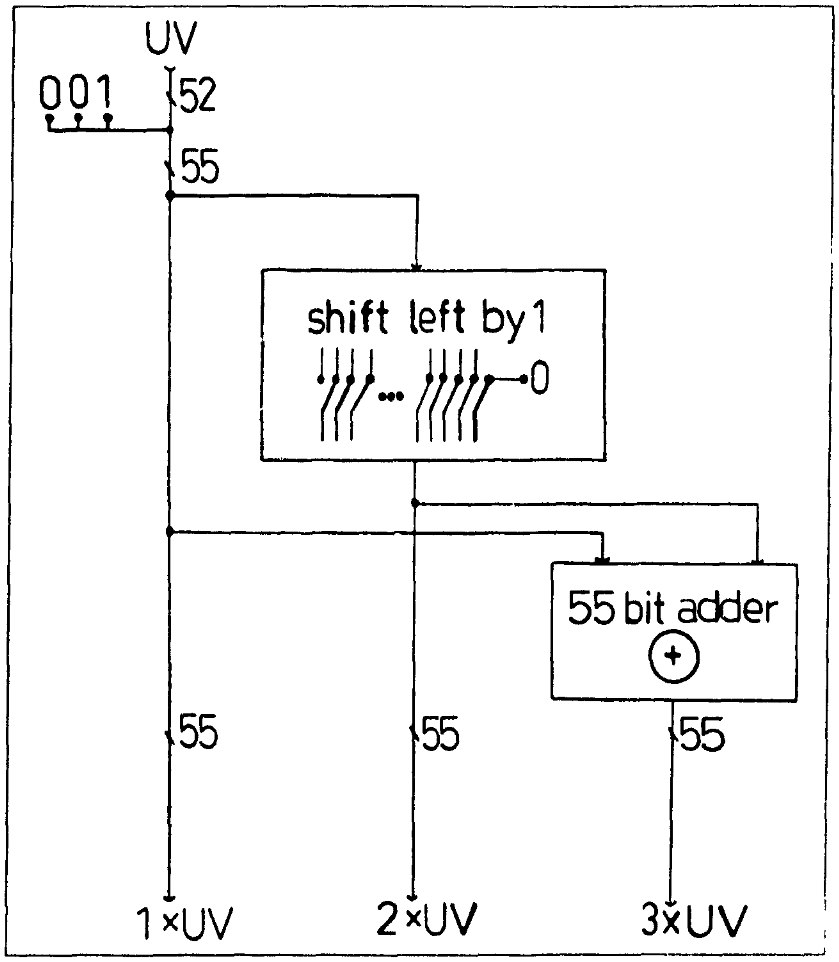


Figure B.3 The Product-Generation Circuit

Only the six UV parameters need be loaded from outside the cell since the product-generation circuits feature combinational logic which realizes the three non-trivial products of each parameter. Because the UV parameters remain constant throughout the processing, the three products of each parameter are stable at the outputs of the logic and no additional delay is required for the selected product to be computed. Hence the products are described as "pre-computed" although they are not pre-loaded from outside the cell. The product-generation circuitry requires about the same silicon area as would registers to hold the three products if they had been computed and loaded from outside.

B.2 The Floating-Point Multiplier Module

The multiplier module is shown in Figure B.4, where part (a) at the left deals with the mantissa multiplication and part (b) at the right deals with the exponent addition and sign generation.

As mentioned in the previous section, the multiplier module uses a "look-up table" style to synthesize the product C X UV by selecting and combining pre-computed products of UV. The input fit-coefficient data C is loaded two bits at a time starting with the least-significant mantissa bits. These two bits are used to select a pre-computed product of one of the UV parameters. For example, if the C bits are '11' binary then the product 3 X UV is enabled (by "mux1" of Figure B.2) into one operand field of the 55-bit 3-stage carry look-ahead adder in Figure B.4(a). Each time a new 55-bit UV product is added, this new slice is conceptually shifted left by 2 bits with respect to the previous partial result. This is actually done by shifting the previous partial result right by 2 bits with respect to the incoming new product as the previous partial result is fed back into the adder. Each time the partial result is fed back, the 53 most significant bits of the partial product register are selected and aligned with the least significant 53 bits of the new 55-bit product to be added. Thus, the required shift of the previous partial result by 2 bits to the right is realized. Note that the least significant 2 bits of the partial product are discarded[†]

