UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

REAL-TIME 3-D SCENE RECONSTRUCTION

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

ERIK ERNST PETRICH
Norman, Oklahoma
2016

REAL-TIME 3-D SCENE RECONSTRUCTION

A DISSERTATION APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

Dr. James J. Sluss Jr., Chair

Dr. Monte P. Tull

Dr. Thordur Runolfsson

Dr. Joseph P. Havlicek

Dr. Murad Özaydin

Dedication

For Donna, Todd, and Clover.

Acknowledgements

I would like to thank my committee members, Dr. James Sluss, Dr. Monte Tull, Dr. Thordur Runolfsson, Dr. Joseph Havlicek, and Dr. Murad Özaydin, for their support throughout the years. Not only have they given their time and useful feedback towards my research and the writing of this dissertation but also have been excellent teachers that increased my understanding of the world and sharpened the mental tools for my own studies. I am particularly grateful to Dr. Tull, who has been patiently supportive as my academic advisor in both the moments of joy as well as the moments of frustration, and also Dr. Sluss, who gave me the opportunity to discover how much I enjoy teaching.

I would also like to thank Dr. Larry Tarrant, who encouraged me to return to school after being away for so many years and was instrumental in helping me overcome some of the initial hurdles.

Table of Contents

List of Tables

List of Figures

Abstract

This dissertation describes a complete system that captures image data from multiple stereoscopic camera pairs and reconstructs a 3-D model of the imaged scene in real-time. To achieve real-time rates, the system is organized in a distributed hierarchical fashion to maximize parallelism and uses algorithms that, in many instances, are suitable for direct implementation in digital hardware rather than software on a general purpose computer. At the lowest level of the hierarchy, image data is acquired from a single camera and processed to compensate for lens distortion and to apply rectification in preparation for stereo image processing. At the next level, data from pairs of cameras is matched to compute a dense stereoscopic disparity map from which 3-D surfaces are inferred and a mesh model is constructed. Finally, at the top level all of the individual 3-D mesh models are merged into a single 3-D model. If desired, the camera image data can be applied to the resultant 3-D model as a texture and the model re-rendered from a virtual camera viewpoint.

Previous 3-D research focuses on individual steps in this process (lens distortion correction, image rectification, stereoscopic disparity computation, and model building). This dissertation considers them instead in the context of a complete end-to-end system. Traditional approaches to model building begin with an unstructured "point cloud" that is neutral with respect to how the data was acquired; this allows model building to be studied independent of data acquisition but may miss some opportunities available in a more tightly coupled interface. By taking a broader view of the problems faced by the entire system, a novel algorithm for 3-D model building has been developed that takes advantage of the organization in the dense stereoscopic disparity map to efficiently

build its model. The core of this novel algorithm is a method of evaluating linear regression error to fit a series of line segments to data points in a way that can be efficiently implemented directly in hardware.

Chapter 1: Introduction

This dissertation describes a design and implementation of an image capture system capable of reconstructing an observed dynamic three dimensional scene. The challenge inherent in this proposed system over previous systems is to operate at “real-time” video frame rates using relatively inexpensive hardware. The constraints implied by the term "real-time" can widely vary, but within this dissertation the specific goal was to acquire and process video a rate of at least 30 frames per second at a 640 by 480 resolution, with a latency of only a few frames. At this modest resolution, the goal for the reconstructed model of the scene is not necessarily high precision but instead one with an image quality that is suitable for casual viewing by a human observer.

The intended application for such a system is 3-D telepresence; once a 3-D model of the scene has been reconstructed, it can be rerendered in 2-D from some virtual camera position and orientation. At the smaller scale, this could be used to draw teleconference participants into a virtual environment with fewer of the artificial barriers of traditional teleconferencing. At the other end of the extreme, with many cameras and with higher resolution, a viewer could watch a sporting event from a position that they dynamically choose.

Within the model building stage (covered in detail in Chapter 6), there are two specific novel developments:

- A method of using least-squares linear regression to segment data into a series of line segments, with each data point evaluated only once.
- A method of processing a dense stereoscopic depth map into a 3-D model that takes advantage of spatial locality in the structure of the depth map.

To achieve the desired frame rate several strategies have been adopted throughout all of the stages of the system:

- Data should be processed and transferred in parallel.
- Perform time critical calculations and data transfers with dedicated special-purpose hardware rather than with a general-purpose microprocessor and software.
- Organize the calculations to use faster operations and data types.
- Complex operations that cannot be avoided entirely should be broken up into a series of simpler operations and pipelined.

Wherever data can be independently processed, hardware parallelization trades what would have been *n* operations spread over time to *n* operations spread over space. In this ideal case, the only limitation is the cost of the additional processing hardware that the operations are spread across. In practice, however, there are often points where data cannot be independently processed or data sets need to be merged. At some point, the cost of distributing data to parallel operations or merging their results exceeds the benefit of the parallelization.

Microprocessors are designed to allow flexible algorithms and quick implementations of these algorithms. The architecture of the microprocessor is designed to give good results for a wide variety of operations, but at the expense of not being optimized for any specific task. For more specialized tasks, one can use correspondingly specialized compute units, such as Digital Signal Processors and Graphical Processing Units, or even implement a fully customized application specific compute unit. However, in all cases to efficiently implement a given algorithm, the algorithm must be a good fit with the capabilities of the underlying hardware. In this system, as much as possible, the algorithms have been chosen or designed so that computations could be implemented directly in hardware with a pipelined data flow model.

Some operations are of greater complexity than others and require more time and/or hardware to implement. For example, multiplication requires the computation of partial products as well as the sum of all these partial products. Therefore, multiplication is considered more computationally expensive than the addition of two numbers. When calculations can be expressed in multiple algebraic forms, the calculation should be organized to minimize this computational expense.

The chapters that follow begin with the relevant literature and provide a background to the concepts of this system in Chapter 2. This is followed in Chapter 3 by a description of the camera calibration process, which only needs to be performed once when the system is initially set up. The operation of the three stages of the system are covered in Chapters 4 through 6. The initial stage (Chapter 4) deals with image acquisition from a single camera and corrects for some shortcomings in the non-ideal nature of real cameras (and particularly, low-cost cameras). The second stage (Chapter 5) processes images from pairs of adjacent cameras to determine depth at each point in the images via stereoscopic disparity. The third and final stage (Chapter 6) process the depth data into a 3-D mesh model that the original images can be applied to as decals to create a realistic 3-D model of the scene. Finally, Chapter 7 summarizes the results and suggests directions for future research.

Chapter 2: Literature / Background

Previous 3-D model building research focuses primarily on reconstruction of a static scene (often as simple as a single object) using various ranging technologies. For example, in structured light analysis a sequence of light/dark patterns is projected [20], or laser scanned [14], onto the scene. Variation in objects' surface contours distort the projected patterns. With knowledge of the patterns, both pre- and post-distortion, and relative positions of the projector and camera, the surfaces can be inferred and the scene reconstructed. Since the scene is static, the process can afford to spend a relatively long period of time sequencing through the various patterns and gathering the resulting data. Furthermore, to reconstruct both the front and back of the objects in the scene, the scene usually undergoes a rotation relative to the camera and light source.

Alternatively, stereoscopic analysis [12] compares corresponding points in two images taken from different points of view to triangulate the position of the corresponding surface. The two images can be gathered with a pair of cameras or a single camera in motion. Again, to reconstruct both the front and back of the objects in the scene, the scene usually undergoes a rotation relative to the camera(s), akin to roll-out photography of cylindrically symmetric objects [18].

Advances in technology have made real-time stereoscopic analysis feasible. The necessary computations have been broken up and implemented in parallel FPGAs [25] or specialty parallel core processors [7]. To date, these real-time implementations have focused solely on gathering range information rather than complete scene reconstruction.

Once the range data has been acquired, a model of the scene can be reconstructed. These models can be represented either volumetrically [4] [22] or in terms of a mesh surface [24]. In a volumetric representation, the space is subdivided into small discrete pieces that indicate occupancy or vacancy of an object. Range information from each of the views is used to mark vacancies in the space, and the final scene model inferred from the pieces not marked as vacant. Surface models are built from a mesh of polygons, usually triangles or quadrilaterals, defined by their vertices. Range information from multiple views may be combined to form a "point cloud" and the mesh formed by using points near each other to define polygons. Alternatively, meshes from each view can be formed on a regular grid, leaving discontinuities in the grid where the range information suggests a sudden change in depth. Then these meshes are combined, eliminating redundant polygons and filling in any small gaps to generate a final model.

With multiple cameras, real-time stereoscope analysis has the potential for dynamic scene reconstruction if the range information can also be processed into scene surface in real-time. Naive reconstruction of the entire scene anew many times per second would handle the requirements of a dynamic scene. However, when changes in the scene are slow relative to the camera frame rate, knowledge of the scene from previous frames could potentially simplify the processing of the current frame. Inter-frame knowledge has also been used to smooth out positional jitter arising from noise in the acquired images [6].

Homogeneous Coordinates

Although the coordinates on an image plane are 2-dimensional, it is often convenient to represent a coordinate as a 3 element vector (or the corresponding 3 element column matrix). The first two elements represent the position on the 2-dimensional axes with the third element representing a scaling factor. If scaling is unneeded, the scaling factor is simply the value 1.

$$\begin{bmatrix} x \\ y \end{bmatrix} \Leftrightarrow \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} \tag{2.1}$$

Likewise, coordinates in the 3-dimensional world are often represented as a 4 element vector (or 4 element column matrix), with again the last element simply a scaling factor for the first three.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Leftrightarrow \begin{bmatrix} sx \\ sy \\ sz \\ s \end{bmatrix} \tag{2.2}$$

Coordinates in these forms that use the last element as a scaling factor are known as homogeneous coordinates and allow a wider range of arithmetic operations to be described in terms of matrix multiplication. Furthermore, these forms are able to reflect a key property of projective geometry in that the apparent size of objects is scaled by their distance from the viewer.

Camera Model

The basic operation of a camera is best described by a *pinhole camera model* (Figure 1) [10]. Light from a point **X** in 3-D space passes through the pinhole point **C** to point **x** on the 2-D image plane. The direction the camera is pointing, the principal axis, coincides with the *Z* axis. The principal axis intersects the image plane at the principal point **p** which is taken as the origin for the image plane axes. The distance between **C** and **p** is the focal length *f*. In a physical camera, the image plane is necessarily behind the pinhole and thus the 2-D axes are reversed from the corresponding 3-D axes. For convenience to avoid axis reversal, a virtual image plane can be imagined at a position an equal distance in front of the pinhole (Figure 2); coordinates of **X** and **x** remain unaltered.

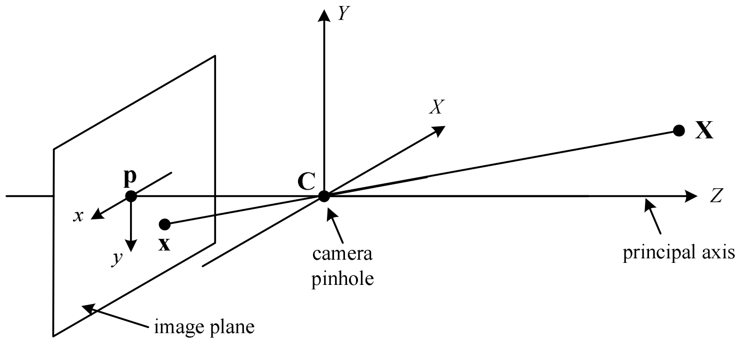

Figure 1. Pinhole camera model (adapted from [10] fig. 6.1)

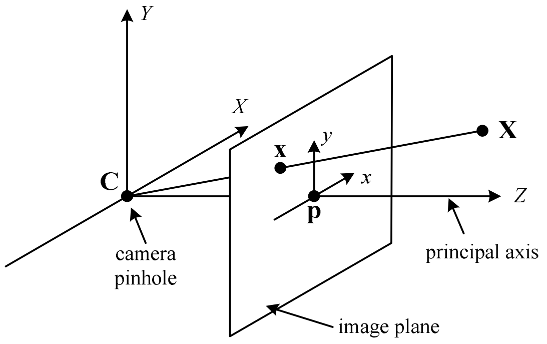

Figure 2. Alternate pinhole camera model (adapted from [10] fig. 6.1).

For simple image processing, basing the world coordinate system relative to the camera may be sufficient. However, with multiple cameras or multiple points of view there needs to be a way to relate each of the image coordinates to a unified world coordinate system. This can be done with the a linear transformation below (Cyganek, 2009).

$$\mathbf{x} = \mathbf{M}\mathbf{X} \tag{2.3}$$

$$\mathbf{M} = \mathbf{M}_i \mathbf{M}_e \tag{2.4}$$

$$\mathbf{M}_i = \begin{bmatrix} \frac{f}{h_x} & 0 & o_x \\ 0 & \frac{f}{h_y} & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$\mathbf{M}_e = \begin{bmatrix} \mathbf{R}_1 & -\mathbf{R}_1\mathbf{T} \\ \mathbf{R}_2 & -\mathbf{R}_2\mathbf{T} \\ \mathbf{R}_3 & -\mathbf{R}_3\mathbf{T} \end{bmatrix} \tag{2.6}$$

The matrix $\mathbf{M}_i$ defines some intrinsic camera parameters that allow some adjustment to the coordinate system for its image plane. In the case of a digital camera, parameters h_x and h_y specify the width and height of a discrete pixel on the image plane so that the coordinates are in terms of pixel units. The parameters o_x and o_y specify an offset for the origin of the image plane coordinates from the principal point **p**.

The matrix $\mathbf{M}_e$ defines extrinsic camera parameters that define how the camera's coordinate system relate to the world coordinate system. The translational motion of the coordinate systems' origins is specified as the column vector **T** and the rotation of the coordinate systems' axes is specified as the orthogonal rotation matrix **R**; in (2.6) $\mathbf{R}_i$ refers to the *i*th row of **R**. Since $\mathbf{M}_e$ is a 3 by 4 matrix, the world coordinate **X** must be in homogeneous coordinate form as a 4 element column vector (this allows the translational motion to be combined with the other matrix multiplication operations rather than a separate addition). If the camera's coordinate system is used as the world coordinate system, then an $\mathbf{M}_e$ without rotation or translation can be used:

$$\mathbf{M}_{e0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad (2.7)$$

Focus

The pinhole camera model is idealized; the pinhole functions as a lens with an infinitely small aperture. For an actual camera, using either a true pinhole or some more sophisticated lens assembly, the aperture is necessarily finite. Thus, **C** is no longer a singular point, which in turn means that 3-D point **X** no longer corresponds to a single 2-D point on the image plane but instead a region around the idealized point **x** known as the *circle of confusion*. When the camera and lens are perfectly focused to the depth of some point in the scene, the corresponding circle of confusion shrinks to a point. Other points closer or farther away have circles of confusion with a non-zero diameter but may still be considered "in focus" if the diameter is small enough. For some arbitrary maximum allowed diameter, one can solve [16] for the range of depths, d_{near} to d_{far}, meeting this restriction:

$$\begin{aligned} c &= \text{maximum circle of confusion diameter} \\ d &= \text{depth lens is perfectly focused to} \\ f &= \text{lens focal length} \\ n &= \text{lens numerical aperture ("f-stop")} \end{aligned}$$

$$h = \frac{f^2}{cn} + f \tag{2.8}$$

$$d_{near} = \begin{cases} 0 & \text{for } d \le 2f - h \\ \dfrac{dh - f^2}{h + d - 2f} & \text{for } d > 2f - h \end{cases} \tag{2.9}$$

$$d_{far} = \begin{cases} \dfrac{dh - 2fd + f^2}{h - d} & \text{for } 0 \le d < h \\ \infty & \text{for } d \ge h \end{cases} \tag{2.10}$$

The depth h is called the hyperfocal distance; the amount of the scene in focus approaches its maximum as d approaches the hyperfocal distance. To improve the depth of field, the lens aperture can be reduced (decrease n), the camera and objects of interest can be moved further apart (increase *d*), the focal length shortened (decrease *f*), or some combination of thereof.

Reducing the lens aperture requires increasing either the exposure time of the camera or increasing the lighting of the scene (since less light can pass through the lens), but otherwise can increase, up to a limit, the depth of field without changing the image scaling. The limitation arises because (2.9)-(2.10) are based on a particle model of light and neglect its wave nature; as the aperture approaches the scale of the light's wavelength, these equations become inapplicable and diffraction patterns become significant enough to actually increase the size of the circle of confusion.

