Sensor Planning for Novel View Generation by Camera Networks

by

James Barabas

B.S., Cornell University (2000)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2006

Author [illegible]
Program in Media Arts and Sciences
August 11, 2006

Certified by [illegible]
V. Michael Bove, JR
Principal Research Scientist
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by [illegible]
Andrew B. Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Sensor Planning for Novel View Generation by Camera Networks

by

James Barabas

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on August 11, 2006, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

This document describes a system for generating novel views of an indoor visual scene by gathering successive 2D images from a set of independent networked robotic cameras. Specifically, the robotic cameras work to seek out texture and geometric information needed to generate the specified synthetic view or views, aiming to–with each successive camera move–increase confidence in the estimates of the pixel intensities in the novel view(s). This system lays the groundwork for future explorations in multi-camera video recording for electroholography and image-based rendering.

Thesis Supervisor: V. Michael Bove, JR
Title: Principal Research Scientist, Program in Media Arts and Sciences

Sensor Planning for Novel View Generation by Camera Networks

by

James Barabas

The following people served as readers for this thesis:

Thesis Reader [illegible]

Wendy Plesniak
Visiting Scholar
Program in Media Arts and Sciences

Sensor Planning for Novel View Generation by Camera Networks

by

James Barabas

The following people served as readers for this thesis:

Thesis Reader [illegible]

Alex (Sandy) Pentland
Toshiba Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Contents

Abstract **3**

1 Introduction **13**

1.1 Motivation . . . 13
1.2 contributions . . . 14
1.3 Overview . . . 15
1.4 Related Work . . . 17

2 Robots for Scene Capture **21**

2.1 Apparatus . . . 21
2.2 Eye Society Robots . . . 22
2.2.1 Mechanical Description . . . 22
2.2.2 Electrical Description . . . 23
2.2.3 Software Infrastructure . . . 25

3 Calibration **27**

3.1 Camera Model . . . 28
3.2 Camera Calibration . . . 31
3.3 Robot Actuator Calibration . . . 32
3.3.1 Servomotor Calibration . . . 33
3.3.2 Track Drive Motor Calibration . . . 33

4 Models from Scenes **35**

4.1 Depth from Stereo Images . . . 35
4.1.1 Graphics Processing Units for Stereo Matching . . . 36
4.1.2 Stereo Matching Algorithm . . . 37
4.1.3 Shader Implementation of Best Match . . . 42
4.1.4 Mesh Discontinuities . . . 43
4.2 3D meshes from many images . . . 45

5 Planning **47**

5.1 Specifying the Scene Capture Task . . . 49
5.2 Determination of Model Completeness . . . 50

5.3 General Planning Strategy 51
5.3.1 Partial Information 53
5.4 Algorithm Overview 54
5.5 Movement Planning for Eye Society Robots 56
5.5.1 Single camera along a rail 57
5.5.2 Translation Planning 59

6 Experiments **61**
6.1 CameraNetworkViewer 61
6.1.1 Simulated planning 62
6.1.2 Depth-map results 63
6.1.3 Reprojection results 63
6.1.4 Real image results 65

7 Discussion **69**
7.1 Calibration 69
7.2 Depth map generation 70
7.3 Mesh generation 70

8 Acknowledgments **73**

List of Figures

1-1 Previously missing scene information is revealed when viewpoint is moved in the direction normal to a depth discontinuity. 17

2-1 Illustration of an "Eye-Society" robot. Drive motor actuates robot along a 12 foot rail. 26

3-1 View volume in rendering model. Space bounded by near and far clipping planes falls in the imaging volume. 30
3-2 Matlab-based interactive camera calibration [Bouguet 2000] used to compute camera geometry from images of checkerboard. 32
3-3 Visualization of camera locations derived from lens distortion computation. 32
3-4 Model edge falls at left edge of a view. When this edge falls inside the synthetic view volume, pivoting camera position left will reveal more of the scene. 34

4-1 Reference image is re-rendered as if it was on a plane in the scene parallel to the reference image plane. 40
4-2 Overview of OpenGL fragment shader used for stereo computation. 41

5-1 Figure from [Se and Jasiobedzki 2005] showing reconstructed scene with areas missing due to occlusion and matching error. 48
5-2 Illustration of depth map and face identifier map for a simulated scene . . . 55
5-3 We only seek to model the approximate visual hull of a scene. Some areas may be occluded in all views. 60
5-4 Model edge falls at left edge of a view. When this edge falls inside the synthetic view volume, pivoting camera position left will reveal more of the scene. 60

6-1 CameraNetworkViewer, an application for interacting with and displaying results from the Eye Society Camera Network 62
6-2 Depth image of elephant model, computed with stereo matching shader. . . 63
6-3 Ground-truth depth image of elephant model, from OpenGL z-buffer. . . . 64
6-4 Reprojected untextured rendering of OpenGL z-buffer depth map above. . . 65
6-5 Reprojected untextured rendering of stereo matched depth map above. . . . 66

6-6 Reprojected textured rendering of stereo matched depth map. 66
6-7 Two images taken by an Eye Society camera, used for depth reconstruction below. 67
6-8 Depth map computed from the stereo pair above. 67
6-9 Reprojection of above depth map. 68

Chapter 1

Introduction

1.1 Motivation

Techniques for recovering geometry from multiple images of the same scene have improved dramatically over the last ten years. In this time, much of the work in the Object Based Media group at the MIT Media Laboratory has explored methodologies for media processing that operate primarily on two-dimensional images. Shawn Becker and Stephan Agamanolis proposed object-based image coding and compositing techniques that stretched the state of the art in the mid and late 90s, but their work has not been revisited since the recent explosion of work in this area. The multi-camera virtualized reality work of Kanande and the image-based rendering techniques employed by Debevec have brought image-based modeling into the research mainstream. Image-based rendering has been used to create convincing 3D models and feature-film visual effects, but despite tremendous recent progress, scene capture for these techniques is far from automatic.

Central to gathering scene data for model generation or image-based rendering is the planning of camera positions to ensure adequate coverage. Since much of the processing of these image sets is done offline, it is usually impossible for a camera operator to know if he or she has gathered enough image data from a scene to generate a model of a desired fidelity. If any part of a scene is out of frame or occluded from view in all captured frames, that portion of the scene will not be available for any subsequently synthesized views. Likewise, if a surface is imaged only from a grazing angle, quality of gathered texture and geometry will suffer. For the simple convex objects often digitized in demonstrations of 3D capture systems, reasonable camera placement usually produces good models. Unfortunately, for complex scenes with many occluding objects, it is quite difficult to be sure when one has gathered adequate coverage. The camera movement planning system I describe here enables automated capture and refinement of static scenes allowing for unattended modeling of indoor spaces using a group of mobile robots. This work also demonstrates an application of scene capture for producers of holographic images. Holographic video generation requires the capture of either many adjacent views of a scene, or computation of display information from 3D models. The planning system described here allows for the automatic capture of data for holographic display.

1.2 contributions

This thesis describes software tools and algorithms that were designed by the author to allow for automated capture of 3D scenes. Specific contributions include:

An algorithm for planning robotic camera movements to iteratively improve 3D models of a visual scene.

Specific optimizations to allow the camera planning algorithm take advantage of programmable graphics hardware.

Software infrastructure for communicating between image-capturing robots and image processing computers.

Software viewing progress of scene capture, resulting 3D models, and for simulating the capture task entirely in software.

1.3 Overview

Calibration

Before experiments begin, the camera attached to each robot is calibrated to determine lens focal length and radial distortion parameters. Although techniques have been documented for automatic calibration of moving cameras, for simplicity, I calibrate using static calibration targets. The derived parameters are used to rectify images gathered by each camera. Angular calibration of the servo pan-tilt system has was performed, while the linear drive system only crudely calibrated.

Scene Capture

Capture is triggered by a request sent via the application running on the laptop computer. The request specifies the camera parameters for the desired view or views of the room to be gathered, a rough scene center location, and a rough scene radius.

Depth Image Capture

Cameras orient themselves towards the view volume of the desired view, capture one image, move linearly to a nearby camera position and capture a second image. Each image is archived and stamped with a unique identifier. A depth image is then computed from the captured stereo pair. Depth images are be computed using the simplifying Lambertian assumption: a given point on the surface of an object will look the same from any direction.

A confidence score is stored for each depth sample, based on the difference between the best-matching luminance patches in the stereo pair.

View Generation

Upon completion of the depth recovery computation, the system, using the GPU on an attached PC, generate the desired view(s) from all recorded depth images. This is done by converting each depth map into a 3D mesh, transforming each mesh from the coordinate system in which it was gathered into the world coordinate system, and then rendering all of the stored meshes in a single view. Confidence scores are likewise coalesced into a 2D confidence map for each of the desired view(s).

Planning

Robotic camera nodes then prepare to move to a new location for further image capture, repeating the above steps until the requested quality criteria are met. To do this, each node combines all depth and estimates computed so far, and locates a region of the camera's linear track that has visual access to depth discontinuities in the depth maps. A camera move towards the area will (except in the degenerate case of a grazing view of a surface) yield a view of scene information that was previously occluded. New pan-tilt direction direction is chosen to center the camera on the depth discontinuity. A map is kept of regions that have been the focus of previous attempts at uncovering scene information. To limit search of scene patches that cannot be be imaged from reachable positions, explored regions that yielded no new information will not be revisited until all other candidate regions have been explored.