[†] These 2 bits are actually part of the 52 least-significant bits of the complete product which

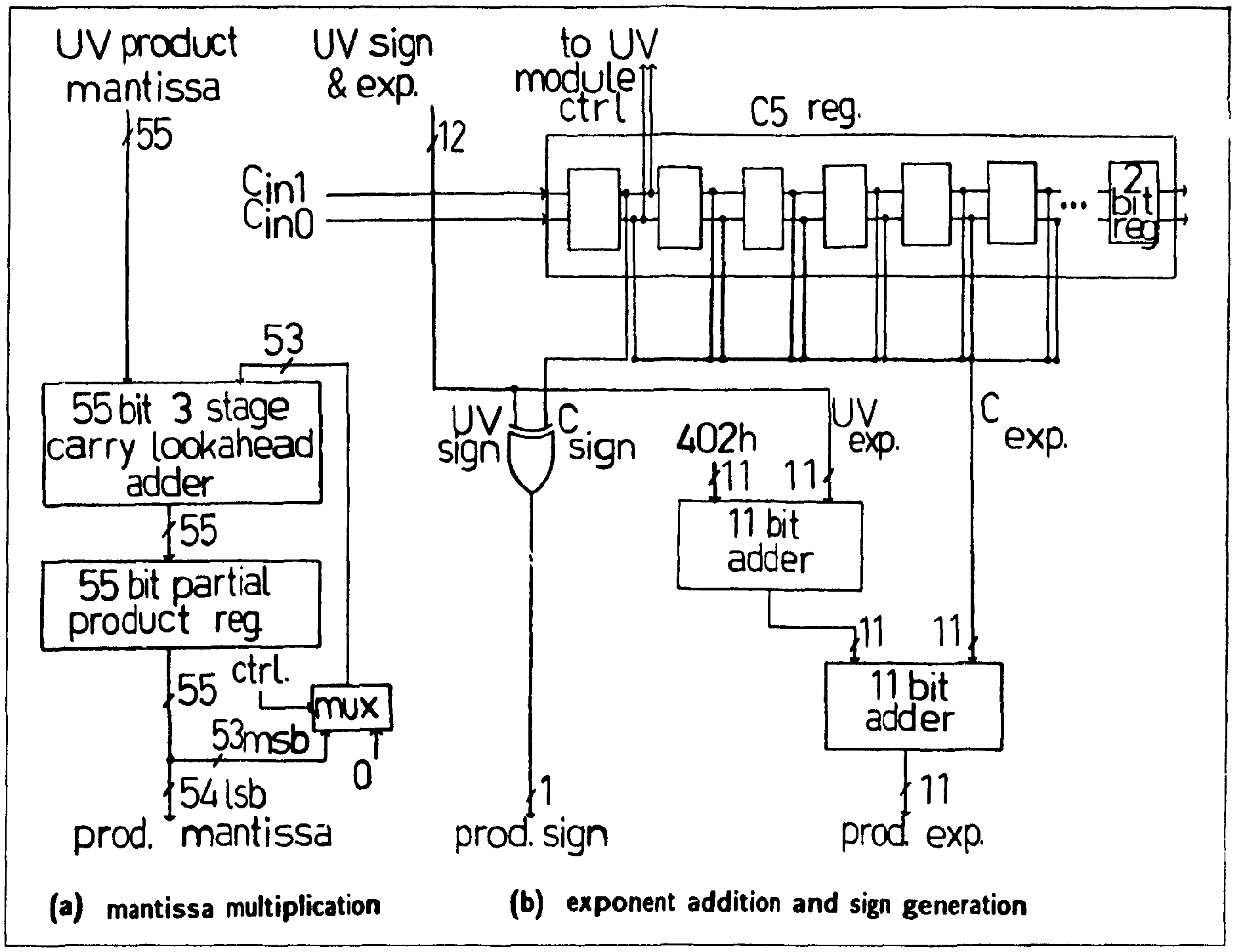


Figure B.4 The Floating-Point Multiplier Module

before each addition as a consequence of this shifting. Also note that carries out of the 55th bit of the adder never occur.

