3D Surface Reconstruction from Multi-Camera Stereo with Distributed Processing

3D SURFACE RECONSTRUCTION FROM MULTI-CAMERA STEREO WITH DISTRIBUTED PROCESSING

By

GORAV ARORA, B. Eng.
McMaster University, Hamilton, Ontario, Canada

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
Master of Engineering

McMaster University

MASTER OF ENGINEERING (2001)	MCMASTER UNIVERSITY
(Electrical and Computer Engineering)	Hamilton, Ontario

TITLE:	3D Surface Reconstruction from Multi-Camera Stereo with Distributed Processing
AUTHOR:	Gorav Arora B. Eng. McMaster University, Hamilton, Ontario, Canada
SUPERVISOR:	Dr. David W. Capson
NUMBER OF PAGES:	xi,118

Abstract

In this thesis a system which extracts 3D surfaces of arbitrary scenes under natural illumination is constructed using low-cost, off-the-shelf components. The system is implemented over a network of workstations using standardized distributed software technology. The architecture of the system is highly influenced by the performance requirements of multimedia applications which require 3D computer vision. Visible scene surfaces are extracted using a passive multi-baseline stereo technique. The implementation efficiently supports any number of cameras in arbitrary positions through an effective rectification strategy. The distributed software components interact through CORBA and work cooperatively in parallel. Experiments are performed to assess the effects of various parameters on the performance of the system and to demonstrate the feasibility of this approach.

Acknowledgements

I would like to extend my deep gratitude to my supervisor Dr. David Capson for his patience, support and for providing me with this research opportunity. Without the love, support and understanding of my family, this would not have been possible. I would also like to thank the other members of the Machine Vision and Image Analysis Laboratory, David Woo, Jason Stavnitzky, Phil Quick and Jeff Fortuna for their insightful discussions, companionship and laughs. I owe special thanks to the staff of the Electrical and Computer Engineering Department, especially to Cheryl Gies and Barb MacDonald for their exceptional care and resourcefulness. Last but not least, I would like to thank Christine Yang for everything she has provided me.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

CORBA	Common Object Request Broker Architecture
DME	Depth Map Extractor
FBSM	Feature Based Stereo Matching
fps	frames per second
$H_{\Pi \to \Gamma}$	Homography from plane Π to plane Γ
IBRM	Image-Based Rendering and Modeling
IBSM	Intensity Based Stereo Matching
IDL	Interface Definition Language
IMS	Information Management Server
MBS	Multi Baseline Stereo
NOW	Network of Workstations
OO	Object Oriented
ORB	Object Request Broker
$\tilde{\mathbf{P}}$	Homogeneous camera projection matrix
R_x, R_y, R_z	Euler rotation angles
SSD	Sum of Squared Differences
SSSD	Sum of SSD
VACF	Vision Application CORBA Framework
$\mathbf{x}$	2D $(x, y)^T$ or 3D $(x, y, z)^T$ vector
$\tilde{\mathbf{x}}$	homogeneous coordinate representation of vector $\mathbf{x}$.

Chapter 1

Introduction

1.1 Computer Vision

"Vision is a process that produces from images of the external world a description that is useful to the viewer and not cluttered with irrelevant information."

- D. Marr and H.K. Nishihara

The projection of light rays upon the retina presents our visual system with an image of the three dimensional environment, that is inherently two dimensional. However, our interactions with the environment incorporate an understanding of the 3D structure contained within it, suggesting that our visual system reconstructs a 3D representation of the scene from the 2D binocular images. The human visual system recovers depth of objects from their small positional differences (or disparities) in the images on the retina. This remarkable ability of 3D perception is clearly demonstrated through random-dot stereograms developed by Bela Julez, an example of which is shown in Fig. (1.1). Although each monocular view consists of random dot patterns, when fused binocularly, the images yield the impression of a square floating above a background.

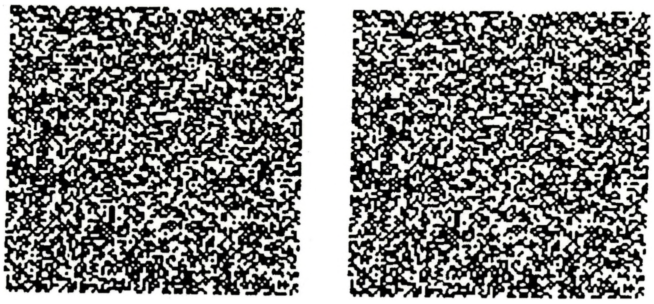


Figure 1.1: A random-dot stereogram of a square[26].

The random dot stereograms propound that our sense of depth perception relies heavily upon the disparity between stereo images. This simplistic notion has fueled decades of research in computer vision to have computers imitate this innate ability of humans.

Initially the research was motivated by autonomous robot navigation, aeriel photogrammetry and to further understand the human visual cognition system. In such applications, a complete model of the underlying 3D surface structure is not necessary. For example, in autonomous robot navigation, the scene need not be represented with complete 3D models, but rather with depth estimates of surfaces facing the robot. Such information is incorporated in a *depth map* which is a $2\frac{1}{2}$D representation of the scene, i.e. depth estimates for all or some pixels in an image of the scene. Figure (1.2) shows an example of a depth map, where darker regions represent surfaces farther away from the camera.

Over the years, the range of applications requiring 3D scene information has grown to include medical imaging, object measurement and object inspection. In recent years, there has been an increasing demand for 3D surface information of a scene in multimedia applications such as telepresence, virtual reality, computer-aided-design (CAD), cinematography and console/computer games. In such applications, real and virtual worlds are often combined to create a new *augmented reality* and require novel

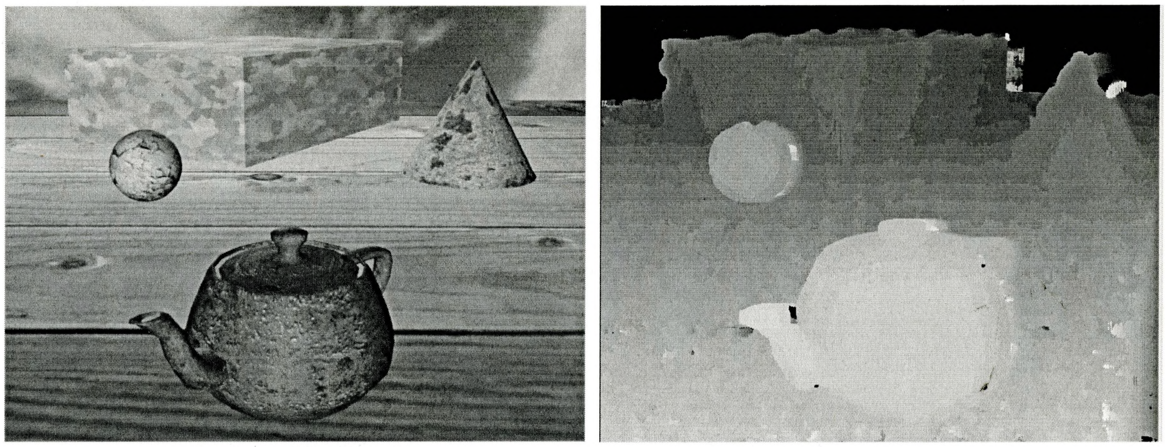


Figure 1.2: A depth map of a synthetic scene. Surfaces with at greater distance from the camera are shown with darker grey-level intensities

views of a real world scene to be synthesized. To provide a truly immersive experience in augmented reality, it is necessary that accurate depth information of the real world scene be known to allow for flawless merging with synthetic data. These applications not only require dense depth information of the real world scenes, but also require surface properties, such as texture and opacity, to facilitate the rendering of realistic visual environments.

In general, multimedia applications requiring 3D computer vision can be broken down into four primary processes: *scene capture, 3D surface extraction, 3D model generation* and *model rendering.* Scene capture digitizes the real world scene, while the 3D surface extraction process builds depth estimates of the scene based on the digitized images. This depth data is used to construct 3D models of the scene in the next process. Finally model rendering concerns itself with the photorealistic rendering of these constructed models for the application.

Three-dimensional surface extraction of the scene is the principal component of such 3D multimedia applications. Although the latter processes of 3D model generation and rendering are important to complete the illusion of realism, the perception of reality will largely depend upon the performance of the 3D surface estimation.

1.2 Existing Systems

The task of extracting the 3D scene geometry in computer vision can be solved using *active* or *passive* techniques. In the active vision techniques, controllable energy (e.g. laser, ultrasound waves) may be projected within the scene, and the differences between the sender and receiver are used to calculate depth maps [38], [27]. Other methods involve direct measuring techniques (e.g. Coordinate Measuring Machine) which provide sparse, yet very accurate depth maps.

Some active vision techniques use controlled lighting to create known surface descriptors (e.g. grids, patterns) and infer the depth of surfaces by triangulating the position of the projected features [23]. Other active techniques include the analysis of illumination variation (*photometric stereo analysis*), variations of focus (*shape from focus/defocus*) and sequence of images (video) (*temporal stereo analysis*). Active vision systems may incorporate a combination of approaches that work cooperatively to determine scene depth [42]. However, active vision techniques for the extraction of 3D scene geometry cannot be used in environments where lighting cannot be controlled (e.g. outdoor scenes) and in environments where the projection of energy is not feasible (e.g. military applications). In such circumstances, passive techniques must be used to gather depth information.

Passive techniques use two or more images of the scene illuminated with ambient lighting to extract the underlying 3D scene geometry. This property is highly desirable in augmented reality applications where the actual scene properties are crucial and must be preserved. A complete overview of passive techniques for 3D surface reconstruction is provided in §2.5.1. In the following section, an overview of existing passive vision systems that are used in multimedia applications is presented.

The Vision and Autonomous System Center at Carnegie Mellon University, Pittsburgh has built an excellent example of such a system [34], [40]. The system utilizes numerous omni-directionally distributed cameras to capture dynamic, real-world events. The arrays of cameras are mounted along the perimeter of the room [19] or geodesic dome [35] to view the scene from all directions. The cameras are calibrated to provide accurate pose information, and the multiple-baseline stereo (MBS) algorithm [32] is used to extract dense depth maps of the scene from each camera. The depth maps calculated from each camera are merged to provide a complete volumetric model of the scene. This is further converted into a 3D model (polygonal mesh) through iso-surface extraction. To improve the depth estimation, the model is reprojected into virtual cameras corresponding to the original physical cameras. From this, object silhouettes and initial depth estimates for each pixel in each camera are calculated. Using these initial depth estimates to limit the range of the searched disparity and ensuring edges obtained from the silhouettes (which are modified by a human operator to improve visual accuracy) are preserved, the MBS algorithm is executed again to obtain the final depth maps. These error-reduced depth maps are once again combined and tessellated as before to provide a refined 3D model of the scene. Finally, these models are texture mapped to provide a complete, photorealistic, 3D representation of the real world scene, which can be combined with other virtual or real scenes.

From the above discussion, it is evident that system is extremely computationally intensive. It is due to this reason, that the system analyzes pre-recorded, dynamic, real-world events using special recording hardware [29], [19]. The system also requires large storage media for the digitized images and intermediate processing.

The MBS algorithm [32], (§2.6) has been used in various systems to provide depth maps [22], [37], [8], [23], [44] and has been developed into systems capable of producing depth maps at 30Hz. Oda et. al. [30] demonstrated a system capable of producing depth maps at 30Hz with 6 cameras, while it merged synthetic and real world at a rate of 15Hz. The system employs special hardware (such as C40 DSP arrays, ALUs and pipeline registers) to accomplish this task. The system imposes limits on the arrangement of the cameras, the size of the images and disparity range to simplify the computational complexity of depth maps.

Similarly the system described by Kang et. al. in [23] produces depth maps at video rate (30Hz). The system utilizes 4 cameras in a converging configuration to maximize the viewpoint overlap among the cameras. However, due to approximations used to simplify the calculations, the vergence angles between the cameras are assumed to be small. It utilizes an 8×8 matrix of iWarp cells, each containing 20 MFLOPS computation engine, low latency communication engine and 16 MB DRAM. Furthermore, to recover accurate and dense depth maps, a sinusoidally varying pattern is projected onto the scene to increase local intensity variation. This limits the extension of the existing system to augmented reality applications, since the artificial lighting masks the true texture of the surfaces in the scene.

To overcome some of the computational requirements for creating depth maps for large scenes Kang et al. in [22] describes an omnidirectional MBS algorithm. The panoramic view of the scene is obtained by constructing cylindrical images from sequences of images taken with a camera rotated 360^o about the vertical axis. Depth extraction is carried out using multiple cylindrical panoramic images through simple stereo techniques. This allows the entire depth map of the entire scene to be constructed without the need to merge multiple depth maps from different view points. The complete model of the scene is constructed and texture mapped with the cylindrical images as a final step. The system is limited by the method of acquiring images of the scene which may not also be possible in all circumstances (such as dynamic events).

Due the increasing interest in videoconferencing, many 3D computer vision systems geared towards these applications have arisen [17], [31]. Due to the specifics of the application, the systems make certain assumptions (e.g. uniform backgrounds, human subjects) and use a priori knowledge (e.g. 3D humanoid models, tracking of facial features) about the scene to guide it in the depth extraction process.

Recently the convergence of computer vision and computer graphics has produced a new subfield known as *Image-Based Rendering and Modeling* (IBRM) [21], [20], [24], [5]. IBRM methods render new views of a scene based on the photometric and geometric information recovered from a number of images of a static scene. Using classical computer vision techniques (e.g. stereo, structure from motion, projective geometry) such methods reproject or interpolate the existing images to synthesize new views. The photorealism of the synthesized views largely depend on the quantity of input images and can thereby become computationally expensive. The quality of the novel views is acutely related to the number of cameras and their placement in the scene. The pre-processing of images required to generate novel views may limit these methods to static scenes.

1.3 Objective and Approach

The visual rendering of 3D surface geometry of real-world scenes is a complex task with stringent requirements. The scene capturing process must adequately capture the dynamics and features within the scene. The estimation of depth within the scene, 3D model generation and finally texture mapped rendering of the these models must provide the degree of accuracy required to convey the illusion of photorealism necessary in the application. These processes require prodigious amounts of computational power. Multimedia applications utilizing 3D computer vision have the further constraint of having to perform in real-time. The exact definition of real-time would vary upon the timing requirements of each application. As with traditional approaches in computer vision, the norm in building such systems is to utilize specialized hardware. Such an approach is not only cost prohibitive, but the resulting systems are predominantly incapable of adapting to new environments due to the inherent assumptions made during their design and construction. Furthermore, there is a large cost of time, energy and finances to evolve such systems over time.

As predicted by Moore's Law, the computing power of standard workstations has made remarkable gains over the recent years. Following suit, networks connecting these workstations have become commonplace and capable of transmitting gigabytes in seconds. Accompanied by their greater affordability, these two resources can provide a practical alternative to specialized hardware solutions in computer vision. The adoption of such general purpose resources inevitably increases the complexity of the software components in the system. However, software components are easier to maintain and evolve and thereby increase the agility of the final system to adapt to changing environments. Furthermore, such systems simplify the evaluation of different techniques.

The networks of workstations (NOW) [1] has given rise to a new paradigm of distributed computing. Such software systems comprise of individual components that work in parallel to solve a complex task. Since such systems can aggregate the power of millions of computers (e.g. SETI project at University of Berkeley), they provide an attractive alternative to massively-parallel processor (MPP) architectures that would be traditionally employed in these environments.

The motivation for the project presented in this thesis stems from these observations. By the novel and unique pairing of ordinary hardware components with distributed software, the project aims to provide insight on the functionality and feasibility of the use of such systems in computer vision. Due to the high demands of resources in 3D scene reconstruction, such an application is a prime candidate for testing such a system. As a result, a system to extract dense depth maps has been built whose architecture has been guided by the strict timing requirements of multimedia applications.

To allow for the conclusions drawn from the project to be valid for a broad range multimedia systems, it is important to build a system with very few constraints. The system is designed to accommodate real world scenes containing arbitrary objects, without any knowledge regarding object genus. To allow for future adaptations for novel view rendering, the system was limited to employing passive stereo techniques (discussed further in §2.5.1). To avoid training and learning, knowledge based or fuzzy logic systems are not employed since they would potentially limit the scope of the system.

Standard off-the-shelf frame grabbers and cameras are utilized. These cameras do not have any enhancements such as zoom, auto iris and pan/tilt. The cameras are arbitrarily placed within the scene and strongly calibrated. This eliminates the need for expensive stereo rigs, and allows the scene to be viewed omnidirectionally from arbitrary viewpoints. Any two cameras having overlapping views can be chosen as a stereo pair, allowing for a single camera to be used in multiple stereo pair configurations to provide dense depth maps for all regions.

The proposed system lays the groundwork for future study and development of distributed computer vision systems.

1.4 Thesis Organization

Chapter 2 details the multi-view geometry of 3D scenes and describes the techniques used within the various components of the system. Included are the discussions for camera calibration, image rectification and depth map generation. Justification for the each approach is provided in context of the project motivation and goals. Chapter 3 describes the software developed for the system. This includes an overview of the underlying distributed software framework and discussion of all the software components that the comprise the system. Chapter 4 provides performance results of the system, providing insight onto its bottlenecks and capabilities. Finally, Chapter 5 draws conclusions based on the presented information and provides possible future directions of research in the area.

Chapter 2

Multiple View 3D Surface Reconstruction

The extrapolation of depth of a scene viewed from two or more views is possible due to the inherent proportional differences in the mappings of common scene points onto the spatially separated projective surfaces. This chapter explores and analyzes this relationship, establishing notation and algorithms.

2.1 Basic Stereo Geometry

To develop the methodology for calculating depth maps from multiple arbitrary views, a simple stereo view setup is first presented. Consider a scene viewed by two perfect pinhole cameras as shown in Fig. (2.1). The world coordinate system is aligned with Camera_1, with $\mathbf{C_1}$ as the origin, and the z-axis as the principal axis of Camera_1. The optical center $\mathbf{C_2}$ of Camera_2 is translated along the x-axis by the B, defining the *baseline*. The principal axes of the cameras are directed towards the scene and are parallel to each other. The cameras have the same focal length f and their image planes are coplanar and parallel. Under such constrained conditions, the epipolar lines of the images coincide with the horizontal scan lines, with the epipoles at infinity.

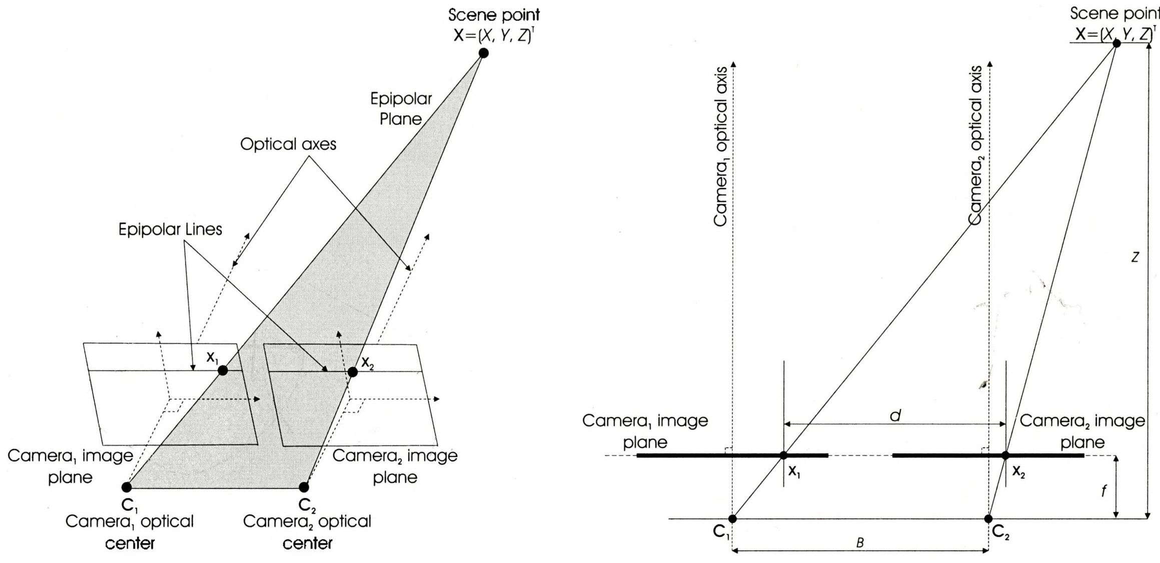


Figure 2.1: Coplanar Stereo Geometry

Thus, a scene point $\mathbf{X} = (X, Y, Z)^T$ is projected onto two corresponding image points $\mathbf{x_1}$ and $\mathbf{x_2}$ given by:

$$\mathbf{x_1} = \begin{pmatrix} u_1 \\ v \end{pmatrix} = \frac{f}{Z}\begin{pmatrix} X \\ Y \end{pmatrix} \qquad \mathbf{x_2} = \begin{pmatrix} u_2 \\ v \end{pmatrix} = \frac{f}{Z}\begin{pmatrix} X - B \\ Y \end{pmatrix} \tag{2.1}$$

The distance between the corresponding image points, or *disparity* $\mathbf{D}$, is:

$$\mathbf{D} = \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} = (\mathbf{x_1} - \mathbf{x_2}) = \begin{pmatrix} \frac{fB}{Z} \\ 0 \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix} \tag{2.2}$$

Consequently, the distance Z or *depth* from $\mathbf{C_1}$ to the scene point $\mathbf{X}$ is defined by:

$$Z = \frac{fB}{d} \tag{2.3}$$

As a result, to obtain a complete depth map, the disparity $\mathbf{D}$ must be measured for each pair of corresponding points in a stereo pair image. Due to the discrete nature of digital images, the disparity values are limited to integers, unless disparity calculations are computed to sub-pixel accuracy using special algorithms. It should be noted from Eq. (2.3) that the calculated depth is directly proportional to the baseline of the cameras. Hence, a large baseline would provide a higher degree of accuracy in the measurement of Z. However, a larger baseline increases the likelihood of false matches, since the overlapping regions of the stereo pair decrease. Additional error in the depth estimation is also introduced by the change in perspective of each camera and occlusion. Wide-baselines also increase the computational load, since greater number of disparities must be searched. Shorter baselines make the search for corresponding points more robust, however the depth estimates are not very accurate. The various depth estimation techniques as a result differ in the way they deal with this fundamental tradeoff between ease of matching and accuracy of depth estimation.

The calculation of depth maps using coplanar views simplifies the task of finding corresponding pixels to a 1D search. However, such a configuration imposes gross limitations on the placement of cameras and thus the flexibility and practicality of the use of such a system. Arbitrarily placed cameras expand the correspondence search from 1D to 2D. This increases the complexity of the system, making matching more difficult, time consuming and error prone. In order to avoid these issues, stereo images taken from arbitrary orientations are *rectified* i.e., resampled in order to produce a pair of images that have epipolar lines corresponding to image rows. In order to perform rectification, the internal and external camera parameters must be known to define the new homography. Homography defines the relationship (in homogeneous coordinates) between two images by an eight parameter perspective equation. The camera parameters can be found through either strong or weak *camera calibration*, from which the homography between the original and rectified image can be calculated. The aforementioned processes are described in the following sections.

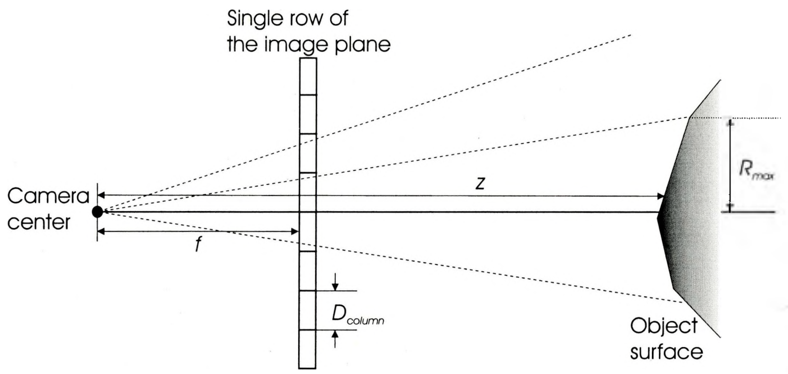


Figure 2.2: Perfect Pinhole camera with a CCD sensor matrix

2.2 Theoretical limitation

The accuracy of 3D surface reconstruction via image acquisition is fundamentally limited by the properties of the cameras. In particular, the spatial resolution of an extracted object surface is directly dependent upon the physical distance between the object and the cameras as well as the focal lengths of the cameras. Furthermore, the pixel resolution of the CCD sensor matrix limits the detail captured of the object surface.

The maximum resolution of the surface reconstruction can be calculated under the assumption of a perfect pinhole camera, with a CCD sensor matrix, as shown in Fig. (2.2). Figure (2.2) depicts a single row of the CCD camera with focal length f, comprising of individual CCD elements of width D_{column}. Given a point on the object surface which is a distance z from the optical center, it holds that:

$$\frac{D_{column}}{f} = \frac{R_{feas}}{z} \tag{2.4}$$

Thus, the theoretical limit of the maximum resolution R_{max} is given by:

$$R_{max} = \frac{D_{column} \cdot z}{f} \tag{2.5}$$

For the cameras used within the setup, $f = 16$ mm and $D_{column} = 7.6\mu m$, $R_{max} = 0.475$ mm for objects 1 m away from the optical center of the camera.

The theoretical limit of surface reconstruction from images can be improved through sub-pixel techniques, such as the iterative technique described by Okutomi and Kanade [32]. However, such an approach inevitably increases the computational requirements.

2.3 Camera Calibration

Since the system presented in this thesis places no restrictions on camera pose and characteristics, the multi-camera setup must be calibrated. Camera calibration establishes the projection of the 3D world coordinates to the 2D image coordinates, allowing 3D information to be inferred from the 2D images. Depending upon the accuracy required by application, a model of the camera is assumed which describes image formation within the camera. Since these parameters are usually technically impossible or not feasible to be measured directly, the calibration problem is thus to compute the numerical parameters for a given camera model. Camera calibration techniques can be classified roughly into two categories: photogrammetric calibration and self-calibration.

Photogrammetric calibration (or "strong" calibration) of a camera is performed with the aid of a special calibration object whose 3D geometric measurements are known very precisely. The calibration object is usually planar, which undergoes precise translation and rotation, or a well defined 3D object. Such methods recover the complete Euclidean structure of the scene [39], [28], [43].

Self calibration (or "weak" calibration) methods do not use any special calibration objects. Images of a static scene taken from multiple viewpoints are used to calculate the parameters of the camera [25], [9]. Such methods recover the camera properties and projective scene geometry, however the exact Euclidean space is not extracted without additional scene or camera information. Due to the numerous parameters that need to be estimated, reliable results may not be always found. Moreover, all depth measurements are relative, which introduces complexity when combining the depth data from different viewpoints.

The cameras used in the system are strongly calibrated since there exist numerous robust methods for strong camera calibration. Furthermore, this approach is necessary in order to combine the depth estimates from multiple stereo camera pairs and viewpoints into a common depth map. The calibration feature detection is simplified in strong calibration since a calibration target is used. Moreover, the complexity involved in the estimation of the camera parameters in strong and weak calibration is relatively similar, leaving little to be gained through weak calibration.

Both *intrinsic* and *extrinsic* parameters of a camera must be computed. The intrinsic parameters of the camera define how a point in the camera coordinate system is mapped to the image. This is dependent on the internal geometric and optical characteristics of the camera (e.g. effective focal length). The extrinsic parameters of the camera however, define the mapping between the points in the world coordinate system and the camera coordinate system. The extrinsic parameters of the camera model thus describe the pose of the camera relative to another coordinate system (such as the world coordinate system) in 3D space.

The cameras used in the experimental setup are characterized by the Tsai camera model [39]. The Tsai camera model describes the camera as a perfect pinhole camera combined with the radial lens distortion and image scanning parameters. The camera model contains eleven parameters that are obtained through direct calibration to explicitly define the camera, namely:

Intrinsic Parameters

f :	effective focal length in mm.
κ_1 :	1st order radial lens distortion.
$(C_x,\ C_y)$:	the row and column image coordinates respectively for the radial lens distortion center (in pixels).

s_x : horizontal uncertainty scale factor introduced by the timing error of the acquisition hardware.

Extrinsic Parameters

$$\mathbf{R} \equiv \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} : \text{ rotation of the camera axes.}$$

$$\mathbf{T} \equiv [T_x, T_y, T_z]^T : \text{ translation of the camera origin.} \tag{2.6}$$

The calibration technique by Tsai [39] was used in the implemented system because of its frequent application in computer vision applications and mature software tools [41]. The method requires at least seven non-coplanar calibration points which have been accurately determined in an arbitrary but known geometric configuration.