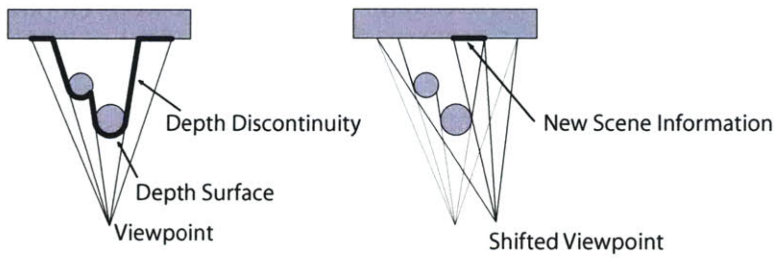


Figure 1-1: Previously missing scene information is revealed when viewpoint is moved in the direction normal to a depth discontinuity.

Completed View Generation

When a node has a set of confidence maps that together meet the requested confidence requirements, the node assembles a full model for generating the scene. This node traverses the depth and confidence maps and finds which texture data will be needed to generate the desired view(s). These texture data are then requested from the appropriate nodes, and a complete model is assembled from accumulated texture and geometry data. This completed model is then sent to the laptop computer. A request to cease further data collection is then sent to the remaining nodes.

1.4 Related Work

The method described here for planning iterative scene capture draws on work in the areas of sensor calibration, correspondence-based depth recovery, image-based rendering, and sensor planning. Much work has been documented in these fields, and this thesis draws extensively on existing techniques. My contribution consists largely of the integration of several of these methods, with contributions in fast GPU-based image processing techniques, and in simple heuristic rules to guide image-driven planning. Here, I will outline research that serves as the foundation for this thesis, as well as other related work.

Mallett and Bove [2003] developed a network of robotic cameras with two degrees of pan-tilt and one degree of translational freedom. These robots were employed for much of the work described in this thesis. Bove and Seongju Chang also developed a stationary system of networked tiles with the same camera and microprocessor hardware as used in the robotic cameras. Algorithms for comparing the images captured by these tiles were explored by Pilpre [2005], who built a software tool to find the 2D offset between images gathered by pairs of tiles. Pilpre's approach employed KLT features [Shi and Tomasi 1994]. Mallett [2005] continued this work by using more robust SIFT features [Lowe 2004] to establish 2D adjacency relationships within groups of these tiles.

Calibration allows a researcher to learn the geometric parameters of an individual camera as well as parameters relating systems of cameras. Calibration of a camera network may simply seek to discover which cameras have overlapping views [Pilpre 2005] [Mallett 2005] [Wren and Rao 2003]. Such calibration may also seek additional parameters including lens parameters [Willson 1994] [Becker and Bove 1995], rotation axes, and relative camera locations [Sinha and Pollefeys 2004] [Davis and Chen 2003].

Once camera geometry is known (or estimated), collections of multiple images, and/or images taken of objects under constrained lighting, have been used to extract the 3D shape and texture of objects.

At the Media Lab, techniques developed for extracting geometry include the work of Bove [1989], where depth of images was recovered from structured lighting, focus, and from camera motion. Successful systems for recovering moving object models by arrays of fixed cameras have been demonstrated by Kanande [1997]. Image-based rendering of such scenes has been explored in detail by Debevec [1998]. The work done for this thesis will use similar techniques for generation of images from synthetic views.

Complete systems for capturing lightfield [Adelson and Wang 1992] models of stationary objects have been also developed [Unger et al. 2003]. Other research at MIT has explored the digitizing of cities using a mobile pan-tilt imaging rig [Teller et al. 2005]. Research at

UCSD [Trivedi et al. 2005] has explored the tracking of people in a room using networks of omnidirectional and pan-tilt cameras.

In addition, work has been done to recover 3D models from images from uncalibrated image sequences. In uncalibrated situations, camera parameters are derived only from relationships between gathered images. Work in the Perceptual Computing Group at the Media Lab [Azarbayejani et al. 1994] demonstrated assisted model recovery and camera tracking from uncalibrated video sequences. Becker [1995] proposed further methods for recovering dynamic camera parameters including focal length and radial distortion by finding vanishing points of rectilinear geometry. More recently, compelling image-based Work done at UNC [Pollefeys et al. 2004] has demonstrated the unassisted registration of frames from an unknown video sequence to recover detailed geometry of scenes. Future automated scene capture tools could use iterative refinement techniques to avoid the explicit calibration I perform for this thesis.

Beyond work to generate models and images from still frames, the work described in this thesis employs online planning of camera movements. Motion planning was the topic of relatively early work in the robotics community [Brooks 1982]. Similar approaches were used for placing an image sensor to observe an object without occlusion. Much of this early sensor planning work assumed prior knowledge of scene geometry to compute object visibility [Abrams et al. 1993]. For capture of 2D video, work in the Vision Modeling group [Pinhanez and Bobick 1995] at the Media Lab used approximate 3D templates of real scenes to perform camera motion planning. Like the planning system proposed here, the system described by Lehel [1999] uses images directly (without prior scene knowledge) to adjust the spacing of a trinocular camera array.

Chapter 2

Robots for Scene Capture

This section outlines the equipment, methodologies and algorithms used in the view planning system. Briefly, the system generates and improves novel scene views by capturing stereo pairs of views, processing the pairs to extract depth and texture information, and making available these locally computed depth and texture estimates. The planning system directs individual robotic camera nodes towards new unseen parts of the scene to survey. The robotic cameras then adjust their position, gather new stereo pairs and repeat the process.

2.1 Apparatus

The physical equipment used to perform the scene capture and view planning described in this thesis consists of four "Eye-Society" robotic cameras, several network-attached desktop PCs for performing computation, a web and file-server for archiving images captured by the system, and a laptop computer for controlling these resources.

2.2 Eye Society Robots

The network of three "Eye-Society" robots developed by the Object Based Media group [Mallett 2005], are used for image capture and camera movement. These cameras are mounted on the ceiling of room 368 along rails forming a roughly 12 foot by 12 foot square. Each robot sits on an independent linear rail, allowing for about 12 feet of linear motion. The camera network was designed for experiments with distributed scene capture, object tracking, and human activity monitoring.

2.2.1 Mechanical Description

To enable future large-scale camera deployments, The Eye Society robots were engineered to be low-cost and easy to produce in quantity. We chose to employ inexpensive Charge-Couple Device (CCD) image sensors used in off-the shelf cameras sold for use with personal computers. In particular, we chose digital cameras that interface via the Universal Serial Bus (USB; www.usb.org) despite availability of more inexpensive analog cameras because digital cameras do not require additional video digitizing hardware for capturing images into computer memory.

The USB cameras selected for the Eye Society robots were the Connectix Quickcam 3000-series "webcams" (www.connectix.com). These cameras have CCD sensors with a resolution of 640x480 pixels. By employing on-chip image compression, they are capable of transferring images over their full-speed USB interface at near-television rate. These cameras also come with reasonably low-distortion fast optics which enables the capture of comparatively sharp, low-noise images under indoor illumination.

The USB cameras used in the robots were stripped of extraneous plastic packaging and mounted in a laser-cut acrylic two-axis (pan-tilt) gimbal. Two small off-the-shelf servos serve as the pivots of this gimbal, and actuate the pan and tilt angle of the camera relative to the body of the robot. Each servo is capable of moving though about 160 degrees of rotation, allowing cameras to be pointed in nearly any downward-pointing direction, covering nearly a hemisphere of visual space.

On each robot, the camera gimbal is mounted to the small horizontal acrylic platform from which the single-board computer is also supported. This platform is in turn suspended from a system of four cylindrical nylon rollers mounted between more laser-cut acrylic pieces. The rollers support the weight of the robot and maintain its alignment to the axis of its extruded aluminum rail.

Each robot is equipped with a small, high-torque drill motor for actuating position along the support rail. The motor assembly includes an inline gear-train and a rubber wheel that rests on the upper face of the aluminum rail.

2.2.2 Electrical Description

The robotic cameras are each built around a “BitsyXb” single-board computer manufactured by Applied Data Systems (www.applieddata.net). These computers were chosen for their small size and low power requirements. As the Eye Society camera platform was designed to be capable of optionally running untethered on battery power, lightweight (requiring less energy to effect camera movement) and low-power components were selected where possible. For our experiments, we wired the robots to 12V ”wall wart” transformers so many datasets could be captured sequentially without needing to recharge batteries. The Bitsy single-board computers have power circuitry capable of running from unregulated DC Voltages from 9 to 18V.

The Bitsy single board computers are built on the Intel X-Scale architecture, with allows them to be capable of relatively high computational performance for their power consumption. Initial robot prototypes were based on the ADS BitsyX board, which contained a

400 Megahertz (MHz) X-Scale processor running on a 100 MHz system bus. The Bitsy Xb boards run at 450 MHz, but are capable of processing images much more quickly due to their 200 MHz system bus. The Bitsy Xb boards contain 64 Megabytes (Mby) of Synchronous Dynamic Random Access Memory (SDRAM) for storing and processing images, as well as 64 Mby of non-volatile flash memory for storing the operating system and other software.