Each addition step of the type described above is achieved within one clock cycle. During the first 26 clock cycles (cycles 0 to 25), this addition process is carried out 26 times. These 26 iterations are directed by the 26 pairs of mantissa bits which make up the 52 bits of explicit fractional mantissa bits in the C input. During cycle 26, the C inputs (which are exponent bits) do not control the selected UV product; a forced product of 1

would have 106 bits if all of them were kept The 52 least-significant bits are discarded since only 53 bits are needed in the IEEE double-precision floating-point format

X UV is added to realize the implicit '1' in front of the binary point. The mantissa part of the product is thus ready at the adder output by the rising edge of clock 27, and on that rising edge it is loaded into the partial product register. Loading of new partial products occurs at the rising edge of the clock. This loading is disabled on the rising edges of cycles 0, 28, 29, 30, and 31. Cycle 32 is really cycle 0 of the next computation.

The final answer for the product mantissa is in the 54 least significant bits of the partial product register after the start of clock cycle 27. The result remains there during cycles 27 through 31 as well as during cycle 0 of the next computation. It is on the rising edge of this next cycle 0 that the next module, the align-and-add module, loads the result in from the multiplier module.

One may wonder why the product mantissa is finally equal to the 54 *least-significant* bits of the partial product register, while the most-significant bit is ignored in the end. The most significant bit of the 55-bit partial product register is always a zero at the end of the mantissa multiplication. This bit is always zero at the end because the last UV product added is always 1 X UV and the previous partial result is "shifted right" by 2 bits when fed back, effectively making the non-existent 2 top bits also zeroes. The fact that no carries from lower bits ripple up to cause a 1 in the highest bit of the partial product register was verified for the case of the largest possible product mantissa (1.111 ... 1111 X 1.111 ... 1111 = 11.111 ... 1100). In both the pre-computed 55-bit UV products and in the 55-bit partial product register the binary point is located to the left of the 52nd bit. Hence, the most-significant 3 of the 55 bits are located to the left of the binary point. In the result mantissa, there may be two non-zero digits above the binary point but not three; hence, 54 mantissa bits are kept. To conserve the 53-significant-bit representation, the least significant bit is forced to zero when the second bit above the binary point is a '1'.

Having looked at the mantissa multiplication, the exponent addition and sign generation will now be considered. The first 26 pairs of C input bits represent the mantissa of the fit coefficient. The last six pairs of bits represent the sign and exponent of the fit coefficient. Since all of these last six bit-pairs are needed simultaneously by the sign and exponent circuit of Figure B.4 (b), the output of this circuit only stabilizes during the last (31st) clock cycle even though the required inputs begin to arrive at clock 27. The C storage register is tapped to obtain the required inputs. It is the $C5$ register that is tapped, because this is the one that contains the currently-arriving fit coefficient. The $C5$ register is sequentially chained to the five other 64-bit registers $C0 - C4$. Together the six C registers hold a set of six fit coefficients (a, b, c, d, e, f) which match with the set of six UV parameters $(u^2, v^2, uv, u, v, 1)$. In fact, if these two data sets are viewed as vectors, the fit-error cell is seen to be forming their dot product which is the fitted depth as given by equation (B.1) cited earlier.

As shown in Figure B.4(b), the exponents of the stored UV parameter and the input fit coefficient C are added to produce the product's exponent and the product's sign is produced by an exclusive-or of the C and UV signs. Note that there is a pre-addition of the UV exponent with 402 hexadecimal. A pre-addition of 401 hexadecimal is equivalent to the subtraction of 1023 in two's complement. This gives the same result as subtracting 1023 from both exponents and adding 1023 to the resulting exponent. It preserves the excess-1023 representation. However, 402h is added rather than 401h. This is because an extra 1 may be generated above the binary point in the product. (This is also why 54 bits rather than 53 bits of mantissa are kept). To see this, recall that the complete product of the two 53-bit mantissas has 106 (= 2 X 53) bits, although some are discarded, and since each mantissa has 52 bits after the binary point, the result has 104 (= 2 X 52) bits after the binary point. Hence, the product has two bits above the binary point; at least one of these two digits will always be a '1'. To avoid normalizing at this point, the circuit interprets the binary point as shifted left by 1 (equivalent to shifting the number right by 1) and hence the exponent is incremented to preserve the value of the product.

B.3 The Floating-Point Align-and-Add Module

The floating-point align-and-add module is shown in Figure B.5 where part (a) gives the mantissa circuitry and part (b) the exponent circuitry.