The Tsai camera model can be described with the following equations. The rigid body transformation of a point $\mathbf{X_W} = (X_W, Y_W, Z_W)^T$ in the object world coordinate system to point $\mathbf{X_C} = (X_C, Y_C, Z_C)^T$ in the camera coordinate system is:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = \mathbf{R} \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} + \mathbf{T}, \tag{2.7}$$

where $\mathbf{R}$ and $\mathbf{T}$ are the rotation and translation matrices respectively. The point $\mathbf{X_C}$ in the 3D camera coordinate system is transformed to the undistorted image coordinate $\mathbf{x_u} = (x_u, y_u)^T$ by pinhole camera model perspective projection:

$$\begin{bmatrix} x_u \\ y_u \end{bmatrix} = \begin{bmatrix} f\frac{X_C}{Z_C} \\ f\frac{Y_C}{Z_C} \end{bmatrix} \tag{2.8}$$

The radial distortion is modeled by:

$$\begin{bmatrix} x_d + D_x \\ y_d + D_y \end{bmatrix} = \begin{bmatrix} x_u \\ y_u \end{bmatrix} \tag{2.9}$$

where $\mathbf{x_d} = (x_d, y_d)^T$ are the true distorted image coordinates and:

$$\begin{bmatrix} D_x \\ D_y \end{bmatrix} = \begin{bmatrix} x_d(\kappa_1 r^2 + \kappa_2 r^4 + ...) \\ y_d(\kappa_1 r^2 + \kappa_2 r^4 + ...) \end{bmatrix} \tag{2.10}$$

$$r = \sqrt{x_d^2 + y_d^2}.$$

The distorted (or real) image point $\mathbf{x_d} = (x_d, y_d)^T$ is transformed into the actual image buffer coordinates $\mathbf{x_b} = (x_b, y_b)^T$ by:

$$\begin{bmatrix} x_b \\ y_b \end{bmatrix} = \begin{bmatrix} \frac{s_x x_d}{d'_x} + C_x \\ \frac{y_d}{d_y} + C_y \end{bmatrix} \tag{2.11}$$

where

(x_b, y_b) : row and column numbers of the pixel in the image buffer in memory,

(C_x, C_y) : row and column numbers of the center of the image buffer (principal point),

$d'_x = d_x \frac{N_{cx}}{N_{fx}}$,

d_x : center to center distance between adjacent sensor elements in X direction,

d_y : center to center distance between adjacent sensor elements in Y direction,

N_{cx} : number of sensor elements in the X direction, and

N_{fx} : number of pixels in a line as sampled by the computer.

The last four of these parameters are usually obtained from the specifications of the imaging device provided by the manufacturer. The following 7 steps describe the calibration procedure in [39] for non-coplanar calibration points.

Step 1: Calculation of the distorted coordinates from the image buffer coordinates

Initially the calibration object containing non-coplanar calibration features are captured to an image buffer. Next, the row and column positions ($\mathbf{x_b}$) for all visible calibration features are determined to sub-pixel accuracy. The calibration object and mark extraction procedure is described in §3.2.2. Assuming that the principal point (C_x, C_y) is identical to the image buffer center and scaling factor s_x is set to one, $\mathbf{x_d}$ is determined for each extracted feature point in the image buffer using Eq. (2.11). These assumptions are later removed.

Step 2: Calculation of seven parameters for the transformation of sensor coordinates into world coordinates

A calibration point $\mathbf{X_W}$ in the world coordinate system is imaged to the distorted image coordinate $\mathbf{x_d}$ characterized by the following equation:

$$x_d = \begin{bmatrix} y_d X_W & y_d Y_W & y_d Z_W & y_d & -x_d X_W & -x_d Y_W & -x_d Z_W \end{bmatrix} \mathbf{L} \tag{2.12}$$

where

$$\mathbf{L} \equiv \begin{bmatrix} \frac{r_1 s_x}{T_y} & \frac{r_2 s_x}{T_y} & \frac{r_3 s_x}{T_y} & \frac{T_x s_x}{T_y} & \frac{r_4}{T_y} & \frac{r_5}{T_y} & \frac{r_6}{T_y} \end{bmatrix}^T \text{for } T_y \neq 0. \tag{2.13}$$

Here T_x and T_y are components of the translation vector $\mathbf{T}$, and r_i represent elements of the rotation matrix $\mathbf{R}$. With the number of feature points much larger than seven, an overdetermined system of linear equations can be established and solved for the elements of $\mathbf{L}$.

Step 3: Calculation of $|T_y|$

Let:

$$a_1 = \frac{r_1 s_x}{T_y}, \quad a_2 = \frac{r_2 s_x}{T_y}, \quad a_3 = \frac{r_3 s_x}{T_y}, \quad a_4 = \frac{T_x s_x}{T_y}, \quad a_5 = \frac{r_4}{T_y}, \quad a_6 = \frac{r_5}{T_y}, \quad a_7 = \frac{r_6}{T_y} \,. \tag{2.14}$$

Using Eq. (2.14) and orthonormality property of $\mathbf{R}$, the value of T_y is determined from:

$$|T_y| = \sqrt{a_5^2 + a_6^2 + a_7^2}. \tag{2.15}$$

To determine the sign of T_y, an imaged calibration point $\mathbf{P} = (X_w, Y_w, Z_w)^T$ is chosen such that the corresponding image point $(x_b, y_b)^T$ lies far away from the principal point $(C_x, C_y)^T$. Assuming T_y is positive, $s_x = 1$ and using Eq. (2.14) the following equations are solved:

$$\begin{aligned} r_1 &= \frac{r_1}{T_y} \cdot T_y \ , \\ r_2 &= \frac{r_2}{T_y} \cdot T_y \ , \\ r_4 &= \frac{r_4}{T_y} \cdot T_y \ , \\ r_5 &= \frac{r_5}{T_y} \cdot T_y \ , \\ T_x &= \frac{T_x}{T_y} \cdot T_y \ , \\ x &= r_1 X_W + r_2 Y_W + T_x \ , \text{ and} \\ y &= r_4 X_W + r_5 Y_W + T_y . \end{aligned} \tag{2.16}$$

If the parameters x and x_d as well as y and y_d have the same sign then $sgn(T_y) = +1$, else $sgn(T_y) = -1$.

Step 4: Calculating the value of scaling factor s_x

Since $\mathbf{R}$ is orthonormal and s_x is positive s_x is determined using the equation:

$$s_x = |T_y| \cdot \sqrt{a_1^2 + a_2^2 + a_3^2}. \tag{2.17}$$

Step 5: Calculation of R and T_x

The elements of the 3D rotation matrix $\mathbf{R}$ is given by:

$$\begin{aligned} r_1 &= a_1 \cdot T_y / s_x, \quad r_2 = a_2 \cdot T_y / s_x, \quad r_3 = a_3 \cdot T_y / s_x, \\ r_4 &= a_5 \cdot T_y, \qquad r_5 = a_6 \cdot T_y, \qquad r_6 = a_7 \cdot T_y. \end{aligned} \tag{2.18}$$

where r_i are the elements of $\mathbf{R}$ and a_i are defined in Eq. (2.14).

The last row of **R**, namely $r7$, $r8$ and $r9$ can be calculated by the cross product the first two rows of **R** and using the orthonormal property $r_7^2 + r_8^2 + r_9^2 = 1$. T_x can be determined by:

$$T_x = \frac{a_4 \cdot T_y}{s_x}. \tag{2.19}$$

Step 6: Calculation of approximate values for f and T_Z

By ignoring the lens distortion, the linear equation:

$$\begin{bmatrix} y & -d_y y_b \end{bmatrix} \begin{bmatrix} f \\ T_z \end{bmatrix} = w d_y y_b \ , \tag{2.20}$$

can be formulated for every calibration point where

$$\begin{aligned} y &= r_4 X_W + r_5 Y_W + r_6 \cdot 0 + T_y, \\ w &= r_7 X_W + r_8 Y_W + r_9 \cdot 0 \ . \end{aligned} \tag{2.21}$$

More that two calibration points, yields an overdetermined set of linear equations which can be solved for the unknowns f and T_z.

Step 7: Calculation of exact solutions for f, T_Z and κ_1

The use of standard optimization techniques allow for the accurate calculation of f, T_Z and κ_1. The values of f and T_Z calculated in the previous step act as starting values, while κ_1 is assumed to be zero initially. The undistorted image coordinates of world coordinate point **X** can be given by the perspective projection equations:

$$x_{u1} = f\frac{r_1 X_W + r_2 Y_W + r_3 Z_W + T_x}{r_7 X_W + r_8 Y_W + r_9 Z_W + T_z} \quad y_{u1} = f\frac{r_4 X_W + r_5 Y_W + r_6 Z_W + T_y}{r_7 X_W + r_8 Y_W + r_9 Z_W + T_z} \tag{2.22}$$

Furthermore, they can also be obtained by the radial rectification of the actual projected points by using Eq. (2.9), namely:

$$x_{u2} = x_d(1 + \kappa_1 r^2) \text{ and} \tag{2.23}$$

$$y_{u2} = y_d(1 + \kappa_1 r^2), \text{ with} \tag{2.24}$$

$$r = \sqrt{x_d^2 + y_d^2} \tag{2.25}$$

Thus, an error function $\varepsilon(\kappa_1, f, T_z) = h(x_{u1}, y_{u1}, x_{u2}, y_{u2})$ can be formulated using the set of equations above, and optimized using standard techniques (e.g. steepest descent). The principal point assumption in the first step can be removed by using Eq. (2.11) for the image point and repeating the whole process with the updated (C_x, C_y) to improve the calibration accuracy. A solution can also be obtained by solving the non-linear equation:

$$\begin{aligned} d'_y y_b + d_y y_b \kappa_1 r^2 &= f\frac{r_4 X_W + r_5 Y_W + r_6 Z_W + T_y}{r_7 X_W + r_8 Y_W + r_9 Z_W + T_z} \\ r &= \sqrt{(s_x^{-1} d'_x y_b)^2 + (d_y y_b)^2} \end{aligned} \tag{2.26}$$

Camera Perspective Projection Matrix

A point $\mathbf{X_W} = (X_W, Y_W, Z_W)^T$ in the world coordinates is transformed into an image buffer point (pixel) $\mathbf{x_b} = (x_b, y_b)^T$ through the linear perspective projective matrix $\tilde{\mathbf{P}}$ (given in homogeneous coordinates):

$$\tilde{\mathbf{x}}_\mathbf{b} = \tilde{\mathbf{P}}\tilde{\mathbf{X}}_\mathbf{W} \tag{2.27}$$

where

$$\tilde{\mathbf{x}}_\mathbf{b} = \begin{bmatrix} \tilde{x}_b \\ \tilde{y}_b \\ s \end{bmatrix}, \tag{2.28}$$

$$\mathbf{x}_b = \begin{bmatrix} \frac{\tilde{x}_b}{s} \\ \frac{\tilde{y}_b}{s} \end{bmatrix} \quad \text{if } s \neq 0, \tag{2.29}$$

$$\tilde{\mathbf{X}}_\mathbf{W} = \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix}. \tag{2.30}$$

The matrix $\tilde{\mathbf{P}}$ can be decomposed into:

$$\tilde{\mathbf{P}} = \tilde{\mathbf{P}}_I \tilde{\mathbf{P}}_E \tag{2.31}$$

where $\tilde{\mathbf{P}}_I$ is the projection matrix which transforms the points in camera coordinate system to image pixels and is given by (assuming that the radial distortion of the camera has already been removed and that the camera pixels are square):

$$\tilde{\mathbf{P}}_I = \begin{bmatrix} \frac{s_x f}{d'_x} & 0 & C_x \\ 0 & \frac{f}{d_y} & C_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.32}$$

and $\tilde{\mathbf{P}}_E$ is the projection matrix which transforms the points in the world coordinate system to the camera coordinate system given by:

$$\tilde{\mathbf{P}}_E = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{2.33}$$

where $\mathbf{R}$ and $\mathbf{T}$ are the rotation and translation matrices respectively, defined in Eq. (2.6). Equation 2.32 defines the *horizontal* and *vertical* focal lengths:

$$\begin{aligned} f_h &= \tfrac{s_x f}{d'_x} \quad \text{(Horizontal focal length in pixels)}, \\ f_v &= \tfrac{f}{d_y} \quad \text{(Vertical focal length in pixels)}. \end{aligned} \tag{2.34}$$

The 3×4 projection matrix $\tilde{\mathbf{P}}$ can be written as:

$$\tilde{\mathbf{P}} = \left[\begin{array}{c|c} \mathbf{p}_1^{\mathbf{T}} & p_{14} \\ \mathbf{p}_2^{\mathbf{T}} & p_{24} \\ \mathbf{p}_3^{\mathbf{T}} & p_{24} \end{array}\right] = [\mathbf{P}|\tilde{\mathbf{p}}]. \tag{2.35}$$

$\tilde{\mathbf{p}}$ is the homogeneous image coordinate of the origin. The one-dimensional right null space of $\mathbf{P}$ represents the camera center $\mathbf{C}$. Therefore:

$$\tilde{\mathbf{P}} \begin{bmatrix} \mathbf{C} \\ 1 \end{bmatrix} = 0 \tag{2.36}$$

and

$$\mathbf{C} = -\mathbf{P}^{-1}\tilde{\mathbf{p}}. \tag{2.37}$$

2.4 Image Pair Rectification

Image Pair Rectification is the process of resampling stereo image pairs to produce a pair of images with matched epipolar projections. These projections are such that the conjugate epipolar lines of the images are collinear and run parallel to the x-axis, thereby limiting the disparities between matching points of the images in the x-direction only. This epipolar constraint is of great benefit to stereo matching algorithms, since the correspondence search space is reduced to one dimension, namely the corresponding rows of the rectified images. The rectified images can be thought of as obtained by a new coplanar stereo camera setup, obtained by rotating the original cameras about their optical centers.

In order to produce a rectified pair of stereo images, the rectification procedure by Fusiello et. al [12] is used. This rectification scheme is well suited for multi-camera configurations since the reference camera is rectified in each stereo pair. Image rectification causes the projection of original images on a new common retinal plane. As a result, rectification of images from cameras that have large vergence angles between their optical axes, can suffer from quantization effects if an ill-suited rectification plane is chosen. This would occur if all the multi-camera stereo image pairs are rectified to a common plane such as the plane containing the reference camera image. Under such conditions, the image of the non-reference camera can undergo severe transformation during rectification thereby decreasing the accuracy of stereo matching.

In the approach outlined in [12], the rectifying projection matrices are calculated by the original perspective projection matrices of the camera obtained through strong calibration. The image planes obtained by rectification are coplanar and parallel to the baseline as shown in Fig. (2.3). The operation can be thought of as physically rotating the two cameras about their optical centers such that their optical axes are parallel and perpendicular to their baseline. All constraints necessary to guarantee

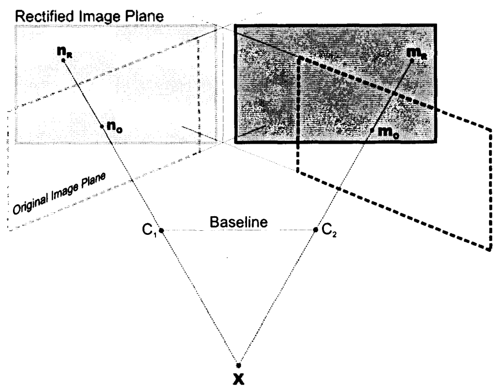


Figure 2.3: Coplanar Rectification

a unique solution are explicitly enforced, resulting in a linear, homogeneous system of equations incorporating explicit quadratic constraints. Furthermore, the camera stereo pair geometry is unrestricted, allowing for arbitrarily placed cameras.

Given that the original projection matrices for a stereo camera pair are given by $\tilde{\mathbf{P}}_{\mathbf{O1}}$ and $\tilde{\mathbf{P}}_{\mathbf{O2}}$, while the new rectifying projecting matrices are given by:

$$\tilde{\mathbf{P}}_{\mathbf{N1}} = \begin{bmatrix} \mathbf{A}_1^T \\ \mathbf{A}_2^T \\ \mathbf{A}_3^T \end{bmatrix} = \left[\begin{array}{c|c} \mathbf{a}_1^T & a_{14} \\ \mathbf{a}_2^T & a_{24} \\ \mathbf{a}_3^T & a_{34} \end{array}\right] \qquad \tilde{\mathbf{P}}_{\mathbf{N2}} = \begin{bmatrix} \mathbf{B}_1^T \\ \mathbf{B}_2^T \\ \mathbf{B}_3^T \end{bmatrix} = \left[\begin{array}{c|c} \mathbf{b}_1^T & b_{14} \\ \mathbf{b}_2^T & b_{24} \\ \mathbf{b}_3^T & b_{34} \end{array}\right] \tag{2.38}$$

the following constraints can be formulated.

Since the rectified projections must share a *common focal plane*, it follows:

$$\mathbf{a_3} = \mathbf{b_3} \quad \text{and} \quad a_{34} = b_{34}. \tag{2.39}$$

Furthermore, the *orientation of the common rectified focal plane* is chosen to lie parallel to the intersection of the two original focal planes:

$$\mathbf{a}_3^T(\mathbf{A}_3^T \wedge \mathbf{B}_3^T) = 0, \tag{2.40}$$

where $\wedge$ is the intersection operator. The conjugate equation $\mathbf{b}_3^T(\mathbf{A}_3^T \wedge \mathbf{B}_3^T) = 0$ is redundant due to Eq. (2.39).

The *position of the optical centers* $\mathbf{C_1}$ and $\mathbf{C_2}$ of the stereo camera pair must remain unchanged:

$$\tilde{\mathbf{P}}_{\mathbf{N1}} \begin{bmatrix} \mathbf{C_1} \\ 1 \end{bmatrix} = \mathbf{0} \quad \text{and} \quad \tilde{\mathbf{P}}_{\mathbf{N2}} \begin{bmatrix} \mathbf{C_2} \\ 1 \end{bmatrix} = \mathbf{0} \tag{2.41}$$

where $\mathbf{C_1}$ and $\mathbf{C_2}$ can be calculated using Eq. (2.37). Equation (2.41) provides the following six linear constraints:

$$\begin{cases} \mathbf{a}_1^{\mathbf{T}}\mathbf{C_1} + a_{14} = 0 \\ \mathbf{a}_2^{\mathbf{T}}\mathbf{C_1} + a_{24} = 0 \\ \mathbf{a}_3^{\mathbf{T}}\mathbf{C_1} + a_{34} = 0 \\ \mathbf{b}_1^{\mathbf{T}}\mathbf{C_2} + b_{14} = 0 \\ \mathbf{b}_2^{\mathbf{T}}\mathbf{C_2} + b_{24} = 0 \\ \mathbf{b}_3^{\mathbf{T}}\mathbf{C_2} + b_{34} = 0. \end{cases} \tag{2.42}$$

Since the purpose of rectification is to *align conjugate epipolar lines*, the vertical coordinate of a 3D point $\mathbf{X_W}$ must be equal under both transformations $\tilde{\mathbf{P}}_{\mathbf{N1}}$ and $\tilde{\mathbf{P}}_{\mathbf{N2}}$, i.e.:

$$\frac{\mathbf{a}_2^{\mathbf{T}}\mathbf{X_W} + a_{24}}{\mathbf{a}_3^{\mathbf{T}}\mathbf{X_W} + a_{34}} = \frac{\mathbf{b}_2^{\mathbf{T}}\mathbf{X_W} + b_{24}}{\mathbf{b}_3^{\mathbf{T}}\mathbf{X_W} + b_{34}}. \tag{2.43}$$

This constraint can be simplified using Eq. (2.39)

$$\mathbf{A}_2^{\mathbf{T}} = \mathbf{B}_2^{\mathbf{T}}. \tag{2.44}$$

The rectified image planes must have *orthogonal $x-$ and $y-$axes*, thus the corresponding planes (given by the first and second rows of the camera projection matrix) must also be orthogonal. Using Eq. 2.44, the constraint can be written as:

$$\mathbf{a}_1^{\mathbf{T}}\mathbf{a_2} = 0 \qquad \mathbf{b}_1^{\mathbf{T}}\mathbf{a_2} = 0. \tag{2.45}$$

The *principal point* (C_x, C_y) is set to $(0,0)$ for each of the rectified image planes and using Eq. (2.39) and (2.44) the following equations are obtained:

$$\begin{cases} \mathbf{a_1^T a_3} = 0 \\ \mathbf{a_2^T a_3} = 0 \\ \mathbf{b_1^T a_3} = 0. \end{cases} \tag{2.46}$$

By keeping the *horizontal and vertical focal lengths* unchanged and using Eq. (2.34) and (2.46) the focal lengths can be obtained by:

$$\begin{cases} ||\mathbf{a_1} \wedge \mathbf{a_3}||^2 = ||\mathbf{a_1}||^2||\mathbf{a_3}||^2 = {f_h}^2 \\ ||\mathbf{a_2} \wedge \mathbf{a_3}||^2 = ||\mathbf{a_2}||^2||\mathbf{a_3}||^2 = {f_v}^2 \\ ||\mathbf{b_1} \wedge \mathbf{a_3}||^2 = ||\mathbf{b_1}||^2||\mathbf{a_3}||^2 = {f_h}^2. \end{cases} \tag{2.47}$$

Finally, the rectification matrices are defined to a scale factor of 1:

$$||\mathbf{a_3}|| = 1 \quad \textit{and} \quad ||\mathbf{b_3}|| = 1 \tag{2.48}$$

All the constraints can be organized to provide four system of equations:

$$\begin{cases} \mathbf{a_3^T C_1} + a_{34} = 0 \\ \mathbf{a_3^T C_2} + a_{34} = 0 \\ \mathbf{a_3^T (A_3^T \wedge B_3^T)} = 0 \\ ||\mathbf{a_3}|| = 1 \end{cases} \tag{2.49}$$

$$\begin{cases} \mathbf{a_2^T C_1} + a_{24} = 0 \\ \mathbf{a_2^T C_2} + a_{24} = 0 \\ \mathbf{a_2^T a_3} = 0 \\ ||\mathbf{a_2}|| = f_v \end{cases} \tag{2.50}$$

$$\begin{cases} \mathbf{a_1^T C_1} + a_{14} = 0 \\ \mathbf{a_1^T C_2} + a_{14} = 0 \\ \mathbf{a_1^T a_3^T} = 0 \\ ||\mathbf{a_1}|| = f_h \end{cases} \tag{2.51}$$

$$\begin{cases} \mathbf{b_1^T C_2} + b_{14} = 0 \\ \mathbf{b_1^T a_2} = 0 \\ \mathbf{b_1^T a_3} = 0 \\ ||\mathbf{b_1}|| = f_h \end{cases} \tag{2.52}$$

with the added equalities:

$$\begin{cases} \mathbf{a_2} = \mathbf{b_2} \\ a_{24} = b_{24} \\ \mathbf{a_3} = \mathbf{b_3} \\ a_{34} = b_{34} \end{cases} \tag{2.53}$$

Each system of linear homogeneous set of equations can be solved under a quadratic constraint written as:

$$\mathbf{Ax} = \mathbf{0} \text{ subject to } ||\mathbf{x}'|| = k, \tag{2.54}$$

where $\mathbf{x}'$ is a vector comprising of the first three components of $\mathbf{x}$, while k is a real valued numeric.

Once the rectifying perspective projection matrices are known, then the homography between the original and rectified image planes can be computed as:

$$\tilde{\mathbf{x}}_\mathrm{N} = \mathbf{P_N P_O^{-1}} \tilde{\mathbf{x}}_\mathrm{o} = \mathbf{H}_{\mathrm{O} \mapsto \mathrm{N}} \tilde{\mathbf{x}}_\mathrm{o} \tag{2.55}$$

where

$$\begin{aligned} \tilde{\mathbf{x}}_{\mathbf{N}} &: \text{New rectified image coordinate} \\ \tilde{\mathbf{x}}_{\mathbf{O}} &: \text{Original image coordinate} \\ \tilde{\mathbf{P}}_{\mathbf{N}} = [\mathbf{P}_{\mathbf{N}}|\tilde{\mathbf{p}}_{\mathbf{N}}] &: \text{Rectifying perspective projection matrix} \\ \tilde{\mathbf{P}}_{\mathbf{O}} = [\mathbf{P}_{\mathbf{O}}|\tilde{\mathbf{p}}_{\mathbf{O}}] &: \text{Original perspective projection matrix} \end{aligned}$$

The homography $\mathbf{H}_{\mathbf{O}\mapsto\mathbf{N}}$ is applied to every pixel in the original image to synthesize the rectified image. Since the integer coordinate values of the rectified image will correspond in general to non-integer values of the original image, the intensity values of the rectified image are therefore computed through bilinear interpolation.

2.5 3D Surface Reconstruction

To accomplish the final task of *3D surface reconstruction*, correspondence between elements that are the projections of the same physical entity among stereo views must be determined *(stereo matching)*. Once correspondence has been established, the depth can be computed *(depth estimation)* for each element using the camera configuration geometry. The depth values can be represented as depth maps or extrapolated to 3D models using mesh structures incorporating boundary or object recognition techniques, which may further be enhanced using texture maps and lighting. In the presented system, the depth values are visually represented using 255 level greyscale depth maps.

2.5.1 Stereo Matching

The process of detecting corresponding image elements that are projections of the same real world surface point between stereo images is commonly known as *stereo matching*. Stereo matching algorithms, differing in their method of image element extraction, can be classified into two general categories. These are feature based and intensity based techniques.

Principles of Feature Based Stereo Matching

In feature based stereo matching (FBSM), first an operator is used to preprocess the images to extract salient features that are stable under different viewpoints of the scene. The matching process then determines correspondence between the detected features in the stereo image pairs based on their attributes. Edges, corners and contours are commonly selected as features to be extracted. Higher level primitives such as ellipses and polygonal regions can also be selected as features in certain controlled scenes. Usually a collection of features are employed in order to increase the regions for which depth can be computed.

A few of the FBSM from the vast literature on such methods are discussed below to illustrate their properties, while a current survey is presented in [16]. Barnard et. al [3] selected centers of highly variable areas in the stereo images as features. A network of nodes corresponding to possible matches is constructed by pairing up each candidate point in one image to all candidate points in the second image within a disparity range. Initial probability estimates of correspondence, based on sum of squares of intensity differences are used to label each possible match. These probabilities are iteratively refined by attributing probabilities which enforce surface consistency (i.e. are associated with nearly the same disparity) with higher weights. The method does not require camera calibration information which is a significant advantage. M. Pilu [33] discusses in a method to determine correspondence among points using the singular value decomposition without using camera calibration information. Although the algorithm does not depend on the selected feature, the author provides examples where "corners" in images were detected and subsequently matched. However the method is very sensitive to errors during the feature extraction process.

Bensrhair et. al [4] define a feature called "declivity" as a cluster of contiguous pixels, limited by two end-points which correspond to two consecutive local extrema of grey-level intensity . The matching algorithm determines correspondence between declivities by evaluating their sum of neighborhood intensity differences and maximizing a non-linear global gain along the entire epipolar line. This can be regarded as finding an optimal path on a 2D search plane defined by the epipolar lines of the stereo pair which maximizes the global gain.

From the above discussion it is evident that FBSM methods generally involve a high degree of complexity in feature extraction and matching. FBSM approaches do not produce dense surface estimates since the extracted features generate a coarse representation of the scene. This however, can increase the speed and accuracy in correspondence analysis due to the significant reduction in ambiguity arising from the diminished number of possible candidates. The correspondence analysis have a high degree of immunity to photometric variations between stereo image pairs since the extracted features represent prominent details of the scene, and their attributes are generally not altered by photometric variations during imaging.

Principles of Intensity Based Stereo Matching

An alternative technique to FBSM is to directly utilize all the grey levels of the pixels to determine correspondence in stereo pair images, consequently called *intensity based stereo matching* (IBSM). Since pixels share individual intensity values, correspondence is determined by a *similarity measure* of *blocks*, i.e. neighborhoods around the pixels (e.g. using n $\times$ n windows), on various values of disparity along the epipolar line. The similarity measure is based on the intensity attributes of the blocks (e.g. mean squared differences (MSD), cross correlation) and various assumptions and constraints (e.g. smooth surfaces, continuity constraints). Each pixel is compared with a number of pixels along the epipolar line, and the disparity providing the optimum similarity measure value is selected. The difference among various IBSM techniques is their similarity measure and a recent survey is contained in [16].

Since the corresponding pixels are determined on the basis of their neighborhoods, the accuracy of the disparity in IBSM approaches innately depend upon the size of the blocks. The larger blocks potentially have a larger variance of intensity, reducing the potential for false matches. However, increasing the size of the window decreases the accuracy of depth estimates since the windows can potentially contain pixels from multiple depths. Therefore, a common technique employed is hierarchical based stereo matching, where disparity estimates from larger blocks is used to guide the estimates for smaller blocks [6], [8].