If there is sufficient room, the camera can be moved back from the objects of interest. However, the last alternative, reducing the focal length of the lens, has the advantage of working in all cases and is attractive as a general solution. Unfortunately, short focal length lenses (also known as wide-angle lenses) tend to also suffer increased distortion.

Lens Distortion

Imperfections in the manufacturing of a lens and misalignments of individual lens elements in a multi-lens assembly lead to unintended distortion in the resulting image; some amount of distortion is unavoidable. To minimize the distortion, one can pay for more expensive lenses that are made to a higher degree of precision. Alternatively, one can cancel out the distortion by applying a second distortion that is the inverse of the first.

For a reasonably well made lens assembly the distortion occurs primarily along the radial axis, symmetric around the principal point.. For a pixel at radius $\tilde{r}$ from the optical axis in the ideal image, the distortion moves it to an apparent radius r due to Seidel aberrations [8]:

$$\tilde{r} = r + a_1 r^3 + a_2 r^5 + a_3 r^7 + \dots \tag{2.11}$$

The constants a_i are related to the curvature of the lenses in the assembly and their relative positioning, but normally are computed through estimation using pairs of r and $\tilde{r}$ associated with a known calibration target rather than physical measurement.

Also possible is distortion that is perpendicular to the radial axis. This is commonly due to the image sensor being slightly off angle from being exactly perpendicular to the principal axis. However, this tangental distortion is normally a much smaller effect than the radial distortion and so is not considered further in this dissertation.

Suppose the actual image is that of a regular grid of straight lines (Figure 3). When $\tilde{r} > r$, the image is said to suffer from pincushion distortion (Figure 4). When $\tilde{r} < r$, the image is said to suffer from barrel distortion (Figure 5). It is also possible for $\tilde{r} < r$ in some portion of the image and $\tilde{r} > r$ in another portion of the image, but this occurs less commonly. In any case, all of these distortions can be modeled by the above equation.

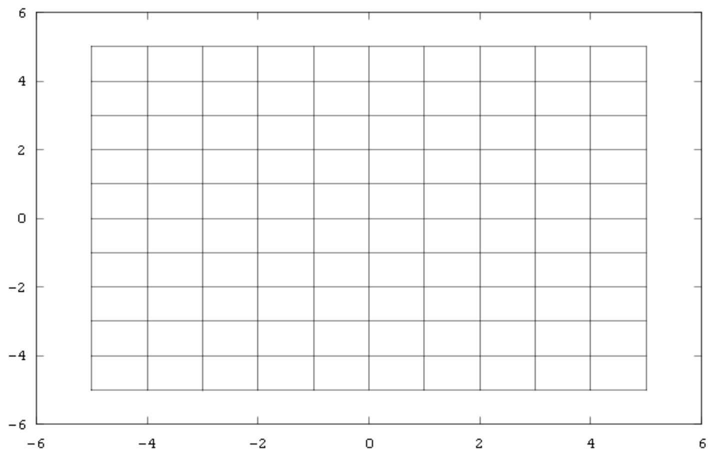

Figure 3. Image of a grid of straight lines.

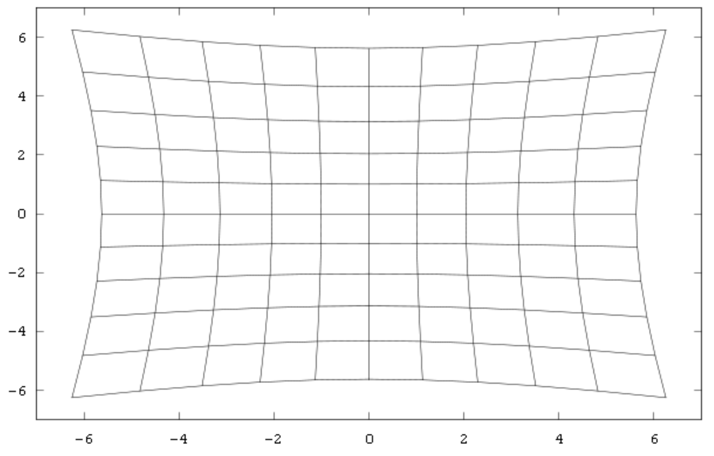

Figure 4. Example of pincushion distortion; straight lines bend inward.

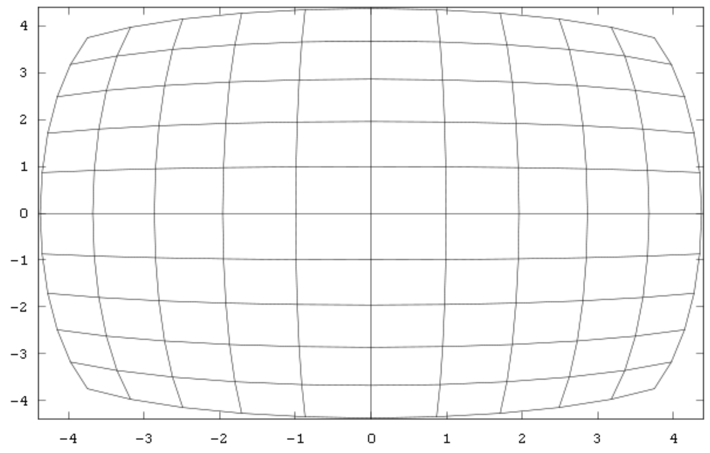

Figure 5. Example of barrel distortion; straight lines bend outward.

The higher order coefficients become progressively smaller and so at some point can be considered negligible. The coefficients are determined as part of the camera calibration process, often by locating straight line edges in either a real-world scene or calibration target and measuring the curvature of those edges in the acquired image [5] [3].

By similar triangles (see Figure 6), the coordinates *x*, *y* are distorted to

$$\tilde{x} = \frac{x\tilde{r}}{r} \tag{2.12}$$

$$\tilde{y} = \frac{y\tilde{r}}{r} \tag{2.13}$$

with $$r = \sqrt{x^2 + y^2} \tag{2.14}$$

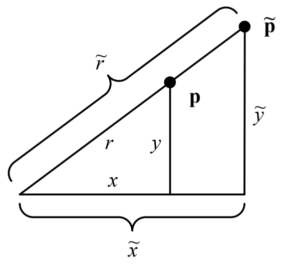

Figure 6. Radial distortion of point p.

After substitution and simplification

$$\tilde{x} = x\left(1 + a_1 r^2 + a_2 r^4 + a_3 r^6 + \ldots\right) \tag{2.15}$$

$$\tilde{y} = y\left(1 + a_1 r^2 + a_2 r^4 + a_3 r^6 + \ldots\right) \tag{2.16}$$

In this form, only even powers of r are needed, so the computationally expensive square root operation can be avoided. The apparent *x,y* coordinates are therefore given by

$$\tilde{x} = x\left(1 + a_1 r^2 + a_2 \left(r^2\right)^2 + a_3 \left(r^2\right)^3 + \ldots\right) \tag{2.17}$$

$$\tilde{y} = y\left(1 + a_1 r^2 + a_2 \left(r^2\right)^2 + a_3 \left(r^2\right)^3 + \ldots\right) \tag{2.18}$$

$$r^2 = x^2 + y^2 \tag{2.19}$$

Iterating over all the x, y coordinates of an image plane, one can compute all the corresponding positions $\tilde{x}, \tilde{y}$ within the distorted image using the above equations. By copying the value of the pixel at the distorted position to the pixel at the undistorted position, the overall undistorted image can be reconstructed.

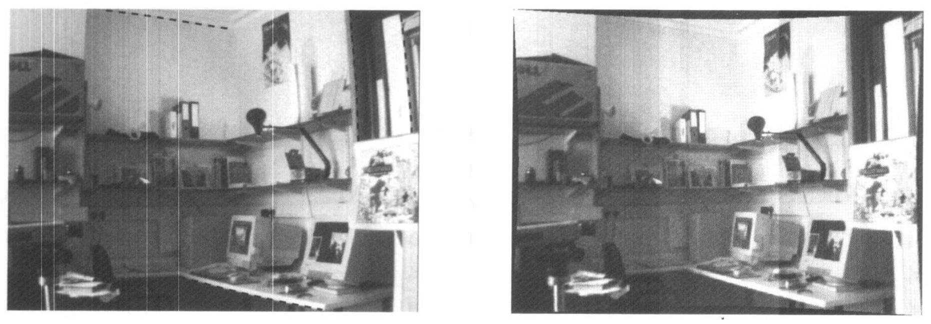

Figure 7. Lens distortion correction. Left: Original distorted image with dashed reference "straight" lines. Right: Distortion corrected image. ([10] fig. 7.6)

Stereoscopic Vision

Suppose a point **X** is visible from two adjacent cameras (see Figure 8), at point $\mathbf{x_1}$ in camera $\mathbf{C_1}$'s image plane and at point $\mathbf{x_2}$ in camera $\mathbf{C_2}$'s image plane. The plane containing the points $\mathbf{C_1}$, $\mathbf{C_2}$, and **X** is called the *epipolar plane*. If the epipolar plane is uniquely determined and the relative positions of $\mathbf{C_1}$, $\mathbf{C_2}$, $\mathbf{x_1}$, and $\mathbf{x_2}$ are known in 3-D space, then the relative position of **X** can be uniquely determined through triangulation.

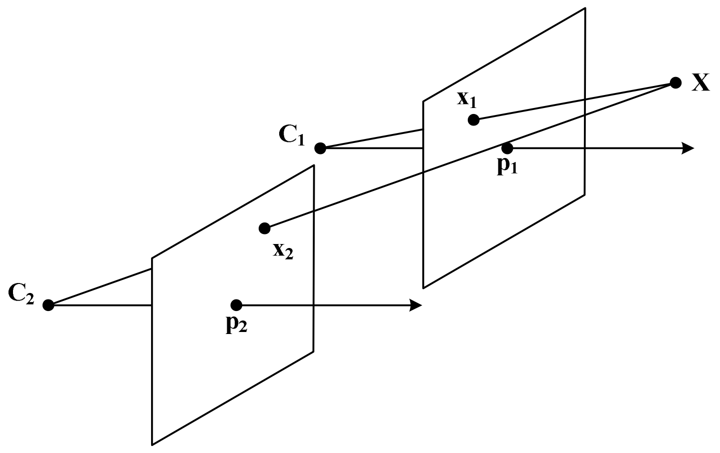

Figure 8. A pair of adjacent cameras view point X

Because $\mathbf{x_1}$ is collinear with $\mathbf{C_1}$ and **X**, the epipolar plane can equivalently be defined as the plane containing the points $\mathbf{C_1}$, $\mathbf{C_2}$, and $\mathbf{x_1}$; this is more practical since the position of **X** is often not known in advance. Since any point that is collinear with $\mathbf{C_1}$ and **X** (other than the degenerate case of **C_1** itself), when taken with the points **C_1** and **C_2**, will define this same epipolar plane, the intersection of the epipolar plane with camera **C_2**'s image plane forms a line, the *epipolar line*, that represents the image of these collinear points from camera **C_2**'s point of view (see Figure 9).

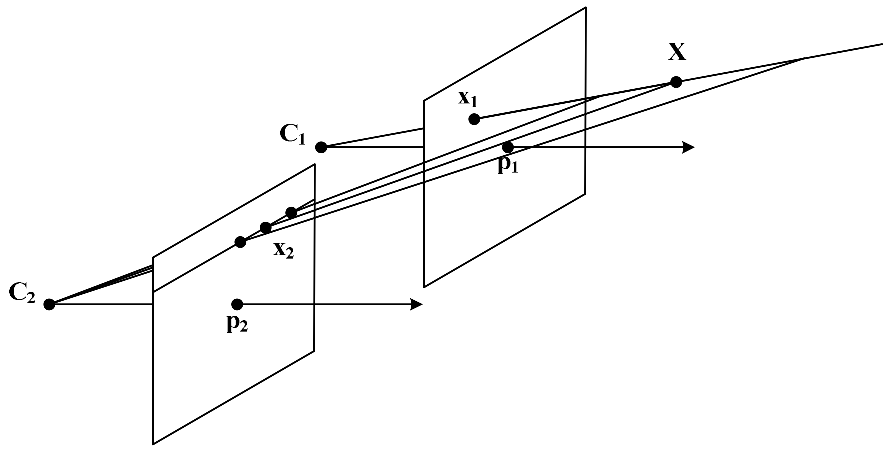

Figure 9. Varying potential depths of X cause its image x_2 to vary along the epipolar line in camera C_2's image plane.

The epipolar line is useful because the position of **x_2** is also not usually known in advance. Instead, camera **C_2**'s image plane must be searched for a region that sufficiently matches the region around **x_1** in camera **C_1**'s image plane. The constraint imposed by the epipolar line reduces the potential search space from a quadratic size to a linear size.

Fundamental Matrix

For particular stereoscopic camera pair there exists a matrix **F**, the *fundamental matrix*, such that

$$\mathbf{x}_2^T \mathbf{F} \mathbf{x}_1 = 0 \tag{2.20}$$

for any pairs of corresponding points $\mathbf{x}_1$ and $\mathbf{x}_2$ given in homogeneous pixel coordinate form. The fundamental matrix encapsulates the relative position and orientation of the camera pair. It also has the useful property that the nullspace of $\mathbf{F}\mathbf{x}_1$ is the epipolar line corresponding to $\mathbf{x}_1$.

Image Rectification

Although the camera pair shown in Figure 8 can be in a wide range of relative positions and orientations that allow the position of **X** to be determined, some are more favorably solved than others. If the cameras are oriented such that their image planes are parallel to the line joining $\mathbf{C_1}$ and $\mathbf{C_2}$, then there will exist a set of parallel lines in the image plane of camera $\mathbf{C_1}$ that all of the points in one of these lines will share the same epipolar line in the image plane of camera $\mathbf{C_2}$.

Each of these parallel lines in conjunction with their corresponding epipolar lines can be processed independently and thus potentially simultaneously. With the proper orientation of the camera about its principal axis, both the set of parallel lines in camera $\mathbf{C_1}$ and the corresponding epipolar lines in camera $\mathbf{C_2}$ can be parallel to a chosen axis (typically x) within the image planes. This would allow the hardware addressing of the image plane data to be partitioned easily so that the potentially simultaneous processing can become actual simultaneous processing.

If the orientation of the cameras do not meet the above conditions, it is still possible to compute the image of a virtual image plane that does meet the above conditions based on the image in the actual image plane; this process is known as *image rectification*.

Image rectification may still be useful even if the cameras are intentionally oriented to meet these conditions, due to limitations in the tolerances achievable in the manufacturing process. A calibration process (discussed in Chapter 3) can be used to determine the actual relative orientation of the cameras and thus compute projective transformations for the two actual images to generate virtual images of an idealized orientation [9].

While computing the appropriate projective transformation matrix **H** for the rectification is somewhat complex, the transformation itself is straightforward. For a coordinate **x** in homogenous form on the actual image plane, the corresponding coordinate $\hat{\mathbf{x}}$ (also in homogenous form) on the idealized image plane is given by:

$$\hat{\mathbf{x}} = \mathbf{H}\mathbf{x} \tag{2.21}$$

Expanding these three matrices into their component elements:

$$\begin{bmatrix} \hat{s}\hat{x} \\ \hat{s}\hat{y} \\ \hat{s} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.22}$$

Applying the matrix multiplication and normalizing the result:

$$\hat{\mathbf{x}} = \begin{bmatrix} \dfrac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \\ \dfrac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \\ 1 \end{bmatrix} \tag{2.23}$$

Image Remapping

Both the lens distortion correction and image rectification involve the mapping of pixels from some positions in a source image to new positions in a destination image. However, the mapping functions are continuous while the pixels occupy spatially discrete positions. Thus, in most cases the mapping function will return a position that will partially overlap four possible discrete pixel locations. Compounding this problem, pixels that are adjacent in the source image may not be exactly adjacent in the destination image. Instead, they might slightly overlap or even leave a gap between their positions in the destination.

A partial solution is to use an inverse mapping function that relates positions of pixels in the destination back to the corresponding position in the source. This inverse mapping avoids the problem of pixel gaps in the destination image because every pixel position has a mapping back to some position in the source. There may still be gaps in terms on which pixels are used in the source image (some source pixels may be left unused), but this would only appear as a loss of detail and is preferable to the alternative of completely missing pixel data that would occur from pixel gaps in the destination image.

Using the inverse mapping does not solve the problem of the discrete versus continuous pixel positions. The simplest solution is to round the continuous position to the nearest discrete position. However, this rounding would introduce a positional error of up to half a pixel position which in turn reduces the reliability of the correspondence matching and increases the possible error in the depth calculations.