In contrast to the multiplier module whose two operands are always obtained from the same sources (the C input stream and the UV product storage and access module), the sources of the two operands for the align-and-add module vary. Selection of which operands enter the align-and-add module is achieved with multiplexers, as indicated in Figure B.1. The first operand can be either $-Z$, Y_{in}, or the product from the multiplier and the second operand can be either zero, the true feedback from the normalization module, or the absolute value of this feedback. The way the operands are chosen at various points in the fit-error computation was discussed in Chapter 3 with reference to Figure 3.4 so will not be discussed here In this section the two operands to be aligned and added will be referred to as operand1 and operand2.

Towards the end of the previous cycle 31, the operand1 and operand2 inputs become stable. This allows a combinational circuit, "comparator1" of Figure B.5, that compares their exponents in the align-and-add module to stabilize. At the start of clock 0, the result of this comparison causes the smaller exponent to be loaded into the 11-bit "decrement-by-1/increment-by-8" variable exponent register of Figure B.5(b) while the larger exponent is loaded into the 11-bit fixed exponent register. The variable exponent register value is incremented by 8 as it is loaded.

Similarly, the result of the exponent comparison causes the mantissa associated with the smaller exponent to be loaded into the 62-bit variable mantissa register of Figure B.5(a) while the mantissa associated with the larger exponent is loaded into the 54-bit fixed mantissa register. The operand for the variable register is shifted right by 8 as it is loaded. This really means that it is loaded into the least-significant end of the variable register along with zeroes in the top 8 bits. Due to the extra bit that can be generated in the product from the multiplier module, the base register length for the mantissas is 54 bits rather than 53. The different operand source paths contain circuitry to add an extra trailing zero as the least significant bit for 52-bit mantissas and to add a leading 1 (for the implicit 1) or a leading 0 (for one operand to be forced to 0) when necessary so that all