Another approach is to increase the robustness of the disparity estimates by considering global attributes rather than local ones. A technique described by Cox et al. [18] produces robust results by simultaneously estimating all disparities in a scan line considering the monotonic ordering of the pixels in the stereo image pair and continuity of the disparity estimates. This technique was improved upon by Falkenhagen [7] by performing matching over small blocks instead of pixels and thereby applying a more sophisticated continuity constraint for neighborhoods. Similarly, Roy et al. [36] solve the stereo correspondence problem by formulating it into a finding the maximum flow in a graph. The depth estimation is performed by considering inter- and intra-epipolar line constraints, while explicitly modeling occlusion and surface discontinuities. By evaluating the minimum cut associated with the maximum flow, the disparity surface of the entire scene is evaluated. The theoretical and actual execution times reported by the authors suggest that the approach is computationally intensive.

Pascal Fua [11] describes a fast stereo algorithm in where a normalized MSD of pixels is used to calculate the optimal disparity to sub-pixel accuracy. The correspondence search is performed for each pixel in *both* images of the stereo pairs and the match between pixels is considered valid if and only if both searches yield the same match. The density of the depth maps were increased by a special hierarchical approach and by the use of multiple images. When calculating depth maps with multiple images, the first image is regarded as the reference frame, and the depth map calculated from each pair is combined in the final step. Consequently, the major drawback in the algorithm is the high computational load.

There are many other approaches to solve the stereo matching problem such as temporal stereo, focus and defocus and shape from shading among others. The system presented in this thesis uses a *multiple baseline stereo* developed by Okutomi and Kanade [32] and discussed in §2.6.

2.6 Multiple Baseline Stereo (MBS)

The *multiple baseline stereo* (MBS) [32] is an intensity based stereo matching technique that improves robustness of disparity estimates by computing correspondences between multiple pairs of stereo images with varying baselines. The motivation of the approach is derived from Eq. (2.3) which can be re-written as:

$$\frac{d}{fB} = \frac{1}{Z} = \zeta \tag{2.56}$$

Disparity calculations from each stereo image pair differ among stereo camera pairs due to the change in camera parameters and hence cannot be directly used across multiple images. Subsequently, Eq. (2.3) is reformulated into Eq. (2.56) to provide a measure that can be performed across multiple image pairs. According to Eq. (2.56), the stereo matching can be performed with respect to the *inverse depth* ζ which has an immutable definition across all stereo pairs formed with a common reference image. The parameter ζ is independent of different disparities, baselines and focal lengths of the cameras. Thus, similarity measures for a pixel in the reference image can be computed across multiple stereo image pairs, with varying baselines, and combined according to inverse depth. The MBS algorithm uses sum of squared difference (SSD) values as a similarity measure between pixels defined as:

$$e_{d_i}(x, y, d_i) \equiv \sum_{a,b \in W} \sum (I_{Ref}(x+a, y+b) - I_{B_n}(x+a+d_i, y+b))^2, i = \{0, 1, 2...\} \tag{2.57}$$

where

(x, y) : pixel coordinate in reference image for which correspondence is being searched

d_i : i^{th} candidate disparity

$e_{d_i}(x, y, d_i)$: SSD value of reference pixel coordinate (x, y) at disparity d_i

$\sum\sum_{a,b\in W}$: summation over the window W

$I_{Ref}(x, y)$: Intensity function of reference image

$I_{B_n}(x, y)$: Intensity function of n^{th} stereo pair image

Since disparity $d = Bf\zeta$, the SSD with respect to the inverse distance ζ is given by:

$$e_{\zeta_i}(x, y, \zeta_i) \equiv \sum\sum_{a,b\in W}(I_{Ref}(x+a, y+b) - I_{B_n}(x+a+B_n f\zeta_i, y+b))^2, i = \{0, 1, 2...\} \tag{2.58}$$

where

$e_{\zeta_i}(x, y, \zeta_i)$: SSD value with respect to inverse depth ζ_i

$\zeta_i = \frac{1}{Z_i}$: ith candidate inverse depth

B_n : Baseline between the reference image and nth stereo pair image

f : focal length

The estimates from N stereo pairs are combined to produced the sum of SSDs (SSSD) for each inverse depth ζ:

$$SSSD(x, y, \zeta) \equiv \sum_{i=1}^{N}\sum\sum_{a,b\in W}(I_{Ref}(x+a, y+b) - I_{B_i}(x+a+B_i f\zeta, y+b))^2 \tag{2.59}$$

The optimal inverse depth estimate $\widehat{\zeta}$ at position (x, y) is given by the ζ value from a defined range which minimizes the SSSD function:

$$\widehat{\zeta} = MIN(SSSD(x, y, \zeta_i)) \; \forall i \tag{2.60}$$

The SSSD has the property of exhibiting an unambiguous and a more pronounced minimum at the correct inverse distance compared to the SSD. This is due to the fact that stereo matching along epipolar lines (using the inverse depth) using SSD may yield multiple matches. However, using SSSD to combine the error values across multiple images would accentuate the correct depth estimate.

The MBS algorithm effectively minimizes the global error by integrating the accuracy of depth calculations obtained by stereo pairs with wide-baselines, with the robustness of stereo matching with stereo pairs of smaller baselines. This contrasts other methods that produce estimates based on consistency checks or filtering of intermediate values. It avoids the need for sophisticated intermediate decisions and the compounding of errors introduced by the propagation of estimates through multiple levels. The technique has the further advantage of handling areas that might be occluded in one or more views and to repetitive patterns to a certain degree. The algorithm is a fast and linear approach that has been implemented by in real-time [30] [23]. The algorithm readily lends itself to a parallel realization since the stereo matching of the each pixel is independent of the rest. As a result, the algorithm can be adapted quite readily in the proposed distributed software architecture.

Since the cameras are allowed to be in arbitrary positions and orientations, the images are rectified (§2.4) before stereo matching via the MBS algorithm. Due to the image rectification, the optical axis of the reference camera is rotated and thus, all measurements of depth are computed parallel to the optical axis of the rectified reference camera. Consequently, in order to use ζ to search across multiple images, depths from the reference camera coordinate system must be transformed to the rectified reference camera coordinate system. Since the rectification method only causes a rotation of the reference camera about the optical center, the rectified coordinates can be written as:

$$\mathbf{P_{Rect}} = \begin{pmatrix} X_R \\ Y_R \\ Z_R \end{pmatrix} = \mathbf{MP_{Orig}} \tag{2.61}$$

where $\mathbf{P_{Rect}}$ is the rectified coordinates of the point $\mathbf{P_{Orig}} = (X_O, Y_O, Z_O)^T$ in the non-rectified (original) reference camera coordinate system . $\mathbf{M}$ is the rotation matrix that transforms points in the original reference camera coordinate system to the rectified reference camera coordinate system. Recall that the measurements of depth are performed in the original reference camera coordinate system. From the pinhole camera model:

$$\mathbf{P_{Orig}} = \frac{Z_O}{f_O}\begin{pmatrix} x_O \\ y_O \\ f_O \end{pmatrix} \tag{2.62}$$

where (x_O, y_O) is the pixel of the original reference image, and f_O is the focal length of the original reference camera. Thus, using Eq. (2.3) the disparity in the rectified image stereo pair is given by:

$$d_R = \frac{B_R f_R}{Z_R} \tag{2.63}$$

Using Eq. (2.62) this can be written as:

$$d_R = \frac{B_R f_R f_O}{\mathbf{m_3}(x_O, y_O, f_O)^T}\zeta \tag{2.64}$$

where $\mathbf{m_3}$ is the third row of $\mathbf{M}$ and $\zeta = Z_O^{-1}$. Since the rectification procedure does not change the optical centers, $B_R = B_O$. For a fully parameterized camera, Eq. (2.64) can be written as:

$$d_R = \frac{B_R f_R}{\mathbf{m_3}(\frac{x_O - C_x}{f_h}, \frac{y_O - C_y}{f_v}, 1)^T}\zeta \tag{2.65}$$

Equation (2.65) relates the depth Z_O in the unrectified reference camera to the disparity in the rectified image pair. This allows for formulation of disparity search space for a given depth across multiple stereo image pairs.

Constraints and Assumptions

Since MBS is an IBSM approach two important assumptions are made about the scene being extracted. First, it inherently assumes that the surfaces in the scene are Lambertian and highly textured, allowing matching to be reliably made from the intensity statistics within a window of the scene. As a result, the algorithm produces depth estimates with low precision for regions of the scene with little texture. In these regions, the constructed windows correspond equally well over a wide range of depths.

Secondly, the MBS technique assumes the scene surfaces are planar and parallel to the coplanar stereo camera pair. This is direct consequence of representing the measure of depth as the perpendicular distance (with respect to the image plane) of an object from the optical center of the camera. This assumption fails when windows are constructed within an image that contain pixels from multiple depths. Due to the variation in camera pose, corresponding windows constructed in images from other viewpoints may not have the same intensity value properties due to the perspective distortion. Consequently, the SSSD function may not have the global minimum at the correct depth, and instead the minimum may lie at an arbitrary depth for which random alignment of textures produces the lowest error. This limitation has been successfully overcome in [34], although with extensive complex computational load.

Chapter 3

System Architecture

3.1 Distributed Computing

The increasing affordability and capability of workstations and computer networks have given rise to notion of a Network of Workstations (NOW) [1]. NOW can be employed to solve complex and tedious tasks that were traditionally solved with expensive, monolithic systems. Monolithic systems extensively (and almost exclusively) use special hardware and software architectures to perform parallel processing, or meet computational load requirements. As shown in Fig. (3.1), this rigid monolithic application architecture is justified, since no processing is done on the terminals accessing the central system responsible for the workload. Over the course of technological evolution, these monolithic applications have been broken into a 2-tier client/server architecture, where some processing is done on the access terminals. NOW on the other hand have given rise to a new computing paradigm, namely a distributed or cooperative computing, that can be regarded as a multi-tier client/server architecture (Fig. (3.2)).

In the distributed computing architecture, the application is broken into functional objects, each of which can use the services provided by other objects in the

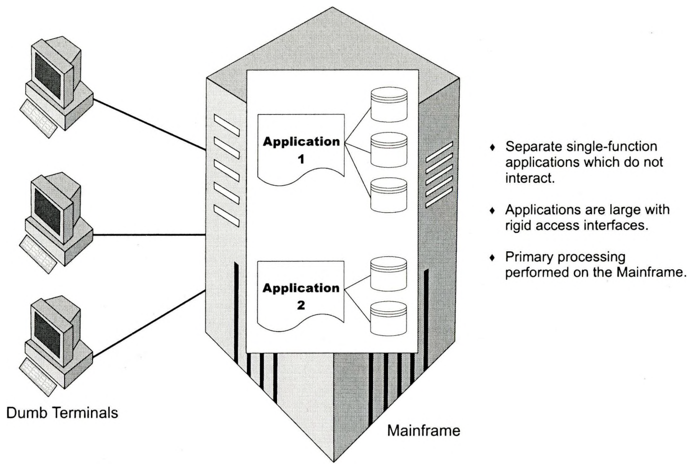

Application 1
Application 2
Separate single-function applications which do not interact.
Applications are large with rigid access interfaces.
Primary processing performed on the Mainframe.
Dumb Terminals
Mainframe

Figure 3.1: Traditional Monolithic computing architecture.

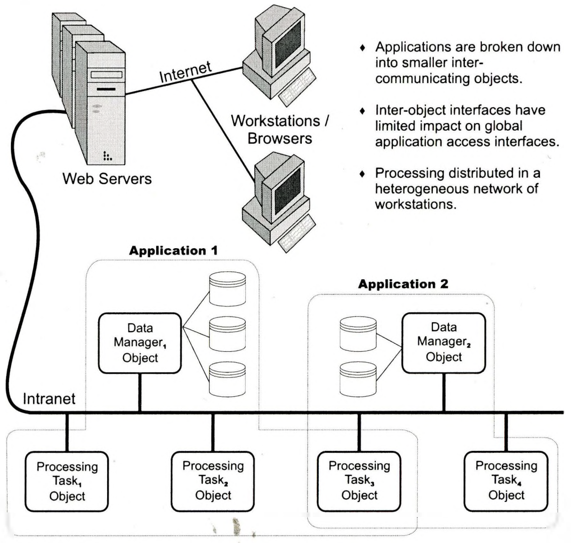

Internet
Workstations / Browsers
Web Servers
Applications are broken down into smaller inter-communicating objects.
Inter-object interfaces have limited impact on global application access interfaces.
Processing distributed in a heterogeneous network of workstations.
Application 1
Application 2
Data Manager₁ Object
Data Manager₂ Object
Intranet
Processing Task₁ Object
Processing Task₂ Object
Processing Task₃ Object
Processing Task₄ Object

Figure 3.2: Distributed computing architecture.

system. Therefore, an object can act both as a client and server. Distributed computing solves a complex task through the notion of "divide and conquer". The complex task is broken into smaller pieces, that can be solved individually. By distributing these smaller tasks over an array of functional units on individual workstations, parallelism is achieved. Due to the decomposition of applications into specific components, the approach encourages (or enforces) the use of Object Oriented software construction techniques, thereby increasing the flexibility, scalability and robustness of the application.

The main advantage to distributed computing is that it allows complex tasks to be solved quickly with affordable resources, while maintaining a scalable platform. The tasks however, are limited to a class of that can be decomposed into smaller units, and solved independently (e.g. calculation of prime numbers, testing of protein strains for drug research and cryptography). Numerous computer vision techniques such as image filtering, segmentation and pattern recognition lie in this class of problems, including methods 3D scene extraction. This property arises from the fact that many algorithms in computer vision are based on the of processing image characteristics that are local and independent.

3.1.1 Common Object Request Broker Architecture

In the distributed computing paradigm, software applications are broken into multiple components (or objects) and distributed over a heterogeneous NOW. Consequently, the efficient exchange of information among these components becomes of primary significance. The Common Object Request Broker Architecture (CORBA) is a reference model that facilitates the communication and inter-operation of distributed components in heterogeneous computing environments. A complete review of CORBA is beyond the scope of this work, and a thus a brief overview follows.

CORBA provides a standard mechanism for defining interfaces between components as well as tools to facilitate the implementation of those interfaces for the

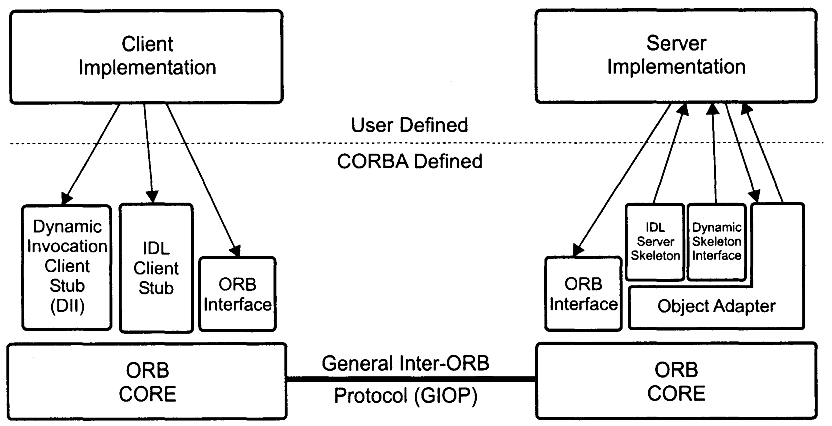


Figure 3.3: Common Object Request Broker Architecture

developer. CORBA defines a structural architecture for the system, based on the client/server paradigm, and is a standard specified by the consortium of over 800 organizations known as the Object Management Group (OMG). The standard allows software vendors to create Object Request Brokers (ORBs) which allow software developers to write distributed software systems. The ORB makes the communication between components transparent with regard to the location on the network, their programming language, operating systems and ultimately their internal implementations. The adherence to a strict standard allows inter-communication of CORBA objects from different vendors and an inter-portability.

CORBA is an example of object oriented (OO) architecture. Through the use of OO mechanisms, it is able to achieve its primary characteristics of reusability and information hiding (through abstraction and encapsulation). This however, does not limit its use to object oriented languages, although CORBA maps particularly well to them. The general characteristics of CORBA are shown in Fig. (3.3). Although CORBA does not have any inherent features for real-time application development, recent developments in CORBA have included the specification of real-time ORBs [14]. CORBA objects can communicate through the ORB with a variety of network protocols, including custom protocols. However, due to the CORBA standard specifications, each ORB inherently supports the TCP/IP network protocol suite.

CORBA was chosen over other alternatives (such as Sockets, DCOM, Java RMI and RPC) since it elegantly extends to a multitude of computing platforms and software languages. It provides a consistent level of abstraction for the interaction of distributed objects, allowing the programmer to solely concentrate on the behaviour of the objects. It is not limited to a certain programming language, operating system or communication protocol. This adds flexibility and adaptability to the presented system, and allows for collaborative research, including across the Internet. Furthermore, the use of an industry standard as the primary infrastructure allows for the research to be applicable to a broad category of practical systems.

3.2 System Overview

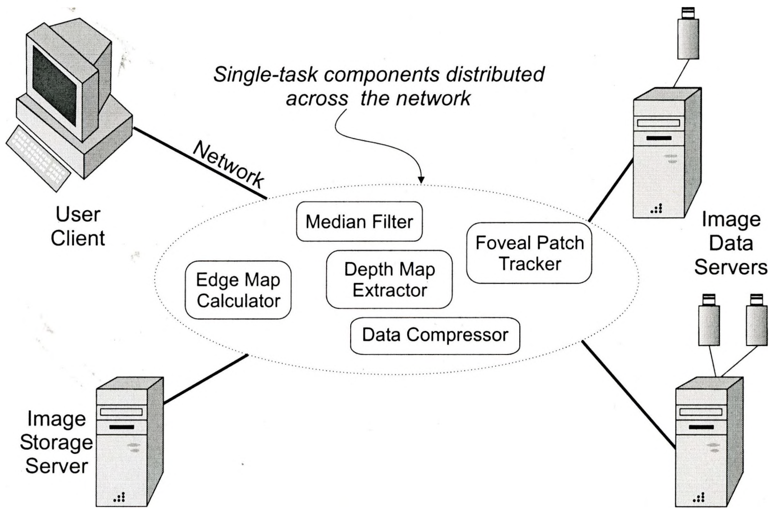


Figure 3.4: General vision system architecture

Vision systems in general utilize a number of interacting, well-defined processes, which can be realized as individual distributed components. This multitier client/server architecture of general vision systems is shown in Fig. (3.4). Here each physical or

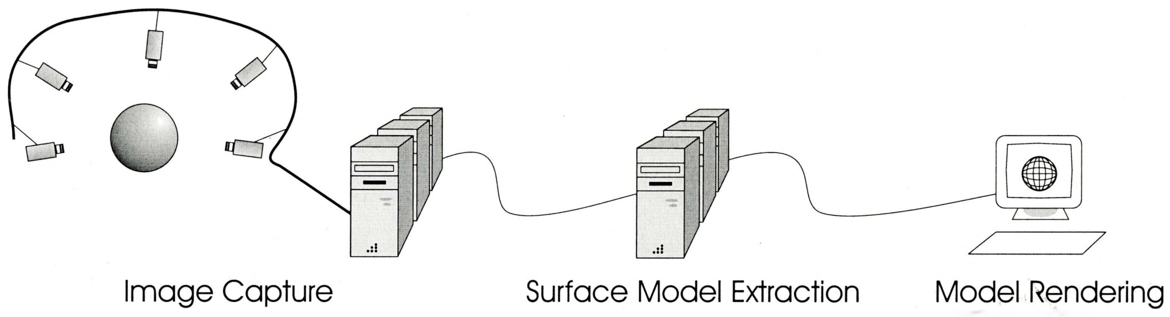


Figure 3.5: General 3D surface extraction system architecture.

logical construct in the vision system (e.g. camera, image display, median filter) has been abstracted as a component, and allowed to arbitrarily utilize the service of other components in the system.

Using this architecture, the extraction of 3D surfaces from 2D images can be broken down into the general components shown in Fig. (3.5). This general architecture can be decomposed into actual distributed components shown in Fig. (3.6).

The *Image Servers* provide images of the scene that are captured in real-time or have been previously stored. Images from a common viewpoint, along with pertinent camera information are retrieved by the *Information Management Server* (IMS), which sends this information to the *Depth Map Extractors* (DMEs). Each DME extracts the 3D surface of a portion of the scene specified by the IMS. This partial 3D surface is collected by the IMS from each DME, combined and sent to the *3D Explorer* which provides the interface to the user. These components are described in following sections.

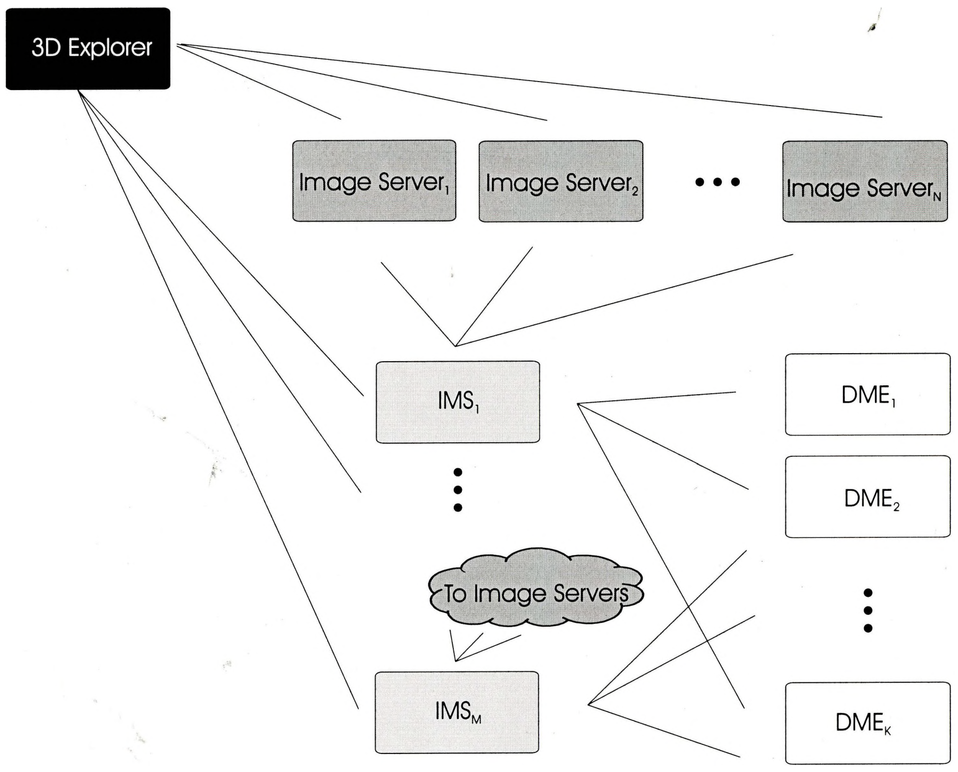


Figure 3.6: Presented system architecture

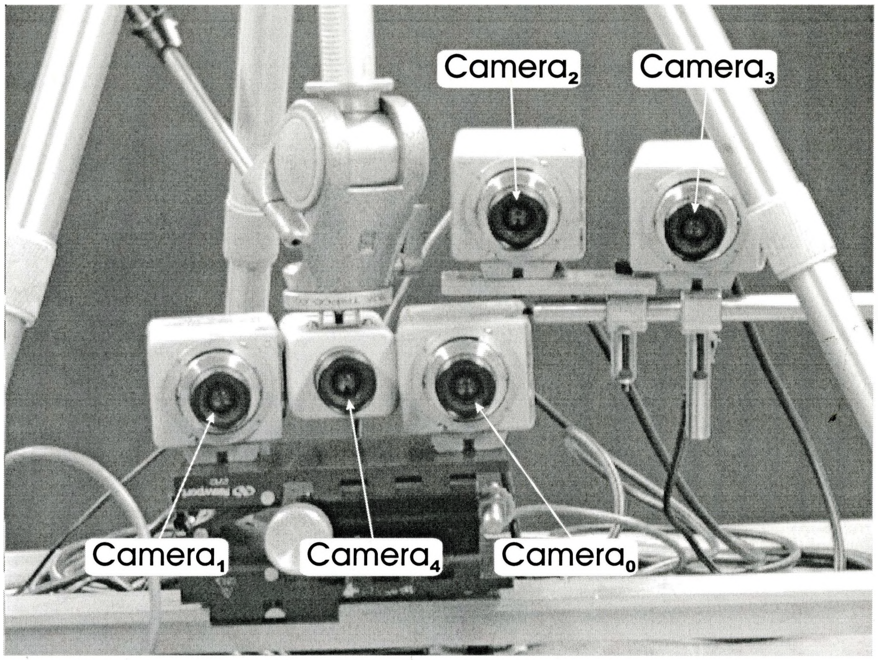


Figure 3.7: Experimental setup used in the project

The current system consists of a number of Intel Pentium II (333/400/450 MHz) and AMD Athlon (500 MHz) workstations running Windows NT (SP6) or Windows 2000 with 128 MB of RAM. The workstations are connected through a 100 MB, switched network through a Cisco 3548XL switch, under general port configuration. Sanyo VCB 3374 and Panasonic WVPB 332 cameras are connected to Matrox Meteor II/MC frame grabber cards. The system utilizes the CORBA ORB (version 3.3.2) provided by Orbacus. The image of the setup is shown in Fig. (3.7). The cameras calibrated parameters are provided in Table (3.2) where the rotation matrix $\mathbf{R}$ is represented by its Euler angles R_x, R_y and R_z. The measurement B is the baseline with respect to Camera$_0$. Any of the cameras viewing the scene can be used as a reference camera, and can be changed dynamically. The cameras are clustered in a general scene due to the limitations of the calibration procedure. It is necessary during calibration for all cameras to have a large overlapping viewable regions. Due to the size of the calibration object characteristics and its movement in space (see §3.2.2), the cameras have a limited volume of placement.

	$\mathbf{Camera_0}$	$\mathbf{Camera_1}$	$\mathbf{Camera_2}$	$\mathbf{Camera_3}$	$\mathbf{Camera_4}$
f (mm)	16.56	16.49	16.53	16.54	16.27
κ_1	3.79×10^{-4}	5.14×10^{-4}	4.41×10^{-4}	6.99×10^{-4}	3.23×10^{-4}
$\mathbf{s_x}$	1.02	1.02	1.02	1.02	1.03
$\mathbf{C_x}$ (pix)	287.89	301.62	293.69	300.93	325.26
$\mathbf{C_y}$ (pix)	254.09	282.82	258.73	218.32	240.30
$\mathbf{T_x}$ (mm)	-66.02	-102.91	-56.97	-40.87	-48.49
$\mathbf{T_y}$ (mm)	-76.66	-91.28	-63.52	-54.97	-85.27
$\mathbf{T_z}$ (mm)	763.44	779.91	776.88	783.16	786.23
$\mathbf{R_x}$ (deg)	0.93	1.79	5.76	4.54	1.73
$\mathbf{R_y}$ (deg)	1.18	7.33	-0.42	-5.54	6.82
$\mathbf{R_z}$ (deg)	-0.44	0.13	1.89	1.53	-0.33
B (mm)	0	-120.78	87.54	137.66	-61.57

Table 3.1: Camera Parameters

3.2.1 Vision Application CORBA Framework (VACF)

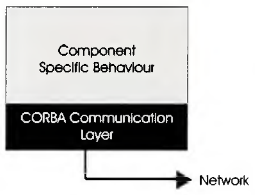


Figure 3.8: Component architecture overview

The system architecture uses multiple components, each of which can be broken into the general architecture shown in Fig. (3.8). Each component has a CORBA communication layer which is responsible for the inter-communication between the objects. The processing layer is the component specific behaviour necessary for providing certain services. Since the communication layer has a common behaviour in all components, a software framework was created to efficiently and effectively deploy the distributed components.

The Vision Application CORBA Framework (VACF) is an OO framework that allows the rapid development of CORBA based software components and for autonomous interactions between distributed components. The design of the framework was influenced by the features commonly found in vision systems. Due to the inherent complexity found in most vision systems, the framework allows for a unique, independent instance of a server for each client, allowing the server to have complex state behaviour. This is in contrast to the common stateless server architecture (such as an HTTP server), which can only support simple client requests, or restricted to limited state behaviour by having the client store state information. Components in vision systems may have multiple behaviours and using the Object-Oriented design methodology for software construction, it is common to encapsulate a behaviour in a unique class. Consequently, the framework contains multi-object support for the components. Finally, the framework provides minimal implementation overhead and component behaviour restrictions.

The client-server architecture used by the framework is described below in detail, providing insight into how the chosen architecture is well suited for distributed vision applications. Each component that uses the VACF includes the framework support classes (linked through a standalone library), and the behaviour implementation provided by the user. The user implementation is assembled into the framework with the aid of a code generating "wizard" written for Microsoft Visual C++ Studio® (screenshots provided in Appendix B). A complete example of an application developed with the VACF is also included in Appendix C.