In addition to the Bitsy Xb single-board computer, each robot contains two additional circuit boards: a power distribution board and a servo controller. The power distribution board is a hand-soldered prototype board containing an H-bridge chip with supporting components for driving the track motor, and a voltage regulator for the low-power circuitry on the servo controller.

The H-bridge chip runs directly from the 12v robot supply, and allows the current-limited 3.3 Volt digital outputs of the Bitsy Xb board to be used to control the speed and direction of the track motor. Three such digital outputs are used to send "direction," "brake" and "PWM" signals to the H-bridge chip. The first two signals configure the output transistors of the H-bridge, selecting a direction of rotation for the motor (although not currently used, the brake control line instructs the h-bridge to short the terminals of the motor together to apply braking force to a moving motor). The Pulse Width Modulated (PWM) drive signal allows specification of the motor's drive power. The PWM line is driven with an asymmetric square waveform, which pulses power to the motor whenever this control line is high.

The servo controller board is an off-the-shelf eight-port RS-232 serial-controlled servo driver. It accepts commands specifying servo position and supplies power and generates the appropriate pulse-trains for standard radio-control type servomotors. This circuit board has two power inputs, one 5V input used by the onboard microcontroller, as well as a 12V input which supplies power to the attached servomotors.

Interfacing between the Bitsy Xb board and the motor controller board is handled by the

"ADSmartIO" controller. ADSmartIO is an Atmel 8538 microcontroller (www.atmel.com). The robots use ADSmartIO output pins PC0, PC1, PC2 and PC3 for the "direction" "brake" "PWM" and "laser" signals respectively. These signals are available on the external header J10 on the Bitsy board, at pins 15, 13, 11, and 9. The use of the ADSmartIO controller to drive the H-bridge chip is problematic due to apparent inconsistent reset behavior. Occasionally, when the Bitsy board reboots, the output lines of the ADSmartIO controller command the H-bridge chip to drive full-on throughout the boot sequence, eventually setting these outputs to float near the end of the boot. This spurious drive command causes the robot to race down the rail, pegging itself at the end and continuing to spinning its drive motor in place. Occasionally this causes damage to the robots, so care should be taken not to reboot them unattended.

Communication among the Eye Society Robots, and with other image processing and storage resources is performed using 802.11b wireless ethernet (WiFi). Each robot has an on-board WiFi card, attached to the PCMCIA interface of the single-board computer. Although these cards are theoretically capable of data transfer of 11 Megabits per second (Mbps), the bus bandwidth of the Bitsy single-board computers limits transfer rates to about 1 Mbps.

Along with standard hardware, one of the Eye Society Robots is additionally fitted with a small laser diode module, mounted with the CCD camera on the pan-tilt gimbal. This laser module was mounted to allow for future experiments with projecting feducial marks into the scene to be captured, as well as to allow for measurement and characterization of the dynamics of robot movement.

2.2.3 Software Infrastructure

The ADS Bitsy Xb single-board computers inside the Eye Society Robots run a version of the Gnu-Linux operating system. Linux was chosen for its modifiability and for its compactness.

Much of the software infrastructure for the Eye Society Robots is shared with the "Smart Architectural Surfaces" project in the Media Lab's Object Based Media Group. Both projects employ variants of the Bitsy single-board computers from ADS, and both use the same camera hardware. Common software was developed with the intent of enabling future collaboration between nodes from both projects.

Communication between robots is mediated by the Remote Python Call (PyRPC) software library (rpyc.sourceforge.net), which allows software objects located across a network to appear to be co-located on a single computer.

Basic robot functions, including movement and image capture are managed by the `RobotCore` software object. This object was written in C++ and contains wrappers to make it accessible from the Python environment (and in turn remotely via PyRPC).

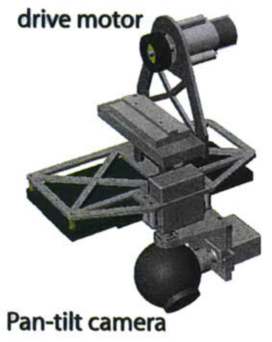


Figure 2-1: Illustration of an "Eye-Society" robot. Drive motor actuates robot along a 12 foot rail.

Chapter 3

Calibration

The cameras used for this project produce two-dimensional images as output. When viewed on a computer screen, these images approximate the appearance of the real objects at which the camera was pointed. Beyond subjective appearance, if we want to draw mathematical conclusions about scene geometry from these images, we must characterize the geometry of the cameras.

This chapter describes the relationship between the light exiting a real scene, and the images produced by the camera. First, the simplified model of camera geometry used throughout this work is presented, and assumptions inherent to this model are described. Next, descriptions are given of the methods for determining the parameters for this model. These parameters seek to describe the spatial distortion in images of objects caused by the true shape of the camera's lens and image sensor. Further, descriptions are given of the calibration methods used for the robotic platforms used to move cameras in experiments conducted for this thesis.

3.1 Camera Model

The cameras used for this work consist of a set of a set of small glass lenses that refract incoming light to form an image on a CCD image sensor. The term "real scene" is used to describe the physical objects that reflect or emit light that is recorded by the camera. This term is used in contrast to "scene model" which describes a 3-dimensional representation of the geometry and appearance of the real scene.

To approximate the properties of the physical cameras, we use a pinhole projection model to describe how points in the real scene map to points on the image sensor. In the pinhole model, only light rays that pass through a single point in space, known as the center of projection, can reach the plane of the image sensor, and only rays within the bounds of the image sensor reaching this "image plane" are recorded by the camera. In this model, the line perpendicular to the image plane that passes through the center of projection is referred to as the "axis" of the camera. Because light rays travel in a straight line through the center of projection, the angle between points in the real scene and the camera's axis always equals the angle between the camera's axis and the image-plane projection of that scene point.

In this thesis, pinhole projection is computed using a mathematical description that is used throughout computer graphics and machine vision research: the homogeneous transformation matrix. Modern computer graphics hardware is highly optimized for handling this representation, as projecting 3D points onto an image plane sits at the core of computer graphics rendering.

Transformation of a point p_o in three-dimensional space to its corresponding point p_i on the camera's image plane can be represented as multiplication by a projective transformation matrix T_{io} as:

$$p_o = T_{io} p_i. \tag{3.1}$$

In this representation, points are represented as 4-vectors, containing three location coordinates and a scale. These vectors are in most cases maintained in canonical form, normalizing all elements by the scale:

$$p_o = [x_o, y_o, z_o, 1]'. \tag{3.2}$$

Projection by a pinhole camera with center of projection at the origin, facing in the positive z direction, can be represented as a multiplication of a location 4-vector by the matrix

$$\begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

where f is the distance from the center of projection to the image plane. This maps scene coordinates to three-dimensional "view" coordinates, of which the first two describe a two-dimensional image plane location.

This projection representation is used for rendering 3D models in computer graphics usually define Two planes, both parallel to the image plane, that bound the volume of space in front of the camera to be considered. These planes are referred to as the "near plane" and the "far plane"(or near and far clipping planes). In this form, the projection matrix is parameterized by Z_n, the distance from the center of projection to the near clipping plane, Z_f, the distance from the center of projection to the far clipping plane, and t b l and r, describing the distance from the camera's axis to the top, bottom, left, and right boundaries of the near plane. When scene point locations are multiplied by this projection matrix, the result is a 2-D view coordinate location, representing position on the image plane, as well as a depth coordinate in the range of 0 to 1, representing relative location of the scene point between the near and far planes.

$$T_{io} = \begin{bmatrix} \frac{2Z_n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2Z_n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{Z_f+Z_n}{Z_f-Z_n} & \frac{-2Z_fZ_n}{Z_f-Z_n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.4}$$

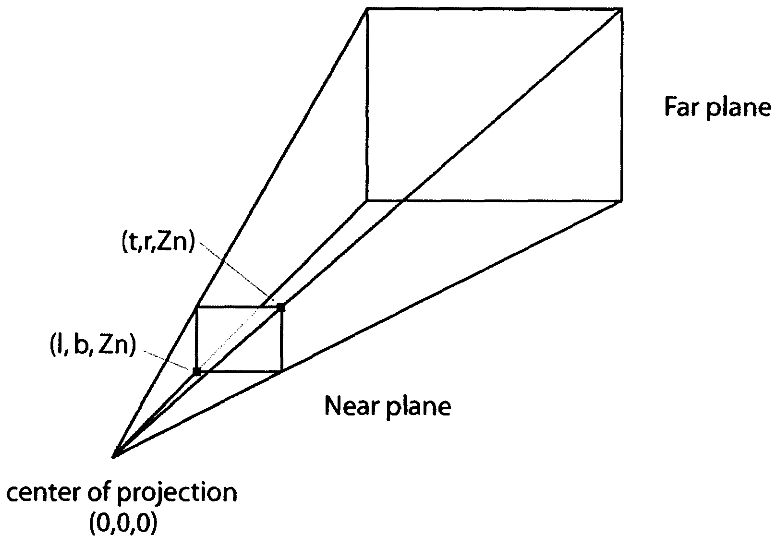


Figure 3-1: View volume in rendering model. Space bounded by near and far clipping planes falls in the imaging volume.

In addition to projection, transformation matrices are also used in this work represent relationships between points after they undergo rotation or translation. Spatial relationships that consist of combinations of translation, rotation, and projection can be represented by combining, via matrix multiplication, the transformation matrices representing these individual relationships.