A better solution to the problem of the continuous positions is to use some interpolation function to estimate the value between the discrete pixel positions. One of the simplest interpolations is bi-linear interpolation, where pixel value is assumed to linearly vary in the horizontal and vertical axes between the four closest discrete pixel positions.

$$u_f = u - \lfloor u \rfloor \tag{2.24}$$

$$v_f = v - \lfloor v \rfloor \tag{2.25}$$

$$\begin{aligned} P(u,v) = &\left(1-v_f\right)\left(\left(1-u_f\right)P\left(\lfloor u \rfloor, \lfloor v \rfloor\right) + u_f P\left(\lfloor u \rfloor + 1, \lfloor v \rfloor\right)\right) \\ &+ v_f\left(\left(1-u_f\right)P\left(\lfloor u \rfloor, \lfloor v \rfloor + 1\right) + u_f P\left(\lfloor u \rfloor + 1, \lfloor v \rfloor + 1\right)\right) \end{aligned} \tag{2.26}$$

In equation (2.26), the pixel's value is determined by the values of the pixels it overlaps, each weighted by the area of overlap (see Figure 10).

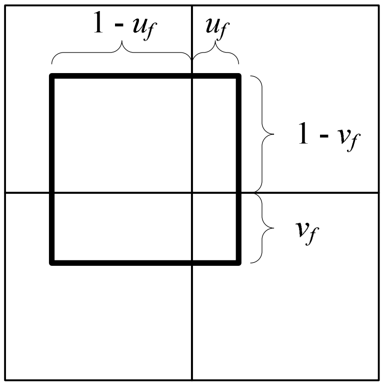

Figure 10. A pixel offset from the usual integer grid coordinates

Disparity and Depth

From a pair of rectified images the depth *z* from the cameras to point **X** is given by [12]:

$$z = \frac{bf}{d} \tag{2.27}$$

$$b = \left\| \mathbf{C}_1 - \mathbf{C}_2 \right\| \tag{2.28}$$

$$d = \left| x_1 - x_2 \right| \tag{2.29}$$

where x_1 and x_2 are the components of $\mathbf{x_1}$ and $\mathbf{x_2}$ in the image plane coordinate system that correspond to the axis parallel to the epipolar lines and f is the effective focal length of the cameras after rectification. Since for any particular configuration of cameras b and f are constant, the depth z and disparity d are always inversely proportional. With the depth known, the coordinate $\mathbf{X}$ can be recovered relative to either camera:

$$\mathbf{X} = \mathbf{C}_1 + \frac{z}{f}\left(\mathbf{x}_1 - \mathbf{C}_1\right) \tag{2.30}$$

$$\mathbf{X} = \mathbf{C}_2 + \frac{z}{f}\left(\mathbf{x}_2 - \mathbf{C}_2\right) \tag{2.31}$$

The set of all such points computed for $\mathbf{X}$ for the set of corresponding points found in the two image planes is known as the *point cloud*.

Scene Reconstruction

Once the point clouds have been determined for each of the camera pairs, an overall model of the scene can be reconstructed. A top-down view of a sample object surrounded by four camera pairs is shown below in Figure 11; while assumed present, the third dimension is omitted for ease of illustration.

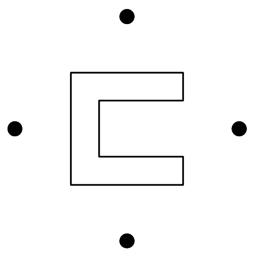

Figure 11. Top down view of an example scene with cameras represented by circles

Volumetric Models

In a volumetric model, the model is represented by a three dimensional array in which the array elements indicated the presence or absence of an object at the corresponding point in the world domain. If corresponding points are visible in both images of a camera pair, then there can be no occluding object between the cameras and the corresponding 3-D world point. Therefore, all array elements representing positions between the camera pair and the 3-D world point can be marked as unoccupied (see Figure 12).

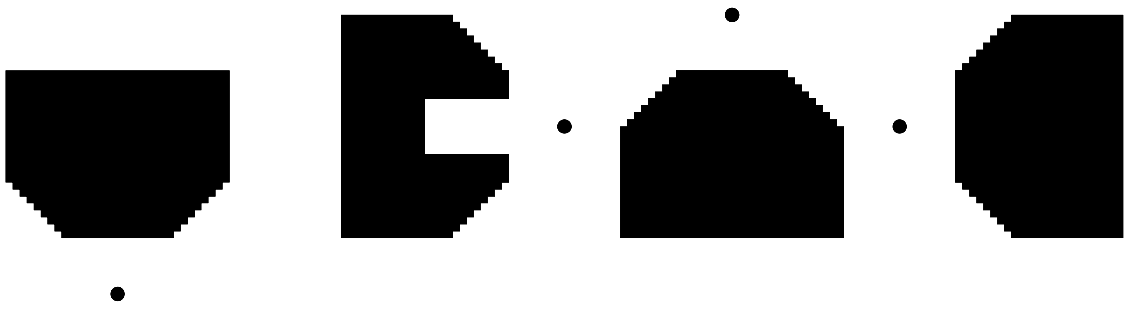

Figure 12. Binary volumetric model formed from each camera's view.

For binary model images, if an object's presence is denoted in the array element by a 1 and its absence denoted by a 0, then a bitwise AND operator can be applied to successive models formed from each view to generate a final composite model (Figure 13). Alternatively, the array elements may be represented as a continuous probability of the presence of an object, to account for the estimated error in the position of a 3-D world point.

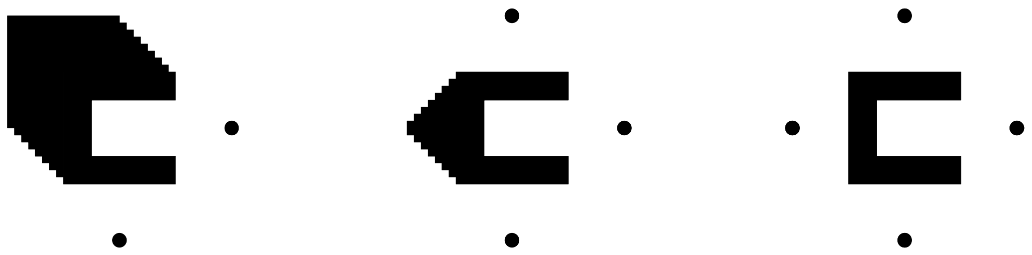
Figure 13. Combining each successive view's volumetric model to form final model.

Surface Mesh Models

In a surface mesh model, the point clouds are assumed to be samples from a relatively smooth and generally continuous surface. This problem is ill-posed meaning there are many solutions to finding a function that defines this surface depending on the working definitions of "relatively smooth" and "generally continuous" [1]. To build the composite model, either all of the point clouds from all of the views (Figure 14) are combined into a global point cloud and then processed to form a surface mesh model (Figure 15) or the individual points clouds are formed into surface meshes (Figure 16) and then these meshes are combined (Figure 17).

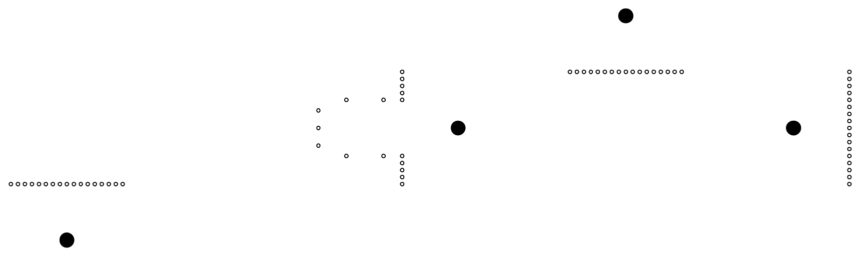
Figure 14. Point clouds formed from each camera's view.

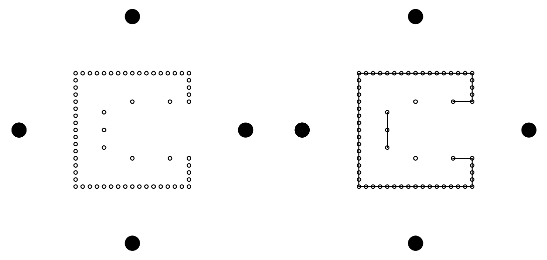

Figure 15. Composite point cloud (left) processed to form surface model (right).

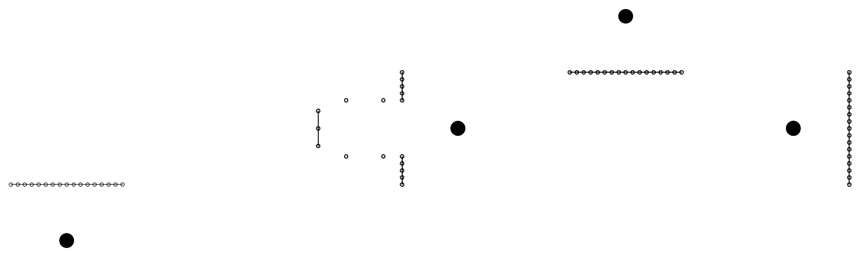

Figure 16. Each view's point cloud processed to form surface models.

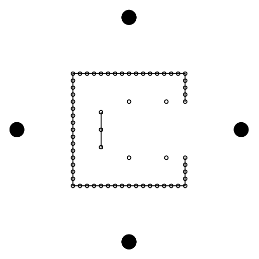

Figure 17. Combining each view's surface model to form final model.

Forming surface meshes from individual point clouds has the advantage that sudden changes in depth relative to the associated camera offer strong hints to find actual discontinuities, whereas in a global point cloud continuity is assumed by close proximity. However, combining multiple surface meshes formed from the individual point clouds may result in redundant or overlapping meshes that may need trimming. If there are slight errors in the mesh positions, the combined meshes may also have small gaps between them that need filling in.

The novel method of model building outlined in Chapter 6 is similar in spirit to this second approach of forming mesh models from individual point clouds and then merging the resulting meshes to form a unified model. However, rather than distilling the range data from disparities in the 2-D images into a pure point cloud form and then processing the point cloud, the spatial relationships present in the original 2-D images are retained and used to guide the mesh formation.

Chapter 3: System Overview, Calibration, and Camera Limitations

System Overview

Image processing is usually a computationally intensive task; image processing in real-time is even more so. To keep the computational requirements and bandwidth reasonable, the system is organized so that operations can be parallelized. While the algorithms are initially implemented in software, many are structured so that they can easily be performed by FPGA hardware for improved performance. Figure 18 shows the top level view of the system organization.

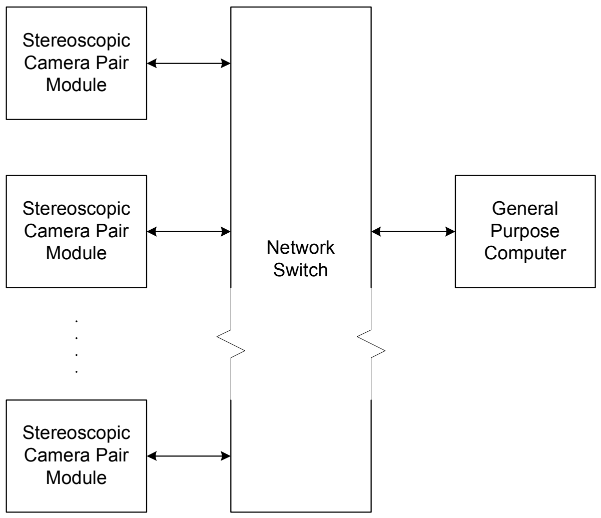
Stereoscopic Camera Pair Module
Stereoscopic Camera Pair Module
Stereoscopic Camera Pair Module
Network Switch
General Purpose Computer

Figure 18. Top-level system organization

The Stereoscopic Camera Pair Module (SCPM, see Figure 19) is much more than just a pair of cameras; it also includes all of the hardware and software needed to process the images from the module's camera pair into a 3-D model of the partial scene visible to those cameras.

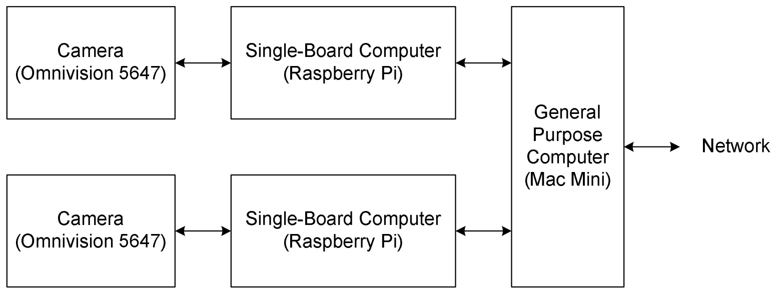

Figure 19. Prototype stereoscopic camera pair module (SCPM) organization

The Omnivision 5647 is a full color, medium resolution (5 megapixel), digital camera. While the color and resolution are beyond the needs of a basic proof-of-concept implementation, the Raspberry Pi single-board computer [19] is designed to work with this specific camera and tightly integrates the camera interface with the combined CPU/GPU. The camera hardware can combine the values for adjacent pixels in the array to produce an image with lower resolution than the native resolution of the sensor array as well as reducing the sensor noise level. Although this could also be done by the CPU or GPU, handling it inside the camera itself frees up processing power that could be used for more complex tasks and reduces the bandwidth needed to manage the camera interface.

The Raspberry Pi is a powerful low-cost single-board computer based on the Broadcom BCM2835 System-on-chip. This device integrates a 700 MHz ARM1176JZFS CPU, a Videocore 4 GPU, 512 MB of RAM, and various peripheral interfaces including the dedicated camera interface and a USB 2.0 interface. This USB interface connects to an SMSC LAN9512 combination USB 2.0 hub and 10/100 Mb Ethernet controller.

The image data from the camera is processed by the GPU before being passed to the CPU. The transformation needed for image rectification fits well with any GPU designed to handle accelerated 3-D operations, including the Videocore 4 GPU. The lens distortion correction probably could be handled by the GPU, although this is less certain because it depends on the details of its architecture. Unfortunately, programming information for the Videocore 4 GPU is not publicly available and its firmware is provided only as a binary image without source code. The published interface to the camera only supports some very basic transformations (rotations in 90 degree increments, flipping the image along the vertical or horizontal axes, and some limited rescaling options) and some predefined artistic special effects. Thus, the lens distortion correction and image rectification is left for the ARM CPU to process. Implementation details of these steps will be covered in Chapter 4.

The Mac Mini is a general purpose desktop computer in a compact form factor. It receives a undistorted rectified image from each of the Raspberry Pis and uses block matching to find the disparity between corresponding points in both images. The disparities are then processed and grouped into polygon regions to form a mesh model. Implementation details of these steps will be covered in Chapter 5 and 6, respectively.

Calibration

Since the ultimate goal of this system is to produce a model of a real-world scene, it is important that the images produced by the cameras can be used to accurately measure the size and position of objects in that scene. This necessitates an accurate model of the camera itself (the intrinsic parameters) as well as the position and orientation of the camera relative to a real-world coordinate system (the extrinsic parameters). The calibration process only needs to be performed one time after the system is set up to observe a particular scene, so there has been no attempt to optimize the speed of the calibration process.

The intrinsic parameters can be grouped into two categories: 1) parameters that map a pixel position in the image to a physical location on the image sensor within the camera, and 2) the parameters that model the distortion in the lens system. With the lens distortion correction and rectification stages disabled, a calibration target consisting of a black-and-white chessboard pattern is shown to each camera. The corners of the squares in the pattern are easy to identify features within the image, even with significant distortion. Furthermore, the stark contrast between the black and white squares allows the corner positions to be inferred with fractions of a pixel accuracy.

Using multiple images of this calibration target in varying positions and orientations, an OpenCV library [2] function is used to compute all of the intrinsic parameters. The coordinates of the corners of the squares in the chessboard pattern image are identified and then processed by separate algorithms for each category of intrinsic parameter.

The lens distortion parameters are determined by a method based on Brown [3]. The corners of the squares of the chessboard pattern should form a grid of implied straight lines; any nonlinearities must be due to lens distortion. Thus the lens distortion model can be recovered through an iterative process that tries to match the observed nonlinearity. Although a single image of the calibration target would be sufficient to uniquely determine the parameters, multiple images are used to find a model that fits all of the available data on a least-squares best-fit basis to minimize the effect of any small errors in locating the square's coordinates.