Server Architecture

Each server component written with the framework exhibits the architecture shown in Fig. (3.9). The basic functionality of the CORBA ORB and related services are encapsulated in the `COrb` utility class. Central to the design are the `CServer` and `CServerFactory_impl` classes that allow for the construction of server components

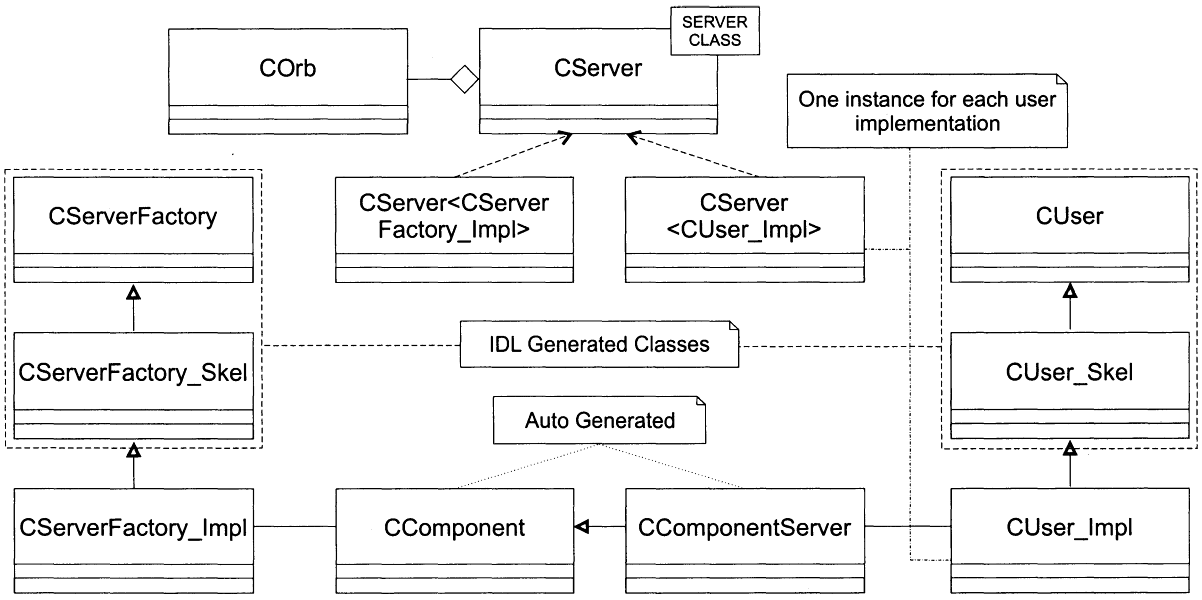


Figure 3.9: VACF Server Class Diagram

with complex state and multiple services. `CServer` is a template class which encapsulates the `COrb` class to provide a higher level of abstraction for the ORB related services specific to servers. For each user object of type (e.g. `CUser_impl`) that is to be exported by the server, a specific `CServer` class is derived. The `CServerFactory_impl` class is an object factory based on the design pattern of the same name[13]. It creates and destroys new instances of all user implementation objects in a component. The use of an object factory avoids coupling of each component to a common communication class through inheritance. Upon instantiation, the server exports only one service, the object factory. Therefore, the library includes a `CServer` derived class of type `CServerFactory_impl`. On the client startup, it connects to the server and invokes the `Create()` method of the factory, subsequently getting references to unique object instances on the server. It can thus invoke the specific methods for these objects to accomplish its task. Since unique instances of user objects are provided to each client, the state of each object can be maintained independently allowing for complex vision processes to be realized easily with the framework. Note that only one object factory persists for the life of the server, while the user objects are created and destroyed as per the requests of the clients.

`CComponent` and `CComponentServer` provide a bridge between the framework library and the server behaviour implementation by the user. Since it was desirable to have the user link to a pre-compiled library, these classes were designed and implemented to handle arbitrary user types used in the implementation of the server behaviour. The object factory `CServerFactory_impl` manages multiple instances of the user objects for each client through a list of `CComponents`. A generic `CComponent` class definition is provided to allow the framework to be compiled into a library , while the actual implementation is generated by the VACF wizard. The wizard also derives a `CComponentServer` from `CComponent` , that includes each of the user objects as member variables. Thus a call to `CServerFactory_impl::Create()` by the client adds a new instance of `CComponent` to the list, and `CComponent::Run()` is subsequently called. Since the library is compiled with a `virtual CComponent::Run()` method, at run time the VACF wizard generated `CComponent::Run()` is called. This in turn calls the VACF wizard generated `CComponentServer::Run()` which actually creates the user objects and exports them to the CORBA service directory managed by the Name Server. It is the service of these objects that the client subsequently employs. Similarly, when the client invokes `CServerFactory_impl::Destroy()` , the user objects are deleted and removed from the service directory, and the specific instance of `CComponent` is removed from the list.

The creation process of objects is shown in the pseudo-code below:

```
//Pre-compiled in the library.
char *CServerFactory_impl::Create()
{
        CComponent *pComponentInstance;
        ...
        //maintain the local list of creations
        AddComponentToList(pComponentInstance);
        //call the generic object creation
        pComponentInstance->Run();
        ...
}
//Called from the library through polymorphism.
//Generated by the VACF Wizard.
bool CComponent::Run()
{
        CComponetServer *m_pComponent;
        ...
        //call the specific object creation
        m_pComponent->Run();
        ...
}

//Called from static link at compile time in
//the application code of the user.
//Generated by the VACF Wizard.
bool CComponentServer::Run()
{
        CServer<CUser0_impl> *m_User0;
        CServer<CUser1_impl> *m_User1;
        ...
        //create the first user's object
        m_User0->new CServer<CUser_impl>(m_pOrb);
        //connect first object to the name server
        m_User0->Connect(m_sInstance.data(), "User0");
        //create the second object
        m_User1->new CServer<CUser_impl>(m_pOrb);
        //connect it to the name server
        m_User1->Connect(m_sInstance.data(), "User1");
        ...
}
```

Pseudo-code of object creation process on the server through VACF

Here, the services of the server are implemented by the user in the classes `CUser0_impl` and `CUser1_impl`. They are exported under the name of `User0` and `User1` respectively.

To allow the objects of the user objects in the component to interact autonomously as both clients and servers, the ORB message loop is started in a separate thread. This allows the component to service requests for the user defined services, while allowing for other simultaneous processes. The component can also interact with itself through another process, providing server side control without client intervention. This multi-threaded architecture of the framework therefore allows for vision components capable of timely interactivity, which may be necessary for higher speed response and the simultaneous response to multiple requests.

Using the VACF wizard, the user can build a complete server application by simply including their implementations in the build. Upon execution, the component will register with the name server, awaiting client requests. The user however, can modify the generated code to add any additional behaviour. The simplicity of this process is illustrated code below, which is auto-generated by the wizard (a complete example is given in Appendix C).

```
//Example of a VACF wizard generated code for an image server application
//which serves images from a local camera to clients across the network.
//Clients gain access to the object by requesting the ORB to connect to
//''RemoteCamera".
void main()
{
        //Create object factory server
        CServer<CServerFactory_impl> Server;

        //initialize the CORBA layer as required by a server
         Server.Init();

        //Make the object factory service available through the ORB
        //via the name ''RemoteCamera"
        Server.ConnectFactory("RemoteCamera");

        //wait for user controlled shutdown.
        cout << "Server Running... press any key to shut down";
        getch();

        //disconnect the factory from the name server
        Server.DisconnectFactory("RemoteCamera");

        //Tear down the CORBA layer for the server
        Server.Stop();
}
```

Auto-generated server startup code by VACF wizard

Client Architecture

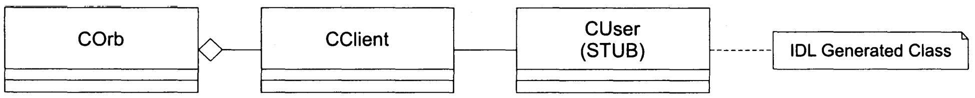


Figure 3.10: VACF Client Class Diagram

There is no "wizard" provided for the client creation since there is no suitable default behaviour of a client. The VACF however does provide abstraction for the CORBA services and automatic connection to the factory. The client architecture is shown in Fig. (3.10). Each client contains a `CClient` derived object and IDL generated stubs, for each server component that the client will connect to. `CClient` provides two important methods `Create()` and `Destroy()` , that call the respective object factory methods to a particular server. The client uses DII (dynamic invocation interface) provided by CORBA to access these services and thus does not require linking across other components. Also provided on the client side of the framework is a `GetRemoteObject()` template function. This function retrieves a specific object reference from the Name Server by name. As mentioned earlier, the name used by each client for an object actually refers to unique instance of the object created by the object factory. The `CServerFactory_impl::Create()` call to the object factory returns a unique name for each newly created server object to the client. This name is maintained *internally* in CClient. This allows multiple clients to use the *same name* to connect to *unique* instances of the server classes. To access the server from the client, the following code example illustrates invoking the GetFrame() method of the previously described camera server:

```
...
CClient *pClient;
CameraServer_var CameraServer_REF;
//Create a new client, initializing the CORBA layer
//as required for a client
pClient = new CClient();
//Connect to the server named "RemoteCamera"
//and have the object factory create a new
//instance of server objects
m_Client->Create("RemoteCamera");

//Get the reference to a "Camera" object from the component
//This actually refers to the newly created object instance.
CClient_GetRemoteObject<CameraServer_var, CameraServer>\
(*pClient, CameraServer_REF, "Camera");

//Get a captured image from the remote server
CameraServer_REF->GetFrame();
...
```

Excerpt of client code for connecting to the image server

A complete listing of a client/server application created with the VACF is provided in Appendix C.

3.2.2 Camera Calibration

The cameras used with the setup are modeled using the Tsai camera model [39] as described in §2.3. A freeware implementation Tsai's algorithm has been provided by Wilson [41] for a number of years. However, this tool does not extract the image and world coordinates of calibration features, but rather calculates the camera parameters given this data. As a result, an application was created to handle basic image processing (such as filtering, histogram equalization and blob analysis) as well as a convenient off-the-shelf camera calibration software tool to extract calibration features from images.

Since the non-coplanar camera calibration algorithm by Tsai is used to determine the camera model, a planar calibration object is moved along an optical rail at discrete depths during the calibration procedure. The origin of the world coordinate system is arbitrarily chosen and is such that the world z-axis is normal to the calibration object.

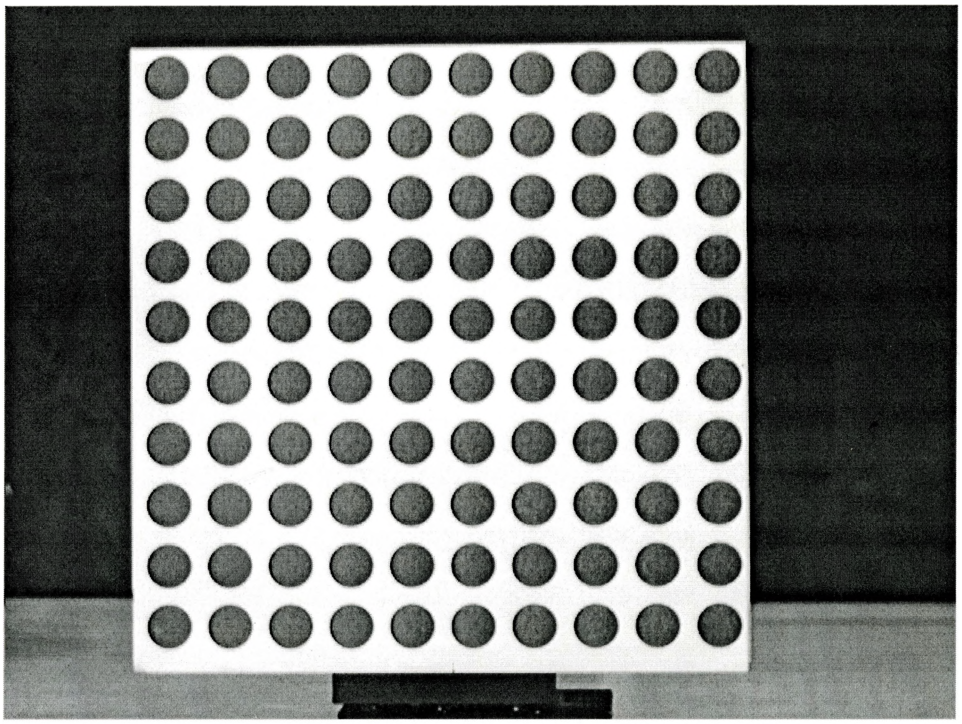


Figure 3.11: Original image of the calibration object.

The calibration object used within the setup is shown in Fig. (3.11). The object consists of 10 $\times$ 10 grid of circles. The circles have a diameter of 15mm and are 20mm apart (center to center). The black grid of circles is printed on a 1200 dpi laser printer on white card-stock paper and mounted on a flat steel plate. The centroid of the circles are selected as the calibration features.

The calculation of the centroid of the circles is performed through local image analysis and each stage is shown in Fig. (3.11 through 3.15). Given a pixel **S** that lies within the circle (either through algorithm estimation, or by user input), the algorithm first traverses vertically and horizontally along pixels with grey level absolute differences $< \delta$ compared to pixel **S** . To accommodate for the inaccuracies in the initial starting position, these four bounds are increased by a certain percentage α to provide a rectangular region that contains only a whole calibration marker (circle) and surrounding whitespace. The gradient of this region is calculated using the Sobel

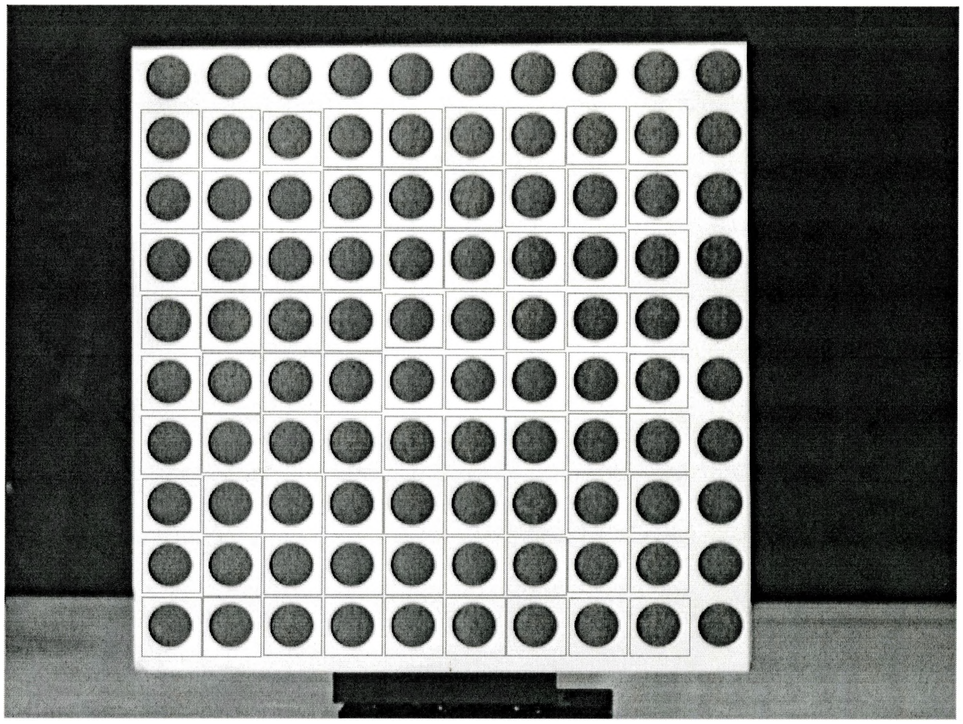


Figure 3.12: Calculation of local regions

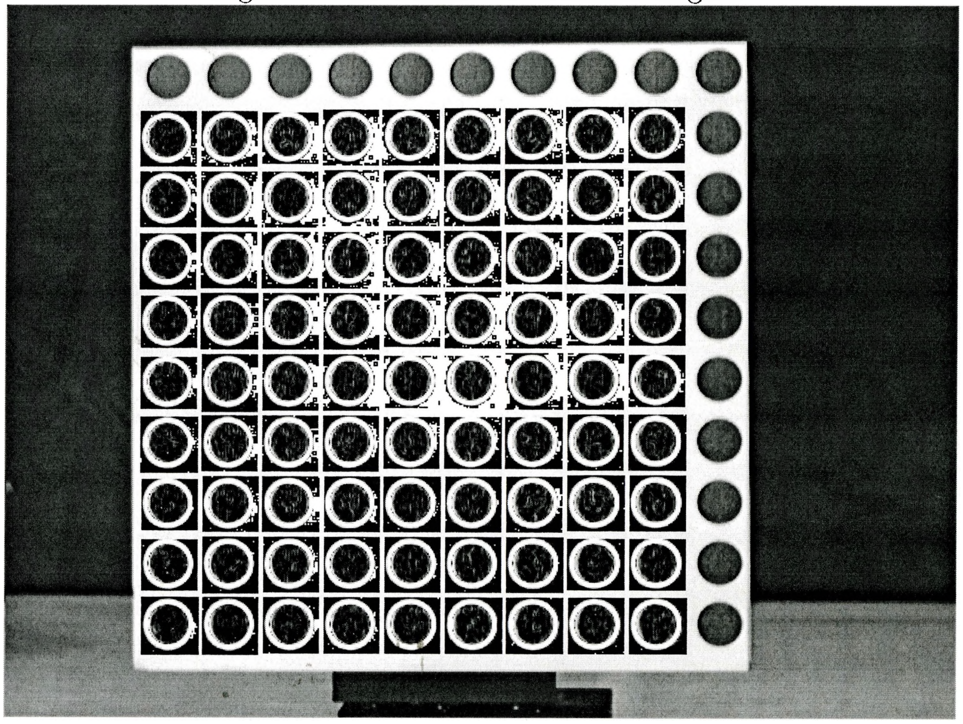


Figure 3.13: Sobel operator output

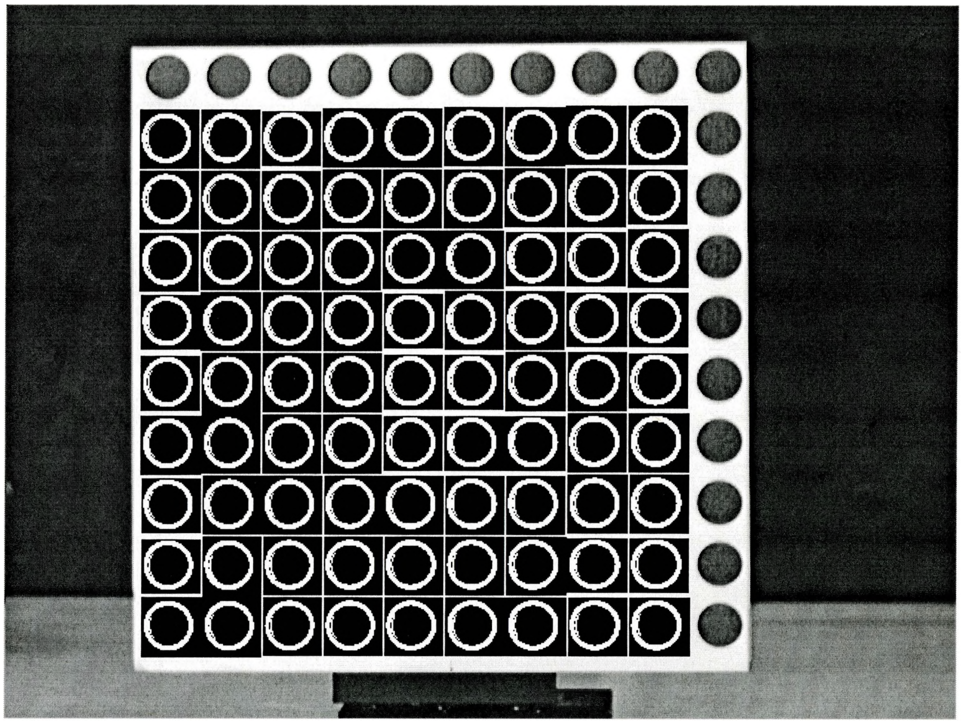


Figure 3.14: Binary thresholding of local regions

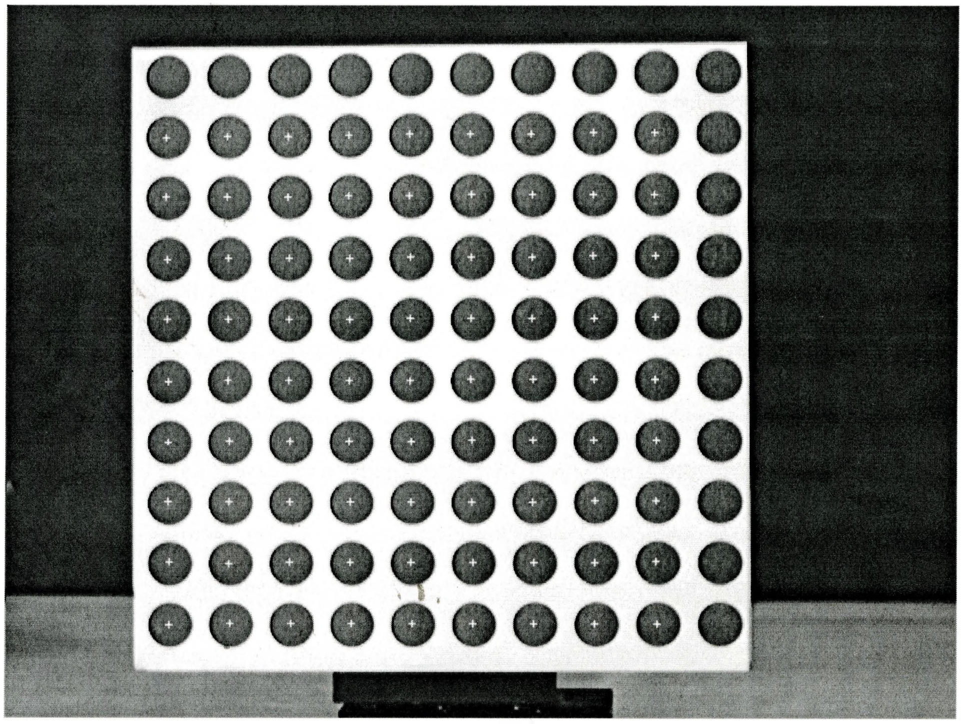


Figure 3.15: Moment calculation of binary regions

operator and converted to a binary image using threshold β. The end effect of this process is the extraction of a local region which only contains edge pixels of the circle. By calculating the moment of this region, the center of the circle is estimated in sub-pixels. $\delta = 127$, $\alpha = 9\%$ and $\beta = 127$ were chosen experimentally. This process is shown in the Figs. (3.11), (3.12), (3.13), (3.14), (3.15). Given an acquired image of the calibration object, only circles fully visible in the image are used to extract a grid of circle centers. This will result in a partial calibration feature grid if the entire calibration object is not visible. This grid of calibration features is extracted in two stages.

In the first stage, the orientation of the grid within the image is determined with the assistance of the user. Since the circles on the calibration object are uniformly spaced horizontally and vertically, the orientation of the calibration grid is completely determined by its origin and the direction of its rows and columns in the image. The origin is selected to be the $\mathbf{C}_{(0,0)}$, the centroid of the circle positioned at the top-left corner (in the world coordinate system) of the calibration grid. The user is asked to click within the image of the top-left corner circle. Note this may not be the top-left fully visible circle in the image, it is however the top-leftmost circle of the physical calibration object that is viewed by the camera. Next, the user clicks within the horizontal and vertical neighbors of the origin. Upon each click, the centers of the selected circles, namely $\mathbf{C}_{(0,1)}\text{and}\mathbf{C}_{(1,0)}$ are calculated. The extraction of these three calibration features allows for the complete estimation of the grid layout.

In the second stage the remaining visible calibration features are automatically extracted. This is done by calculating initial estimates for feature centers, and passing this as a starting position to the aforementioned centroid calculation algorithm. Since it is known that the circles lie uniformly in parallel rows and columns, the initial estimates are calculated by adding the horizontal ($\epsilon_{\mathbf{H}}$) and vertical ($\epsilon_{\mathbf{V}}$) displacements between the previously calculated calibration features using equations:

$$\begin{aligned} \epsilon_{\mathbf{H}}^{(\mathbf{i},\mathbf{j})} &= \mathbf{C}_{(\mathbf{i},\mathbf{j}-\mathbf{1})} - \mathbf{C}_{(\mathbf{i},\mathbf{j}-\mathbf{2})} \\ \epsilon_{\mathbf{V}}^{(\mathbf{i},\mathbf{0})} &= \mathbf{C}_{(\mathbf{i}-\mathbf{1},\mathbf{0})} - \mathbf{C}_{(\mathbf{i}-\mathbf{2},\mathbf{0})}. \end{aligned} \tag{3.1}$$

Therefore, calculation of the approximate centers for circles on adjacent columns can be done by:

$$\mathbf{C}_{(\mathbf{i},\mathbf{j})} = \mathbf{C}_{(\mathbf{i},\mathbf{j}-\mathbf{1})} + \epsilon_{\mathbf{H}}^{(\mathbf{i},\mathbf{j})} \tag{3.2}$$

and initial estimates for the first circle on each row is given by

$$\mathbf{C}_{(\mathbf{i},\mathbf{0})} = \mathbf{C}_{(\mathbf{i}-\mathbf{1},\mathbf{0})} + \epsilon_{\mathbf{V}}^{(\mathbf{i},\mathbf{j})} \tag{3.3}$$

This recursive technique compensates for the skewing of the calibration markers to provide robust local region estimates of their position. To simplify the search for full calibration markers without the loss of generality, the number of calibration features to be extracted is provided by the user.

Furthermore, the user specifies the location of the 3D world origin, the row and column number for $\mathbf{C}_{(\mathbf{0},\mathbf{0})}$ on the physical grid, the depth of the calibration object begin viewed and the spacing between adjacent circle centers. This information allows the calculation of world coordinates for all extracted features. Once the grid of calibration features have been extracted, their image position, as well as world coordinates are written to a file, suitable for input into Wilson's [41] implementation of Tsai's algorithm. The calibrated camera parameters calculated by [41] are saved to a file for later use.

3.2.3 Image Capture and Processing

Generally, image capturing facilities are limited to be used by one application. However, in a distributed environment there is an opportunity (or requirement) for the sharing of these resources. In order to accomplish this, the camera and the frame grabber must be encapsulated within a CORBA aware component, exposing the basic functionality (capturing of images) to other components. Since this requires that a camera be coupled with a workstation and be treated as a single entity, a host of new possibilities emerge. Image processing can be performed by the workstation upon the captured raw image, and this processed image can be sent across the network. Consequently a *smart camera* can be designed.

A smart camera provides increased functionality over a traditional camera that provides a raw video stream. Instead, a smart camera transmits information more relevant to the application. For example, raw images could be compressed before transmission over the network, which may be desired to reduce the network load. Similarly, a smart camera could only transmit location information of a object being tracked, eliminating the need for image transmission altogether. Smart cameras eliminate the need for specialized hardware in many applications, since their behaviour can be programmed in software. Furthermore, in a distributed computing environment, it allows for the possibility of multiple applications sharing camera resources, even across different computing platforms and network protocols. In general, this concept can be applied to any sensor used within a vision system to create a *smart sensor*.

In the presented system, smart cameras are built with the application named *Image Server*. The Image Server provides the basic functionality of cameras such as transmitting raw image frames from the online camera, but also provides additional functionality by allowing foveal patch extraction, streaming of image sequences, and image rectification.

Foveal patch extraction was added as a camera feature to allow for the reduction of network load by sending smaller images. The external components communicating to the Image Server can thereby request an image of the desired portion of the scene. This feature is not utilized in the construction of 3D surfaces, it was however used to test the performance of the system, as outlined in §4.1. The Image Server can also stream stored images whose sequence is defined in a file. This allows synthetic images and images captured earlier in time or from different environments to be used by an application. This feature is especially useful during the testing and construction of components, when the same images are to be processed.

The rectification of images is provided to aid the stereo pair image analysis (§2.5). The smart camera is capable of providing rectified images, for both reference and normal views, such that their epipolar geometry lies on the horizontal scanlines (as detailed in §2.4). Using the direct calibration information for each camera, the server precomputes a reverse rectification table for each reference camera, that maps each pixel in the rectified image to a pixel in the original image. Due to the "many to one" correspondence between the rectified image and original image, the table actually stores the bilinear interpolation parameters for each rectified pixel. These tables are computed and stored on disk before the server comes online, and cached in memory at runtime to improve performance. To further enhance the smart camera capability, the camera can rectify images with respect to cameras serviced by other Image Servers.

The Image Server is also capable of supplying camera specific information that is needed during stereo image analysis such as baseline, focal length and transformation matrices for a given stereo pair. Since the mapping of the original pixel to the rectified image pixel is needed in the implemented algorithm, the server also precomputes and stores these forward rectification tables. The server also supports the rectification of the offline images.

The configuration of the Image Server is controlled through command line options and configuration files. Through the command line options, the camera can be requested to compute the reverse and forward rectification tables and stream offline or online images. The reference camera for the Image Server can be changed dynamically by the request of any client. This is crucial in situations where a camera is part of more than one stereo pair in the multi-camera stereo pair setup.