These matrices are used to model relationships between locations in the scene and locations on the image plane that arise over movements of robotic cameras. Such situations are represented here by computing a separate transformation matrix to represent joints attached to the camera, and matrices to represent the spatial relationships between these joints.

3.2 Camera Calibration

The real lenses used for this work use several glass lenses instead of a simple pinhole to project light onto the image sensor. Lenses, like pinholes allow light to be projected, but have the advantage of allowing much more light to reach the image sensor. Tradeoffs in the design of lens systems however, generally produce optics that distort images of scenes. The standard approach in the machine vision community for accounting for this distortion is to capture images of a known scene, and then measure how the resulting image differs from how it would appear under the pinhole assumption. Following this, a function is constructed that transforms expected locations to their actual measured locations. Once found, the inverse of this function can be approximated to rectify the distorted images. Functions used to model this distortion usually include parameters to account for offset between the center of the image sensor and the camera's axis, shrinking of the image that varies with distance from image center (a third-degree polynomial is used), and sometimes skew and sheer of the image sensor pixels.

Calibration for the Connectix Quickcams used for camera planning experiments were calibrated using the Matlab Camera Calibration Toolbox [Bouguet 2000]. This utility allows for semi-automated computation of camera intrinsic parameters. The toolbox solves for parameters from a set of images that contain a planar checkerboard calibration target. Corners of the checkerboard squares are found in a semi-automatic fashion, requiring the user to click at the corners of the checkerboard in each image, after which the tool solves for sub-pixel estimates of checkerboard square locations.

Camera calibration was performed using a set of images of a checkerboard grid containing 10x10 squares, with each square measuring 25.4 mm (1 in) on a side. The checkerboard pattern was balanced on a table in the middle of a room and cameras were moved interactively to acquire 18 images confining the checkerboard. Effort was made to acquire checkerboard images taken from a variety of locations. The images used were captured at a resolution of 320x240 pixels, at a distance of about 3 meters from the checkerboard.

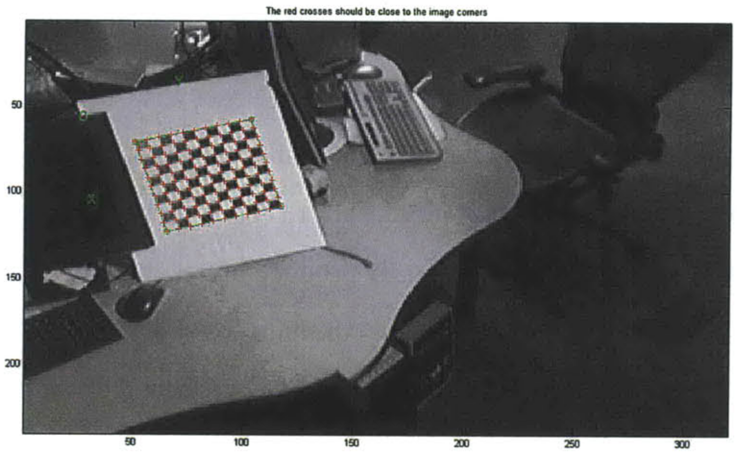


Figure 3-2: Matlab-based interactive camera calibration [Bouguet 2000] used to compute camera geometry from images of checkerboard.

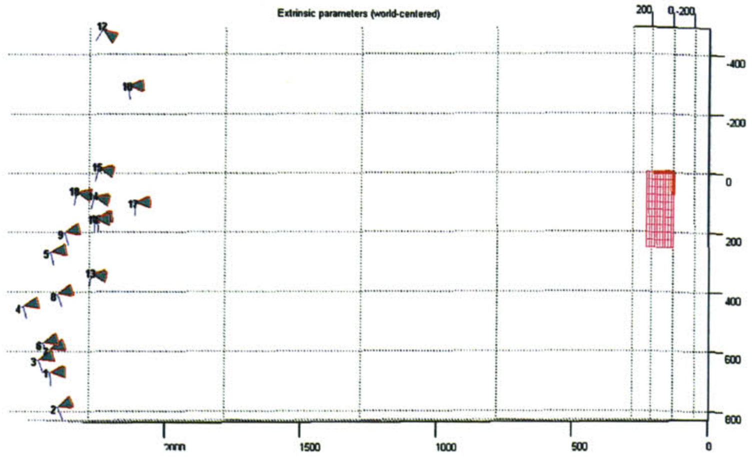


Figure 3-3: Visualization of camera locations derived from lens distortion computation.

3.3 Robot Actuator Calibration

The Eye Society robots used for this project can direct their cameras through three degrees of freedom. The cameras can be rotated, using servomotors, along two (nearly intersecting) axes, and can be translated along a linear track. This section details calibration procedures used to find the mapping between commands sent to the actuators, and the resulting camera movements.

3.3.1 Servomotor Calibration

The servomotors used for camera positioning employ an internal feedback mechanism to rotate an output shaft in response to an input voltage waveform. The relationship between this input signal and the resulting camera waveform is nearly linear at the middle of the servo's range of motion but the servos tend to undershoot as the are commanded towards the ends of their range. This seems to occur in part due to the design of the servo gear system, and in part due to additional forces on the servos as the camera cables become tighter at the edges of the movement envelope.

To calibrate the response of the servomotors, the servos were commanded to move the camera to positions throughout the envelope of movement, and images of a checkerboard target were taken at each position. View planning experiments use a linear approximation found from the servo positions within the center two-thirds of the servos' motion range. Future work could use a lookup table, or polynomial approximation of this function to allow for more accurate positioning at the ends of the servos range.

3.3.2 Track Drive Motor Calibration

Unlike the pan-tilt actuators, the motors that actuates the Eye Society Robots along their guide rails have no feedback control loop. Additionally, slight differences in construction between the individual robots result in a wide range of mappings between commanded and actual movement. Motion along this axis is specified by the view planning system as a movement duration. To approximate the relationship between movement duration and travel distance, robots were directed to make a series of small short-distance fixed-time

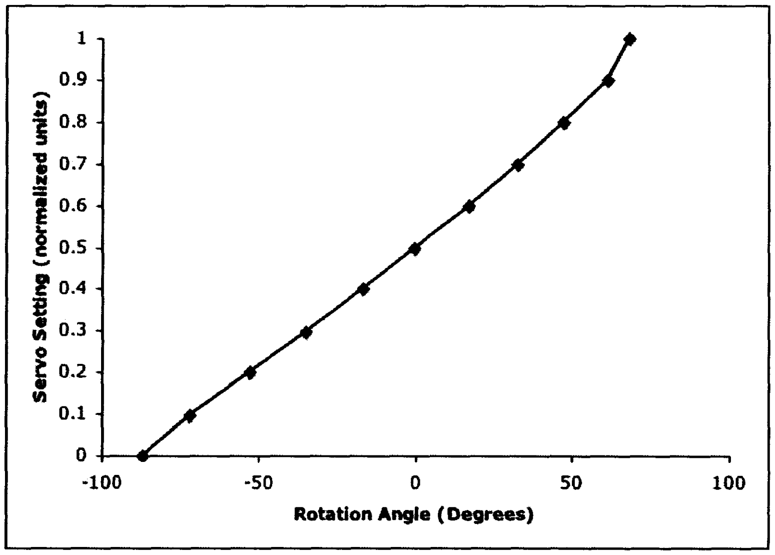


Figure 3-4: Model edge falls at left edge of a view. When this edge falls inside the synthetic view volume, pivoting camera position left will reveal more of the scene.

movements along their rails, and the number of movements required to reach the end of the rail was recorded.

Chapter 4

Models from Scenes

The overall goal of the work presented here is to generate 3D models of scenes for synthesizing new views. This section describes methods for generating these models from sets of images. Discussion of methods for choosing the locations from which these images should be captured are described in the chapter that follows. Models are generated by first computing depth maps from pairs of images. Those depth maps are then combined with scene appearance information to generate sets of 3D patches that model the scene geometry. Sets of patches from many view pairs are combined to form complete models.

4.1 Depth from Stereo Images

Many technologies exist for extracting 3D information from object or scenes. These technologies include acoustic methods, where pressure (in air or water) waves are measured as they reflect of a scene. Other methods project structured light onto a scene (either laser, or other projected light) and triangulate scene geometry based on the locations from which light reflects back to an optical sensor. Additionally, depth information can be gathered by measuring image properties as they change with lens focus. The method for depth recovery chosen for this work is stereo matching: pairs of images taken from different camera locations are compared, and the spatial displacement of matching regions is used to triangulate distance to that scene region.

This method was chosen for the simplicity of the equipment it requires. Since the images to be generated will also need surface appearance information, cameras can be employed as a single sensor for gathering both kinds of information. The cameras used are not capable of actuated focus adjustment, so depth-from-focus methods were not considered. One shortcoming of the stereo matching approach is that extraction of depth information relies on variation in surface radiance information (appearance). The performance of stereo matching diminishes for objects that appear more self-similar. For example, two images of the surface of an empty white desk, taken from a few inches apart, appear nearly identical. Without being able to see variations on a surface, we determine the shift of regions in one image relative to the other; we can not determine its distance from the pair of camera locations. Beyond this limitation, when surfaces have radiance that changes as we view it from different directions (such as shiny or finely textured surfaces), appearance of that surface may not match itself between camera views. For these kind of surfaces (referred to as non-Lambertian), stereo matching is prone to error.