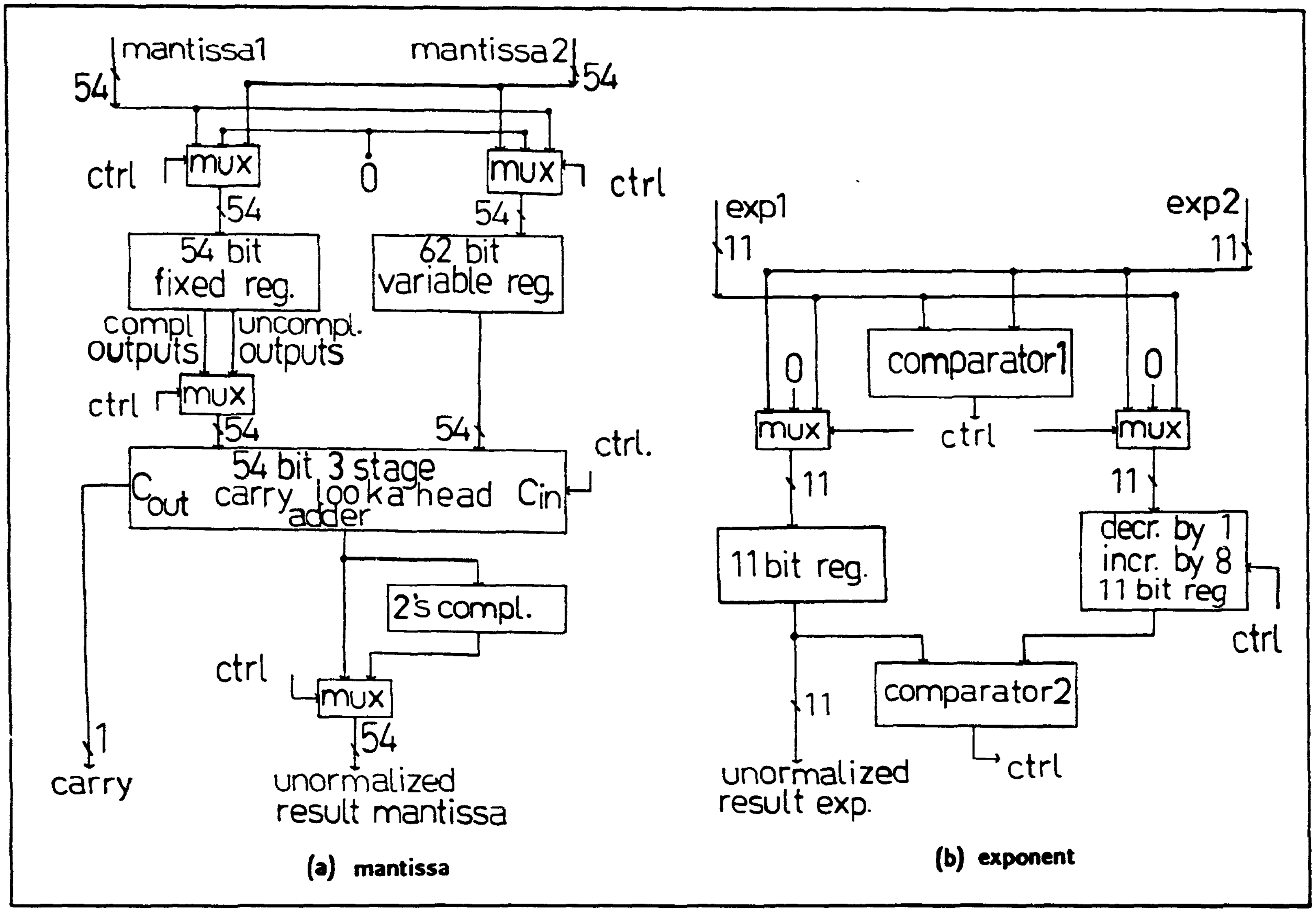


Figure B.5 The Floating-Point Align-and-Add Module

choices of operand1 and operand2 mantissas have the same 54-bit format.

Before addition of the mantissas is carried out, the binary points must be aligned to give equal exponents. This is done by incrementing the smaller exponent (in the variable 11-bit register) by 8 and shifting the corresponding mantissa (in the variable 62-bit register) right by 8 until the variable exponent value exceeds the fixed exponent value. The variable exponent is subsequently decremented by 1 and the variable mantissa is shifted left 1 bit until the two exponents are equal. The additional 8 bits of the variable mantissa register ensure that nothing is lost when the variable exponent value exceeds the fixed one while aligning. In Figure B.5(b), the connections of "comparator2" for the above operations can be seen.

If the two exponents differ by more than 53, the value of the sum is simply the larger of the two numbers. This is due to the 53-bit precision limit of the mantissa field. In such a case, the aligning procedure results in all zeroes in the variable mantissa register for the addition with the fixed mantissa register. The exponent of the sum is that of the fixed exponent register, i.e. that of the larger number, since that of the smaller will still be unequal after the alignment steps for the case where the difference in exponents exceeds 53. The worst case occurs when the two exponents differ by 49. This requires seven increment-by-8/shift-right-by-8 steps plus seven decrement-by-1/shift-left-by-1 steps for a total of 14 steps. However, the first step is performed automatically as the registers are loaded on clock 0 so the worst case is aligned at clock 14 (i.e. the result has stabilized by the end of clock 13).

The clock cycles 14 and 15 are used to perform the addition of the fixed mantissa register with the 54 most significant bits of the variable mantissa register. Treatment of the sign bits is as follows. If the two sign bits of the operands agree, the sign of the result is set to the common value and the complemented paths shown in Figure B.5(a) are inactive. If the signs of the operands differ, the fixed mantissa register is subtracted from the variable rather than added; this is done by having the exclusive-or of the sign bits control whether true or complemented outputs of the fixed register enter the adder and also whether a 0 or a 1 enters the carry-in of the adder. This permits addition of the two's complement of the fixed register, i.e. subtraction of this register from the variable one. If the adder output is negative and the operand signs differed, the fixed register mantissa was of larger magnitude so its sign is taken as the output sign and the negative sum is two's complemented to become positive again. If the adder output is positive and the operand signs differed, the variable register mantissa was of larger magnitude so its sign is taken as the output sign.

The carry bit shown in Figure B.5(a) is passed on to the normalization module as part of the resulting sum mantissa since a carry-out can occur in the case where the two operands have the same sign. If the two operands differ in sign, the carry bit is forced to zero since in this case it is a spurious bit.

The resulting sum of operand1 and operand2 stabilizes at the output of the align-and-add circuit at the end of clock 15. It is loaded into the next module, the normalization module, on the rising edge of clock 16.

The special 11-bit decrement-by-1/increment-by-8 register and the variable 62-bit register are actually implemented with internal multiplexers. These multiplexers allow loading of new data or loading of the previous contents altered by combinational circuitry in the feedback path. The circuitry in the feedback path accomplishes the required increment, decrement, or shift. When alignment of the two operands is detected, loading of the registers is inhibited so that the aligned contents are preserved for addition during cycles 14 and 15.