Hence, the Image Server successfully fabricates a smart camera by efficiently encapsulating of the actual image capturing hardware, and providing elegant extensions that alleviate the computional/information load on the client components. The behaviour of the server can be easily controlled through configuration files, allowing it to adapt to many environments.

3.2.4 Depth Map Extractor

The Depth Map Extractor (DME) component implements the MBS algorithm outlined in §2.6. Given a set of stereo pairs and camera parameters, the DME provides the depth map for any specified portion of the image. The DME assumes that the camera stereo pairs satisfy coplanar stereo geometry, and hence is provided with rectified images. The DME provides depth estimation with respect to the reference camera coordinate system. Consequently, it requires the forward rectification tables that map the original reference camera image to each of the provided rectified reference images.

The DME extracts provides depth from a specified $m \times n$ region of the image, and returns an $m \times n$ array of depth values. This is done to allow for parallel computation of the same scene across multiple DMEs and limit the network load. Finally, the mask size, depth range and depth sampling size are controlled by the user. This allows the DME to be used in systems with varying requirements.

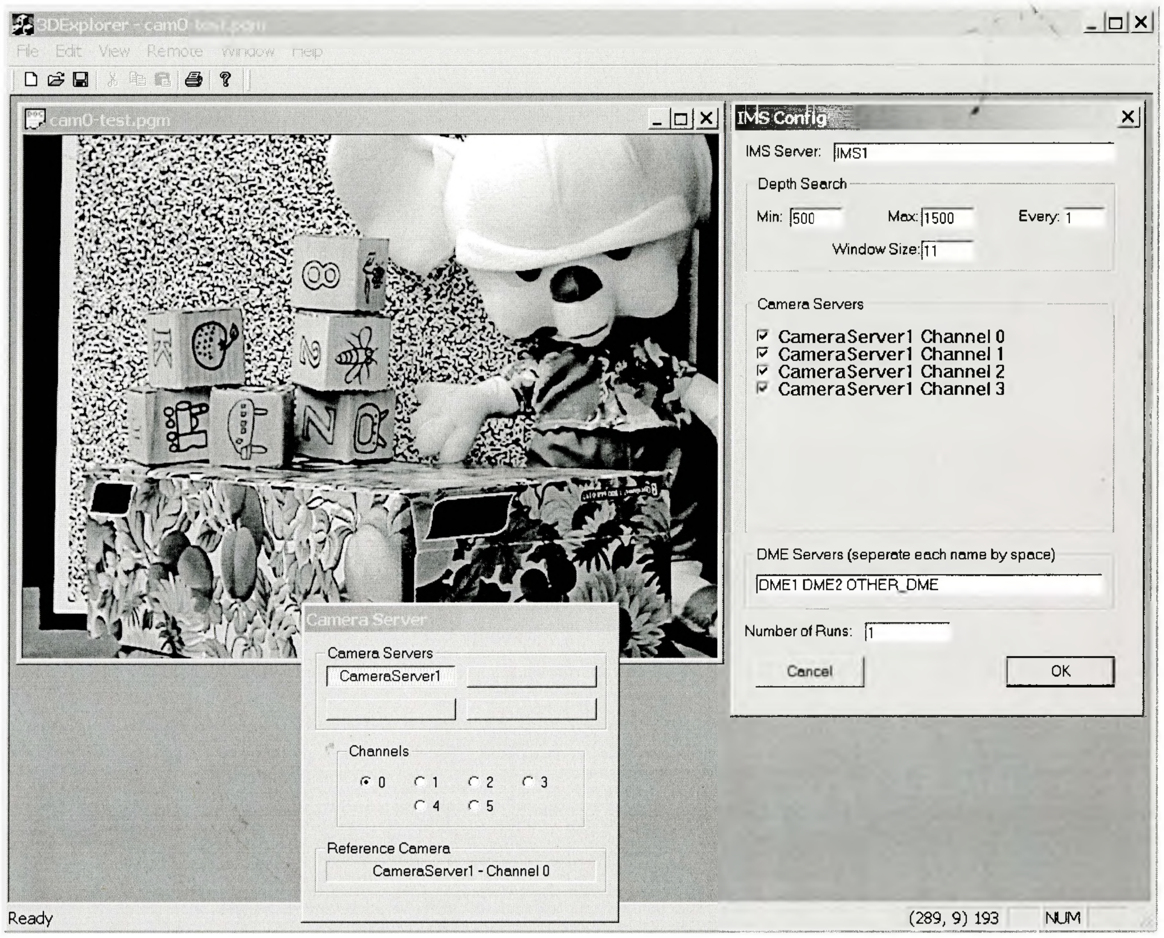
3.2.5 Information Management Server

The Information Management Server (IMS) coordinates the exchange of information among the different components as per the requirements of the end user. Upon receiving the request and configuration information from the end user, the IMS first requests the necessary Image Servers for the required images and camera parameters. Accordingly, the IMS requests the user specified DMEs to work on portions of the scene using separate threads. Currently, the IMS divides the image into N equal sections, where N is the number of DMEs. However, the IMS could potentially perform load balancing by distributing portions of the image proportional to the performance of each DME. Upon the completion of processing by all DMEs, the IMS assembles the individual depth maps provided by each DME into a complete depth map of the imaged scene and transmits it to the client. Thus, in the current system architecture, the 3D surface from each "viewpoint" of the scene (viewed from a group of stereo pairs) is extracted through unique IMSs. These IMSs however, may share DMEs and Image Servers.

The task of the IMS substantially reduces the network traffic by limiting the interaction of the components within the system. For example, in a fully connected architecture, the end user would need to send a request to each DME to process a specified portion of the image, which in turn, would request images from the Image Servers. Consequently, the Image Servers would have to process the same request for each of the DMEs. For dynamic scene environments, this would impose considerable limitations. The IMS encapsulates the interaction specifics between each of the components in the presented system, allowing the behaviour of the other components to remain general and thus adaptable to other systems.

3.2.6 User Interface

The application *3D Explorer* provides the user interface whereby the user can configure a the extraction of surfaces from a scene. It passes the request from the user to an IMS server and currently displays the computed 3d surface as a 8 bpp greyscale image (depth maps). It can also request foveal patches and rectified/non-rectified images from multiple Image Servers. A screenshot of the application is shown in Fig. (3.16).



IMS Config

IMS Server: IMS1

Depth Search

Min: 500 Max: 1500 Every: 1

Window Size: 11

Camera Servers

☑ CameraServer1 Channel 0
☑ CameraServer1 Channel 1
☑ CameraServer1 Channel 2
☑ CameraServer1 Channel 3

DME Servers (seperate each name by space)

DME1 DME2 OTHER_DME

Number of Runs: 1

Camera Servers

CameraServer1

Channels

[✓ 0] 1 2 3 4 5

Reference Camera

CameraServer1 - Channel 0

Ready (289, 9) 193 NUM

Figure 3.16: A screenshot of 3D Explorer

Chapter 4

System Performance

4.1 VACF Performance

Since the system incorporates the notion of NOW as the primary building block for vision systems, the performance of this building block is first measured [2]. The proposed test application extracts and displays foveal patches from cameras viewing an arbitrary scene. The application consists of two components, namely a Image Server (§3.2.3) and client(s) which request images from the Image Server and provide the user interface. The components are situated on separate workstations connected through the network (§3.2).

To stress the load on the network and server, the clients request images from the Image Server repeatedly and as quickly as possible (the use of a non real-time, multi-threaded operating system does not allow the client call to be deterministic). This allows for the identification and quantification of the bottlenecks in the system. The overhead associated with retrieving a single frame (8 bpp) by the client is shown in Fig. (4.1). Figure (4.1) depicts the overhead associated with each of the underlying components, namely the physical medium of the network, the TCP/IP stack in WindowsNT® and the VACF. The overhead associated to the TCP/IP stack in the operating system was measured by transferring the same amount of data using

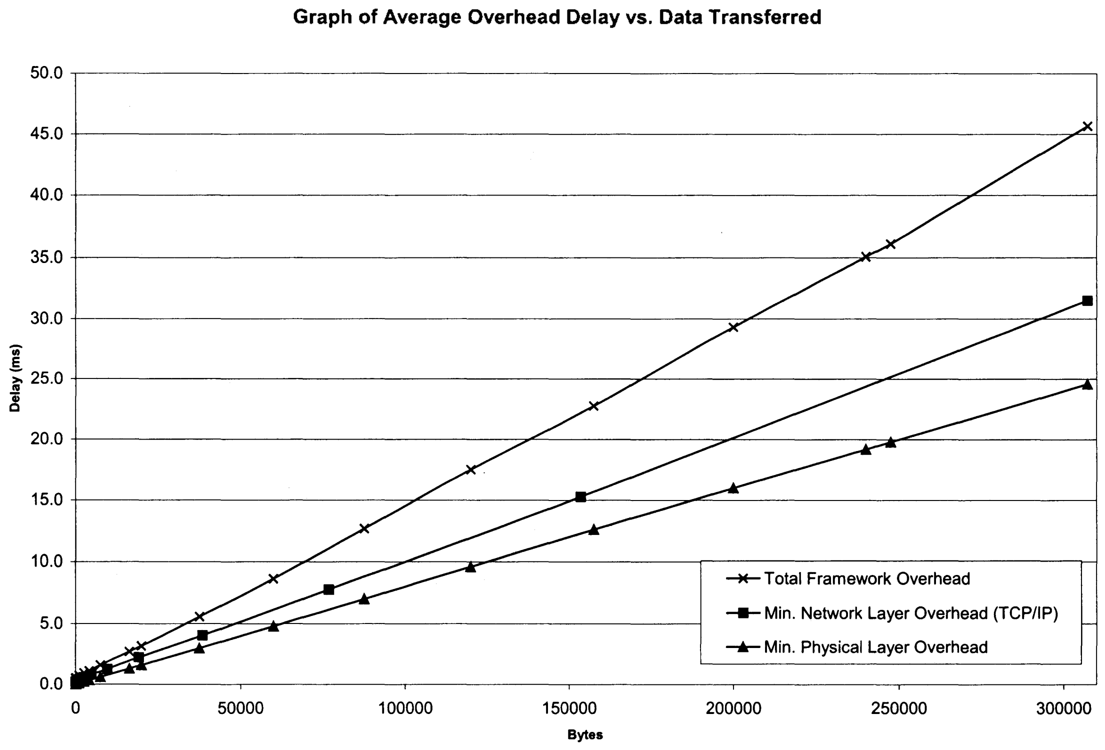


Figure 4.1: Performance measurement of VACF

socket calls. The overhead due to the physical medium is calculated assuming *only* the image data is transferred (i.e. minimum overhead ignoring packet header data). Each overhead measurement is obtained from the average of 10,000 image requests.

Under single client load conditions, the system is able to achieve a frame rate 21.9Hz. The overhead due to CORBA and VACF under heavy load conditions (images of 640x480@8bpp) is approximately 50% over the network transfer using TCP/IP. The overhead drastically increases for very small data transfers due to the peculiarities of TCP/IP (such as the backoff algorithm). This test consolidates the feasibility of smart cameras, since if real-time operating systems and ORBs are employed the system should be able to transmit another 2MBs to achieve 30Hz frame rate. The upgrading to faster networks and real-time network protocols will also improve system performance. However the CORBA processing overhead is more closely coupled to the computer hardware performance than network data throughput in the current system since peak inferred network utilization is $\approx 55\%$ in the above test.

4.2 Rectification Results

The rectification is an indirect method to test the camera calibration accuracy. Given two arbitrary scenes and the camera parameters determined through calibration, the images should be rectified such that their epipolar geometry lies on the horizontal scanlines. Figure (4.2) shows two original images, while Fig. (4.3) depicts the rectified versions. The epipolar geometry has been calculated by calculating the fundamental matrix parameter **F** using the projection matrices of the cameras. As seen from the presented figures, the epipolar geometry is indeed horizontal and lies on the scanlines of the images.

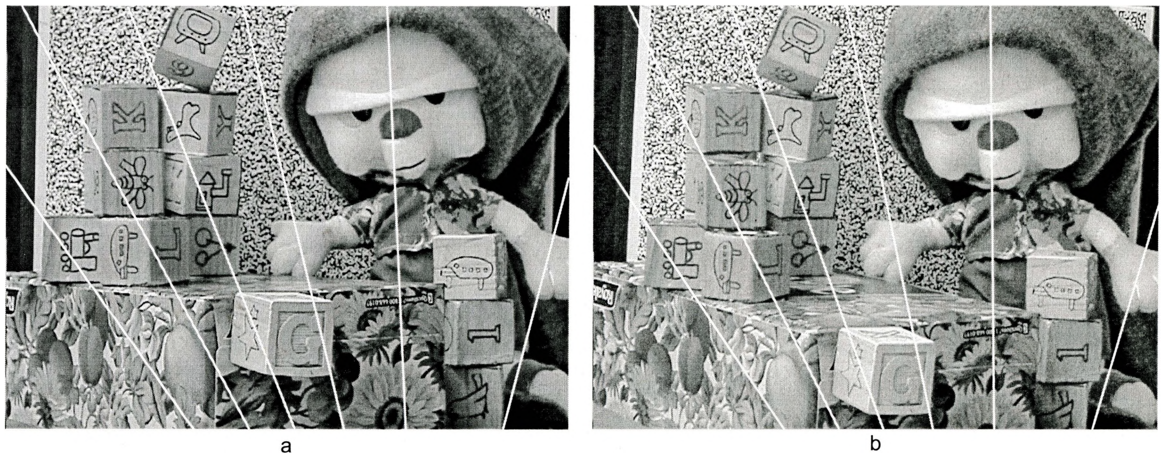


Figure 4.2: Original (a)Reference Camera Image (b)Non-Reference Camera Image

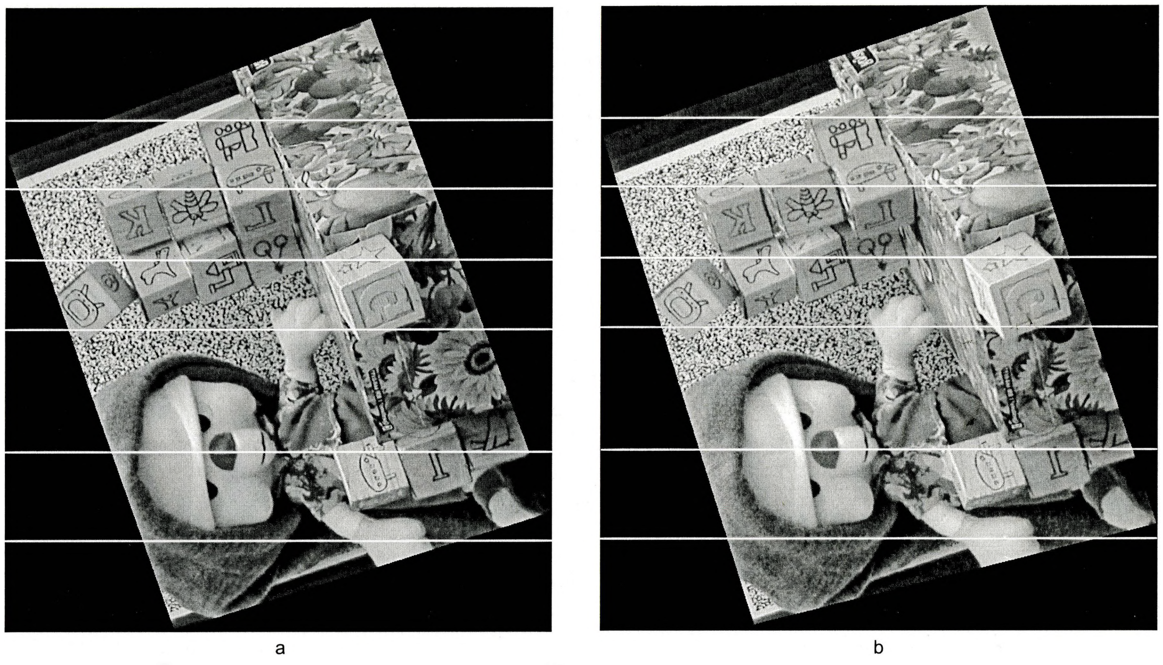


Figure 4.3: Rectified (a)Reference Camera Image (b)Non-Reference Camera Image

4.3 3D Surface Extraction

In this section the performance of the complete system is presented. In the presented sections, one Image Server is used which services 5 arbitrarily placed cameras (refer to Fig. (3.7) for details). These cameras are placed such that they all have overlapping viewpoints, forming 4 stereo image pairs with a specified reference camera. In the following tests, the reference camera is kept the same to ensure uniformity.

Since the radial distortion parameters are under constrained in the camera calibration process, they are largely inaccurate and ignored. The system employs a maximum of three DMEs, each running on identical workstations. Similarly, one IMS and one 3D Explorer are used, each running on separate workstations. All the component threads run under normal process/thread environments. The performance of the system is assessed in both timing and accuracy.

Timing Analysis

For the proposed architecture to be viable for vision systems, it is necessary to evaluate the timing characteristics of the system. The timing performance of the system was measured under various conditions using arbitrary scenes (the properties of the scene do not affect the performance of the algorithm). The presented results have been averaged over 30 runs. On the following graphs depicting the individual timing breakdowns for a certain system configuration, the "Miscellaneous" category predominantly represents the time taken to transmit information to between the IMS, 3D Explorer and Image Server(s).

Varying Number of DMEs

In the proposed architecture for vision systems, parallelism is achieved through the distributed computing software paradigm. In the given setup, parallelism is realized by distributing the extraction of depth maps over multiple DMEs. Thus, to measure the performance, the 3D surface extraction process was performed varying only the number of DMEs. Tables (4.2), (4.3) and (4.4) list the times taken for computation (with different mask sizes) and are graphed in Fig. (4.4).

Figures (4.5, 4.6 and 4.7) depict the percentage breakdown of time per individual task for the 9 $\times$ 9 mask. These figures illustrate that over 90% of the time is spent in the processing of images to extract depth maps. Figures(4.8, 4.9 and 4.10) further

No. of DMEs	**Computation Time (ms)**	σ	**Performance Gain**
1	38202.93	119.28	-
2	20421.70	30.02	46.54%
3	14464.50	182.81	62.14%

Table 4.2: Computation time using varying number of DMEs (Image Size = 320$\times$240, Mask Size = 3 $\times$ 3, Number of Depths = 128, Number of Stereo Pairs = 4)

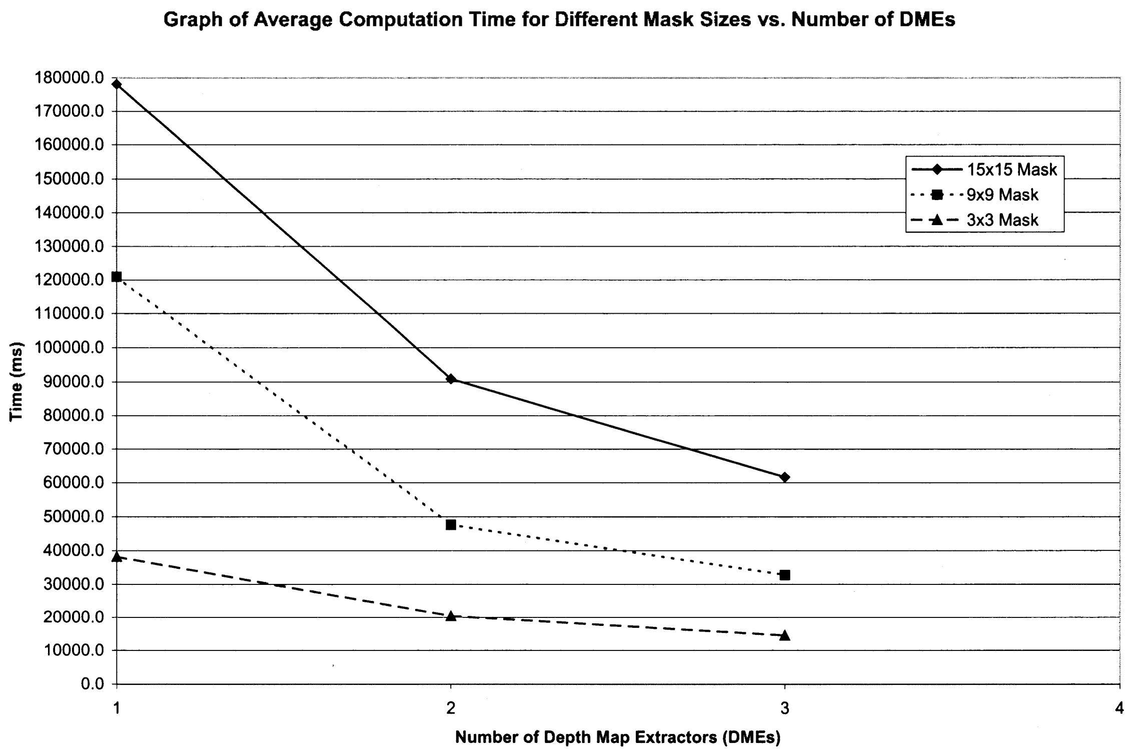


Figure 4.4: System performance for a varying number of Depth Map Extractors (DMEs)

No. of DMEs	Computation Time (ms)	σ	Performance Gain
1	120996.73	317.21	-
2	47676.37	99.57	60.60%
3	32764.77	159.33	72.92%

Table 4.3: Computation time using varying number of DMEs (Image Size = 320×240, Mask Size = 9 × 9, Number of Depths = 128, Number of Stereo Pairs = 4)

No. of DMEs	Computation Time (ms)	σ	Performance Gain
1	178067.33	345.81	-
2	90885.33	154.54	48.96%
3	61668.17	251.70	65.37%

Table 4.4: Computation time using varying number of DMEs (Image Size = 320×240, Mask Size = 15 × 15, Number of Depths = 128, Number of Stereo Pairs = 4)

show a consistent overhead for the other tasks in all the three systems. They only differ in the amount of time spent in the exchange of information (IMS data packing).

From the above results, it is evident that substantial gains can be made through the adoption of the distributed computing architecture. However, the increase in the number of components increases the flow of information. Thus, eventually the coordination and exchange of information over a large number of components will become the dominating factor in the total computation time, limiting the feasible number of components that can be employed. This trend is illustrated by the incremental performance gain of 15% by adding a third DME and the increased time spent packing the data by the IMS. Although the gains are incremental, they are significant from the total time spent perspective. The time spent performing the calculations suggest that the optimization of the DME software component is necessary, both from the algorithm implementation and operating system point of views.

The processing of the images to extract depth maps is directly various parameters

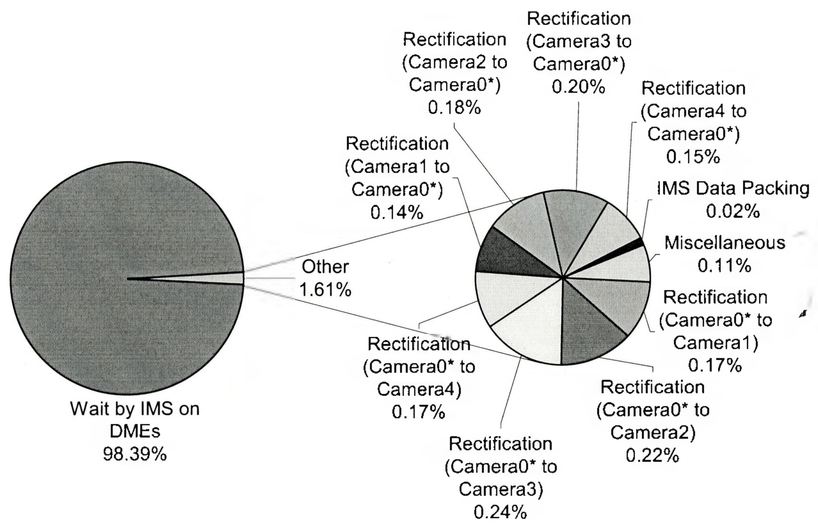


Figure 4.5: Timing breakdown for system employing 1 DME (9 × 9 mask)

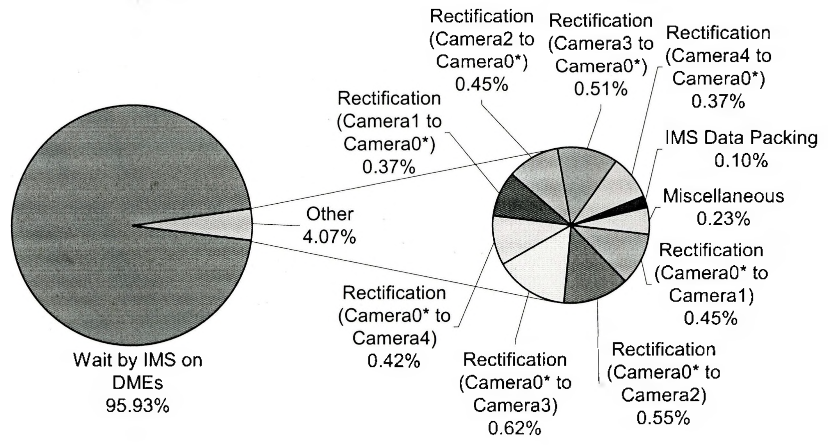


Figure 4.6: Timing breakdown for system employing 2 DMEs (9 × 9 mask)

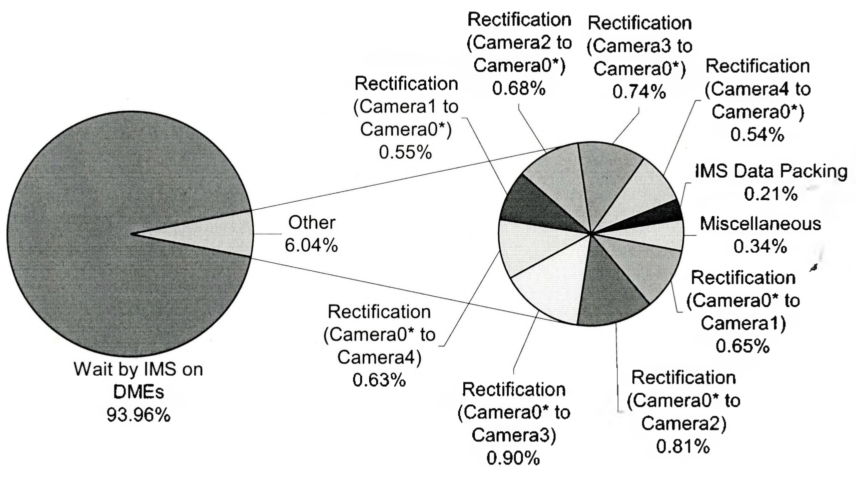


Figure 4.7: Timing breakdown for system employing 3 DMEs (9 × 9 mask)

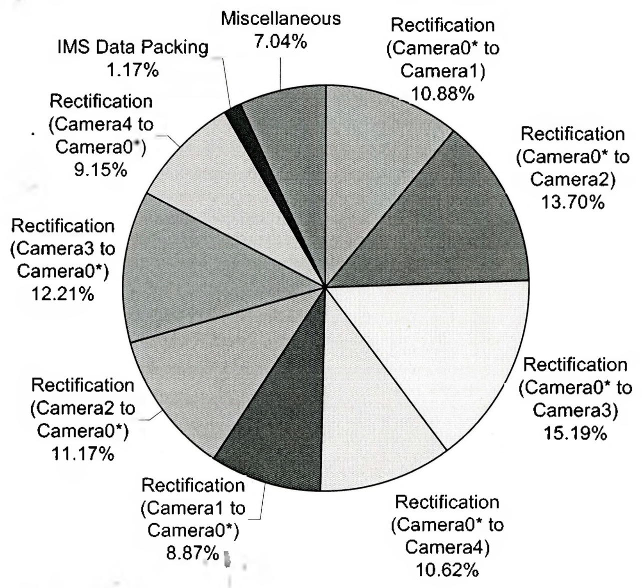


Figure 4.8: Data gathering and exchange overhead for 1 DME (9 × 9 mask)

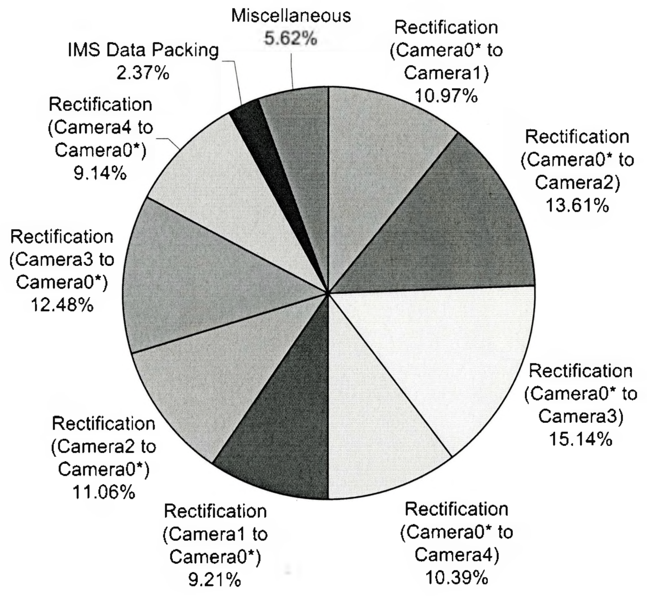


Figure 4.9: Data gathering and exchange overhead for 2 DME (9 × 9 mask)

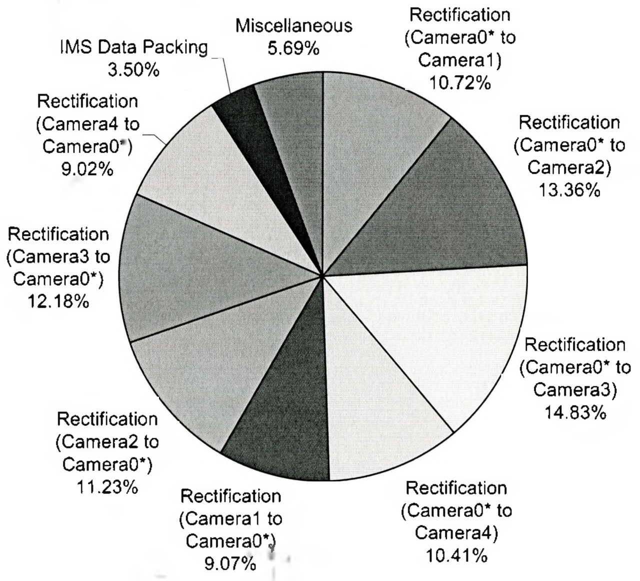


Figure 4.10: Data gathering and exchange overhead for 3 DME (9 × 9 mask)

such as mask size, image sizes, etc. The relationship etc. The relationship between these parameters and the overall computation time is studied in the following sections

Varying mask size

The mask is the size of window W (or block) created around each pixel for which the SSSD is computed (§2.6). The size of the mask directly dictates the number of operations that need to be performed for a given image of the scene. To extract the relationship between the mask size and performance timing measurements were taken for different mask sizes. Figure mask sizes. Figure (4.11) graphs the values in Table (4.5) that list the time required to compute the depth of a scene with varying mask size.