4.1.1 Graphics Processing Units for Stereo Matching

Many computational methods have been described for depth recovery by matching patches in images. For this thesis, the author explored methods that employ graphics processing units (GPUs) present in most modern computers. Recent graphics processing units contain many parallel execution units that are ideal for computational problems that divide easily into independent subproblems. While support for writing programs for these processors is not as mature as for conventional central processors, tremendous performance gains can be had for algorithms that can be implemented in the graphics processing pipeline. Several programming languages for describing programs that run on graphics processing units were evaluated. These included assembly language, Nvidia's Cg language (Nvidia Inc. www.nvidia.com), and the OpenGL Shader Language (GLSL). The OpenGL Shader language was chosen for this work because it seems to be the most likely to be supported in the long term. Specifically, Fragment Shaders, which execute a short program on each pixel location of rendered geometry, were used for computation.

Hardware

For this thesis, experiments were performed primarily on an ATI Mobility Radeon X1600 graphics processor (ATI Technologies Inc. www.ati.com). This processing unit contains twelve individual "pixel shader" processors, designed for performing mathematical operations on each pixel of a computer-graphics image. The processor used operates at 310 million clock cycles per second (MHz) and has access to 256 million bytes (MB) of dedicated memory running at 278MHz.

4.1.2 Stereo Matching Algorithm

Capture of 3D scene models consists of a series of applications of a stereo matching algorithm. Each application of the algorithm uses a pair of strategically captured images to compute a single depth map. The terms "reference camera" and "comparison camera" will be used to describe the two cameras used for matching. Depth maps are computed to describe distance between the reference camera and objects that appear in the image taken by that camera. To find this map, The comparison camera images the same scene from a nearby location, chosen to provide parallax information while still viewing most of the same scene objects as the reference camera. Values in the depth map are populated by testing the similarity of regions (here similarity is defined as the absolute difference between corresponding pixel values, averaged over the region) in the reference and comparison images.

Before actual matching is be performed, the stereo matching system estimates the location and orientation of the camera corresponding to each image. This spatial information allows the system to recover the actual physical location from relative object locations in the two images. When the Eye Society Robots are used for gathering images, the robots record with each captured image, the pan-tilt servo settings, and the approximate track position. These pieces of information, along with a calibrated model of the spatial relationships between the robot actuators, allows for computation of the two camera locations in a common room coordinate system. This in turn allows the depth map generated by region matching to be described in this room coordinate system.

To compute depth maps, this project uses several applications of the standard graphics pipeline (running on the graphics processor, accessed through the Open GL graphics library) to match regions in pairs of images. To perform matching, patches of pixel values in the reference image are compared to pixel values over a set of locations in the comparison image. If the relative placement of the two camera positions is known, Matches for regions at one pixel need only be sought along a single line in the second image. This line of locations corresponds to the set of places in the comparison image to which a scene region can possibly project in the comparison image, when location in the reference image is known. Locations along this line correspond to the scene locations that all appear at one single image location in the reference camera (along a ray through that camera's center of projection). Limiting search to this single line provides dramatic savings over searching the entire image for matches, but uncalibrated lens distortion or incorrectly estimated camera pose can lead cause the wrong line to be searched, producing incorrect matches. Many traditional stereo systems simplify computation by using camera locations that exhibit symmetry (either with co-planar image sensors, or at least keeping corresponding rows of sensor pixels each in a common plane, allowing for converging view directions). This is done so that matches for a given pixel location can only appear along the same horizontal image line in the corresponding image.

Reprojection

The implementation used in this work uses properties of pinhole projection to choose image patches to compare that could have feasibly come from the same scene location. Instead of explicitly describing the candidate line of possible comparison regions for a given reference region, OpenGL is used to render a set of reprojected reference images to match against the comparison image. Each reprojected reference image is generated to simulate the appearance of a planar scene, sitting parallel to the reference camera's image plane, imaged from the viewpoint of the comparison camera. The planar scene used for generating these reprojected image is simply a rectangle containing the image captured by the reference camera projected out into the scene using a pinhole camera model. In this way, each of the reprojected scenes as viewed from the reference camera look identical, while from any other viewpoint, the reprojected planar scenes appear to have different sizes and locations. This method can be thought of as creating a hypothesized scenes, scenes that could feasibly have generated the reference image, and then testing these hypotheses by examining whether when viewed by the comparison camera, the hypothesized scene is consistent with the real photograph captured by that camera. Clearly, we don't expect any of these hypothesized planar scenes to be entirely correct, we can pick some parts of several hypotheses to construct our model of the scene. Since the hypothesis scenes are planar, no parts of the scene block other parts from view. With no interaction between reference pixel locations in the hypothesis scene, we are free to evaluate different hypotheses on a pixel by pixel basis, deciding to accept some regions of the hypothesis scene, but not others. This method also allows for verification of completed depth maps (assembled from feasible regions across many hypothesis scenes). A depth map can be tested by using the map itself to create another kind of non-planar hypothesis scene. Verification can be performed by back-projecting patches of the reference image to depths indicated by the depth map, and then imaging them from the comparison camera location. Discrepancies between the reprojected image and the real comparison image indicate errors in the depth map (or invalid assumptions in the camera or scene models).

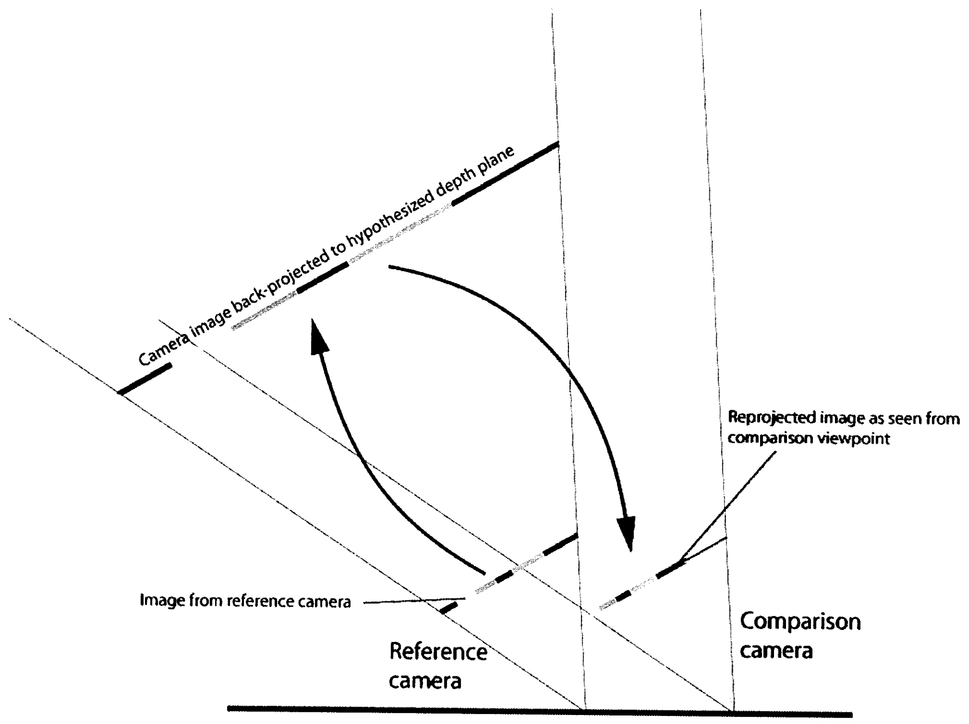

Camera image back-projected to hypothesized depth plane
Reprojected image as seen from comparison viewpoint
Image from reference camera
Reference camera
Comparison camera

Figure 4-1: Reference image is re-rendered as if it was on a plane in the scene parallel to the reference image plane.

Computational cost of Reprojection

Performing comparisons in this way incurs little or no additional computational cost, and simplifies implementation for unconstrained cameras. Since image comparisons in the OpenGL environment require the rendering the two images as if they were a 3D scene (applying a differencing operator as the second image is rendered), rotating and projecting the scene as above can be performed as part of this rendering pass. Since rendering requires the specification of virtual camera parameters and scene orientation, performing comparisons on reprojected images takes about as long as comparing flat, unprojected images.

Shader Implementation of Region Similarity

Once a set of reprojected hypothesis images has been generated, an OpenGL pixel shader is used to decide, for each reference image pixel, which hypothesis image is likely to be a projection of the scene point seen in the comparison image.