Once the corner positions have been adjusted to compensate for any distortion, the remaining intrinsic parameters are determined using the method given in [26]. Because the calibration target is planar, for each image there must exist a translation and rotation matrix $\mathbf{M}_e$ such that the world coordinate for all of the corners within that image have a z-axis component of 0. Each matrix $\mathbf{M}_e$ has 6 degrees of freedom (3 rotational axes and 3 translational axes). The matrix $\mathbf{M}_i$ that is common to all of the images, has 4 degrees of freedom. With a sufficient number of images and corner coordinates within each image, the constraint that the world z-axis component is 0 along with the image corner coordinates can be used to set up a large system of equations to solve for the common $\mathbf{M}_i$ (and incidentally, the $\mathbf{M}_e$ for each image), at least up to a scaling factor. However, since the size of the squares in the chessboard pattern is known, the scaling factor can also be calculated as well.

To some extent, the extrinsic parameters can be arbitrary, since the coordinate system for the real-world scene and resulting model can also be arbitrary. However, the position and orientation of the cameras relative to each other need to be known at two different levels: 1) the relative position and orientation of the camera pair within the SCPM, and 2) the relative position and orientations of the SCPMs.

For the camera pair within the SCPM, the same calibration target used for intrinsic calibration can be used. In fact, as long as the calibration target is visible to both cameras at the same time, images for both calibrations can be acquired at the same time. The OpenCV library also provides a calibration function to compute the fundamental matrix based on two views of the same calibration target. The fundamental matrix can then be used to compute the appropriate transformations to both images for rectification.

Camera Limitations

For a static scene, it does particularly matter when the images from the various cameras are acquired. However, for a dynamic scene it becomes critical that all the images are from a particular point in time. The computed depth of a point is inversely proportional to the distance that point has apparently shifted between the left and right rectified images. If the underlying object associated with that point is in motion and the left and right images are from different times, the apparent shifted distance becomes a function not only of the depth but also the motion. Generally, any motion in the scene is not known in advance, so correctly calculating the depth becomes impossible in this case. Thus, to minimize errors in the calculated depth, the cameras must be sufficiently synchronized that any motion difference between the cameras is negligible.

In the best case, the two cameras in an SCPM are driven by a single digital clock source from within the SCPM. The offset in synchronization from one camera to the other would relate to the different propagation delays of the clock signal from the clock generator to each of the two cameras. Because of the small distances involved, the offset could easily be kept to the order of nanoseconds or less.

Unfortunately, the camera synchronization problem is not just limited to the two cameras within each SCPM; all of the SCPMs also need to be synchronized. While a single digital clock could also drive all the SCPMs, maintaining the integrity of the clock signal over the comparatively longer distances makes this implementation strategy more difficult in this situation.

As an alternative, one can take advantage of modern high-speed network interfaces. Rather than a single digital clock source for the entire system, there could be a multitude of clocks driving the cameras, as long any timing differences between the clocks are minimized. Keeping many clocks synchronized over a network is exactly the problem the Precision Time Protocol[1] (PTP) [11] was designed to solve. In a local network, PTP will typically maintain clock synchronization within a microsecond but performance can be as good as a nanosecond when used with network interface hardware that was designed specifically to support PTP.

The notion that a camera acquires its image at an instance in time is also a simplification. The image is actually acquired in a brief span of time, the length of

[1] Also known as the IEEE 1588-2002 standard, later amended as IEEE 1588-2008.

which is determined by the camera’s exposure setting. Any motion in the time span results in light from a single point to correspondingly move across the image sensor, resulting in blur. Lower lighting levels require either a longer exposure time which increases this blur, or an increase in the image sensor’s gain which also increases the magnitude of the noise.

Finally, the span of time for the exposure might not be the same span of time for all regions of the image sensor. Traditional film cameras use a mechanical shutter that essentially exposes the entire frame of the film simultaneously (the time it takes for the shutter to open and close is assumed to be negligible compared to the exposure time). Digital cameras generally do not use a mechanical shutter but instead leave the image sensor permanently exposed. Instead, the pixels on the image sensor are sampled after the desired exposure time has elapsed and then reset.

In the less expensive image sensors, this sampling happens a line at a time as the data is read out of the sensor and is known as *rolling shutter*. While the duration of exposure remains the same for all lines, the absolute start and stop time of the exposure is slightly offset from one line to the next by the amount of time it takes to output each line’s pixel data. More expensive image sensors have additional hardware to buffer the sampled pixel data, decoupling the sampling from the data output process. This allows the exposure start and stop time to be the same for the entire image and is thus known as a *global shutter*. If the imaged scene is in motion, cameras with global shutter are preferable for the same reasons given previously with regard to camera synchronization

The initial proof-of-concept implementation is using cameras with rolling shutter and all synchronized over the network with PTP. The rolling shutter will cause distortion in the model for objects moving quickly relative to the motion of the virtual shutter across the image sensor, but is considered a reasonable cost trade-off, at least for proof-of-concept purposes. With additional funding, the cameras could be upgraded to more sophisticated cameras that use global shutter.

Chapter 4: Image Acquisition, Lens Corrections, and Rectification

Overview

While the prototype system is using an single-board computer to do the initial processing associated with each individual camera, the proposed system will use dedicated hardware to accelerate the processing to achieve a faster frame rate needed for quality video. A block diagram of this hardware is shown below in Figure 20. These blocks are organized conceptually, and do not necessarily reflect the boundaries between physical components.

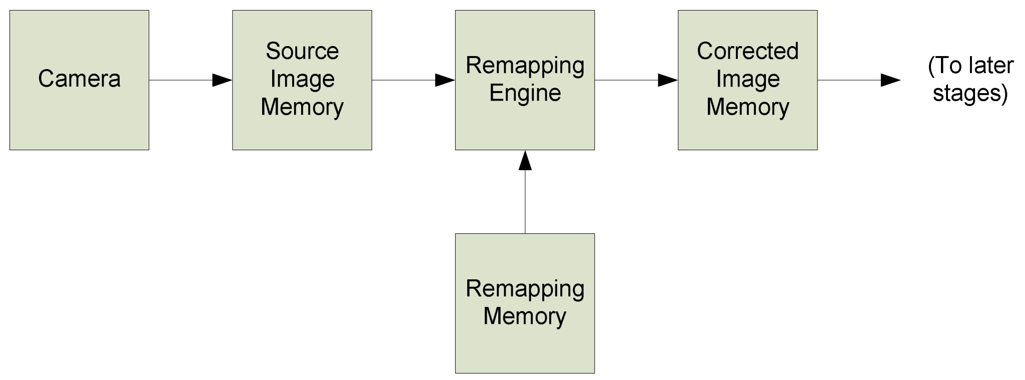

Figure 20. Block diagram showing the image data flow for an individual camera

The memories shown in Figure 20 are random access memory (RAM). By their very nature the RAM’s contents may be read or written in any order. However, the time each access takes may vary depending on the order of the accesses. Generally, modern high-speed RAM large enough to hold a complete image is fastest when the locations are accessed sequentially, somewhat slower when accessed locations are in the same region but not sequential, and slowest when accesses are in different regions. Therefore, whenever possible, the data flow should be organized so that either the RAM is accessed sequentially or smaller RAM is used that has no speed penalty for random access.

Camera

The Omnivision 5647 camera outputs its data through a Camera Serial Interface 2 (CSI-2) interface in which the data bits are serialized and sent synchronously using a low-voltage differential-signal physical interface. While this interface offers good signal integrity over the cable from the camera, it necessitates a much higher speed clock and the serial nature is completely at odds with parallel processing. All of the individual bits associated with each pixel need to be accumulated in a shift register so that the full pixel data can be reformatted into parallel output.

Source Image Memory

Within the source image memory, the pixel data is parallelized in two different ways. In the first case, the pixel data has three components, usually abbreviated with the letters Y, U, and V[2]. The Y component encodes the luminance, the grayscale brightness. The U and V components collectively encode the hue and saturation of the color. All three of these components can be processed in parallel. Alternatively, the U and V components can be discarded at this point if a grayscale image is acceptable in the final model.

The second way the pixel data can be parallelized is spatially. Recall from Equation (2.26) in Chapter 2 that the pixel remapping process in general needs the values of four

[2] This usage of U and V is completely unrelated to the convention of using U and V as pixel coordinates.

adjacent source pixels: $P(u,v), P(u+1,v), P(u,v+1), P(u+1,v+1)$ for some integer pixel coordinates *u* and *v*. If *u* is even, then $u+1$ is odd. If *u* is odd, then $u+1$ is even. Of course this applies to *v* as well. Thus, the four combinations of even and odd for u and v will always map to the four required source pixel positions above and a memory for each combination will allow all four pixel values to be accessed simultaneously.

Since the evenness or oddness of the two components of the pixel coordinate determines which of the four memories will hold the pixel value and bit 0 indicates this evenness or oddness, only bits 1 and higher of the components are needed for the memories' address lines. The addresses to each memory, however, are not necessarily the same. Without loss of generality, consider the portion of the address derived from *u*. When *u* is even, $u+1$ only differs from *u* in bit 0, which is not part of the derived memory address so the addresses to both the even and odd memory are the same. But when *u* is odd, then $u+1$ causes a carry out of bit 0 into bit 1 (and perhaps a cascade of further carries, depending on the values of the rest of the bits) causing the derived address for the even memory to be 1 higher than the odd. This can be implemented in hardware with an adder that adds 0 to the upper bits of *u* with a carry-in from bit 0 of *u* (so this can be a half-adder). This address line related circuitry for the source memory is shown below in Figure 21.

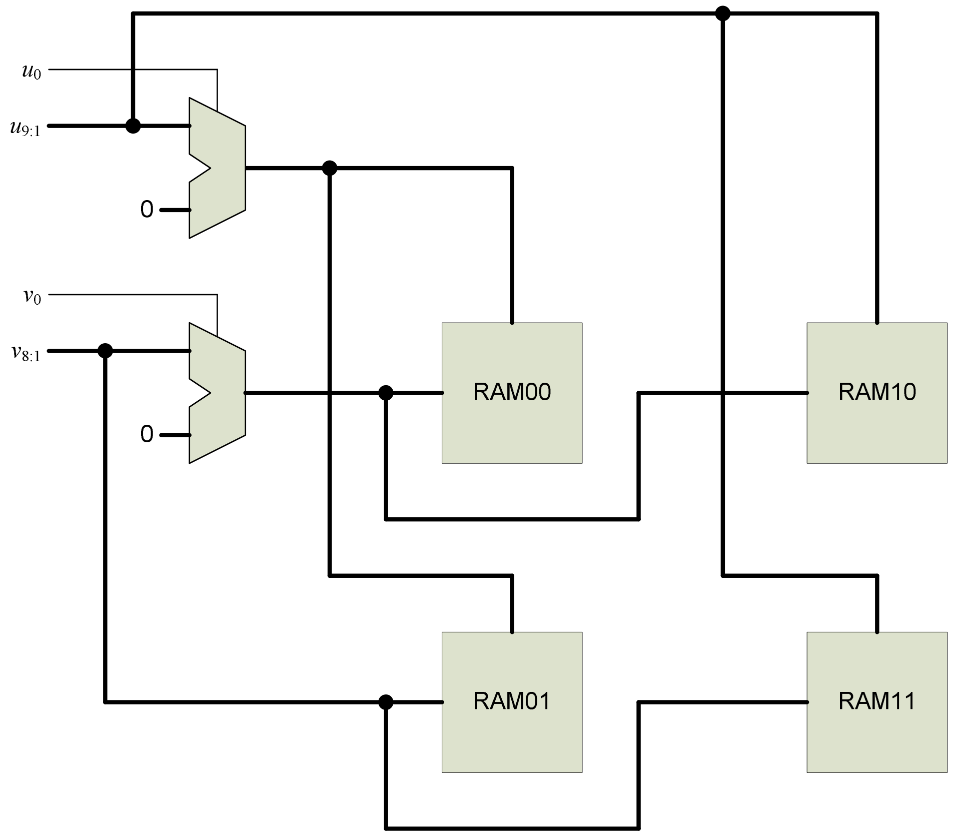

Figure 21. Source image memory organization (only address lines shown)

Remapping Engine

The remapping engine iterates through all of the coordinates of the image in the corrected image memory, applying the transformations to correct the lens distortion and achieve rectification in order to determine the corresponding coordinate in the source image memory. The engine then fetches the four closest pixels values at integer coordinates from the source image memory and applies linear interpolation to compute the pixel value to store in the corrected image memory.

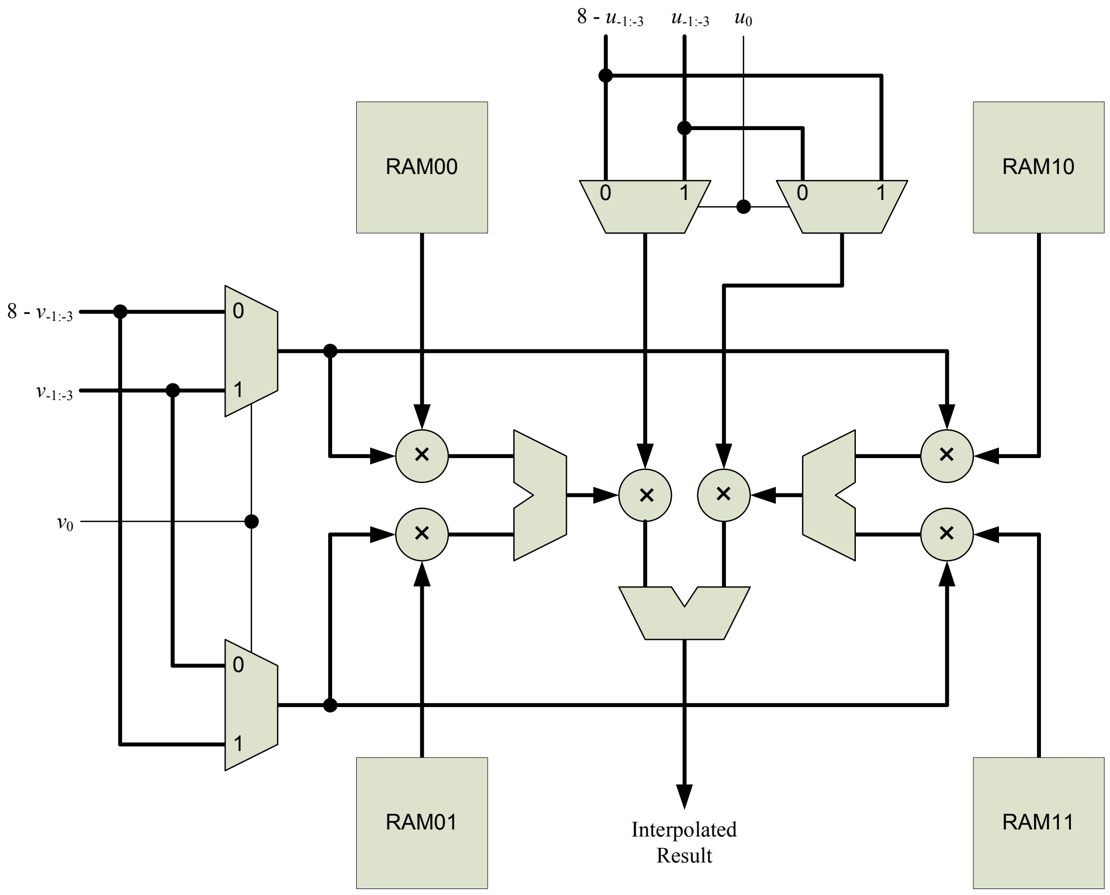

Figure 22. Remapping engine pixel value data flow.

Figure 22 shows the data flow computing the interpolated pixel value and is basically a hardware implementation of Equation (2.26). The only complication is that since the RAMs are dedicated to a particular combination of even and odd horizontal and vertical components, the pixel’s value should be multiplied by the fractional component or one less the fractional component depending on the evenness or oddness of *u* and *v*. In general, the multiplexers could be applied to either of the factors, but by associating them, as in Figure 22, with the fractional parts instead of the pixel value from the RAMs allows their propagation delay time to occur in parallel with the RAM access time, rather than adding to it.

The remapping engine fetches the *u* and *v* pixel coordinate from the remapping memory. Each pixel coordinate in the corrected image is transformed into an image plane coordinate, the lens distortion Equations (2.17) - (2.19) applied, the image rectification Equation (2.21) applied, and then finally transformed back to pixel coordinates *u* and *v* within the source image memory. Rather than perform all of these calculations in the midst of the remapping process, these calculations can be performed once after SCPM calibration and the results stored in the remapping memory. Since these calculations are only done once per system setup and are not particularly time critical, they can be handled by a general purpose processor instead of dedicated special-purpose arithmetic hardware.

If the locations in the destination image memory are accessed sequentially, the corresponding locations in the remapping memory will also be accessed sequentially. This sequential access pattern allows the fastest access times for both of these memories.