A linear relationship relates the increase cost of computation with respect to the mask size. The depth map calculation dominates the required computation time, with 63% for the smallest mask size of 3×3 requiring 2304 computations (9 pixels $\times$ 8 images $\times$ 32 depths) while increasing to 92% for the mask size of 19×19, requiring 92416 computations per pixel as shown in Figs. (4.12, 4.13). The overall overheads of each of the other components remain consistent throughout each run of different mask size, as shown in Figs. (4.14, 4.15). Interestingly, although the computations increased 40 fold, the time required to process the images only increased 7 fold. This

Mask Size	Computation Time (ms)	σ
3×3	5366.33	103.85
7×7	7611.70	96.58
11×11	12176.76	33.71
15×15	17799.50	106.79
19×19	248383.00	61.18

Table 4.5: System performance for different mask sizes (Image Size = 320×240, Number of DMEs = 3, Number of Depths = 32, Number of Stereo Pairs = 4)

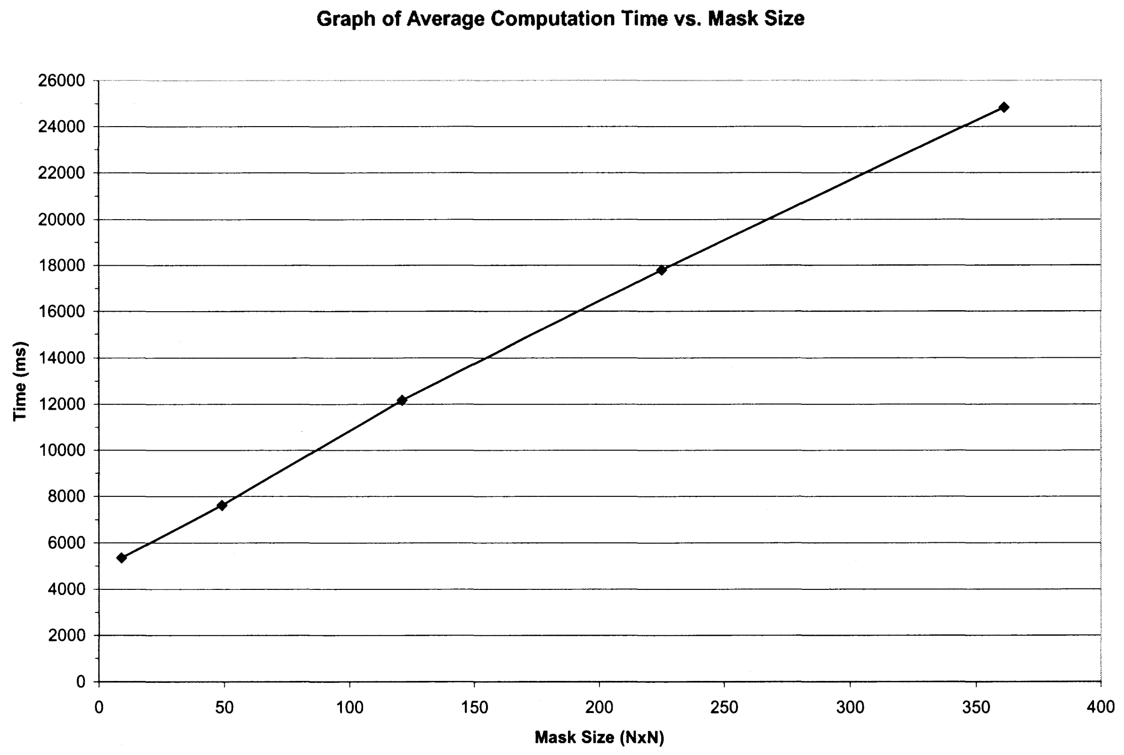


Figure 4.11: Computation time for varying Mask Size

is due to the initial setup overhead by the DME (i.e. the gathering of data, and collection of forward camera rectification tables) dominates the process for smaller mask sizes. This is not true for larger mask sizes, where the performance penalty is proportional to the increase in computation per pixel.

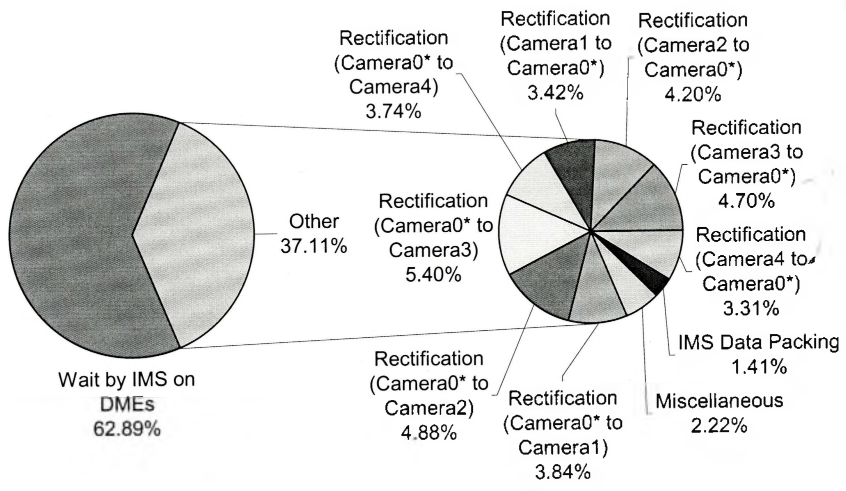


Figure 4.12: Timing breakdown for 3x3 mask

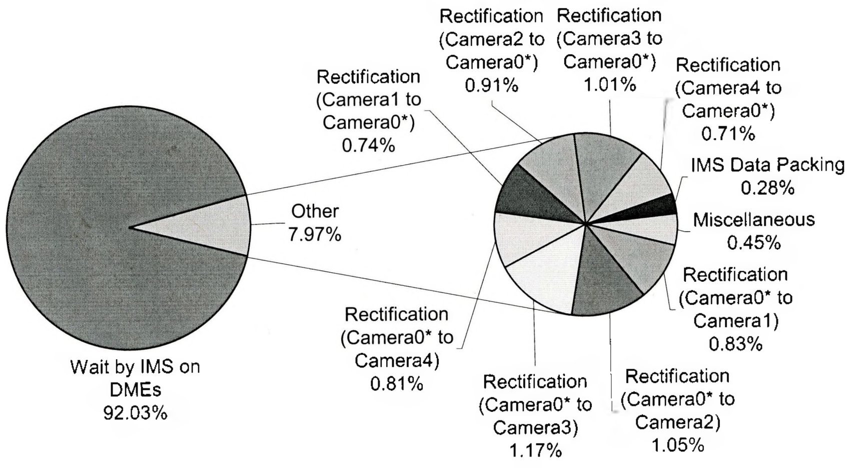


Figure 4.13: Timing breakdown for 19x19 mask

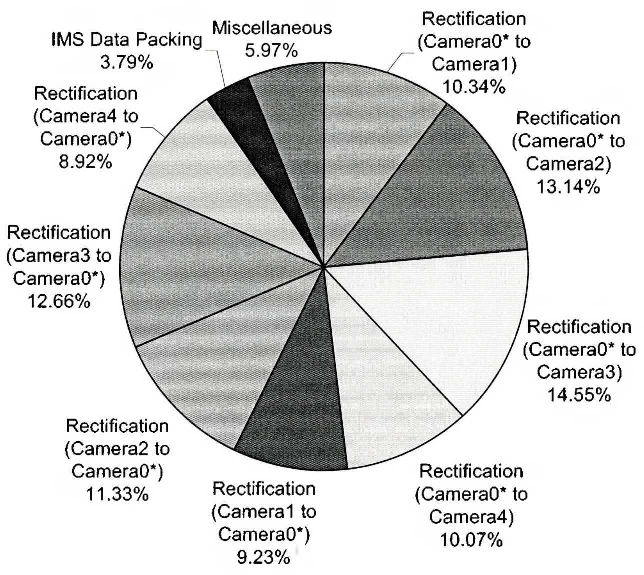


Figure 4.14: Data gathering and exchange overhead for 3x3 mask

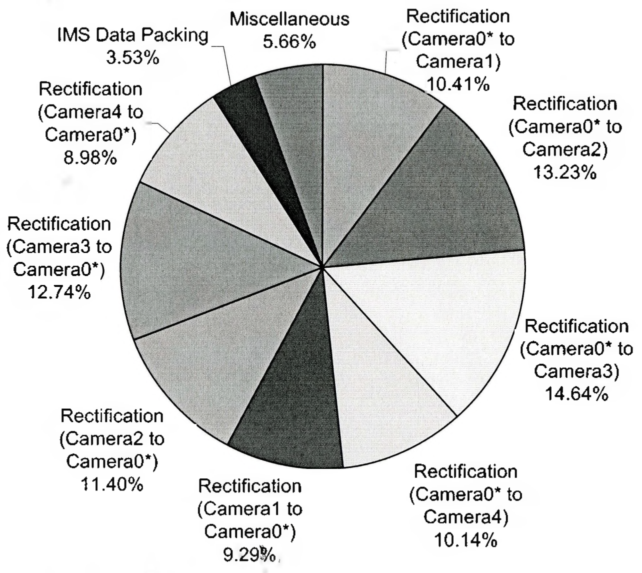


Figure 4.15: Data gathering and exchange overhead for 19x19 mask

Varying number of depths searched

Since the number of depths searched varies the computation required per pixel, the results are similar to those of those obtained in the previous section. Table (4.6) shows the results of varying the depths searched, and is plotted in Fig. (4.16). During the course of the experiment it was ensured that the range of depths lay in all the camera viewpoints.

No. of Depths Searched	Computation Time (ms)	σ
16	3806.36	90.77
32	5278.50	88.16
64	8303.13	41.14
128	14407.97	89.67
256	26684.57	77.51
513	51554.20	151.77

Table 4.6: Computation time for varying depth ranges (Image Size = 320×240, Mask Size = 3×3, Number of DMEs = 3, Number of Stereo Pairs = 4)

Varying number of stereo pairs

The varying of the number of stereo pairs shows changes the data that needs to be transported, as well as searched by the MBS algorithm. Here, the relationship is linear and directly proportional to the number of images. This is to be expected, since under the tested configuration the processing of images still dominates the computation time, while the acquiring and transporting of images is comparatively smaller. Table (4.7) lists the computation times, while Figure (4.17) depicts the linear relationship.

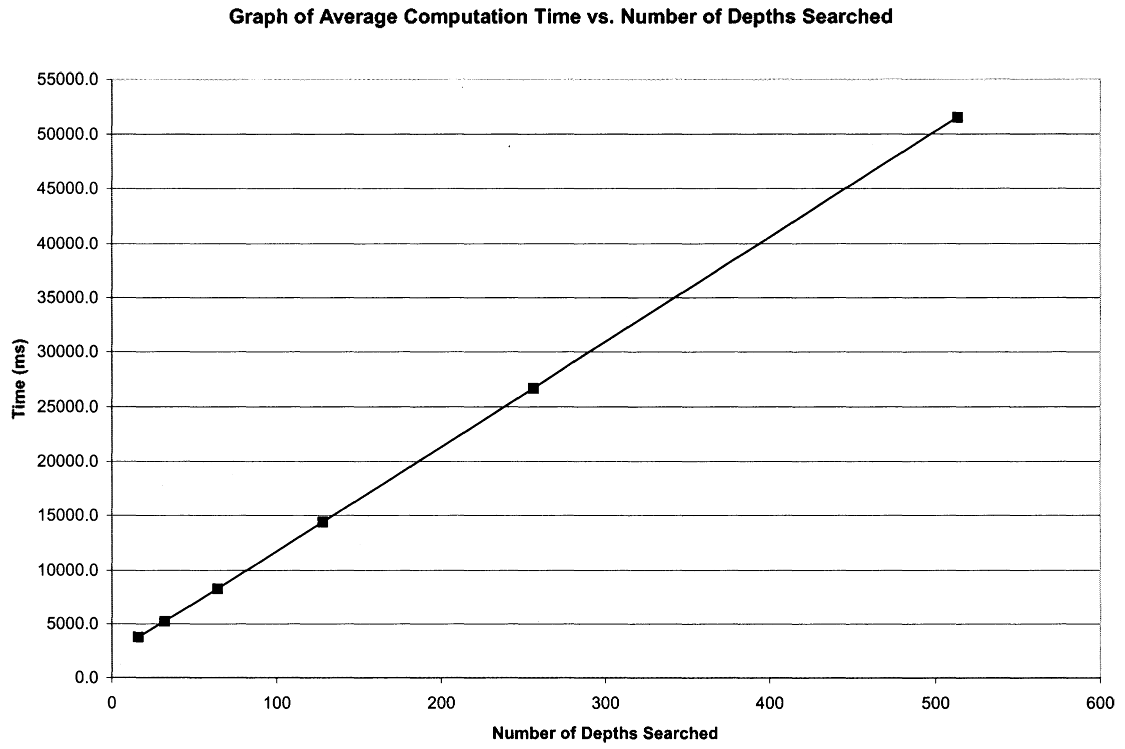


Figure 4.16: System performance for different number of searched depths

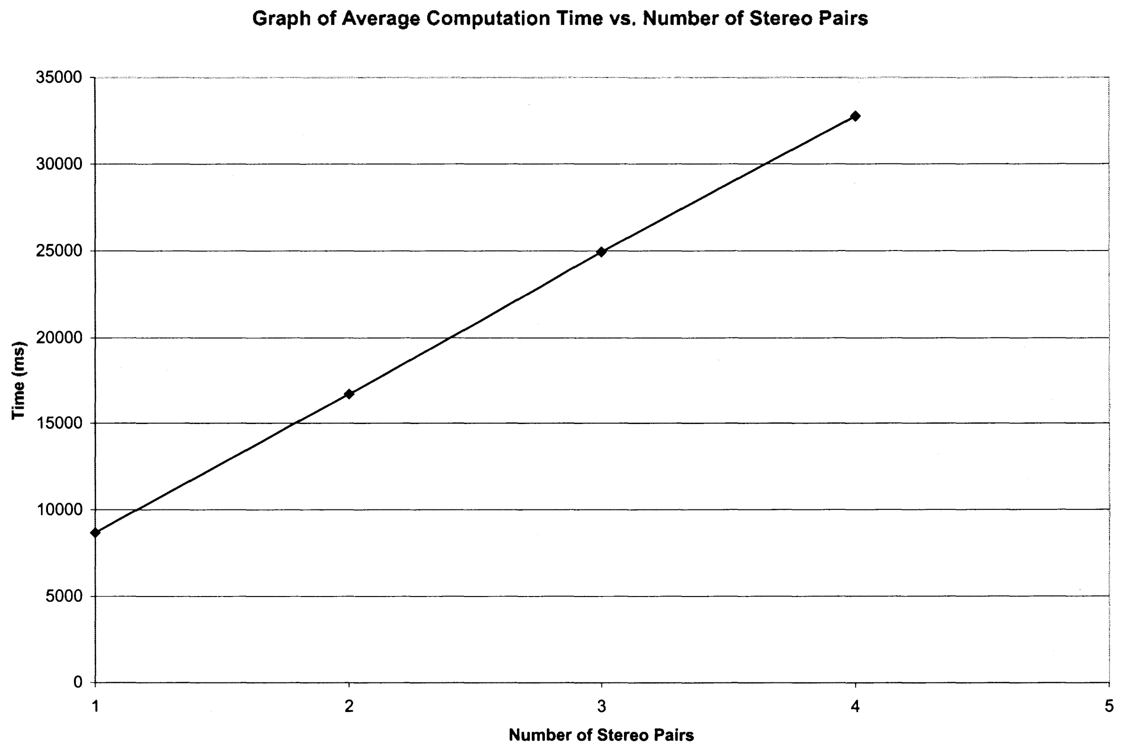


Figure 4.17: System performance for different stereo pairs

No. of Stereo Pairs	Computation Time (ms)	σ
1	8673.37	20.27
2	16706.37	33.03
3	24943.03	86.33
4	32764.77	159.33

Table 4.7: Computation time for different number of stereo pairs (Image Size = 320×240, Mask Size = 9×9, Number of Depths = 128, Number of DMEs = 3)

Varying image size

Figure 4.18 shows the total time taken to compute extract 3D surfaces using different image sizes. The values are listed in Table (4.8). As indicated by Fig. (4.18), there is a linear relationship between the time required for the computation of 3D surfaces and the image size. As expected, the performance increases four fold for the halving of large images. However for small images, the performance bottleneck lies in the actual imaging devices rather than in the extraction of 3D surfaces. For example, nearly 81% of the overall time is spent acquiring and rectifying 80×60 images in contrast to 30% for 640×480 images.

The smart cameras operate (best case for online images) at 2.43Hz for rectifying

Image Size	No. of Pixels	Computation Time (ms)	σ
80×60	4800	1101.20	24.17
160×120	19200	1941.5	48.75
224×168	37362	2990.37	26.95
320×240	76800	5345.07	98.93
640×480	307200	22551.03	2534.95

Table 4.8: Computation time for different image sizes (Mask Size = 9×9, Number of Depths = 128, Number of DMEs = 3, Number of Stereo Pairs = 4)

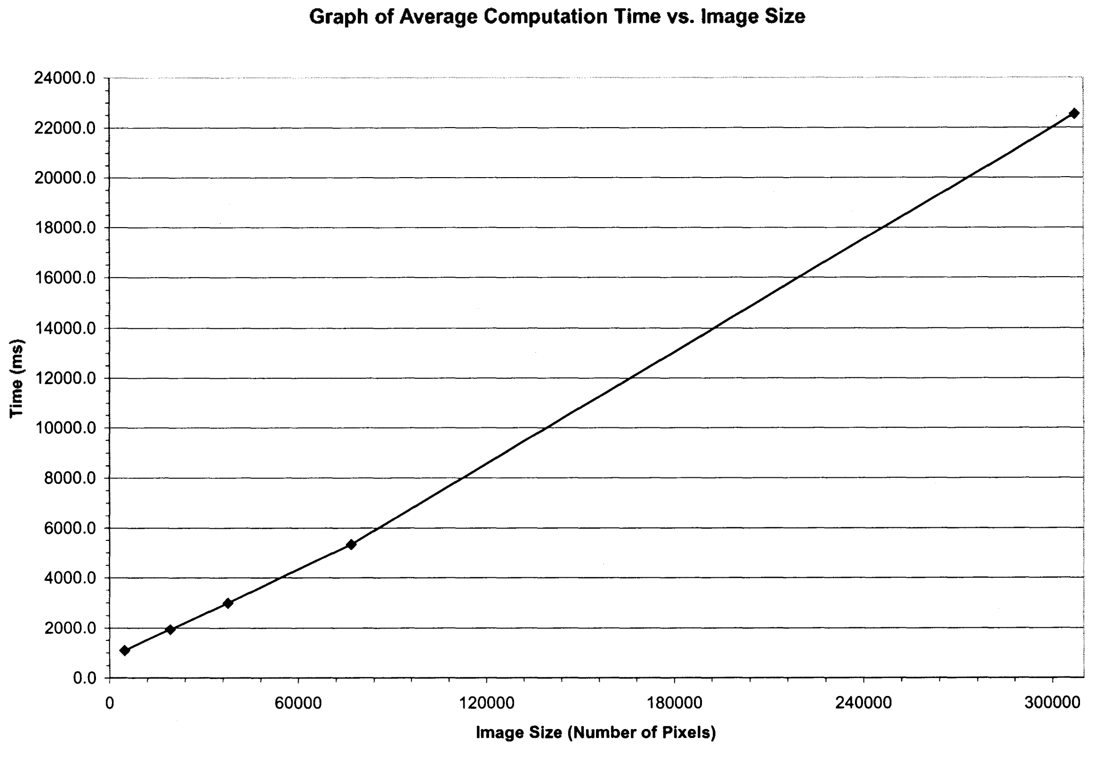


Figure 4.18: System performance for different Image Sizes (Mask Size = 9×9, Number of Depths = 128, Number of DMEs = 3, Number of Stereo Pairs = 4)

640×480 images while increasing to 4.97Hz and 9.71Hz for 50% and 25% shrinking of image size respectively. Image rectification for online images experience the overhead of the actual image acquisition, which varies from vendor to vendor. In the presented system, due to limitations in the actual hardware, image acquisition can take upto 3 times longer than the specified operating frequency of 30Hz, operating at approximately at 11.11Hz on average.

It should be noted that the above temporal performance of the system is directly dependent upon the pose of the camera. Rectification causes the image sizes to enlarge from their original image size. Hence, the greater the divergence of views among the cameras, the greater the severity of their image rectification. Therefore, the rectified images can drastically enlarge, even for very small images. However the overall relationship between the original image size and computing time would still remain.

4.3.1 3D Surface Extraction Accuracy

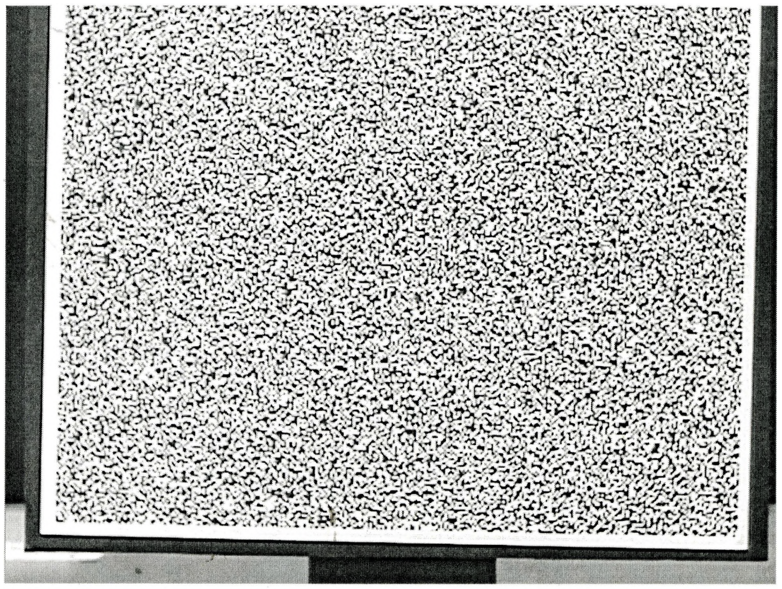


Figure 4.19: Testing object used to measure the surface extraction accuracy of the system

To convey a complete illusion of virtual or augmented reality, it is necessary that the 3D surface extraction process measures the real world scene with sufficient accuracy. The accuracy of the presented system is determined by extracting the 3D surface of a plane. From the computed depth map of the plane, the degree of measured "flatness" provides a gauge for the accuracy of the system. The actual physical plane shown in Fig. (4.19) consists of a rectangle cardboard that is covered with random dots to allow for accurate stereo matching. Given the arrangement of cameras shown in Fig. (3.7), the plane is mounted on the optical rail at a distance of approximately 1040mm from Camera_0.

The depth map computed using all five cameras, with Camera_0 as the reference camera, is shown in Fig. (4.20). For clarity, the 3D plot of this depth map is shown in Fig. (4.21). These figures depict the calculated surface of the plane to contain widespread error on the order of 5mm. They also show that the camera is oriented on a slight angle with respect to the plane, with the top-left corner of the plane being closer to the camera than the bottom-right.

The errors in the 3D surface of the plane are produced from the aggregation of inaccuracies in the pose estimation of each camera. The depth map calculated from the *rectified* stereo pair of Camera_0 and Camera_2 (with Camera_0 as the reference camera) is shown in Fig. (4.22) and Fig. (4.23). These figures show that indeed a plane is measured from such a camera setup, and the pose of the rectified reference camera is such that the bottom-right of the plane is closer to the camera than the upper-left.

The depth map calculations of the plane from the *unrectified* reference camera (using the same camera pair) is shown in Fig. (4.24) and graphed in Fig. (4.25). These figures depict the plane comprising of small saw-tooth like ridges that are approximately 5mm high. Recall that the depth (or Z_O) in the unrectified reference camera coordinate system is related to the rectified reference coordinate system (Z_R) by Eq. (2.65). As each pixel is transformed into the world coordinate system form the rectified camera coordinate system, a uniform error is introduced, whereby the pose of Camera$_0$ is erroneously determined to have gradual inclination (from the bottom-right to the top-left). This error is linear with respect to the rows and columns of the image, thereby forming a the saw-tooth like appearance when applied to the discrete depth map calculated from the rectified reference camera. Due to the inaccuracies in determining the pose of each camera used within the system, the MBS algorithm consequently provides "best-fit" estimation (within 5mm) when searching across multiple images using inverse depth.

The discrete depth measurements of the plane surface are a direct consequence of the integer pixel disparity calculations. The relationship between change of depth (Z) with respect to the change in disparity (d) is given by (using Eq. (2.3)):

$$\frac{\partial d}{\partial Z} = -\frac{Bf}{Z^2} \tag{4.1}$$

For the given setup, baseline between Camera$_0$ and Camera$_2$ B=87.54mm, focal length of Camera$_0$ f=16.56mm and the average plane distance from Camera$_0$ Z = 1040mm, thus providing a resolution of 1.34×10^{-3}mm disparity measurement per 1mm change in depth. Since the camera pixels are 7.6μm, 1 pixel disparity corresponds to Z_{min} = 5.67mm change in depth. This minimum measurement of depth due to the integer disparity measurements is evident in the previous plots. Furthermore, it is due to Z_{min} that the saw-tooth like ridges in Fig. (4.23) peak at 5mm before the depth changes.

For every rectified camera pair, the measured plane varies in orientation. Hence when they are all combined to form a unified depth map via the MBS algorithm, they produce a result that retains the macro properties of the plane (as seen from Fig. (4.21)), however micro properties contain errors on the order of 5mm.

These small surface errors are omissible for scenes with large arbitrary objects. An example of such a depth map is shown in Fig. (4.27) of the arbitrary scene shown in Fig. (4.26). The depth captures the significant features of all objects in the scene with sufficient detail. The areas of gross errors are due to insufficient texture, or viewpoint occlusion.

The advantage of using the MBS algorithm is to harness the advantages of small and large baselines. To illustrate this feature explicitly, the 3D surface of a cylinder with diameter 100mm placed approximately 935mm from reference camera (Camera_0) (shown in Fig. (4.28)) is extracted. Figure (4.29) shows the depth map of the scene using Camera_0 and Camera_3. The 3D plot of this depth map in Fig. (4.30) shows the maximum depth of the cylinder is determined to be approximately 25mm and comprised of a number of discrete planar depths obeying Eq. (4.1). The extreme errors in matching or "noise" (due to occlusion) can also be seen on the right side of the object in Fig. (4.29).