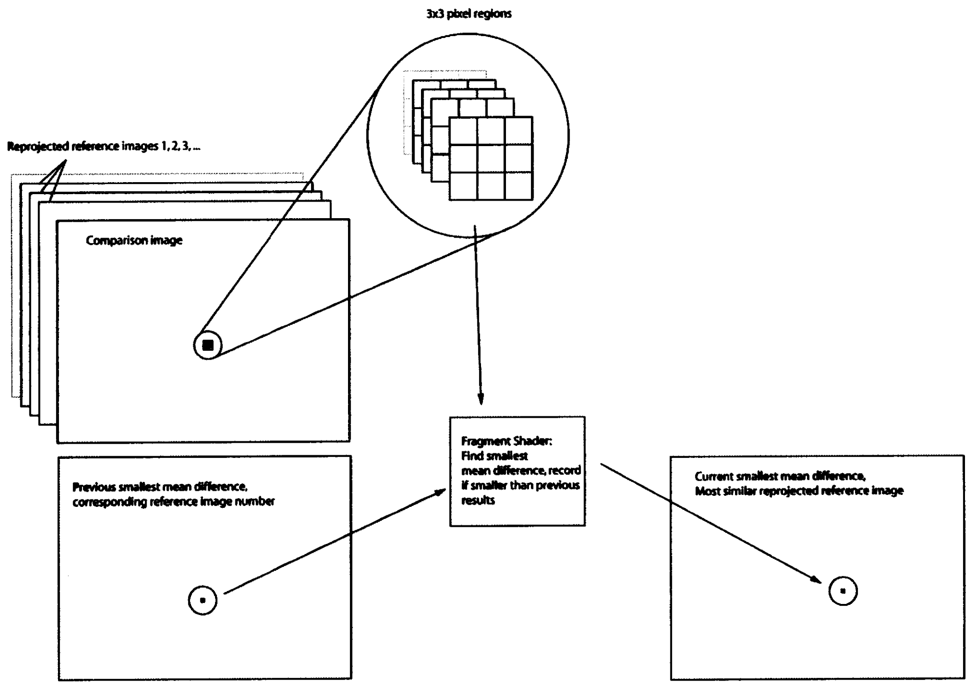

3x3 pixel regions
Reprojected reference images 1, 2, 3, ...
Comparison image
Fragment Shader:
Find smallest
mean difference, record
if smaller than previous
results
Previous smallest mean difference,
corresponding reference image number
Current smallest mean difference,
Most similar reprojected reference image

Figure 4-2: Overview of OpenGL fragment shader used for stereo computation.

This comparison is performed by first rendering each reprojected image into a rectangular OpenGL texture. Each such texture is applied to a rectangular polygon, along with a texture containing the reference image. (OpenGL allows geometry to be mapped with several textures simultaneously, and then use a fragment shader to compute how those textures are combined to produce the final appearance.) The fragment shader used to perform comparison (executed at each pixel location on the supplied polygon) subtracts pixel intensities at corresponding locations in the two textures, computes the absolute value of this difference, and averages these absolute differences over a 3x3 window surrounding the current pixel location. Since fragment shaders can (depending on graphics hardware used) be given access to several textures at once, the final version of the stereo matching algorithm used here simultaneously performs matching between the comparison image and four candidate reference images. The resulting average absolute difference value for each candidate is output to a different channel (normally used for red, green, blue, and transparency values) of the output image. Several rendering passes are used to evaluate all hypothesis images, performing evaluations in batches of four per pass. At the end of K passes, similarity values for all hypotheses are available in K/4 4-channel images.

4.1.3 Shader Implementation of Best Match

The goal of the stereo matching algorithm is to choose the depth estimate at each reference image pixel that is most likely to be correct. This estimate is made by finding the smallest absolute difference score at each pixel location, taking into consideration similarity scores from each of the reprojected hypothesis images. This simple minimum operation can be performed in the same fragment shader program as the similarity computation, or can be performed in the CPU using the stack of "similarity images" described in the previous section. When performed in the fragment shader, the output of the shader is changed from recording the similarity for each of the input hypotheses to recording the best similarity score seen so far, and the image number at which it was seen. Only a single output image is produced by this process, describing the best match, and its similarity. This technique is very fast, but is not compatible with more advanced stereo matching techniques[Brown et al. 2003] that use the similarity scores for all pixels and all hypotheses simultaneously to gain resilience to errors from noisy or untextured regions.

3D Mesh Generation

The map of chosen depth values for each pixel in the reference image is then transformed into a 3D mesh for re-rendering to generate novel scene views. Meshes consist of a lattice of 3D vertex locations, and triangular facets that connect triplets of these vertices. Simple meshes can be constructed by using the same back-projection technique used to generate the hypothesis images above, but back-projecting the corners of each pixel individually to an appropriate depth instead of back-projecting the entire image as a single flat plane. Vertices are back-projected to a camera distance corresponding to that of the the best match for that image location found above. For simple, continuous depth maps, the shape of the scene can be well represented by tessellating the resulting back-projected vertices, producing a 3D polygon connecting the vertices bordering each image pixel. To assign color to this mesh, OpenGL texture mapping is used. A texture map, usually a 2D array of color values, can be used to color a 3D polygon by describing a relationship between surface locations on the polygon and locations in the texture map. OpenGL allows this relationship to be described by specifying the location in the texture map that should be used to color each of the corners of the polygon. The locations in the texture map, specified for each vertex, are called "texture coordinates." Color values for the intermediate locations between vertices on the polygon are computed by sampling the texture map at corresponding intermediate locations between the texture coordinates. For constructing 3D scene meshes, texture coordinates are assigned to the mesh to describe the mapping between locations in the reference image and locations on the mesh. The mesh created describes one rectangular polygon (represented internally as a pair of triangles) corresponding to each pixel of the reference image, so texture coordinates for each polygon are simply the corners of the corresponding pixel in the reference image. Regardless of resulting mesh shape, the texture coordinates fall on a regular grid.

4.1.4 Mesh Discontinuities

The above method works well for smooth, highly textured scenes where no surfaces are blocked from view by either camera. In real scenes however, objects usually do occlude parts of themselves, or other objects from view. When occlusion occurs, depth maps, and corresponding depth meshes, will have discontinuities at locations where the edge of a near object falls in front of a far object. As we scan through sequential values in such a depth map, values describing distance to the near object change in value abruptly at that object's edge to begin describing distance to the far object. If continuous 3D mesh was used to describe such a scene, a surface would appear connecting the boundary of the near object to the representation of the far object. Such a model viewed from the reference camera location would look fine, but viewed from nearby locations, this surface would appear noticeably at odds with a corresponding view of the real scene. This situation is worse that simply having information missing from a generated view since this incorrect surface connecting near and far objects would be capable of obscuring other correctly modeled geometry. Spurious surfaces connecting disjoint objects are an extreme case of another problem that appears in meshes generated through reprojection. Similar problems occur at any locations in a depth map that have depth values dissimilar to those at neighboring locations. Such situations appear at occlusions, but also at locations where the surface of objects is steeply inclined as viewed from the reference camera. Such large depth changes describe polygons that have steep surfaces relative to the reference camera's image plane. Such polygons are textured with information drawn from only a single pixel in the reference image, but have much larger area than polygons that are parallel to the image plane. This means that when viewed from other directions, these steep polygons, informed by only small areas in the reference image, can cover relatively large parts of the synthetic image. This is undesirable as these large polygons provide coarse representations of the surfaces they represent, and small errors in the reference depth or texture maps can impact large areas of synthetic images.

To address the problem of surfaces spanning depth discontinuities, or steep regions in the depth map, 3D mesh facets are drawn only where the gradient of the depth surface is relatively small. In practice, a threshold is chosen, limiting the resulting 3D area of facets in the mesh. If the polygon connecting the four corners of a depth-map pixel would have an area larger than this threshold, it is created, but not drawn in synthetic views. In practice, since generated image error is governed by relative area of created facets compared to the image regions that inform them, a limit of 2 is used for this ratio.

4.2 3D meshes from many images

To coalesce depth meshes from many images, we employ a counterintuitive approach, that produces some visual artifacts in intermediate generated views, but produces correct images once enough images have been collected. The approach employed removes the need to merge geometry across multiple depth meshes, and instead uses only the "best" polygons from each mesh. For the purposes of view generation, best is defined as having the smallest area. Facets from depth meshes have smaller area when they are generated from surfaces that are close to parallel to the capturing image plane. In addition to surface orientation, meshes representing objects that are closer to the camera's center of projection will have smaller facets. Intuitively, each facet represents data from a single sensor pixel, but a single pixel can record much finer detail (small area per pixel) on a close object than a far one. The selection of smallest facets is performed per-pixel in rendering. When many meshes overlapping meshes are available for the same physical space, each is rendered into a separate buffer, and results are combined using a fragment shader. Faces are given an alpha value that represents their area, and the fragment shader culls fragments that have larger area, scaled by distance to the camera.

Chapter 5

Planning

The computer graphics literature exhibits an increasing number of 3D illustrations of scenes generated from sets of still images. One frequent artifact in these images is the presence of visual tears or holes in the generated images. These holes are caused by two phenomena: failed matching, and occlusion. Holes in 3D models that are the result of failed matching appear when the algorithms used to construct models from the source images fail to determine the depth value for texture data present in one of the images. This can happen, for example, when depth is reconstructed via stereo matching from objects with little surface detail. Since many matching algorithms rely on local similarity between image regions in a stereo pair, when little texture is present these regions no longer appear to have a unique best match (they match many regions in the pair image equally well). This can make it difficult to determine the depth value for such low-texture regions. Another source of the holes present in many recovered 3D models is outright lack of information in the source images due to occlusion. Casually gathered still images inevitably do not record complete detail of the contents of a scene. In photographing a dining room for example, detail from some surfaces like the underside of the table, or the inside surfaces of a ceramic vase, are not likely to be photographed, unless specific effort is made to do so. These surfaces are likely to be blocked from view by other parts of the object in question, or by other objects.

Beyond obscure locations whose surfaces may not be present in generated views, it is also very easy to miss larger areas as well. The following figure from a system for generating 3D models of crime scenes [Se and Jasiobedzki 2005] shows such regions behind the copier, and at the front edge of the cabinet.