.

Chapter 5: Stereo Disparity Computation

The heart of determining depth from a pair of stereoscopic images is the correspondence problem; if a point in 3-D space can be identified as corresponding points on the two 2-D images taken by calibrated stereoscopic cameras then the depth of that point can be computed from the two 2-D coordinates and the calibration data. As noted in Chapter 2, epipolar lines can be used to constrain the search space. Image rectification warps the image so that the epipolar lines become parallel to one of the image's axes; this alignment makes a search along the epipolar line coincide with a search along consecutive locations in the memory holding the image, which in turn allows more efficient memory accesses and/or pipelining. However, finding corresponding points is still a challenging problem.

While there are many methods for identifying corresponding points [21], the chosen method is the "StereoBM" algorithm of the OpenCV library [2]. This method is classified as a block matching algorithm; a square block of pixels in one 2-D image is compared to a block window of the same size in the other 2-D image. This window is moved along the epipolar line and a comparison is made at each location.

In general, the location of the window when the best match is found is considered to be the corresponding point. However, the actual corresponding point may not be visible in both images or there may be multiple, ambiguous matches. To reduce the chances of a false match in the first case, the OpenCV StereoBM algorithm requires that the best match also meets a minimum threshold so that poor matches are simply discarded. For the second case, the algorithm also keeps track of the second best match that is not immediately adjacent to the best match and then requires that the ratio of the best match to the second best also exceeds a minimum threshold.

When comparing the block, the OpenCV StereoBM algorithm computes the absolute difference between corresponding pixel values in the block and window and then sums this absolute difference over all of the pixels within the block. Some other block matching algorithms use the square of the difference between corresponding pixel values, which has some nice mathematical properties such as easier to analyze derivatives, but the multiplication is more computationally intensive than the absolute value operation.

The size of the block is an important trade-off. If the block size is too small, there may not be enough pixels to compare to have sufficient confidence that the match is unambiguous. Alternatively, if the block size is too large, the block may encompass regions of significantly different depths and so fail to match. The default block size is 17 by 17 pixels.

The output of the OpenCV StereoBM algorithm is a set of disparity values, that is, how far a point in one image has apparently shifted to reach the position of the corresponding point in the other image. If the disparity value is 0, the point is hypothetically an infinite distance away; in practice, the point is simply far enough away that there is not enough detail available to resolve the actual distance. There is also a special value used for the disparity if the point could not be confidently matched to a corresponding point.

When demonstrating new stereoscopic related algorithms, many papers use the so-called "Tsubuka head and lamp" (often shorted to just "Tsubuka") images from [17] and shown below as Figure 23 and Figure 24.

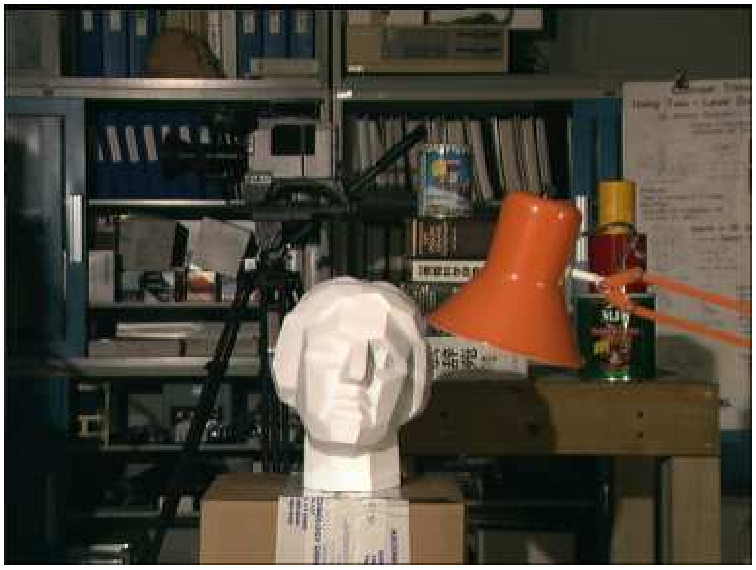

Figure 23. "Tsukuba" left image. [17]

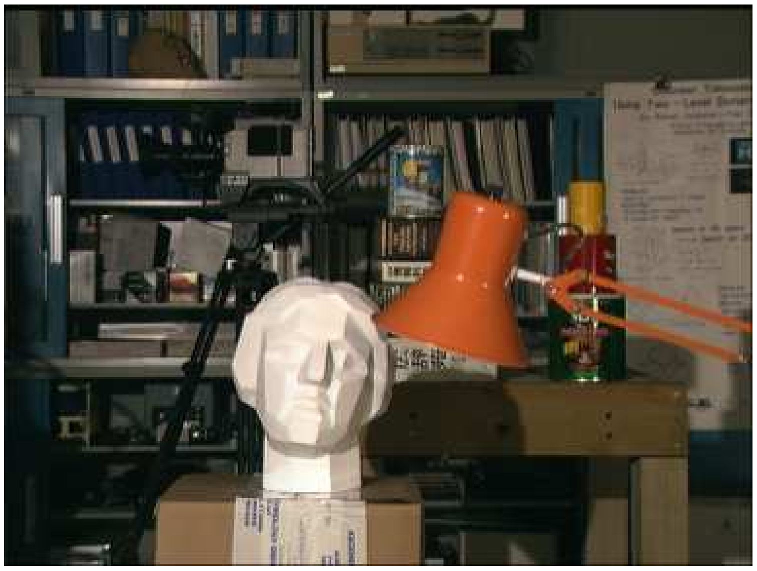

Figure 24. "Tsukuba" right image. [17]

The output of the OpenCV StereoBM algorithm for the "Tsukuba" images is shown in Figure 25; increased darkness indicates increased disparity and pure white regions indicate that the disparity is unknown. For comparison purposes, the ground-true disparity is shown in Figure 26. Generally, errors in the computed disparity occur in regions where there are many changes in depth over a short span, such as along the arm of the lamp. Disparity information is missing in larger regions where the corresponding point is occluded by other objects in the second image (the camera stand is partially obscured by the head).

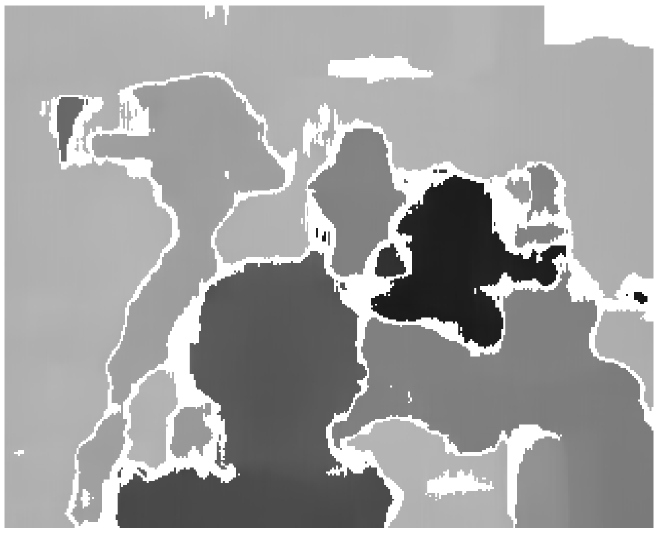

Figure 25. "Tsukuba" disparity map computed by the OpenCV block matching algorithm.

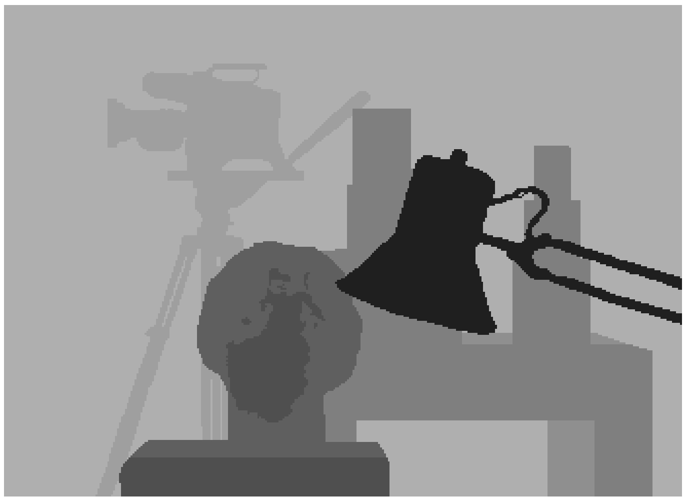

Figure 26. "Tsukuba" ground-truth disparity map. [17]

Overall, the OpenCV StereoBM algorithm has many attributes, some already noted above, that are conducive to direct implementation in hardware. In fact, a FPGA based design already exists [23] and has been verified that its output matches the OpenCV software version bit-for-bit.

Strother's design [23] accepts the rectified images as a stream of pixel data that is scanned left-to-right and top-to-bottom. After processing, the design outputs disparity values in the same left-to-right and top-to-bottom order. As such, there need not be any intermediate memory between the remapping engine of the preceding rectification and lens distortion correction stage and this stereo disparity computation stage. Likewise, the model building stage, discussed in the next chapter, can also accept the disparity values as a stream in left-to-right and top-to-bottom order.

The performance of Strother's design at a number of resolutions in various FPGAs is shown in Table 1. All of these rates are adequate for typical non-high definition video rates.

Table 1. Performance of hardware implementation of OpenCV StereoBM [23]

Pixel resolution	FPGA	Frame rate achieved
320 x 240	Xilinx Spartan 3E 250	120 frames/second
640 x 480	Xilinx Spartan 3E 500	30 frames/second
800 x 480	Xilinx Spartan 6 LX25	60 frames/second
800 x 480	Altera Cyclone IV EP4CE22	60 frames/second
1920 x 1080	Xilinx Spartan 6 LX75	30 frames/second

Chapter 6: Model Building

Building the model is a matter of finding the polygons that closely approximate the surface(s) implied by the range data. A traditional approach is to transform the ranges into a point cloud, the set of points in 3-D space that correspond to the 2-D points with known range, and then use these as vertices for a mesh of triangles that imply the surface(s). When less detail is desired, groups of adjacent triangles can be merged to form a single triangle that resembles the original group.

However, working purely from a point cloud loses some useful information in the original 2-D organization of the range data. Adjacent pixels within the 2-D image frequently represent adjacent regions in 3-D space, and this adjacency is exploited in my new approach to quickly identify which points in the point cloud are near each other when forming the surface mesh. Furthermore, the adjacency can also be used to help identify which points might be omitted from the surface mesh at the outset, rather than assembling a detailed mesh that requires further processing to simplify.

It is well known and easy to show that a line segment in 3-D space maps to a line segment on the image plane. Thus, the outline of a polygon in 3-D space will also map to a polygon on the image plane. To find these polygons, one needs to identify the pixels on the image plane that are both contiguous and members of the same approximate plane in 3-D space.

Simple Linear Regression

Linear regression is a method for modeling a set of coordinates as points on a hypothetical line [15]. For a pair of unique coordinates one may directly compute the equation of the line that contains those two coordinates, but when more than two unique points are used generally there is no longer a line that includes all the coordinates. In this case, a line is found that minimizes the error between the line and the coordinates.

The equation of a 2-dimensional line in slope-intercept form is:

$$y = mx + b \tag{6.1}$$

The error for an individual coordinate is usually computed as the square difference between the dependent variable *y* of the equation of the line and the *y* value of the coordinate:

$$E_i = \left(mx_i + b - y_i\right)^2 \tag{6.2}$$

The square of the difference in Equation (6.2) is used so that its value is always non-negative and can be summed over all the coordinates without the possibility of an error at one coordinate canceling or reducing the error at another coordinate:

$$E = \sum_{i=1}^{N} E_i = \sum_{i=1}^{N} \left(mx_i + b - y_i\right)^2 \tag{6.3}$$

Find the values for *m* and *b* that minimize this error (for conciseness, assume all of the summations are over the range $i = 1$ to *N*):

$$0 = \frac{\partial E}{\partial m} = \sum 2\left(mx_i + b - y_i\right)x_i = m\sum x_i^2 + b\sum x_i - \sum x_i y_i \tag{6.4}$$

$$0 = \frac{\partial E}{\partial b} = \sum 2\left(mx_i + b - y_i\right) = m\sum x_i + Nb - \sum y_i \tag{6.5}$$

$$\begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & N \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} \sum x_i y_i \\ \sum y_i \end{bmatrix} \tag{6.6}$$

$$m = \frac{\begin{vmatrix} \sum x_i y_i & \sum x_i \\ \sum y_i & N \end{vmatrix}}{\begin{vmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & N \end{vmatrix}} = \frac{N \sum x_i y_i - \sum x_i \sum y_i}{N \sum x_i^2 - \left(\sum x_i \right)^2} \tag{6.7}$$

$$b = \frac{\begin{vmatrix} \sum x_i^2 & \sum x_i y_i \\ \sum x_i & \sum y_i \end{vmatrix}}{\begin{vmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & N \end{vmatrix}} = \frac{\sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i}{N \sum x_i^2 - \left(\sum x_i \right)^2} \tag{6.8}$$

Using Linear Regression Error to Group Points into Line Segments

Linear regression attempts to model a set of points as a single line. If the points are actually based on multiple line segments, linear regression will determine the best line that reflects a composite of the underlying line segments. If those segments were close to being co-linear, the resulting line will have only a small error and will probably be an acceptable approximation. Otherwise, the poor fit of the resulting line to the underlying segments is reflected in a large error.

By deciding on an acceptable level of error, one can use the linear regression error to group points into different line segments. With the points sorted by their independent coordinate, the points are processed sequentially. Initially, a point is assumed to be part of the current line segment and Equations (6.7) and (6.8) are used to find the parameters for the current line segment. Then Equation (6.3) is used to compute the resulting error. If the error now exceeds the maximum acceptable error, the most recent point is removed from the current line segment and is instead used as the starting point for a new line segment.

Unfortunately, using Equation (6.3) to recompute the error for each line segment as each new point is added requires reevaluating the error between each point used and the fit line. In the underlying computer hardware these accesses of the point's coordinates are difficult to parallelize, and so become a bottleneck to computing the error quickly. However, one can also substitute Equations (6.7) and (6.8) into Equation (6.3) to directly compute the overall error based on the various summations:

$$E = \frac{2\sum x_i \sum y_i \sum x_i y_i - \sum x_i^2 \left(\sum y_i\right)^2 - N\left(\sum x_i y_i\right)^2 + N\sum x_i^2 \sum y_i^2 - \left(\sum x_i\right)^2 \sum y_i^2}{N\sum x_i^2 - \left(\sum x_i\right)^2} \tag{6.9}$$

Initially, Equation (6.9) may appear to be a step backwards since it has considerably more operations than Equation (6.3). Its virtue is that all of the summations can be computed incrementally; as each new point is considered, the corresponding newest term in the summation can be computed and added to the total. Then Equation (6.9) can be reevaluated based on these new totals, without any need to access any prior point coordinates.

The expansion of Equation (6.3) into (6.9) is somewhat complicated because of the non-zero value for the intercept *b*. Suppose the coordinate system undergoes a translation such that the line passes through this new origin, forcing:

$$b = 0 \tag{6.10}$$

Equation (6.10) now makes solving Equation (6.4) for *m* considerably simpler:

$$m = \frac{\sum x_i y_i}{\sum x_i^2} \tag{6.11}$$

Substituting Equations (6.10) and (6.11) into (6.3) results in:

$$E = \frac{\left(\sum x_i^2\right)\left(\sum y_i^2\right) - \left(\sum x_i y_i\right)^2}{\sum x_i^2} \tag{6.12}$$

Equation (6.12) is much simpler than Equation (6.9), but it depends on an initial coordinate translation. The translation itself is fairly cheap computationally, only two additions; the difficulty is determining where to place the new origin. For Equation (6.12) to be equal to Equation (6.9), the origin must be on the final fitted line, but the position of the line won't be known exactly until after the points have been grouped. This circular dependency makes Equation (6.12) appear useless for evaluating the error when grouping.

The circular dependency can be broken by accepting Equation (6.12) instead as an estimate of Equation (6.9). If the new origin is not actually on the fitted line, but slightly off the line, then the best-fit line would have a non-zero intercept *b*, with a magnitude dependent on how far the line passes from the origin. If the maximum acceptable error is small, then all of the points grouped for a line segment must be close to the fitted line. Any of these points (the first point of the segment being the most convenient) could be used as an origin when computing Equation (6.12) with the understanding that the true best-fit line is in most cases a better fit than the line passing through this origin, but never worse. Thus Equation (6.12) may overestimate the error in comparison to

Equation (6.9). So if Equation (6.12) is still less than the maximum acceptable error, Equation (6.9) is as well.