Figure (4.31) shows the depth map of the cylinder extracted using all camera pairs, and is graphed in Fig. (4.32). These figures portray the cylinder to have a maximum depth of 40mm which is approximately the viewable surface of the cylinder. Although discrete planes may be distinguished in the depth map, the region between them is interpolated, producing a more accurate representation of the object. Furthermore, there is a large reduction in noise in the extracted 3D surface. These gains are a direct consequence of combining different viewpoints via the MBS algorithm.

Sources of Error

Various sources contribute to error of depth maps acquired from this technique. Firstly, the camera calibration technique although provides accurate data for rectification, the inaccuracies play a larger role during 3D surface extraction. This is due to the fact that camera pose errors are absorbed in projective transformation to produce correct rectified images. However, they provide erroneous disparity measurements when performing the SSSD calculations. The effect of these errors decreases as measurements are made at greater distance from the camera due to the spatial quantization. The calibration errors are due to limited resolution of measuring tools and human error in reading amidst calibration. This source of error can be statistically reduced through extensive measurements. Largely however, the calibration errors lie in the detection of the calibration features in images.

Secondly, the cameras do not capture the scene synchronously. Thus they not only differ from each other due to the camera internal characteristics (thermal noise, etc.), but also due to slight change in lighting. Furthermore, each of the cameras have a manual iris and auto-gain functions, which cause the images of the same scene to differ in greyscale value. The use of SSSD theoretically would still yield the correct result under normal circumstances. However it is likely that the camera gain is non-linear, which may cause saturation of the CCD and aberrations in the histogram. The effect of these sources can be minimized through histogram equalization.

Thirdly, errors are introduced numerically due to the conversion of numbers to and from pixels (such as during depth search). The effect is minimal, but does advocate that detectable features must be at least one pixel wide.

Finally, the performance measurement of the system over time provides a estimate at best, due to the timing inaccuracies introduced by the non-real time operating system and the limited resolution of the timer (1ms). The network switch is configured for normal operation, which does not isolate the workstations from the normal network operations such as broadcasts, ARP requests and browse requests.

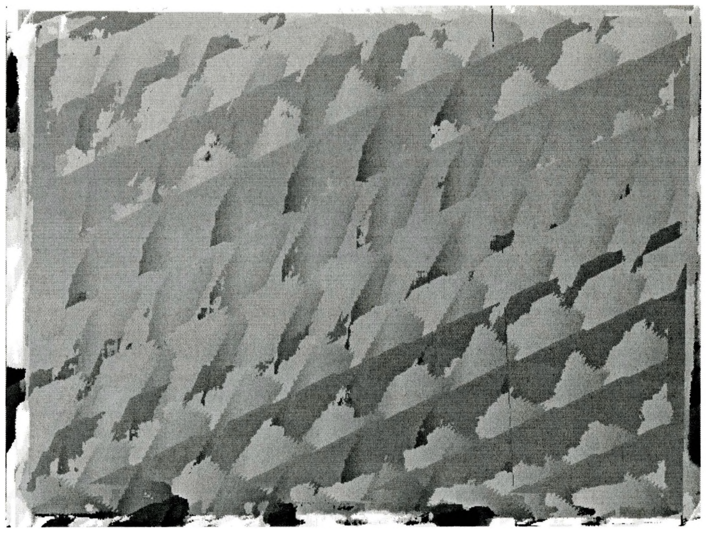


Figure 4.20: Depth map of the plane using five cameras

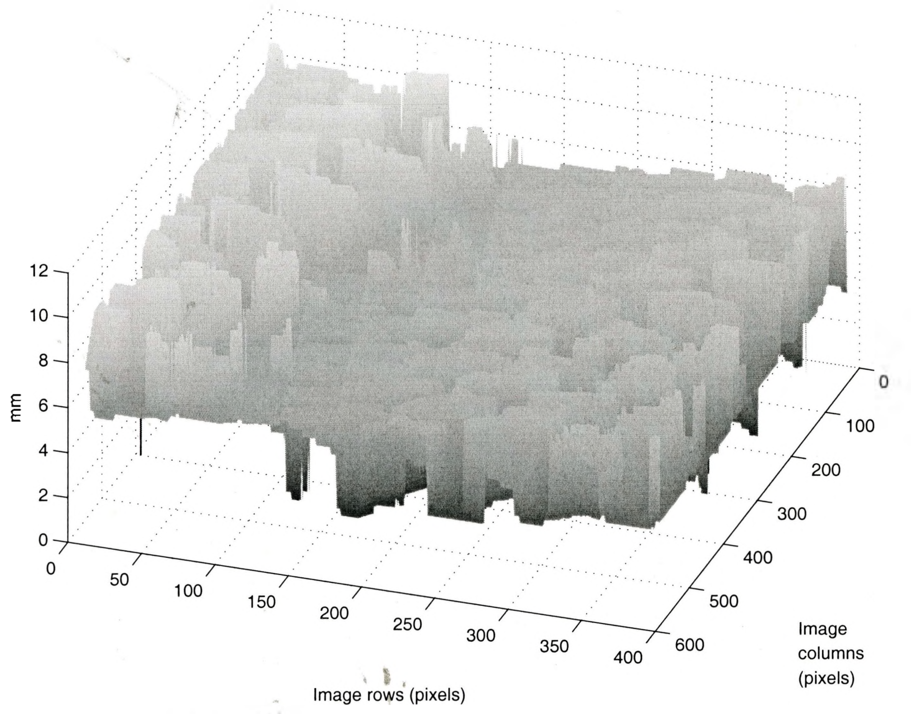


Figure 4.21: 3D plot of depth map of the plane using five cameras

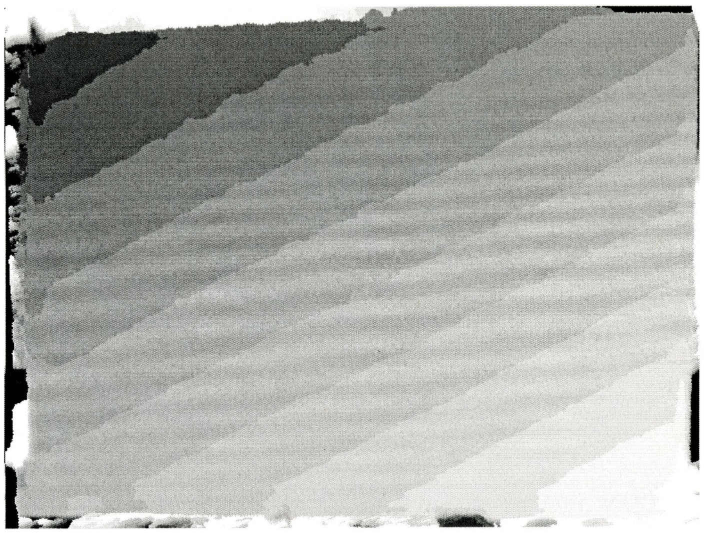


Figure 4.22: Depth map of the plane in rectified Camera$_0$ coordinate system (using Camera$_2$)

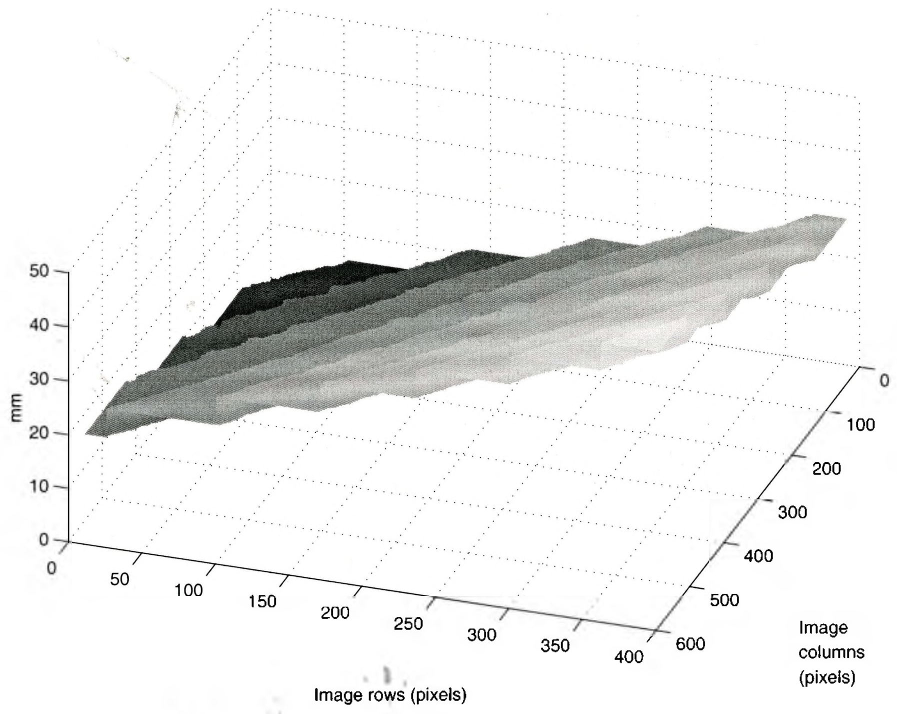


Figure 4.23: 3D plot of depth map of the plane in rectified Camera$_0$ coordinate system (using Camera$_2$)

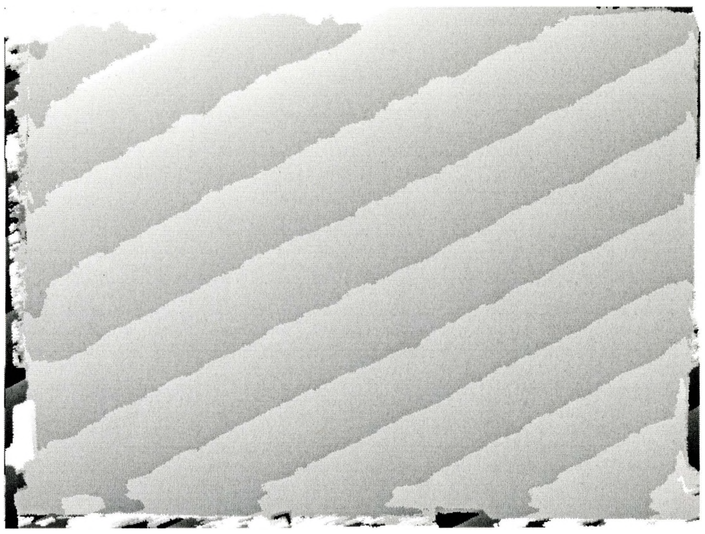


Figure 4.24: Depth map of the plane in unrectified Camera$_0$ coordinate system (using Camera$_2$)

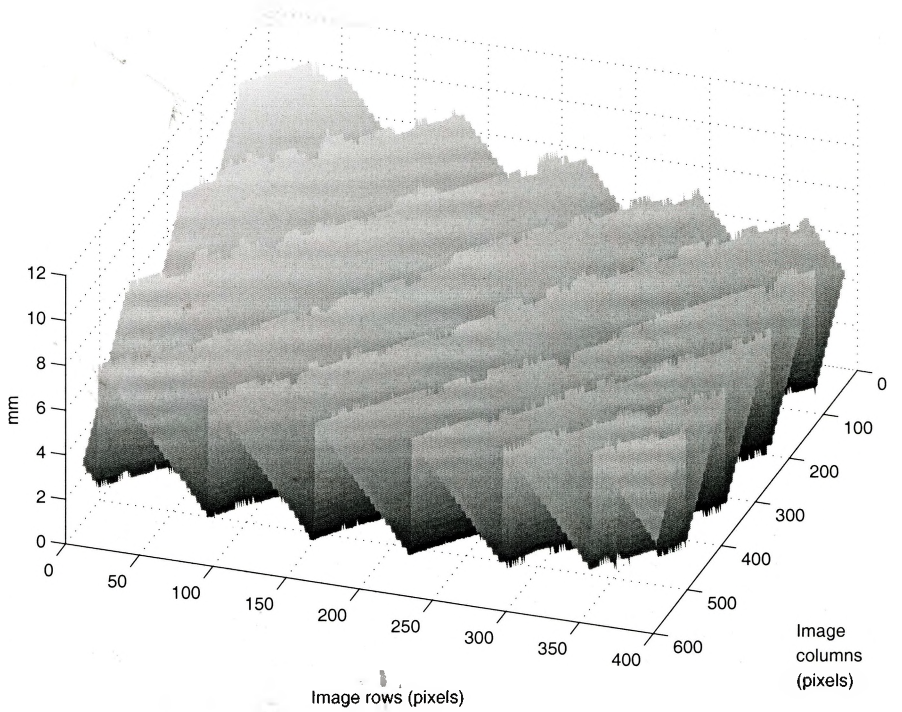


Figure 4.25: 3D plot of depth map of the plane in unrectified Camera$_0$ (using Camera$_2$)

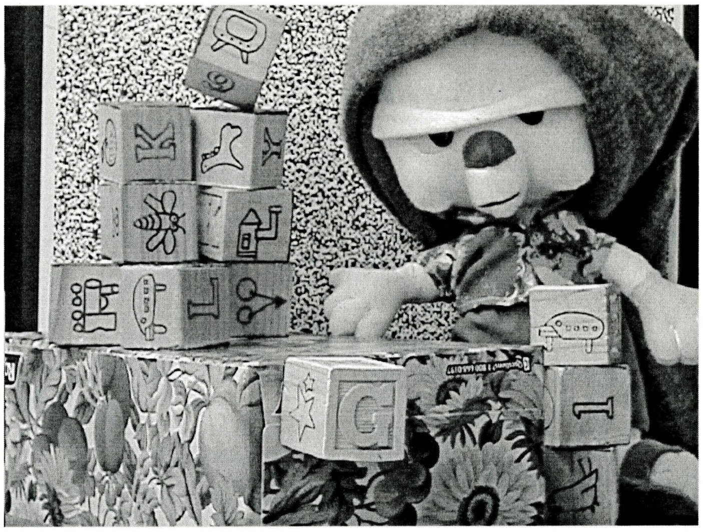


Figure 4.26: Arbitrary scene as viewed by reference Camera$_0$

Figure 4.27: Depth map of the arbitrary scene

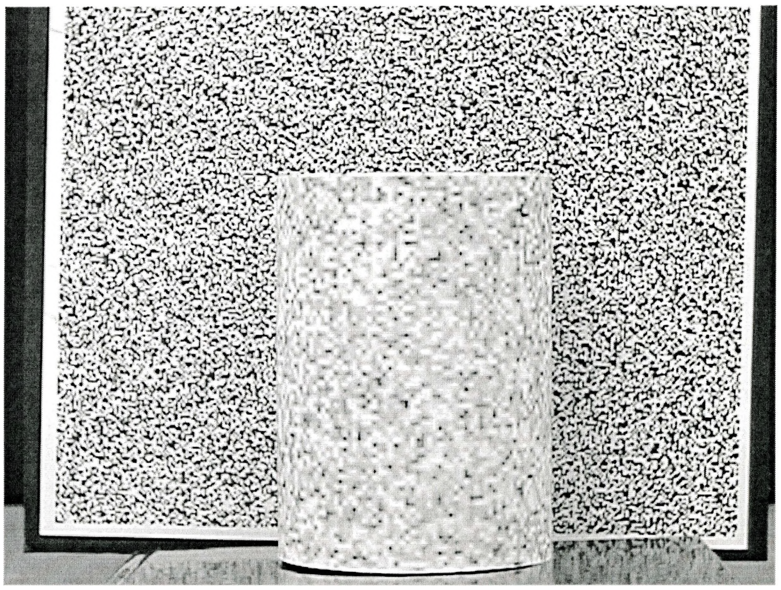


Figure 4.28: Cylinder scene viewed by Camera$_0$

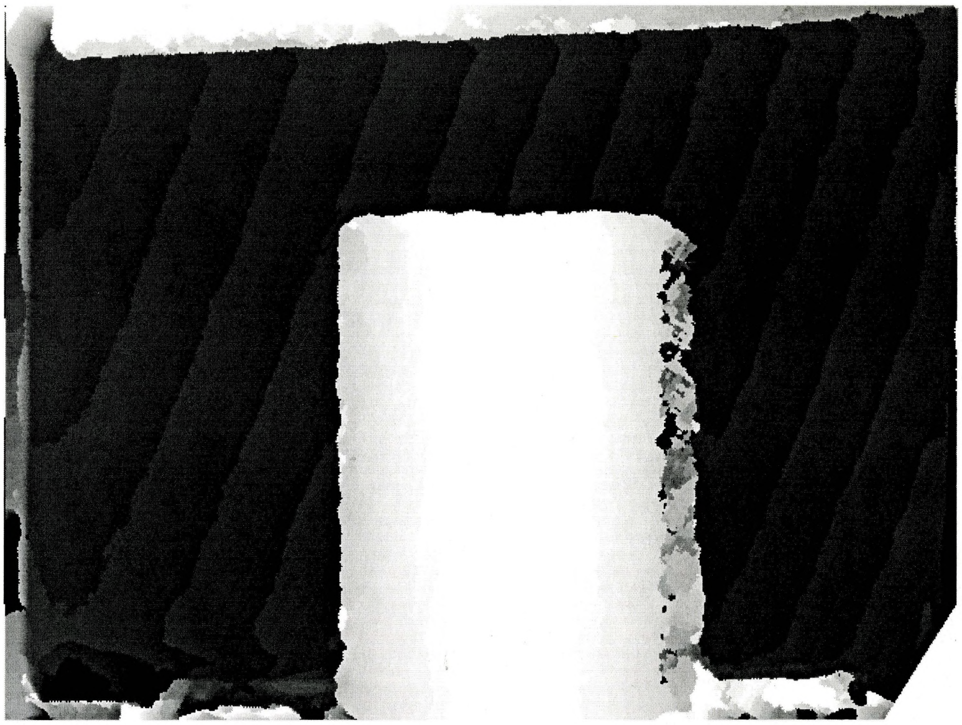


Figure 4.29: Depth map of the cylinder from reference Camera$_0$ and Camera$_3$

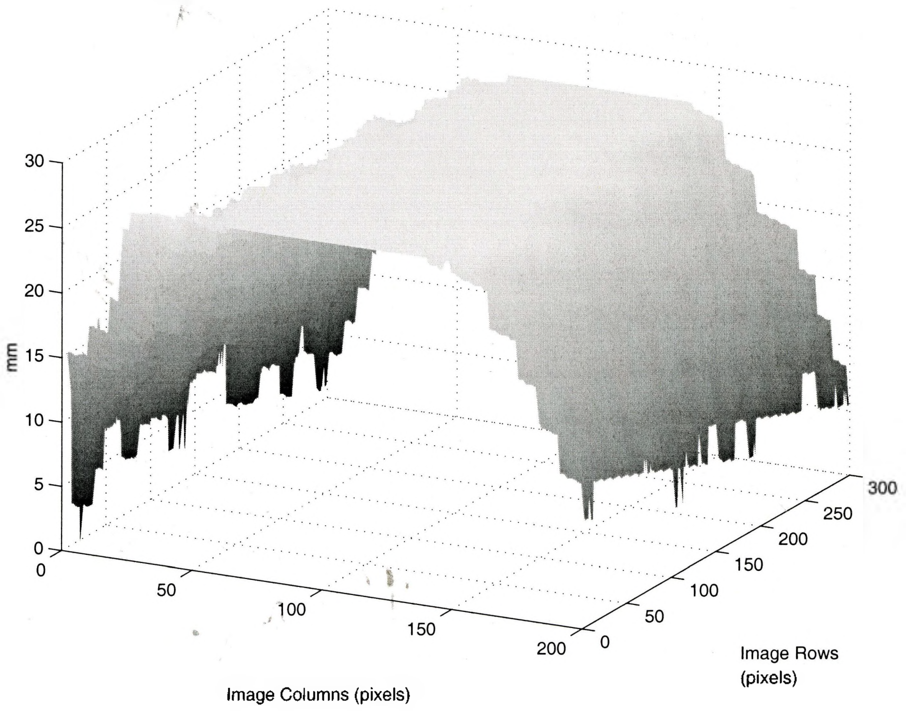


Figure 4.30: 3D plot of depth map of the cylinder from reference Camera$_0$ and Camera$_3$

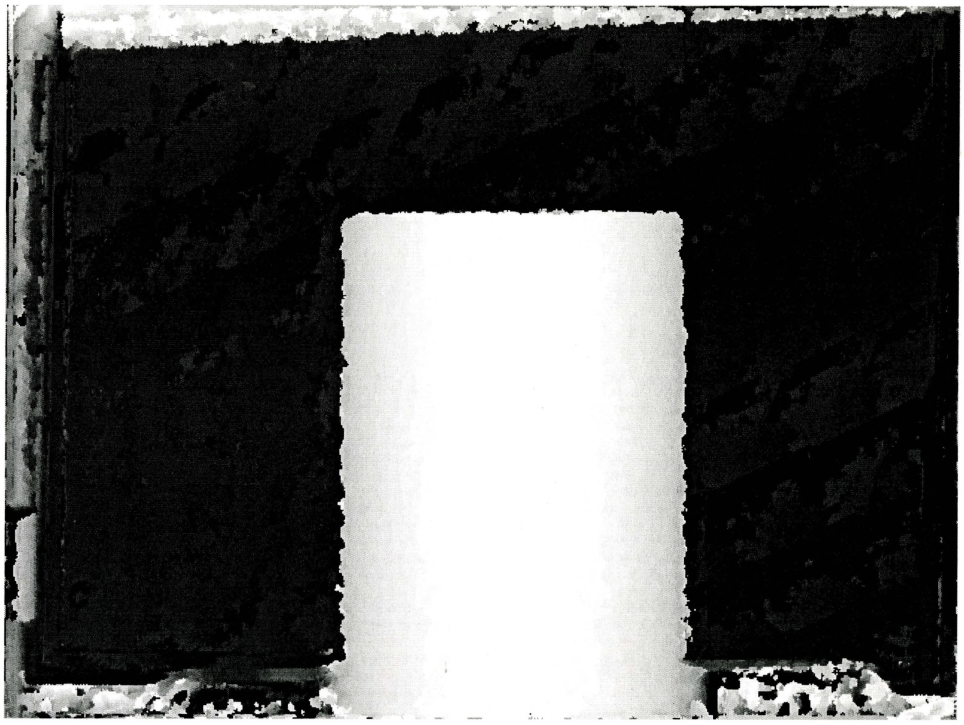


Figure 4.31: Depth map of the cylinder from all five cameras

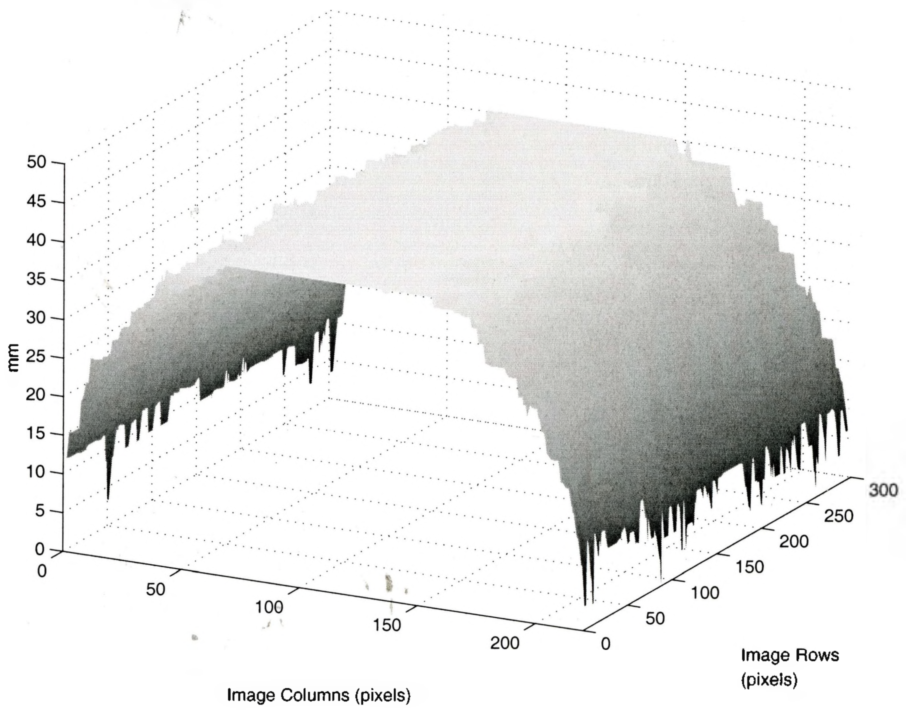


Figure 4.32: 3D plot of depth map of the cylinder from all five cameras

Chapter 5

Final Thoughts

5.1 Towards Distributed Vision Systems

In this thesis, it has been shown that 3D surfaces of an arbitrary scene can be extracted through passive multi-baseline stereo techniques over a network of workstations. Moreover, a software system was built using standardized distributed object technology (CORBA) which clearly demonstrates the feasibility and flexibility of the adopted software architecture. Although the cameras can be placed arbitrarily, they are assumed to be stationary with respect to the world coordinate system. Most scenarios requiring 3D surface estimation of a scene generally meet this requirement.

The placement of arbitrary cameras eliminates the need for expensive and custom stereo rigs, while allowing the scene to be viewed omnidirectionally. The accuracy of the system is shown to closely approximate the theoretical limit dependent on the physical camera resolution. The accuracy is directly related to the correspondence detection and consequently highly sensitive to camera calibration errors. Moreover, the use of distributed object technology allowed for the creation of smart cameras which are capable of performing complex tasks in addition to image capture. This also allows multiple applications to share imaging resources when feasible. The smart cameras are shown to operate at acceptable (and predictable) levels.

The system uses standard off-the-shelf hardware, which not only reduces the cost of the final system, but also eliminates the need for any special software considerations (such as special parallel processing architectures and communication protocols).

The built system can only be employed in non-real time environments, primarily due to the non-real time operating systems and ORBs that comprise the system. The performance of the system is dominated by the non-optimized implementation of the MBS algorithm and linearly dependent upon the system configuration parameters.

The large scale deployment of network of workstations (forming the Internet) has offered ubiquitous connectivity and economies of scale. As shown in this thesis, this has provided a practical alternative to expensive, specialized hardware solutions in computer vision by moving the solution into the software domain. This allows the solution to gain the advantages of software solutions, namely re-usability, flexibility, adaptability and manageability. This does come at the cost of performance. However, from the analysis of the presented system, it is evident that current technologies make it feasible to implement vision systems where hardware performance gains may be unnecessary and the elegance and simplicity of a distributed software solution would suffice.

5.2 Future Work

This study has successfully shown the application of distributed computing concept to computer vision, however there are many practical considerations which warrant further research.

In general, the use of real-time operating systems and ORBs would improve the performance of the overall system significantly. Much of the code adheres to ANSI C++ standards and thus would port to other platforms readily. System performance gains may also be achieved by simply upgrading to faster workstations.

5.2.1 Scene Capture

The current technique for camera calibration requires that all cameras view a large volume of space. This imposes some limitations on the camera placement within the scene. The remedy could be to adopt another calibration object that is visible from all directions (such as an LED light source, or a calibration cube) or via another calibration technique. Furthermore, weak calibration may be used in the system if relative measurements among cameras can be integrated.

The calibration object detection algorithm currently relies on a simple method which is highly susceptible to errors due to lighting conditions and noise. Although the results are accurate enough to perform proper image rectification, these calibration errors propagate into the final 3D surface estimates. Greater accuracy could be achieved through more robust detection of calibration features, such as the ellipse extraction method described in [15] and [10]. The feature extraction process can also be extended to extract features autonomously.

At present, stationary cameras are used within the system. The system could be extended to use cameras with motorized zoom, pan/tilt capabilities to enhance the resolution in desired regions. Although this may be seen as the use of specialized hardware, these features have become inexpensive and commonplace for standard CCD cameras due to their high demand in surveillance. The use of zoom or pan/tilt cameras would increase the viewable scene area without the need for additional cameras. Furthermore, baselines could be dynamically adjusted based on environment sensing. The groupings of cameras into stereo pairs is currently performed by the user along with the selection of the reference camera. This could be automated based on the calibrated parameters of the cameras to optimize the benefits gained from cameras with varying baselines. Mobile cameras may require online calibration techniques.

Although the thesis is largely motivated by the requirements for 3D surface extraction of dynamic events, the major issue of synchronous capture of such a scene over a distributed network has been left unaddressed. This is a complex issue, with no simple solution in sight. As vision systems adopt the distributed computing environment, this will become of greater importance.

The enhancements for smart cameras could include histogram equalization and LoG filtering. These would aid in the reduction of errors during correspondence of errors during correspondence search.

5.2.2 3D surface reconstruction

The software implementation of the MBS algorithm could be optimized for faster performance. The current implementation was generated by "ease-of-coding" model, with little attention to performance.

To improve the MBS algorithm itself, dynamic window selection methods could be adopted to reduce the blurring of object boundaries. This could be based on the variance local region analysis to detect texture or edges. A coarse-to-fine strategy would also help to reduce the number of false matches and preserve object boundaries. The depth estimates can be improved through interpolation and noise may be reduced through median filtering.