B.4 The Floating-Point Normalization Module

The floating-point normalization module is shown in Figure B.6. The input number to be normalized by this module is always obtained from the output of the align-and-add module. To normalize a floating-point number, the mantissa must be shifted and the exponent adjusted such that the left-most '1' in the mantissa appears just to the left of the binary point. In the diagram of Figure B.6, the binary point is located to the right of the most-significant bit in the 54-bit special shift register.

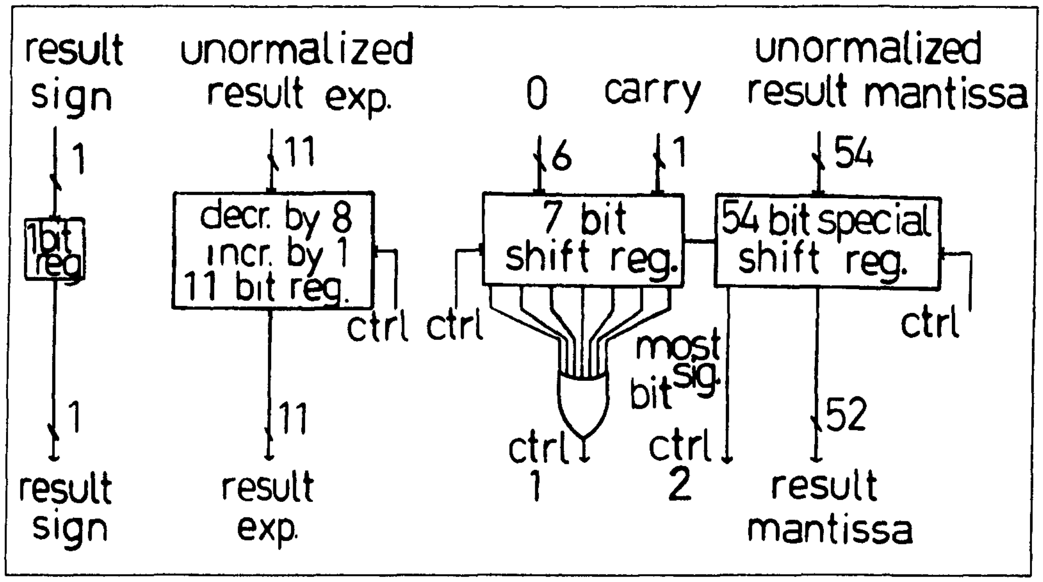


Figure B.6 The Floating-Point Normalization Module

The technique used for normalization is very similar to that used for alignment in the align-and-add module. The unormalized number is loaded into the 54-bit special shift register of Figure B.6 on the rising edge of clock 16. The mantissa is shifted left by 8 and the exponent is decreased by 8 until a '1' is detected to the left of the binary point. In terms of the circuit, this means that the steps of 8 continue until either the OR gate output (*ctrl1*) is '1' or the most-significant bit of the 54-bit shift register (*ctrl2*) is '1' or both. If at this point all the bits above the binary point are zero except the first (or in circuit terms if "*ctrl1*" is zero and "*ctrl2*" is '1'), then the normalization is complete and the process halts. Otherwise, the mantissa is shifted right by 1 and the exponent is increased in by 1 until this last condition is achieved. Zeroes are shifted in on either side as bits are vacated during the normalization. The extra 7-bit register prevents the loss of significant digits while normalizing.

The worst case occurs when the leading '1' starts off in bit 4 of the 54-bit special shift register (using the convention that the least-significant bit is counted as bit

0). For this case, 7 shift-left-by-8 steps are required to get the '1' above the binary point; then 7 shift-right-by-1 steps place the '1' exactly to the left of the binary point. A total of 14 steps are required, each taking one clock cycle, and none are performed automatically upon loading. The shifting requirements are detected and set up with combinational logic during the clock cycle and the operation is performed with a register load on the next rising clock edge. The special shift registers are implemented with internal multiplexers and logic in the feedback path as explained for the align-and-add module. Hence, the worst valid case is normalized by clock 31. If normalization is not achieved by clock 31, it is because there were only zeroes in the mantissa and this is detected as an underflow.

Once the completion of normalization is detected, further changes are inhibited and the result is preserved until the next clock 0 when it is loaded into the Y_{out} register shown in Figure B.1.

Note that due to the need for feedback between the output of the normalization module and the input of the align-and-add module in the fit-error computation, the normalization module is inactive during cycles 0 to 15 and the align-and-add module is inactive during cycles 16 to 31.