Grouping Example

As an example, consider the three line segments sampled with a small amount of added Gaussian noise shown below in Figure 27.

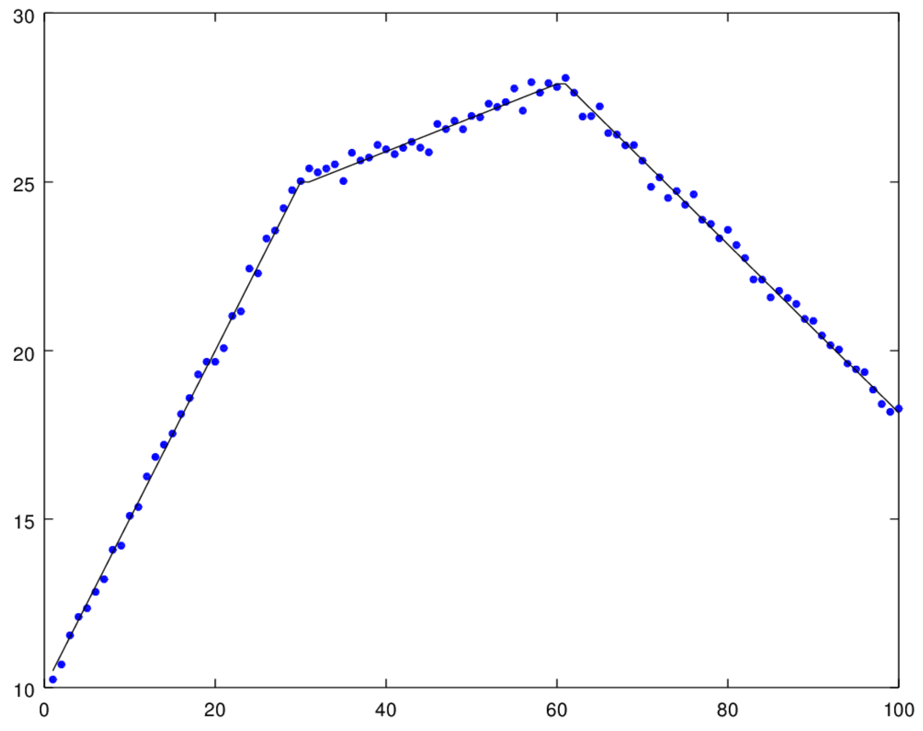

Figure 27. Sample line segment data.

If the maximum allowed error is arbitrarily set to 2, plotting the accumulated error *E* using Equation (6.9) results in Figure 28 and gives rise to the three line segments shown in Figure 29.

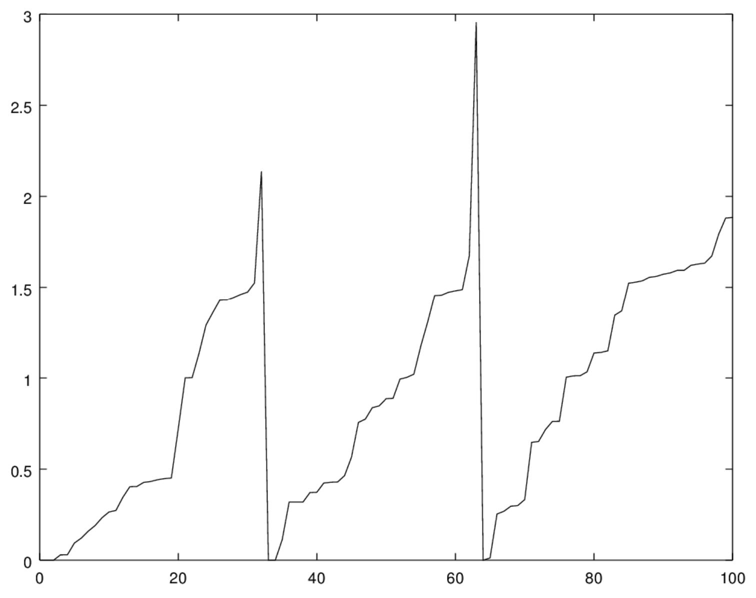

Figure 28. Accumulated error *E* using Equation (6.9).

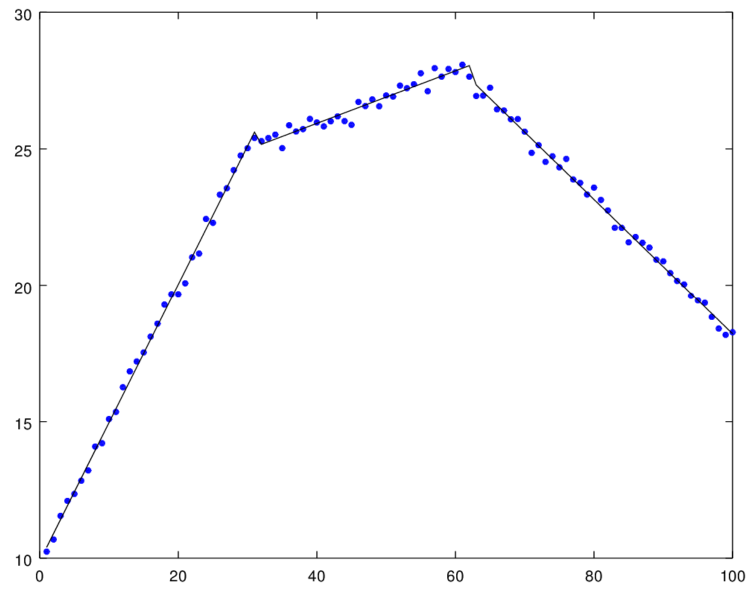

Figure 29. Three line segments fit to data using grouping based on Figure 28.

Similarly, if the maximum allowed error is arbitrarily set to 2, plotting the accumulated error *E* using Equation (6.12) results in Figure 28 and gives rise to the four line segments shown in Figure 29. Note that since Equation (6.12) slightly overestimates the true error, the final line segment happens to be broken into two line segments with this particular threshold for the maximum error.

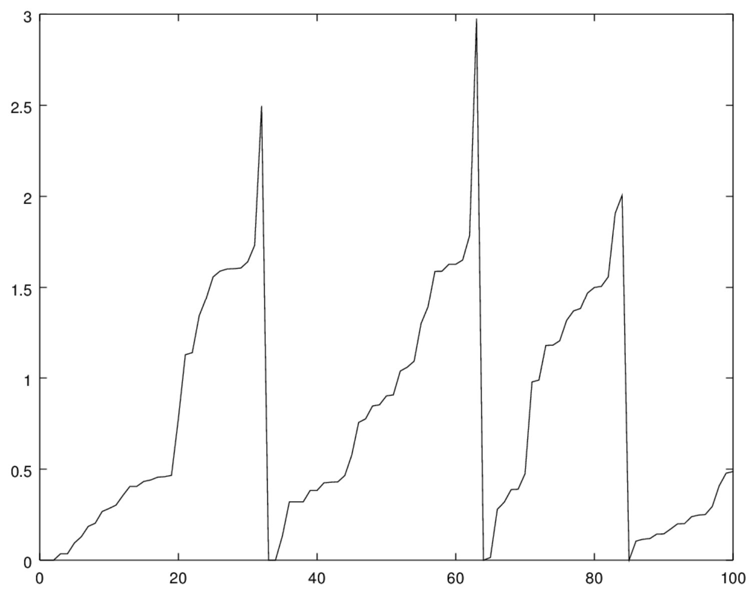

Figure 30. Accumulated error *E* using Equation (6.12)

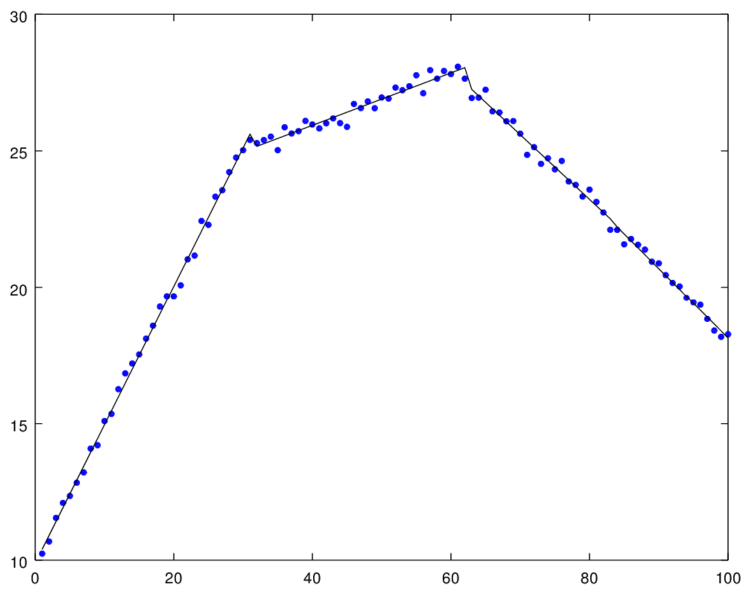

Figure 31. line segments fit to data using grouping based on Figure 30.

Identifying Line Segments

The image plane is scanned top-to-bottom and left-to-right to find line segments in the 2-D image that correspond to line segments in 3-D space and thus may be part of a polygon in 3-D space. The left-to-right scan inherently gives rise to 2-D line segments; the question really is how well do the corresponding points in 3-D space fit a line?

A 3-D line can be represented in parametric form [13]:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} + t \begin{bmatrix} X_D \\ Y_D \\ Z_D \end{bmatrix} \tag{6.13}$$

$\begin{bmatrix} X_0 & Y_0 & Z_0 \end{bmatrix}^T$ specifies some arbitrary reference point on the line, $\begin{bmatrix} X_D & Y_D & Z_D \end{bmatrix}^T$ specifies a non-zero direction vector that the line follows from that point., and t varies over the set of real numbers to generate all the points on the line. However, for a particular line the values in Equation (6.13) are not necessarily unique. The direction vector can be multiplied by any non-zero scalar and while the magnitude may change, the direction itself does not. Also, since the reference point can be any point on the line and t can be any real value, any reference point component can take on any value if the corresponding component of the direction vector is non-zero.

For 3-D lines that also appear as lines in the 2-D image plane, $X_D \neq 0$ and $Y_D \neq 0$. With these restrictions on X_D and Y_D, Equation (6.13) can be rewritten but still represent the same line as

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0 \\ Y_0 - X_0 Y_D / X_D \\ Z_0 - X_0 Z_D / X_D \end{bmatrix} + t_1 \begin{bmatrix} 1 \\ Y_D / X_D \\ Z_D / X_D \end{bmatrix} \tag{6.14}$$

and

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_0 - Y_0 X_D / Y_D \\ 0 \\ Z_0 - Y_0 Z_D / Y_D \end{bmatrix} + t_2 \begin{bmatrix} X_D / Y_D \\ 1 \\ Z_D / Y_D \end{bmatrix} \tag{6.15}$$

$$t_1 = X \tag{6.16}$$

$$Z = Z_0 - \frac{X_0 Z_D}{X_D} + \frac{X Z_D}{X_D} \tag{6.17}$$

$$X = \frac{X_D}{Z_D} Z - \frac{X_D}{Z_D} Z_0 + X_0 \tag{6.18}$$

$$t_2 = Y \tag{6.19}$$

$$Z = Z_0 - \frac{Y_0 Z_D}{Y_D} + \frac{Y Z_D}{Y_D} \tag{6.20}$$

$$Y = \frac{Y_D}{Z_D} Z - \frac{Y_D}{Z_D} Z_0 + Y_0 \tag{6.21}$$

Equations (6.18) and (6.21) allow the line to be described in slope-intercept form in the X-Z and Y-Z planes:

$$X = m_x Z + b_x \tag{6.22}$$

$$Y = m_y Z + b_y \tag{6.23}$$

with

$$m_x = \frac{X_D}{Z_D} \tag{6.24}$$

$$b_x = X_0 - \frac{X_D}{Z_D} Z_0 \tag{6.25}$$

$$m_y = \frac{Y_D}{Z_D} \tag{6.26}$$

$$b_y = Y_0 - \frac{Y_D}{Z_D} Z_0 \tag{6.27}$$

Use the sum-of-square-residuals (*Ssr*) to evaluate how well the data points fit Equations (6.22) and (6.23) :

$$S_{sr} = \sum_{i=1}^{N} \left(X - X_i \right)^2 + \sum_{i=1}^{N} \left(Y - Y_i \right)^2 \tag{6.28}$$

$$S_{sr} = \sum_{i=1}^{N} \left(m_x Z_i + b_x - X_i \right)^2 + \sum_{i=1}^{N} \left(m_y Z_i + b_y - Y_i \right)^2 \tag{6.29}$$

$$S_{sr} = S_{srx} + S_{sry} \tag{6.30}$$

$$S_{srx} = \sum_{i=1}^{N} \left(m_x Z_i + b_x - X_i \right)^2 \tag{6.31}$$

$$S_{sry} = \sum_{i=1}^{N} \left(m_y Z_i + b_y - Y_i \right)^2 \tag{6.32}$$

If the line is being fitted to data from a horizontal slice in the image plane (*y* is constant) then Equation (6.32) can be simplified. Recall Equations (2.3) through (2.7):

$$\mathbf{x} = \mathbf{M}\mathbf{X} \tag{2.3}$$

$$\mathbf{M} = \mathbf{M}_i \mathbf{M}_e \tag{2.4}$$

$$\mathbf{M}_i = \begin{bmatrix} \frac{f}{h_x} & 0 & o_x \\ 0 & \frac{f}{h_y} & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$\mathbf{M}_e = \begin{bmatrix} \mathbf{R}_1 & -\mathbf{R}_1\mathbf{T} \\ \mathbf{R}_2 & -\mathbf{R}_2\mathbf{T} \\ \mathbf{R}_3 & -\mathbf{R}_3\mathbf{T} \end{bmatrix} \tag{2.6}$$

$$\mathbf{M}_{e0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.7}$$

Choosing no coordinate rotation or translations ($\mathbf{M}_e = \mathbf{M}_{e0}$):

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} \frac{f}{h_x} & 0 & o_x \\ 0 & \frac{f}{h_y} & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{6.33}$$

Then

$$y = \frac{Y_i f}{Z_i h_y} + o_y \tag{6.34}$$

for all Y_i and Z_i visible through that slice. Solve for Z_i:

$$Z_i = \frac{Y_i f}{h_y \left(y - o_y \right)} \tag{6.35}$$

Substituting Equation (6.35) into Equation (6.32):

$$S_{sry} = \sum_{i=1}^{N} \left(\frac{m_y Y_i f}{h_y \left(y - o_y \right)} + b_y - Y_i \right)^2 \tag{6.36}$$

Then choosing

$$m_y = \frac{h_y \left(y - o_y \right)}{f} \tag{6.37}$$

and

$$b_y = 0 \tag{6.38}$$

forces every term in the summation of (6.36) to be zero: a perfect fit, at least in the Y-Z plane. So to evaluate the goodness of fit, only equation (6.31) needs to be evaluated. However, Equation (6.31) is the same as Equation (6.3) but with different variable names, so Equations (6.9) and (6.12) can be updated with appropriately renamed variables:

$$E = \frac{2\sum Z_i \sum X_i \sum Z_i X_i - \sum Z_i^2 \left(\sum X_i\right)^2 - N\left(\sum Z_i X_i\right)^2}{N\sum Z_i^2 - \left(\sum Z_i\right)^2} + \frac{N\sum Z_i^2 \sum X_i^2 - \left(\sum Z_i\right)^2 \sum X_i^2}{N\sum Z_i^2 - \left(\sum Z_i\right)^2} \tag{6.39}$$

$$E = \frac{\left(\sum \tilde{Z}_i^2\right)\left(\sum \tilde{X}_i^2\right) - \left(\sum \tilde{Z}_i \tilde{X}_i\right)^2}{\sum \tilde{Z}_i^2} \tag{6.40}$$

with the coordinate system origin shifted to the first point by:

$$\tilde{X}_i = X_i - X_1 \tag{6.41}$$

$$\tilde{Z}_i = Z_i - Z_1 \tag{6.42}$$

Division is one of the harder operations to implement in hardware and so should be avoided when possible. If the error, *E*, itself is only needed as part of a test to ensure that it does not exceed some $E_{\max}$ then the division can be algebraically replaced with a multiplication. For example, Equation (6.40) can be rearranged into the test:

$$E_{\max} \sum \tilde{Z}_i^2 \overset{?}{>} \left(\sum \tilde{Z}_i^2\right)\left(\sum \tilde{X}_i^2\right) - \left(\sum \tilde{Z}_i \tilde{X}_i\right)^2 \tag{6.43}$$

Line Segment Variables

Each line segment has the following associated variables:

- L_u - the *u* in the pixel coordinate space for the left endpoint of the segment
- R_u - the *u* in the pixel coordinate space for the right endpoint of the segment
- **L** - the 3-D coordinate $\begin{bmatrix} L_X & L_Y & L_Z \end{bmatrix}^T$ of the left endpoint of the segment
- **R** - the 3-D coordinate $\begin{bmatrix} R_X & R_Y & R_Z \end{bmatrix}^T$ of the right endpoint of the segment

- ΣX - the sum of the 3-D X coordinates of the points in the line segment
- ΣZ - the sum of the 3-D Z coordinates of the points in the line segment
- $\Sigma \tilde{X}\tilde{X}$ - the sum of the squared relative 3-D X coordinates of the points in the line segment
- $\Sigma \tilde{Z}\tilde{Z}$ - the sum of the squared relative 3-D Z coordinates of the points in the line segment
- $\Sigma \tilde{X}\tilde{Z}$ - the sum of the product of the relative 3-D X and Z coordinates of the points in the line segment
- T - the index of an associated "top" line segment or 0 if there is no associated segment.