The performance of the system suggests that it may possibly be better suited for algorithms with fewer computation requirements, such as model based 3d surface estimation. Given certain assumptions of structures in the scene, faster stereo techniques may be employed. For example, a scene primarily consisting of cubes can be decomposed to a edge map, and multiple components could simultaneously track separate cubes. These models may be may be dynamically updated (such as through Kalman filtering).

Pre-processing of images could be performed, whereby textureless regions could be identified, and consequently avoided by the correlation based stereo techniques. Other estimation techniques (such as shape from shading, shape from focus/defocus) could be applied in these regions and the combined with the results from MBS.

Since the system is distributed and parallel, 3D surface extraction can be performed by several different techniques simultaneously, and then combined through discrete or fuzzy logic.

Currently the system displays the 3D surfaces for one viewpoint in a form of a grey scale depth map. This could be enhanced to extract a 3D mesh model for the scene and render it using shading or texture mapping. Due to the peculiarities of human vision, inaccuracies of a 3D model to a certain extent can be hidden through correct texture-mapping to a certain degree. Furthermore, a complete voxel representation of the scene can be built by stitching depth maps from different viewpoints together. These enhancements would allow for novel view generation and virtual/augmented reality applications.

5.2.3 Distributed Computing

Currently, the VACF does not provide for a convenient method for objects residing on the same server to interact locally. If used to build more complex systems, this would surely be required. As mentioned earlier, the adoption of a real-time implementation of CORBA would reduce the overhead and increase the quality of service.

Multiple Image Servers could be used to distribute the capturing and rectification of the images over several workstations. This would reduce the overall overhead and moves the system one step closer to synchronous capture of dynamic events.

Appendix A

Camera Model

A.1 Pinhole Camera Model

A camera captures a 3D scene in 2D, and thus represents a mapping between the 3D world space and a 2D image. This mapping is derived under the assumption that the camera is a perfect pinhole camera. The *pinhole camera model* assumes that light rays pass through an infinitesimal aperture at the front of the camera, to form a proportional image of scene on the image plane as shown in figure A.1(a). In such a configuration, the image captured on the image plane is inverted vertically, and thus

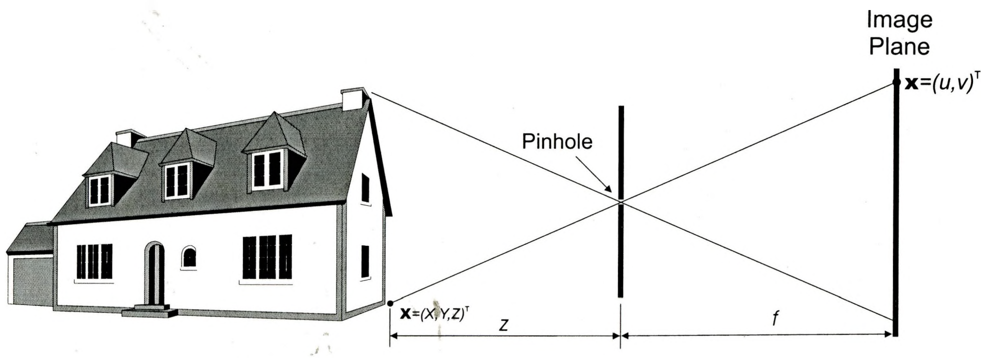


Figure A.1: Pinhole Camera

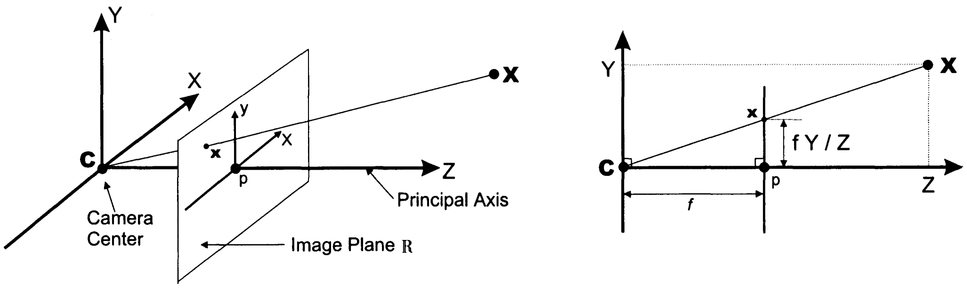


Figure A.2: Pinhole camera geometry

it is customary to avoid this inversion by positioning the image plane in front of the camera center (as shown in figure A.1(b)). This model is a substantial simplification of the actual cameras, since it ignores their property of variable aperture sizes and use of lenses which allow them to work under different lighting conditions. However, these features do not violate any of the assumptions defined by the ideal pinhole camera model and describe the CCD cameras used in the system with sufficient accuracy.

The pinhole camera geometry is shown in figure A.2, where the camera is modeled with an optical center **C** and image plane $\mathbb{R}$. **C** coincides with the origin of the 3D coordinate system, with $\mathbb{R}$ parallel to the **XY** plane at a distance f from the camera center. The $z - axis$ (also called the *optical axis* or *principal axis*) perpendicular to $\mathbb{R}$, intersecting it at the *principal point* **p**.

A.2 Central Projection

According to the *law of collinearity*, the scene point $\mathbf{X}$, the corresponding image point $\mathbf{x}$ and the camera center $\mathbf{C}$ all lie on the same line (figure A.2). Using similar triangles $\mathbf{X} = (X, Y, Z)^T$ is mapped to $\mathbf{x} = (u, v)^T$ by

$$(X, Y, Z) \mapsto \begin{pmatrix} u \\ v \end{pmatrix} = \frac{f}{Z} \begin{pmatrix} X \\ Y \end{pmatrix} \tag{A.1}$$

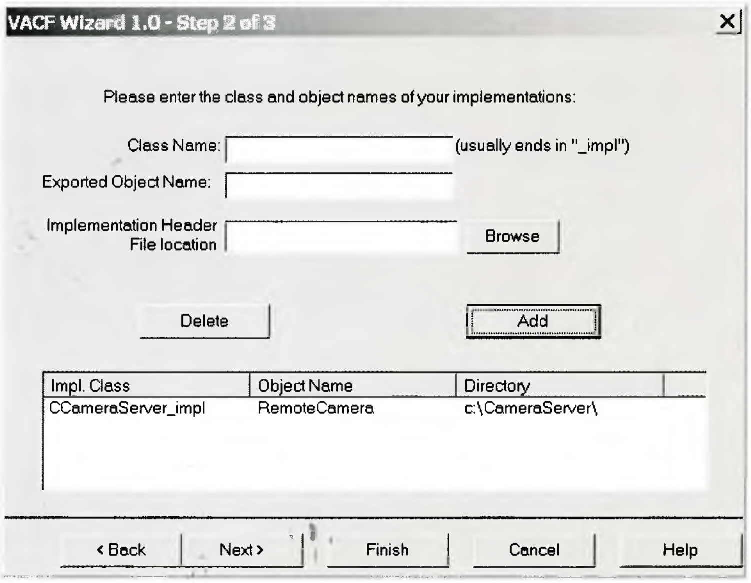
which describes the *central projection* mapping from world to image coordinates. Equation (A.1) can be described as a linear mapping using homogeneous coordinates as

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 1 & 1 & 0 \\ 0 & f & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{A.2}$$

Appendix B

VACF Wizard Screenshots

The VACF Wizard screenshots are presented. Note that the "Step 1 of 3" in the wizard is a greeting to the user. In the first step, the information for each object being exported is supplied. This includes the class name and the name the object that will be supplied to the name server. In the final step, the user provides the default name of the server.


VACF Wizard 1.0 - Step 2 of 3

Please enter the class and object names of your implementations:

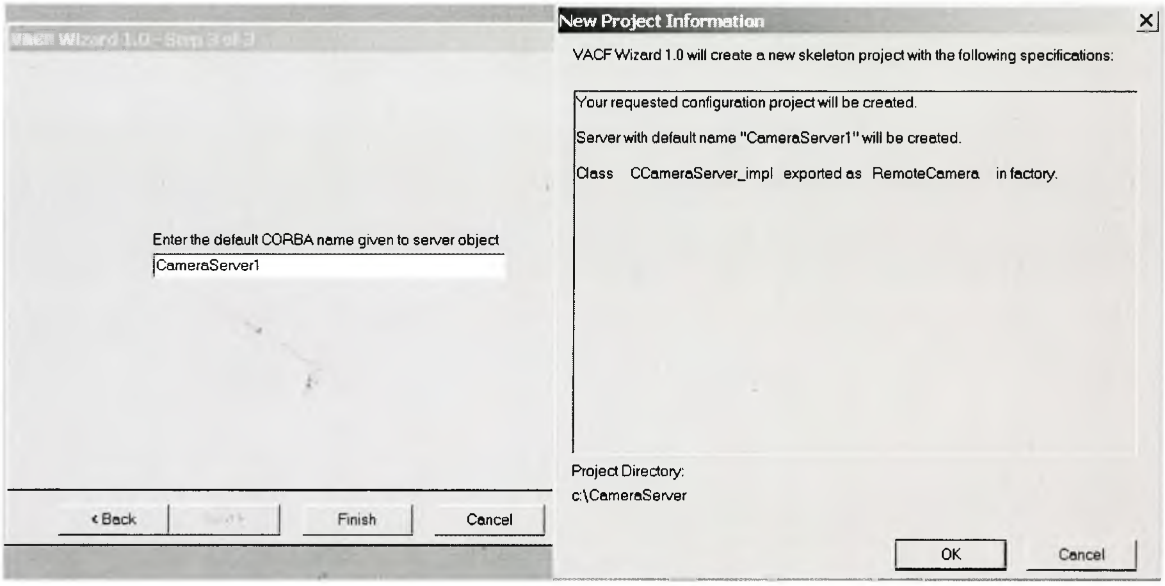
Class Name: (usually ends in "_impl")

Exported Object Name:

Implementation Header File location

Impl. Class	Object Name	Directory
CCameraServer_impl	RemoteCamera	c:\CameraServer\

Figure B.1: Step 1 of the VACF Wizard



Enter the default CORBA name given to server object

CameraServer1

VACF Wizard 1.0 will create a new skeleton project with the following specifications:

Your requested configuration project will be created.

Server with default name "CameraServer1" will be created.

Class CCameraServer_impl exported as RemoteCamera in factory.

Project Directory:
c:\CameraServer

Figure B.2: Step 2 of the VACF Wizard

Appendix C

Sample application using VACF

Presented is an example of a complete application (both client and server) built through VACF. In the example, an image server is built which transmits a sequence of images as listed in a file. The client simply makes a number of calls to retrieve the images from the remote server. The IDL generated class definitions and code stubs have been omitted for brevity.

```
typedef sequence<char> UnboundCharSeq;

interface CImageSeqServer {
        //called to initialize the server
        boolean Init();
        //get an image from the server
        boolean GetNextImage(out UnboundCharSeq ImageBuf, out long ImageWidth, out long ImageHeight);
};
```

ImageSeqServer.idl: Interface of the image server defined in IDL

```
/********* VACF Auto Generated File *******/
// Component.cpp: implementation of the CComponent class
//////////////////////////////////////////////////////////////////////

#include "Component.h" //part of VACF library
#include <string.h>
#include <assert.h>
#include "ComponentServer.h"

CComponent::CComponent() { }

CComponent::CComponent(COrb* pOrb)
{
    // check preconditions
    assert(pOrb != NULL);

    // start up the user's object
    m_pApp = new CComponentServer(pOrb);

    // flag to indicate not to call the base class destructor twice
    bBaseClassDestructorCalled = false;
}

CComponent::~CComponent()
{
    // m_pApp is a subclass of CComponent, so CComponent
    // destructor will get called twice.
    if(bBaseClassDestructorCalled == false)
    {
        bBaseClassDestructorCalled = true;
        delete m_pApp;
    }
}

void CComponent::SetAppInstance(const char* lpszApplication)
{
    // check preconditions
    assert(lpszApplication != NULL);

    //remember the "client" we are associated with
    m_pApp->SetAppInstance(lpszApplication);
}

void CComponent::Run()
{
    // check preconditions
    assert(m_pApp != NULL);

    //execute
    m_pApp->Run();
}
```

Component.cpp: Source Code

```
/********* VACF Auto Generated File *******/

#include "Component.h"
#include "Server.h"
#include "ImageSeqServer_impl.h"

class CComponentServer : public CComponent
{
public:

    CComponentServer(COrb*);
    ~CComponentServer();

    void Run();

    void SetAppInstance(const char*);

    CServer<CImageSeqServer_impl>* m_User0; //user type
};
```

ComponentServer.h: Source Code

```
/********* VACF Auto Generated File *******/
// ComponentServer.cpp: implementation of the user application
//////////////////////////////////////////////////////////////////////
#include "ComponentServer.h"
#include <string.h>
#include <assert.h>

CComponentServer::CComponentServer(COrb* pOrb)
{
    // check preconditions
    assert(pOrb != NULL);
    // assign the reference to the orb object
    m_pOrb = pOrb;
}

CComponentServer::~CComponentServer()
{
    //each user type needs to be disconnected
    m_User0->Disconnect(m_sInstance.data(), "ImageSeqServer");
    delete m_User0;
}

void CComponentServer::SetAppInstance(const char* lpszApplication)
{
    // check preconditions
    assert(lpszApplication != NULL);

    m_sInstance = lpszApplication;
}
```

ComponentServer.cpp: Source Code (Server Side)

```
void CComponentServer::Run()
{
    // check preconditions
    assert(m_pOrb != NULL);

    //startup all user types
    m_User0 = new CServer<CImageSeqServer_impl>(m_pOrb);
    m_User0->Connect(m_sInstance.data(), "ImageServer");
}
```

ComponentServer.cpp: Source Code (continued)

```
#ifndef IMAGE_SEQ_SERVER_H_INCLUDED
#define IMAGE_SEQ_SERVER_H_INCLUDED

#include <OB\Corba.h>
#include "idl\ImageSeqServer_skel.h"
#include "PGM.h"

#define MAX_IMAGE_SIZE 307200 //640x480

//user defined implementation
class CImageSeqServer_impl : public CImageSeqServer_skel {
private:
    FILE *m_pImageFile;     //file pointer to the image file
    FILE *m_pSequenceFile;  //file pointer to the sequence file
    long m_nImageNumber;    //the number of images served
    CPGM m_Image;           //image data (CPGM is a user helper class to read
                            //and write PGM files).

    bool NextFileName(char *FileName);  //private function to determine the next image
                                        //file to be served as listed in the sequence file.

public:
    CImageSeqServer_impl();
    ~CImageSeqServer_impl();

    //idl interface
    CORBA_Boolean Init();
    CORBA_Boolean GetNextImage(UnboundCharSeq*& Image, CORBA_Long &ImageWidth, CORBA_Long &ImageHeight)
};

#endif
```

ImageSeqServer_impl.h: Source Code (Server Side)

```
#include <conio.h>
#include <stdio.h>
#include "ImageSeqServer_impl.h"

CImageSeqServer_impl::CImageSeqServer_impl()
{
    //initialize variables
    m_pImageFile = NULL;
    m_pSequenceFile = NULL;
    m_nImageNumber = 0;
}

CImageSeqServer_impl::~CImageSeqServer_impl()
{
    if (m_pImageFile)        fclose(m_pImageFile);
    if (m_pSequenceFile)     fclose(m_pSequenceFile);
}

CORBA_Boolean CImageSeqServer_impl::Init()
{
    //open the sequence file
    if ((m_pSequenceFile = fopen("ImageSeq.seq", "rt")) == NULL)
    {
        printf("\nCannot Open Sequence File\n"); return false;
    }
    return true;
}

CORBA_Boolean CImageSeqServer_impl::GetNextImage(UnboundCharSeq*&
Image, CORBA_Long &ImageWidth, CORBA_Long &ImageHeight)
{
    char ImageFileName[_MAX_PATH];

    //read the next filename
    if (NextFileName(ImageFileName) == false)
    {
        printf("Error determining next file from sequence file"); return false;
    }
    if (m_Image.Load(ImageFileName)) //load the image from the file
    {
        //set the array to be passed to the requesting object
        Image = new UnboundCharSeq(MAX_IMAGE_SIZE, m_Image.GetWidth() * m_Image.GetHeight(), \
                                   m_Image.m_pImageData->GetPtr());
        //copy the image stats
        ImageWidth = m_Image.GetWidth();
        ImageHeight = m_Image.GetHeight();
        //print out statistics
        m_nImageNumber++;
        printf("ImagesServed %d (%s)\n", m_nImageNumber, ImageFileName);
        return true;
    }
    else
       return false;
}
```

ImageSeqServer_impl.cpp: Source Code (Server Side)

```
bool CImageSeqServer_impl::NextFileName(char *FileName)
{
    //sanity checks
    if (FileName == NULL)
    {
        printf("Invalid filename passed.\n");
        return false;
    }
    if (m_pSequenceFile == NULL) //sequence file must be open
    {
        printf("No sequence file open (file pointer == NULL)\n");
        return false;
    }

    //read the next filename.
    if (feof(m_pSequenceFile))
    {
        fseek(m_pSequenceFile, 0, SEEK_SET); //rewind to the beginning
    }

    fscanf(m_pSequenceFile, "%s\n", FileName);

    return true;
}
```

ImageSeqServer_impl.cpp: Source Code (continued)

```
/********* VACF Auto Generated File *******/

// This file is the MAIN entry point of the server. It will offer these services:
//BEGIN EXPORT
/*

Class     CImageSeqServer_impl   exported as   ImageServer    in
factory.

 */
//END EXPORT

#include <process.h>
#include <conio.h>
#include "ServerFactory_impl.h"
#include "Server.h"
#include "ErrorHandler.h"

#define DEFAULT_SERVER_NAME "MyImageServer"

bool ProcessCommandLine(int argc, char *argv[]);
void LowerCase(char *string);

char ServerName[80];

void main(int argc, char *argv[])
{
    if (!ProcessCommandLine(argc, argv))
        return;

    // start up the factory and make services available
    CServer<CServerFactory_impl> Server;
    Server.Init();
    Server.ConnectFactory(ServerName);
    Server.Start();

    printf("ServerName: %s\nPress Any Key to shutdown.\n", ServerName);

    getch(); //wait for user to shutdown

    Server.Stop();

    // disconnect the factory from the servers
    Server.DisconnectFactory(ServerName);
}

void LowerCase(char *string)
{
    for (int i=0; i<strlen(string);i++)
    {
        tolower(string[i]);
    }
}
```

ServerApp.cpp: Source Code (Server Side)

```
bool ProcessCommandLine(int argc, char *argv[])
{
    argc--; //get rid of the program name

    if (argc == 1)
    {
exit:
        printf("Incorrect Parameters!\n");
        printf("Usage: %s -name ServerName\n", argv[0]);
        printf("No command line parameters starts up default server.\n");
        exit(1);
    }
    else if (argc > 1)  //we have at least one option
    {
        for (int i = 1; i < argc; i++)
        {
            LowerCase(argv[i]);
            if (strcmp(argv[i], "-name") == 0)
            {
                strcpy(ServerName, argv[i+1]);
            }
        }

        if ((strlen(ServerName) == 0))
        {
            goto exit;
        }

    }
    else // no command line options
        sprintf(ServerName, "%s\0", DEFAULT_SERVER_NAME);

    return true;
}
```

ServerApp.cpp: Source Code (continued)

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <OB/Corba.h>
#include "ImageSeqServer.h"
#include "Client.h"
#include "PGM.h"

#define MAX_IMAGE_SIZE 307200 //640x480
#define NUM_IMAGES 4

void main(void)
{
    CClient *ImgSeqClient=NULL;
    CImageSeqServer_var ImageSeqServer;
    UnboundCharSeq_var InputImages[NUM_IMAGES];
    CORBA_Long CRBImageWidth[NUM_IMAGES], CRBImageHeight[NUM_IMAGES];
    int i;

    printf("Creating ImageSeqServer CORBA Client...");
    ImgSeqClient = new CClient();

    ImgSeqClient->Create("MyImageServer");

    printf("Getting ImageSeqServer...");
    CClient_GetRemoteObject<CImageSeqServer_var, CImageSeqServer>
        (*ImgSeqClient, ImageSeqServer,"ImageServer");


    printf("Initialzing Remote Server....");
    if (!ImageSeqServer->Init())
    {
        printf("\n*****Remote server failed to initialize, cannot continue....\n");
        printf("Shutting Down...");
        ImgSeqClient->Destroy();
        printf("Done.\n\nPress any key to exit.\n");
        getch();
    };

    printf("Getting Images...");
    for(i=0; i < NUM_IMAGES; i++)
    {
        ImageSeqServer->GetNextImage(InputImages[i].out(), CRBImageWidth[i], CRBImageHeight[i]);
    }

    printf("Done.\n");
    return;
}
```

Main.cpp: Source Code (Client Side)

References

[1] T. Anderson, D. Culler, and D. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15:54–64, February 1995.

[2] Gorav Arora, Jeff Fortuna, and David W. Capson. A Framework for Distributed, Image-Based Measurement Systems. In *IEEE Instrumentation and Measurement Technology Conference*, May 2000.

[3] Stephen T. Barnard and William B. Thompson. Disparity Analysis of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(4):333–340, July 1980.

[4] Bensrhair, A., Miche, P., and R. Debrie. Fast and Automatic Stereo Vision Matching Algorithm-Based on Dynamic-Programming Method. *Pattern Recognition Letters*, 17(5):457–466, May 1996.

[5] Qien Chen. *Multi-View Image-Based Rendering and Modeling.* PhD thesis, University of Southern California, 2000.

[6] L. Falkenhagen. Hierarchical Block-Based Disparity Estimation Considering Neighborhood Constraints. In *International workshop on SNHC and 3D Imaging*, 1997.

[7] Lutz Falkenhagen. Depth Estimation from Stereoscopic Image Pairs Assuming Piecewise Continuous Surfaces. In *Proceedings of European Workshop on combined Real and Synthetic Image Processing for Broadcast and Video Production*, 1994.

[8] H. Farid, S. W. Lee, and R. Bajcsy. View Selection Strategies for Multi-View, Wide-Baseline Stereo. Technical Report MS-CIS-94-18, Department of Information Science, University of Pennsylvania, Philadelphia, PA, 1994.

[9] O. Faugeras, D. Luong, and Q. Maybank. *ECCV'92, Lecture notes in Computer Science, Vol. 588*, chapter Camera Self-Calibration: Theory and Experiments, pages 321–334. Springer-Verlag, New York, 1992.

[10] A. Fitzgibbon, M. Pilu, and R. Fisher. Direct Least Squares Fitting of Ellipses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5), May 1996.

[11] Pascal Fua. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6(1):35–49, 1993.

[12] Andrea Fusiello, Emanuele Trucco, and Alessandro Verri. Rectification with Unconstrained Stereo Geometry. In *British Machine Vision Conference*, 1997.

[13] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G.Booch. *Design Patterns CD*. Addison Wesley Longman, New York, 1995.

[14] Object Management Group. Real-Time CORBA Specification. http://www.omg.org/ cgi-bin/doc?orbos/99-02-12, December 2000.

[15] J. Heikkil. Geometric Camera Calibration Using Circular Control Points. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1066–1077, 2000.

[16] E.E. Hemayed, A. Sandbek, A.G. Wassal, and A.A. Farag. Investigation of stereo-based 3D surface reconstruction. *Proceedings of SPIE*, 3023, February 1997.

[17] S. Weik J. Wingbermühle. Highly Realistic Modeling of Persons for 3D Videoconferencing systems. In *IEEE Signal Processing Society 1997 Workshop on Multimedia Signal Processing*, June 1997.

[18] Ingemar J.Cox, Sunity L. Hingorani, and Satish B. Rao. A Maximum Likelihood Stereo Algorithm. *Computer Vision and Image Understanding*, 63(3):542–567, May 1996.

[19] Takeo Kanade, Hideo Saito, and Sundar Vendula. The 3D Room: Digitizing Time-varying 3D Events by Synchronized Multiple Video Streams. Technical Report CMU-RI-TR-98-34, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213-3890 USA, December 1998.

[20] S.B. Kang. A survey of image-based rendering techniques. Technical Report CRL 97/4, Digital Equipment Corporation, Cambridge Research Lab, Cambridge, Massachusetts, 1997.

[21] S.B. Kang and H.Q. Dinh. Multi-layered Image-Based Rendering. *Graphics Interface*, pages 98–106, 1999.

[22] S.B. Kang and R. Szeliski. 3-D Scene Data Recovery Using Omnidirectional Multibaseline Stereo. Technical Report CRL 95/6, Digital Equipment Corporation, Cambridge Research Lab, Cambridge, Massachusetts, 1995.

[23] Sing Bing Kang, J.A. Webb, Charles Zitnick, and Takeo Kanade. A Multibaseline Stereo System with Active Illumination and Real-time Image Acquisition. In *The Fifth International Conference on Computer Vision*, pages 88–93, June 1995.

[24] Makoto Kimura, Hideo Saito, and Takeo Kanade. 3D Voxel Construction based on Epipolar Geometry. In *ICIP99*, 1999.

[25] Q.T. Luong and O. Faugeras. Camera calibration, scene motion, and structure recovery from point correspondences and fundamental matrices. *International Journal of Computer Vision*, 22(3):261–289, 1997.

[26] David Marr. *Vision.* W.H. Freeman and Company, San Francisco, 1982.

[27] D. K. McAllister, L. Nyland, V. Popescu, A. Lastra, and C. McCue. Real-Time Rendering of Real World Environments. Technical Report UNCCH TR99-019, North Carolina University, North Carolina, NC, 1999.

[28] J. Mellor. Realtime Camera Calibration for Enhanced Reality Visualization. In *First International Conference on Computer Vision, Virtual Reality and Robotics in Medicine*, pages 471–475, April 1995.

[29] P. J. Narayanan, Peter Rander, and Takeo Kanade. Synchronous Capture of Image Sequences from Multiple Cameras. Technical Report CMU-RI-TR-95-25, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1995.

[30] K. Oda, M. Tanaka, A. Yoshida, H. Kano, and Takeo Kanade. A Video-Rate Stereo Machine and Its Application to Virtual Reality. In *Proceedings of ISPRS '99*, 1999.

[31] J.R. Ohm, K. Grneberg, E. Hendriks, E.M. Izquierdo, D. Kalivas, M. Karl, D. Papadimatos, and A. Redert. A Realtime Hardware System for Stereoscopic Videoconferencing with Viewpoint Adaption. In *Image Communication, special issue on 3D technology*, November 1998.

[32] Masatoshi Okutomi and Takeo Kanade. A Multiple Baseline Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353–363, April 1993.

[33] Maurizio Pilu. Uncalibrated Stereo Correspondence by Singular Value Decomposition. Technical Report HPL-97-96, Digital Media Department, Hewlett Packard Labs, Bristol, August 1997.

[34] Peter Rander. *A multi-camera method for 3D digitization of dynamic, real-world events.* PhD thesis, Carnegie Mellon University, 1998.

[35] Peter Rander, P.J. Narayanan, and Takeo Kanade. Virtualized Reality: Constructing Time-varying Virtual Worlds from Real World Events. In *Proceedings of IEEE Visualization 1997*, pages 277–283, October 1997.

[36] Sebastien Roy and Ingemar J. Cox. A Maximum-flow Formulation of the N-camera Stereo Correspondence Problem. In *Proceedings of International Conference on Computer Vision*, 1998.

[37] Hideo Saito, Shigeyuki Baba, Makoto Kimura, Sundar Vedula, and Takeo Kanade. Appearance-Based Virtual View Generation of Temporally-Varying Events from Multi-Camera Images in the 3D Room. In *Proceedings of Second International Conference on 3-D Digital Imaging and Modeling*, October 1999.

[38] Ioannis Stamos and Peter K. Allen. 3D Model Construction Using Range and Image Data. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2000.

[39] R. Tsai. A versatile camera calibration technique using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, August 1987.

[40] Sundar Vedula, Peter Rander, Hideo Saito, and Takeo Kanade. Modeling, Combining, and Rendering Dynamic Real-World Events from Image Sequences. In *Proceedings of Virtual Systems and Multimedia (VSMM98)*, pages 326 - 332, November 1998.

[41] Reg Wilson. Tsai Camera Calibration Software. http://www.cs.cmu.edu/afs/cs.cmu.edu/user/rgw/www/TsaiCode.html, December 2000.

[42] T. Yuan and M. Subbarao. Integration of Multiple-Baseline Color Stereo Vision with Focus and Defocus Analysis for 3D Shape Measurement. In *Proceedings of SPIE*, volume 3520, pages 44–51, December 1998.

[43] Zhengyou Zhang. A Flexible New Technique for Camera Calibration. Technical Report MSR-TR-98-71, Microsoft Research, Microsoft Corporation, Redmond, WA, December 1998.

[44] C. Lawrence Zitnick and Jon A. Webb. Multi-Baseline Stereo Using Surface Extraction. Technical Report CMU/CS-96-196, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, November 1996.