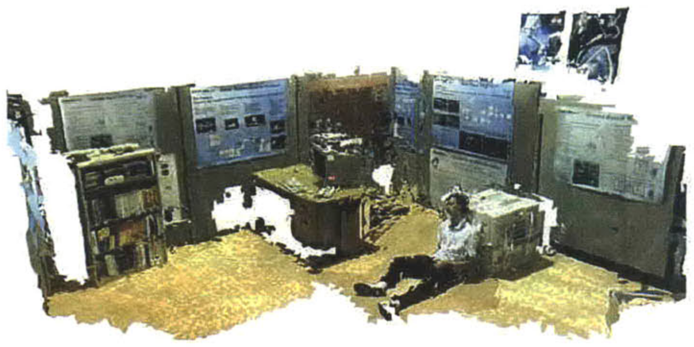


Figure 5-1: Figure from [Se and Jasiobedzki 2005] showing reconstructed scene with areas missing due to occlusion and matching error.

The main contribution of the work described here, is to provide tools for generating models from 3D scenes which provide methods for computing when sufficient scene information has been gathered. In addition, when desired scene information is found to be missing, this work describes methods for computing new camera locations (optionally subject to given constraints) that are likely to provide views of missing portions of the scene.

When reconstructing scenes from still images, there are three possible requirements for what information is available for reconstruction: we may wish to have all information about a scene so that it may be reconstructed from any imaginable viewpoint, we may wish to only gather enough information to reconstruct the scene from some continuous set or sets (lines, surfaces, or volumes) of camera positions, or lastly, we may wish to gather enough information to reconstruct scenes from a closed set of predefined point camera locations.

This work proposes methods for solving the third such problem, where desired viewpoints for scene reconstruction are known in advance. By pre-specifying virtual camera locations for reconstruction, we can address this problem without requiring the computation of watertight scene geometry. Geometry gathered may have holes, but this method describes how to seek models whose holes are not visible in the reconstructed views. Problems for continuous sets of camera positions can be approximated by sampling the spaces of desired camera locations and applying the methods described here.

5.1 Specifying the Scene Capture Task

The camera planning algorithm I describe here seeks to generate images from a set of novel viewpoints. The algorithm accepts as input, virtual camera descriptions and basic scene information both described below, in addition to a description of the available camera resources, and optionally intermediate results from previous executions of the algorithm. The algorithm provides as output, a synthetic image created for each given novel viewpoint, a description of a new camera position (or positions for multiple cameras) that is likely to provide visual information to improve the synthetic images. Also available at completion of execution is a set of 3D models, with texture and confidence information, which is used for subsequent executions. The algorithm seeks to maximize the "pixel gain per second," seeking to improve the total confidence over all pixels of all synthetic views.

Camera descriptions supplied to the algorithm consist of one or more virtual pinhole camera descriptions from which novel views will be generated. These descriptions consist of, for each camera, location, viewing direction, up direction, horizontal view angle and horizontal and vertical resolution. (Information describing locations of cameras are expressed in a common "world" coordinate system.)

Basic scene information provided as input is used to initialize system, and limit search for geometry. This information includes specification of a rough scene center point as well as an approximate scene radius, both expressed in the same world coordinate system as the virtual camera descriptions.

5.2 Determination of Model Completeness

Before describing methods for planning camera movement, it is useful to have way to determine if additional camera movements (additional images) are necessary. At some point, enough images may exist to completely synthesize views the scene from the desired viewpoints. To determine this, the algorithm maintains a confidence map that has a value defined for each pixel of the desired synthesized images representing roughly the probability that the information gathered so far for that pixel accurately represents the real scene. A confidence value between zero (no information) and one (high confidence) is maintained for each location on the geometry constructed from the existing set of still images. (Geometry consists of triangular facets with a single confidence value per facet.) To form the confidence maps, this geometry is projected into the framebuffers of virtual cameras with parameters determined by the desired view generation locations. These framebuffers are initialized to contain zero at all locations. Then, each facet of scene geometry with confidence value greater than some `min_confidence_to_render`, is simply colored with its confidence value and the standard computer graphics rendering pipeline is applied. After rendering, these confidence values then appear in the framebuffers in locations corresponding to locations where that facet would be the visible in each of the desired views. These framebuffer contents are the confidence maps. When all values in all confidence maps are greater than some completion threshold `min_confidence_per_pixel`, we consider the models to be complete for the set of desired views. footnote: This method fails to address situations where a low-confidence facet – generated erroneously due to a sensing or stereo matching error – obscures high-confidence, correctly generated facets. Future work will include methods for detecting and correcting for these situations, possibly adapting thresholds for rejecting facets from rendering.

```
Confidence_Map_Generate
1: for all facets in model do
2:     if confidence is greater than min_confidence_to_render then
```

```
3:       color facet with confidence
4:       render facet in all desired views
5:     end if
6:   end for
```

`Check_Gather_Complete`

```
1: for all desired views do
2:   Confidence_Map_Generate
3:   return: are all pixels in confidence map greater than min_confidence_per_pixel?
4: end for
```

5.3 General Planning Strategy

The core strategy employed for actual planning of new camera positions is simulation of potential views. Throughout the planning process, the system improves an internal 3D model of the physical space to be imaged, and uses that model both for image generation, and for consideration of new views.

Visibility of scene locations can be expensive to determine with certainty, especially when we are considering a range of potential camera locations. To reason about this, consider having a known patch of scene that we want to re-image (for example one we have seen at a grazing angle and want to re-image to gain a better view of surface texture). The task of deciding whether the patch will be visible from some new viewpoint under consideration is one of deciding whether other geometry will block our view.

From a single camera viewpoint, we can determine the visibility of a scene point by considering only other objects along the line connecting the point to view's center of projection. For the purposes of novel view generation, if the location of the novel view is specified, and is some 3D representation of a single camera-facing surface known to be in the real scene, we can be sure that either that some object exists between the camera and the surface, or that surface will appear in the camera's view. To decide which of these possibilities is the case, we have to search for geometry between the camera and the surface in question. Presence of any geometry means that the initially considered surface will be occluded. The process for finding the content of a stationary camera is fairly straightforward though. Only a small portion of space must be explored to answer the visibility question. If the scene is represented internally as a set of 3D triangles, the list may be traversed once, computing intersection between a line and a plane for each triangle. (Simply rendering a scene using the OpenGL pipeline can also answer this question, and can be an efficient way to evaluate the visibility of many surfaces simultaneously) In view planning, we are considering a range of camera locations, choosing one that gives us an improved view of (some part) of our scene. If we wish view a specific scene location, but want to consider a range of camera locations, visibility determination is no longer a search of a single line of 3D locations. Especially for fully unconstrained camera systems, the visibility of a scene surface requires much more information. For example, if we are considering camera locations from any location on a sphere around the patch in question, entire 3D volumes of space must be searched for geometry that may occlude the patch from some portions of this sphere. To compute the areas of this sphere from which a surface is visible, all scene geometry must be projected onto the sphere of camera locations, and regions must be found that contain the projections of no geometry. When computing this for several patches simultaneously, it would be easy to evaluate individual locations on the sphere for being able to see a location in question, the search for a "best" location on this sphere (one that say, could see as many patches as possible) would require merging all projections from all patches of interest. While possible, an approximation using sampling was chosen for this work.

5.3.1 Partial Information

In general, the planning task is one of determining visibility in the presence of only partial information. The goals of a new view of a scene are twofold. First, we want to use new camera views to discover surfaces in the real scene that a) have not yet been seen and b) will be visible in one of the synthetic views. Second, we want to refine views of surfaces we have seen that a) have been seen previously but with insufficient resolution and b) that will be visible in one of the synthetic views.

To guide this search, this thesis takes advantage of the relative continuity of real scenes. In general, objects are in contact with other objects, and open space is next to more open space. If we happen to have observed some 3D location to contain an object, we have a better chance of finding more occupied space if we look near that object than if we sample some other randomly chosen 3D location. Locations in space that are not adjacent to other objects can be considered in the planning system, but only after we have explored the extent of objects that we have already seen.

The relative continuity of real scenes also works to our advantage when considering occlusions by unseen geometry. When we choose a new camera location, we do so with the hope that the image taken from that location will contain portions of the scene also visible by the desired view(s), so we can come closer to our overall goal of populating those views. When we consider a camera location, clearly we don't want one that has its view blocked by some object outside of the desired view volumes – such an object will not contribute to the desired views. Unfortunately, we can only compute how to avoid occlusions by geometry we have previously seen. For this reason, we adopt a search strategy that assigns a low but present priority to exploring geometry that falls outside the desired views, but blocks parts of them from view. If the system cannot see objects in the desired view, geometry of outside objects is still modeled so that knowledge of its shape can be used to find ways to see around it. As such blocking objects are, in most scenes, generally spatially continuous, accidentally stumbling onto one of these occluding objects will lead to its exploration, after which it can be avoided.