All of these variables are retained in memory. There may be other values associated with each line segment, such as its midpoint location, that are not retained in memory but instead computed as needed from the variables above.

Line Segment Identification Algorithm

Each horizontal slice of the 2-D disparity data is broken into a series of line segments using the following algorithm:

- Each pixel position in the slice is evaluated exactly once, in any order that results in adjacent pixels being processed consecutively
- If there is no disparity data available for that pixel position and there is a current line segment that has not been ended, then that line segment is marked as ended with a discontinuity.
- If there is disparity data available for that pixel position and there is a current line segment that has not been ended:

  - Provisionally update the ΣX, ΣZ, $\Sigma\tilde{X}\tilde{X}$, $\Sigma\tilde{X}\tilde{Z}$, and $\Sigma\tilde{Z}\tilde{Z}$ sums with the *X*, *Z*, $\tilde{X}^2$, $\tilde{X}\tilde{Z}$, and $\tilde{Z}^2$ values for the current pixel position.
  - Test if the error, *E*, exceeds the maximum chosen threshold $E_{\max}$ by evaluating the inequality in Equation (6.43)
  - If *E* exceed the chosen threshold, then the provisional sums are discarded, the current line segment is marked as ended with continuity, and a new line segment is begun.
  - Otherwise, if *E* did not exceed the chosen threshold, then the provisional sums are accepted as the actual sums, R_u is assigned *u*, and **R** is assigned $\begin{bmatrix} X & Y & Z \end{bmatrix}^T$.
- If there is disparity data available for that pixel position and there is no current line segment or the current line segment has already been ended, then a new line segment is begun.
- If all of the pixel positions in the slice have been considered and there is a current line segment that has not been ended, then that segment is marked as ended with a discontinuity.
- Whenever a new line segment is begun:
  - The ΣX and ΣZ sums are reset to *X* and *Z* respectively.
  - The $\Sigma\tilde{X}\tilde{X}$, $\Sigma\tilde{X}\tilde{Z}$, and $\Sigma\tilde{Z}\tilde{Z}$ sums are reset to 0.
  - L_u and R_u are assigned *u*.
  - **L** and **R** are assigned $\begin{bmatrix} X & Y & Z \end{bmatrix}^T$
  - *T* is assigned 0.

Examples of the line segment extraction at various error thresholds using the "Tsukuba" disparity values from Figure 25 are shown below as Figure 32 through Figure 34.

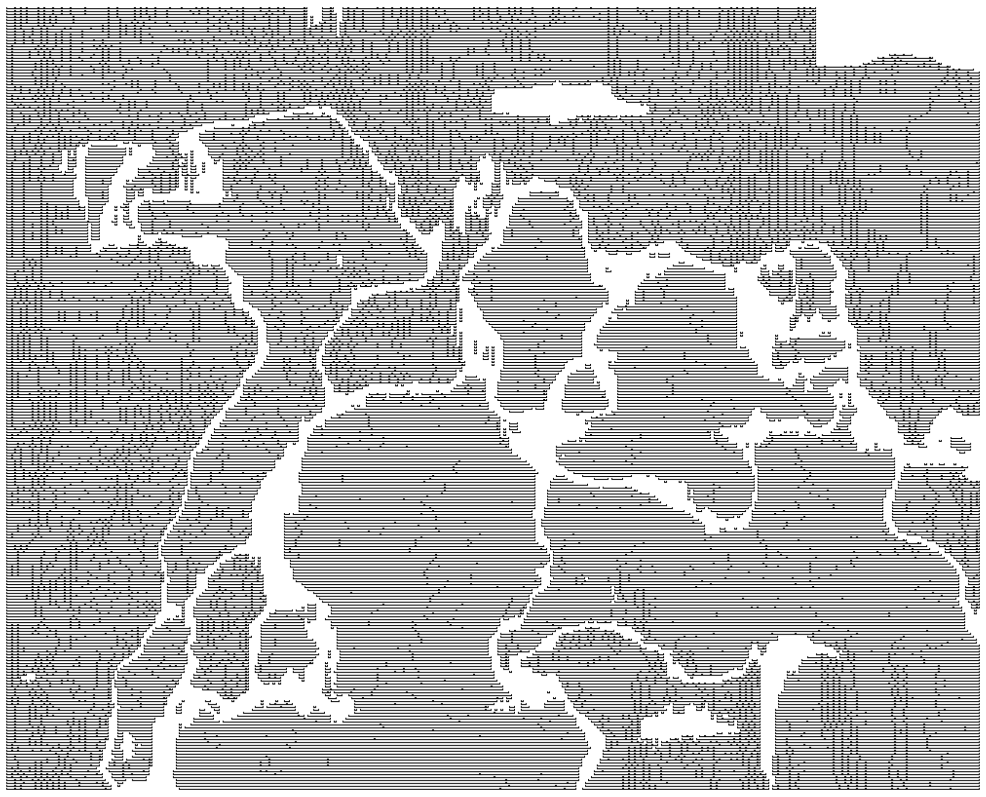

Figure 32. Segment map with maximum error = 1: 12622 segments

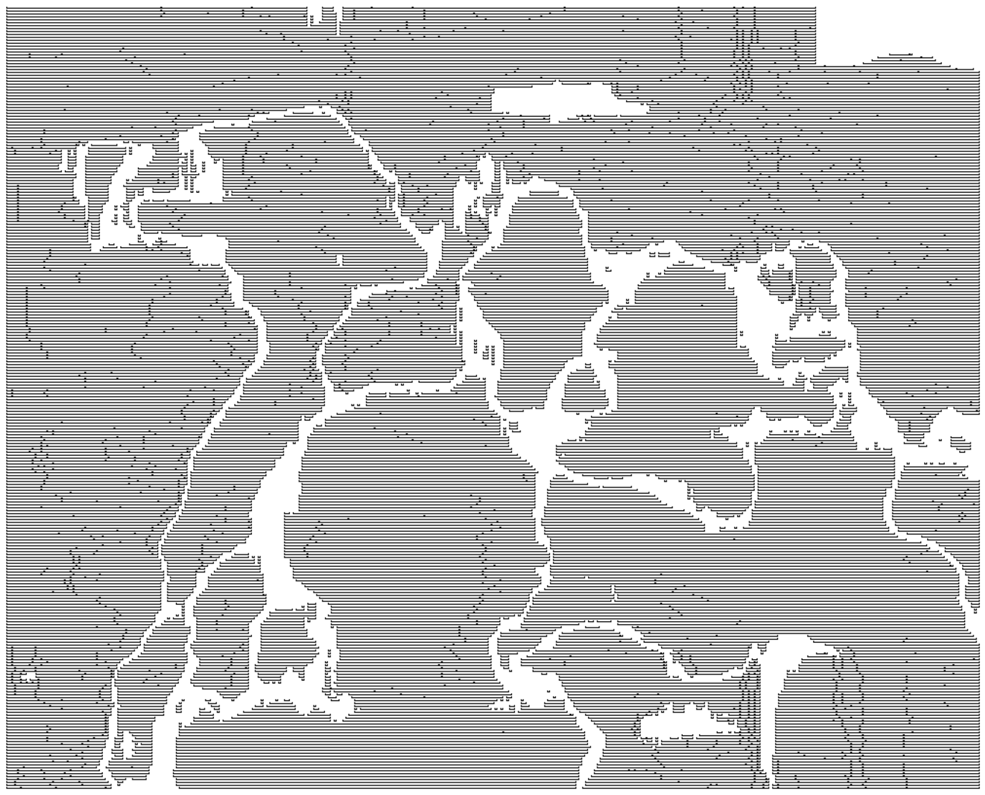

Figure 33. Segment map with maximum error = 10: 4072 segments

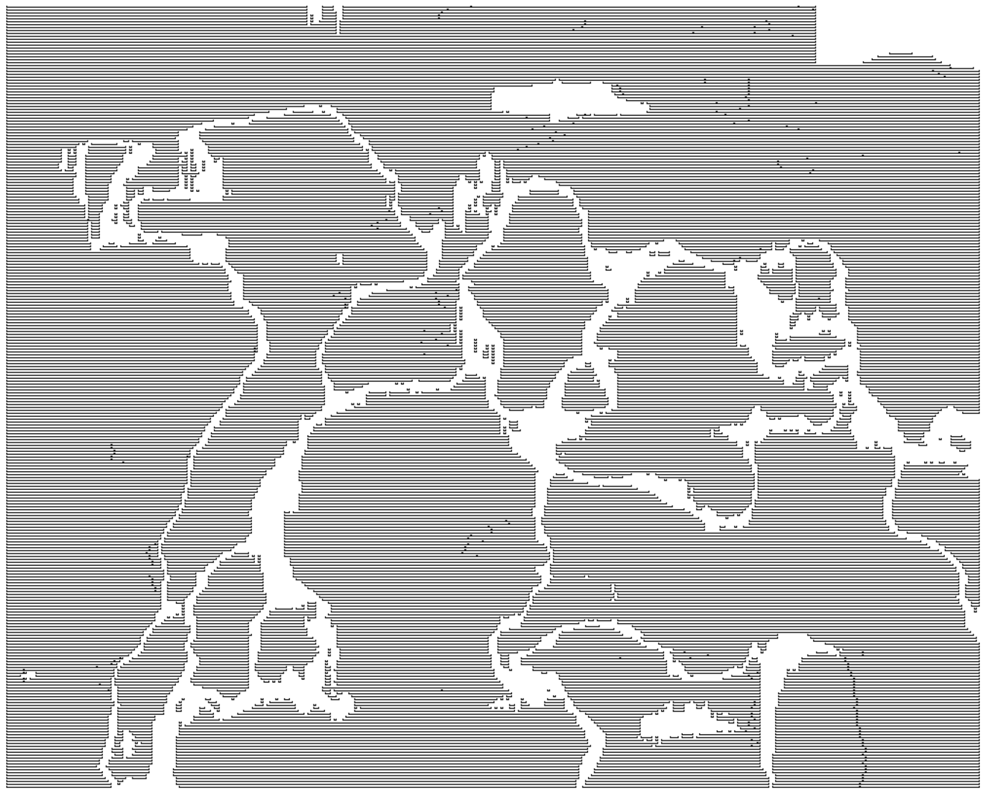

Figure 34. Segment map with maximum error = 100: 2318 segments

Merging Line Segments into Polygons

Once all of the line segments have been identified, adjacent line segments that are coplanar and have acceptably straight edges can be grouped into polygons. Horizontally adjacent line segments do not need to be considered because the underlying points are not coplanar; if they were coplanar, the underlying points of these two segments would have been colinear and grouped into a single segment rather than two. Thus, only vertically adjacent line segments need to be considered.

Within a pair of horizontal slices, all of the line segments are tested in pairs for adjacency. In general, the line segments could be considered adjacent if there is any overlap in their horizontal spans. That is, either the horizontal minimum or maximum of one segment *i* is within the horizontal minimum and maximum of the other segment *j*:

$$\left(L_u\right)_i < \left(L_u\right)_j < \left(R_u\right)_i \tag{6.44}$$

$$\left(L_u\right)_i < \left(R_u\right)_j < \left(R_u\right)_i \tag{6.45}$$

A better test, with only slightly higher computational cost, is to require that the midpoint of each segment be within the horizontal minimum and maximum of the other. This assures that there is a significant region in common between the two segments:

$$\left(L_u\right)_i < \frac{\left(L_u\right)_j + \left(R_u\right)_j}{2} < \left(R_u\right)_i \tag{6.46}$$

$$\left(L_u\right)_j < \frac{\left(L_u\right)_i + \left(R_u\right)_i}{2} < \left(R_u\right)_j \tag{6.47}$$

Initially, the index *i* is set to the index of the first segment of a horizontal slice and the index *j* is set to the index of the first segment of the following horizontal slice. If the indexed segments' midpoints overlap with each other's span, i.e., both Equation (6.46) and (6.47) hold true, then more computationally expensive tests can be evaluated and both *i* and *j* are advanced to the next index. Otherwise, the only one of *i* or *j* is advanced to the next index; whichever corresponds to the segment that horizontally "precedes" the other. Specifically, *i* is advanced when either:

$$\frac{(L_u)_i + (R_u)_i}{2} < (L_u)_j \tag{6.48}$$

or

$$(L_u)_i < \frac{(L_u)_j + (R_u)_j}{2} \tag{6.49}$$

Otherwise *j* is advanced.

Once a pair of vertically adjacent line segments have been identified, there are three tests to see if the lower segment can be combined into a polygon with the upper segment and any previous segments that the upper segment had already been combined with: 1) are the left edges sufficiently colinear, 2) are the right edges sufficiently colinear, and 3) are all of the underlying points sufficiently coplanar? Tests 1 and 2 use the same linear regression incremental error evaluation described earlier in this chapter using the *u*, *v* pixel space coordinates of the left or right edge (depending on the test) of the line segments.

Test 3, coplanarity, is somewhat more complicated. While one can extend the derivation of Equation (6.9) to find its equivalent for the multivariate regression needed to evaluate how a plane would fit the data points, the number of terms in this equation quickly becomes unreasonable. As an alternative, the scalar triple product can be used to evaluate coplanarity:

$$E = \left| \left[(\mathbf{L})_i - (\mathbf{R})_i \right] \times \left[(\mathbf{L})_i - (\mathbf{L})_j \right] \cdot \left[(\mathbf{L})_j - (\mathbf{R})_j \right] \right| \tag{6.50}$$

Equation (6.50) computes the volume of the parallelepiped implied by the two line segments; if the volume is 0 (or sufficiently close to 0) then the line segments are (sufficiently) coplanar.

If all three tests have passed, then the two line segments are considered to be part of the same polygon. $(T)_j$ is assigned $(T)_i + i$ and then $(T)_i$ is assigned zero. Thus, $(T)_j$ will have the index of the topmost line segment of a polygon when *j* is the index of the bottommost line segment of a polygon. Otherwise, $(T)_j$ is zero because the line segment is not part of any polygon or occupies some other position within the polygon.

These final three tests could be implemented in hardware as well, however, it is expected that a software implementation would be adequate if the line segments are built directly by a hardware implementation. For example, the "Tsukuba" images (which have a resolution of 384 by 288 pixels) take approximately 10 ms to merge on an AMD FX-6300 CPU running at 3.5 GHz.

As an example of this line segment merging process, consider the line segments shown in Figure 35. Gaps have been inserted between these line segments to show the distinct segments for illustration purposes, but they represent segments that are continuous in both the horizontal and vertical directions. Assuming the segments are sufficiently coplanar, these segments could be grouped into the polygons shown in bold in Figure 36. Note that the edges of these polygon need not exactly coincide with the line segment endpoints, as shown in the upper right polygon; this is determined by the linear regression error tolerance chosen for the left and right edges. There is also the possibility that a segment cannot be merged with any other segment and becomes orphaned; see the line segment in the upper right corner.

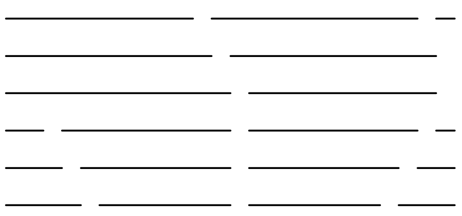

Figure 35. Enlarged detail of sample line segments.