B.5 Treatment of Overflows and Underflows

Circuitry for detecting overflows and underflows was not included in the diagrams of the modules presented above so that these diagrams would not be rendered overly complex. The manner in which overflows and underflows are handled by the fit-error cell will be now be outlined.

In the IEEE floating-point standard, an overflow is denoted by a number with the largest possible exponent (all exponent bits set to 1) and an underflow is denoted by a number with the smallest possible exponent (all exponent bits set to 0). Since biased exponent notation is used in this standard and the fit-error cell uses double-precision format with 11 exponent bits, the smallest possible exponent is 0 (representing -1023) and the largest possible exponent is 2047 (representing +1024).

The approach to signalling overflows or underflows at the output is cumulative from one module to the next. Each module receives overflow/underflow status information from the preceding module The result of overflow/underflow detection in each module is logically ORed with the previous status and passed on to the next module so that overflow or underflow occurring at any stage of the computation causes overflow or underflow of the result. The Y_{out} shift register shown in Figure B.1 finally acts on the overflow/underflow status by setting the exponent field of the result to all 1's if an overflow was detected or to all 0's if an underflow was detected. Hence the overflow or underflow is encoded in the result data as required by the IEEE standard. The overflow/underflow status is reset at the beginning of each compute-interval of 256 clock pulses. Hence, a separate overflow/underflow detection is performed individually on each fit-error partial-sum output Y computed.

The detection of overflows and underflows in the individual modules basically involves checking the exponents of the incoming data and the generated results. The exponents are checked to see if they take on the extreme values (-1023, +1024) or if any of them wrap around during processing. A zero mantissa may be generated during addition in the align-and-add module; this also constitutes an underflow and is detected when the result is processed by the normalization module. If one of the operands entering the align-and-add module is detected as an underflow, that operand is replaced with a zero allowing the other number to emerge unaffected after the addition.

B.6 A Note on the Processing and Transfer of Operands in Two-Bit Slices

The four modules in the fit-error cell are modelled after the "stages" of the Larochelle convolution cell [Laro89]. In the Larochelle design, the image data input is processed 4 bits at a time. The number of constant coefficients handled by the fit-error cell is increased to six (these are the UV parameters) compared to the single convolution coefficient handled in the Larochelle cell. Hence, if the fit-error cell were to process the input 4 bits at a time, 6 X16 = 96 possibilities for the pre-computed product would have had to be dealt with compared to the 16 possibilities for the Larochelle design. This would require a large amount of silicon area in the storage and access system for the pre-computed UV products. Also, the speed advantage incurred by processing 4 rather than 2 bits of the image input at a time would be defeated in the case of the fit-error cell due to the feedback requirement. As mentioned in Chapter 3, the Larochelle convolution cell design is truly pipelined and successive stages operate concurrently with no idle intervals; in the fit-error cell, the feedback of operands forces idle periods in the operation of the modules. The feedback is from the output of the normalization module to the input of the align-and-add module and this feedback causes the two modules to operate as a single module with a processing time equal to the sum of the individual processing times. Hence, if 4 bits were processed at a time by the multiplier module, the product would be ready in 16 clock cycles. But the product must match up with the feedback from the previous addition during the accumulation of the fitted depth and this feedback requires 32 clock pulses to compute. Hence the multiplier would have to be idle for the last 16 of these 32 clock pulses and no speed improvement would be achieved.

The slow-down caused by the feedback in the fit-error cell and the increased area requirement in the storage and access system for pre-computed UV products were the two determining factors in the design choice to process two input-image bits at the same time in the multiplier module. Hence there is a basic periodicity of 32 clock pulses for each module. This is the time for the multiplier to compute a product and also the time for the combination of the align-and-add plus the normalize units to produce the feedback. Movement of data into and out of the fit-error cell is thus synchronized to the 32-clock-pulse time unit and all I/O data paths are thus two bits wide and deliver 64-bit operands in 32 clock pulses. However, to compute an output (which is a partial result Y), a cell requires 8 of these 32-clock-pulse time units. These time units were called "sub-intervals" in Chapter 3 and the sequence of 8 such sub-intervals was called a "compute-interval".

The sequencing of operations within the cell was discussed in Chapter 3 (with the aid of the sequence table of Figure 3.4) and need not be repeated. The goal here has been to explain the selection of the two-bit wide data path for the cell I/O and for the multiplier unit.