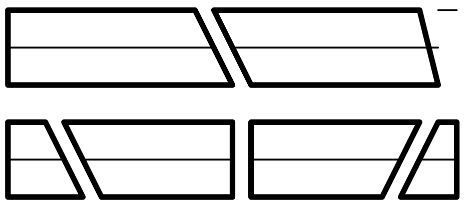

Figure 36. Line segments (thin) from Figure 35 grouped into polygons (bold).

Extracting the Polygons

Once the line segment merging process has been completed, the line segments are iterated over to find each instance that $(T)_i \neq 0$. Each instance represents a trapezoidal polygon that is part of a surface in the 3-D model. This polygon can be approximated by the vertices $(\mathbf{L})_i$, $(\mathbf{R})_i$, $(\mathbf{R})_j$, and $(\mathbf{L})_j$ with $j = (T)_i$. However, these vertices may not be exactly coplanar, so the trapezoid is split along the diagonal into two triangles, one with vertices $(\mathbf{L})_i$, $(\mathbf{R})_i$, and $(\mathbf{R})_j$, and the other with vertices $(\mathbf{L})_i$, $(\mathbf{R})_j$, and $(\mathbf{L})_j$.

Examples of the resulting polygons built from the line segments in Figure 32, Figure 33, and Figure 34 are shown below as Figure 37, Figure 38, and Figure 39, respectively. As the allowed error increases, the number of polygons/triangles decreases. Potentially, the maximum allowed error could be dynamically adjusted according to the bandwidth available and complexity of the image so that more detail is retained when the bandwidth needed to transmit a more complex model is available.

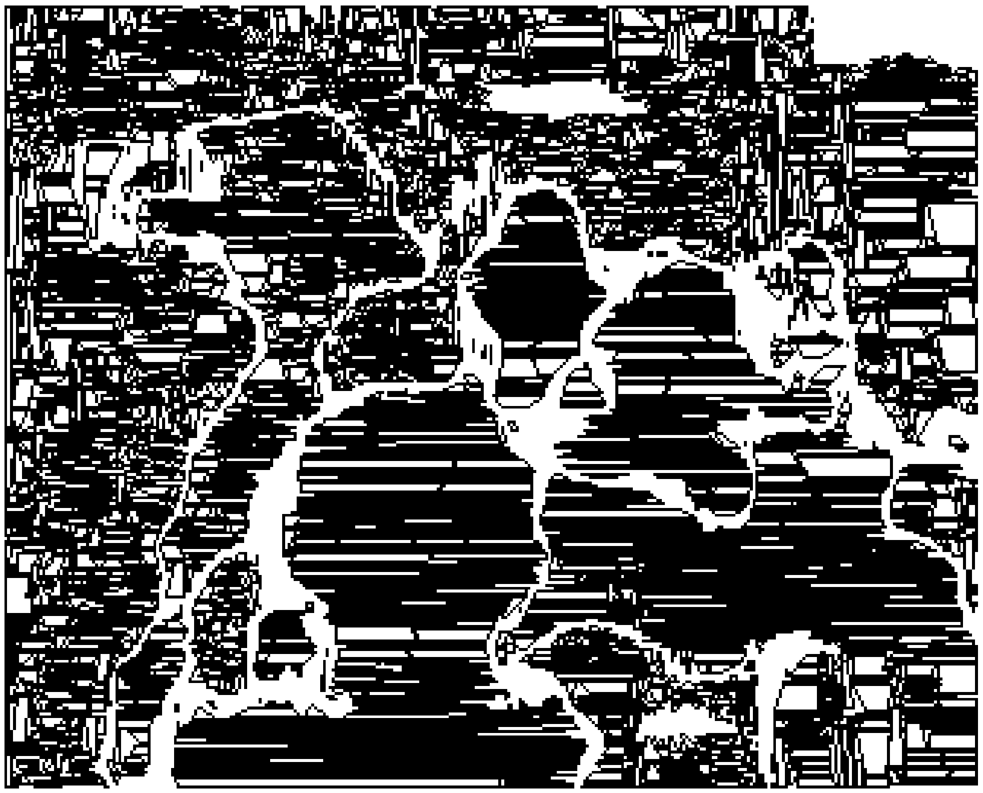

Figure 37. Polygons with maximum error = 1: 6128 triangles

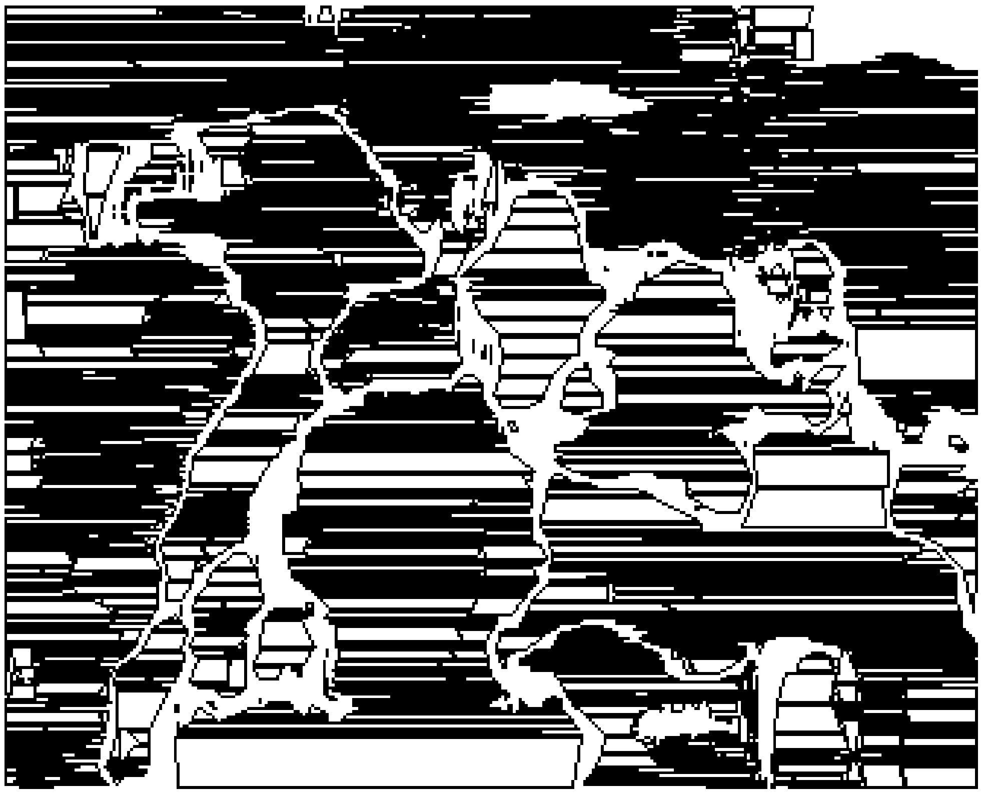

Figure 38. Polygons with maximum error = 10: 2310 triangles

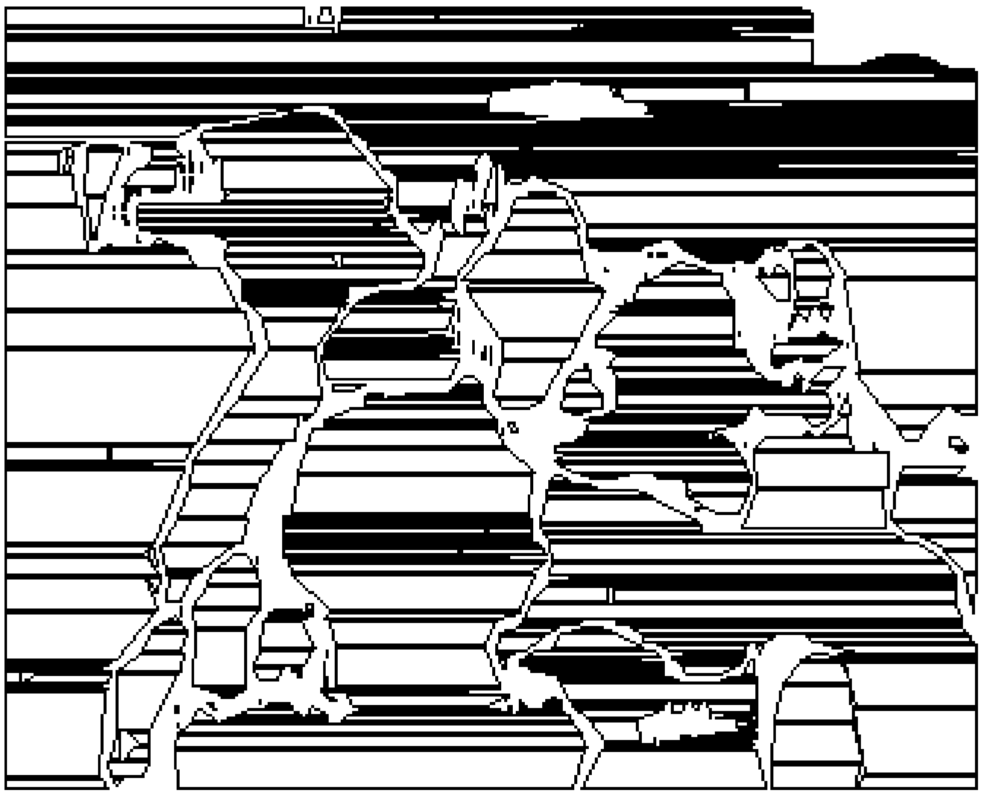

Figure 39. Polygons with maximum error = 100: 912 triangles

Chapter 7: Conclusions / Recommendations

This dissertation describes a complete end-to-end system for acquiring images of a scene and reconstructing a 3-D model of that scene in real-time. For the most part, the system's image processing has been structured so that it can be implemented in logic within FPGAs, with the number of clock cycles needed proportional to the number of pixels in the source images. Each stage can complete its processing within the period of a single video frame, and pipeline its results to the next stage on a frame-by-frame basis. Thus, the system has an overall latency of only a few frames and meets its real-time objective, both in terms of frame rate and latency.

Within the model building process, an algorithm for using linear regression error to group a series of data points into multiple line segments, rather than a single line, was discovered. This algorithm has the efficient property that each point need only be evaluated once and so not only has a run-time cost of only O(n), but also fits well with a pipelined data flow model in hardware. This line segmenting algorithm was then used as a basis for discovering planar surfaces and polygons in a novel model building algorithm.

Nevertheless, there are certainly some areas for improvement and future research. The model building relies entirely on the quality of data in the stereo disparity map. The OpenCV StereoBM algorithm was chosen because it is already implemented both as a software library function as well as a verified hardware FPGA design. However, there are other algorithms with better accuracy; it may be possible to create an FPGA implementation of them as well. Alternatively, the OpenCV StereoBM algorithm could be retained as an initial estimate of the disparity, but a secondary algorithm added that considers inter-frame motion to refine the disparity (corresponding points must both either be in regions with motion or regions without motion; otherwise, they cannot correspond even if the pixel values are similar).

Within the model building algorithm, the line segments are grouped into trapezoidal polygons, which are then split into two triangles. While the 3-D mesh models are generally built from triangle primitives, there are various ways to break down the higher order polygons into triangles. Limiting the polygons to trapezoids was a decision based on simplicity; allowing more complex polygons may permit a better breakdown of triangles (triangles with very sharp angles are more difficult to render properly when regenerating the model on a display).

The generated polygon vertices could be better selected so that the surface fits the underlying data points more closely. Many of the sums used for evaluating the regression error in the line segments are also sums needed to compute a least-squares regression-fit of a plane to the points; with a few more sums computed and retained, a best-fit polygon could be computed. Somewhat at odds with this, however, is another problem with the polygon vertices. Adjacent polygons that are both part of what should be a smooth surface may have gaps on their edges. The line segmentation algorithm tracks where the line segments, and thus also the resulting polygons, should be continuous, but this information is not currently being used.

Finally, the maximum allowed error when grouping adjacent regions into polygons is currently a fixed parameter. A larger allowed error allows rougher surfaces to be approximated as planar, which in turn reduces the number of polygons in the model and the amount of bandwidth necessary to transmit the model. Alternatively, a smaller allowed error preserves more detail but at the expense of more polygons, a more complex model, and more bandwidth required. Rather than use a fixed value for the maximum allowed error, this value could be dynamically adjusted so that if the available bandwidth changes or the complexity of the scene changes, the actual bandwidth used could track the available bandwidth.

References

[1] A. Blake and A. Zisserman, *Visual Reconstruction*. Cambridge, MA: MIT Press, 1987.

[2] G. Bradski, “The OpenCV library,” *Dr. Dobb’s Journal of Software Tools*, vol. 25, no. 11, pp.120-126, 2000.

[3] D. C. Brown, “Close-range camera calibration,” *Photogrammetric Engineering*, vol. 37, no. 8, pp. 855-866, 1971.

[4] B. Curless and M. Levoy, “A volumetric method for building complex models from range images,” in *Proc. 23rd Annu. Conf. Computer Graphics and Interactive Techniques*, 1996, pp. 303-312.

[5] F. Devernay and O. D. Faugeras, “Automatic calibration and removal of distortion from scenes of structured environments,” in *SPIE’s 1995 Int. Symp. Optical Science, Engineering, and Instrumentation*, pp 62-72, Int. Soc. for Optics and Photonics.

[6] L. Falkenhagen, “3D object-based depth estimation from stereoscopic image sequences,” in *Proc. of Int. workshop on Stereoscopic and Three Dimensional Imaging*, 1995, vol. 95, pp. 81-86.

[7] X. Gao et al., “Implementation of auto-rectification and depth estimation of stereo video in a real-time smart camera system,” in *IEEE Computer Society Conf. Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1-7.

[8] A. Gruen and T. S. Huang, Eds., *Calibration and Orientation of Cameras in Computer Vision*, Berlin: Springer-Verlag, 2001.

[9] R. Hartley, “Theory and practice of projective rectification,” *International Journal of Computer Vision*, vol. 35, no. 2, pp. 115-127, 1999.

[10] R. Hartley and A. Zisserman, *Multiple View Geometry in computer vision*, 2nd ed., Cambridge: Cambridge University Press, 2003.

[11] *Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE 1588-2008, 2008.

[12] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*, Boston: McGraw-Hill, 1995.

[13] B. Kolman and D. Hill, *Elementary Linear Algebra*, Upper Saddle River, NJ: Pearson Education, 2004.

[14] M. Levoy et al., “The digital Michelangelo project: 3D scanning of large statues,” in *Proc. 27th Annu. Conf. Computer Graphics and Interactive Techniques*, 2000, pp. 131-144.

[15] W. Mendenhall and T. Sincich, *Statistics for Engineering and the Sciences*, Upper Saddle River, NJ: Prentice-Hall, 1995.

[16] H. M. Merklinger, *The INs and OUTs of FOCUS*, Dartmouth, NS: Merklinger, 2002, pp. 14-15, [Online]. Available: http://www.trenholm.org/hmmerk/TIAOOFe.pdf

[17] Y. Nakamura et al., "Occlusion detectable stereo-occlusion patterns in camera matrix," in *1996 IEEE Computer Society Conf. Computer Vision and Pattern Recognition*, pp. 371-378.

[18] E. Petrich, “Image processing methods for product label identification on cylindrical surfaces,” M.S. thesis, Elec. and Comp. Engr., Univ. of Oklahoma. Norman, OK, 2004.

[19] Raspberry Pi Foundation, “Raspberry Pi FAQ” [Online]. Available: http://www.raspberrypi.org/help/faqs/

[20] H. Rushmeier et al., “Acquiring input for rendering at appropriate levels of detail: digitizing a Pieta,” in *Proc. 9th Eurographics Rendering Workshop*, Vienna: Springer-Verlag, 1998, pp. 81-92.

[21] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *International journal of computer vision*, vol. 47, no.1-3, pp. 7-42, 2002.

[22] M. Sormann et al., “Watertight multi-view reconstruction based on volumetric graph-cuts,” *Image Analysis*, Berlin: Springer, 2007, pp. 393-402.

[23] D. Strother, “Open-source FPGA Stereo Vision Core released,” [Online]. Available: http://danstrother.com/2011/06/10/fpga-stereo-vision-core-released/

[24] G. Turk and M. Levoy, “Zippered polygon meshes from range images,” in *Proc. 21st Annu. Conf. Computer Graphics and Interactive Techniques*, 1994, pp.311-318.

[25] J. Woodfill and B. Von Herzen, “Real-time stereo vision on the PARTS reconfigurable computer”, in *Proc. 5th Annu. IEEE Symp. on Field-Programmable Custom Computing Machines,* 1997, pp. 201-210.

[26] Z. Zhang, “A flexible new technique for camera calibration”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, 2000, pp. 1330-1334