5.4 Algorithm Overview

The planning algorithm used in experiments performs the following steps to choose the next view:

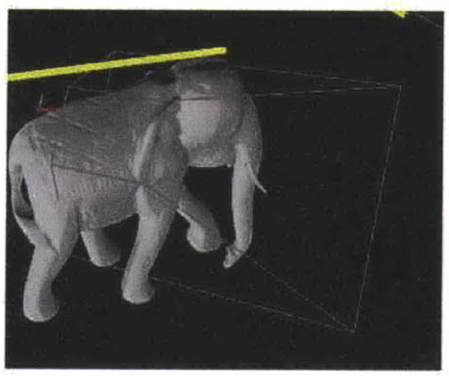
```
Plan_Next_View
 1: Image Capture
 2: Depth Map Generation
 3: Depth Mesh Generation
 4: Novel View Projection of face ID numbers
    For each novel view, the newly captured mesh is rendered, coloring each face with a
    unique color value. Lighting is disabled.
 5: Weighting of visible mesh faces
    All mesh faces are assigned a small base weight, then mesh face weights are incre-
    mented based on the number of novel-view pixels in which they appear.
 6: Re-imaging of weighted mesh from candidate new view locations
    The space of possible camera center of projection locations is discreteized into an
    array spanning possible camera locations. For each location, a wide-angle image is
    synthesized of the current scene mesh. Mesh faces are colored with their weights, and
    lighting is turned off. After all candidate view buffers have been rendered, weights are
    totaled across the entire buffer and the location with the largest total is chosen as the
    new camera location.
```

Although not explicitly computed within the planning process, novel views are computed from meshes as follows:

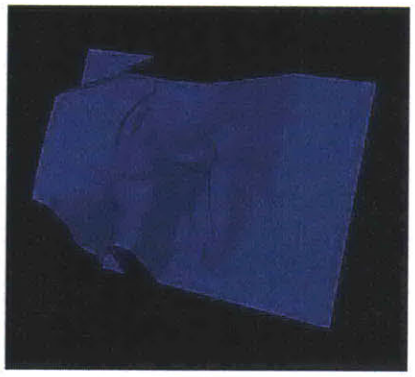
```
Populate_Novel_View
```

1: Each mesh is rendered to an individual framebuffer from the viewpoint of the novel view, using RGB texture mapping, and alpha value determined by mesh polygon area. Rendering of back-faces is disabled.

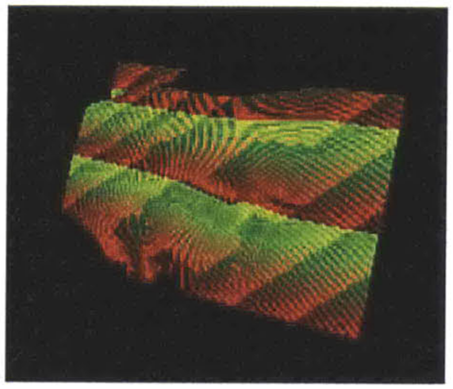
2: Framebuffers are merged by a fragment shader, keeping the smallest polygons over all views at each screen pixel.



Simulated elephant scene, with robot and camera bounds



Mesh from simulated camera using OpenGL z-buffer



Depth mesh colored with face identifiers

Figure 5-2: Illustration of depth map and face identifier map for a simulated scene

5.5 Movement Planning for Eye Society Robots

When using the Eye Society Robots to gather images for scene modeling, the range of possible locations from which we can gather scene information is restricted approximately to a set of line segments defined by the camera tracks. The center of projection of each of the cameras is very close to the cameras' pan and tilt axes and the aperture of the lenses on the cameras is only a couple of millimeters. This means that after gathering an image, actuating one of the pan-tilt servos can reveal parts of the scene that were outside the bounds of the camera's image sensor in the first photograph, but will not substantially change views of objects that are fully in both images. Parts of objects that are occluded in one of these views will, for the most part, be occluded in any view taken after a pan or tilt. Considering this, image information needed to generate the desired views that is missing due to occlusion is unlikely to become after a pan or tilt, so the only camera movements available for recovering occluded information are movements along their one-dimensional support rail. For this reason, we can separate the planning of camera position and camera direction. Pan-tilt movements are used to fill in information available from the current position but outside of the current field of view while lateral movements are used to position the camera so that it can look behind objects that occluded parts of the scene in previous views.

Internally, to plan movement, we compute, based on several considerations described in the next section, a goodness score for potential new camera locations. Cameras are directed to move to the position with the highest goodness score. Currently, this goodness score is approximated as a discrete array of goodness values representing small bins of position along the rail, but future work could explore a continuous representation.

5.5.1 Single camera along a rail

When a single Eye Society camera is used for generating models of a scene, first a stereo pair of images are captured to create an initial estimate of scene geometry. Then, plans to move the camera and gather subsequent images are computed based on the confidence maps created from this geometry estimate.

For indoor operation of the Eye Society Robots, cameras operate with a long (about half of a second) sensor integration time. Because of this, substantial motion blur results when the cameras are moving during image capture. Also, because of the geometry of the track support mechanism and high torque of the track drive motor, linear movements of the robots along their rails cause robots to exhibit swaying motion that can continue up to about two seconds after the robots are commanded to stop moving. In addition, the movement itself takes time (about ten seconds to traverse the entire rail), so movements generally require several seconds to complete. Pan and tilt operations however only move a small part of the robot. This leads to quick (sub-second) movement completion time, and very little shaking. and image capture can commence about a half-second after these movements are made. For these reasons, pan operations are prioritized over movement operations when possible, and movements of short distances are preferable to long moves.

At the beginning of scene capture, the first pair of images is gathered from the current location of the camera along its track. This capture is intended to help the system "get it's bearings." As no geometry information is available until images have been gathered, computational resources sit idle until images are available. Even if the images gathered here are not ideal, computation can begin early, and continue as the robot is unavailable while subsequent movements are made.

For the first images, the pan and tilt actuators are adjusted to point the camera approximately to the center of the scene to be captured. This center, expressed in world coordinates is specified by the operator of the system to direct scene capture, but in principle an arbitrary starting "center" could be chosen as well. For the capture of subsequent images, the camera is also directed, in pan and tilt, towards a specific scene location. The method for choosing pan and tilt values is described next.

Pan and tilt values for pointing the camera at a specific scene point are found as follows: First an "object" vector is found by transforming the scene point into the the robot-centered coordinate system. Pan angle is found by projecting the object vector onto the robot's pan plane, and computing the signed angle between this and the robot's calibrated zero-pan direction in the pan plane. Relative tilt angle is found by computing the angle between the object vector and its pan-plane projection. This relative tilt angle is added to the robot's calibrated zero-tilt angle.

After this first stereo pair has been gathered, matching is performed to generate a depth map, and then 3D meshes with texture and confidence values are computed as described in the previous chapter. These meshes are used for planning the next view as described next.

Pivot planning

The most inexpensive and reliable camera movement for the Eye Society robots is a simple change in pan and tilt settings. We can quickly tell if such a camera movement will be beneficial by looking for edges of the most recently gathered 3D mesh in the synthesized view(s). This can be done by projecting vertices from the perimeter of the most recent reprojected image pair into the coordinate system of the synthesized views. If these edge vertices lie within the view volume of the synthesized view, this most recent camera orientation "cuts off" part of the scene projected into the desired view. Furthermore, if no geometry information is available (zero appears in the confidence map) at locations adjacent to projected edge vertex locations, the scene information that lies outside of the frame of the camera has not been gathered by previous imaging. This makes this location a good candidate for a pan-tilt movement.

If, for example, a physical camera has a much longer focal length than the desired view, most of the images it can capture that overlap the desired view will produce the mesh edges (described in the previous paragraph) at most if not all of the edges of the most recent physical view. To plan subsequent pan-tilt movements, exploration proceeds towards the edge vertex that falls closest to the desired view (or has the smallest distance from any of the multiple desired views). Because objects close to a virtual camera appear larger, and due to the piecewise smoothness of most scenes, we generally have a good chance of retrieving useful scene information from areas nearby known object locations, especially if they are close to the center of projection of the desired view. To prevent repeated search of the same location, cameras maintain a list of visited directions for the current track position. Views are removed from consideration if they contain more than some number of these directions.

5.5.2 Translation Planning

Planning of translational movements for the Eye Society Robots proceeds with consideration given to speed of the track motor actuator. As candidate views are evaluated as possible camera destinations, views farther along the track are penalized for their distance from the current camera position. Penalty is chosen by dividing the score previously assigned to each view by the total time required to move the camera and capture an image.

$$\frac{\text{candidate\_view\_score}}{\text{image\_grab\_time} + \text{transit\_time\_to\_location}} \tag{5.1}$$

This weighting allows movements to be chosen to maximize the estimated number of novel view pixels improved per second.

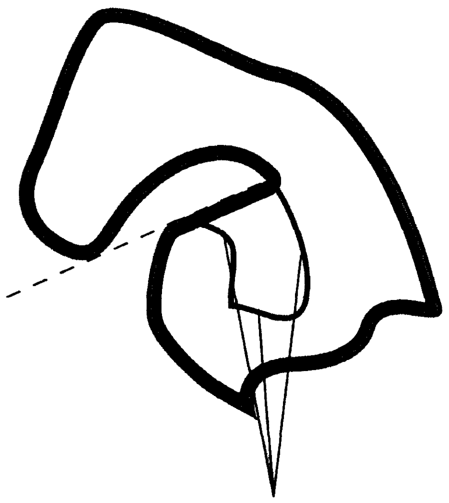


Figure 5-3: We only seek to model the approximate visual hull of a scene. Some areas may be occluded in all views.

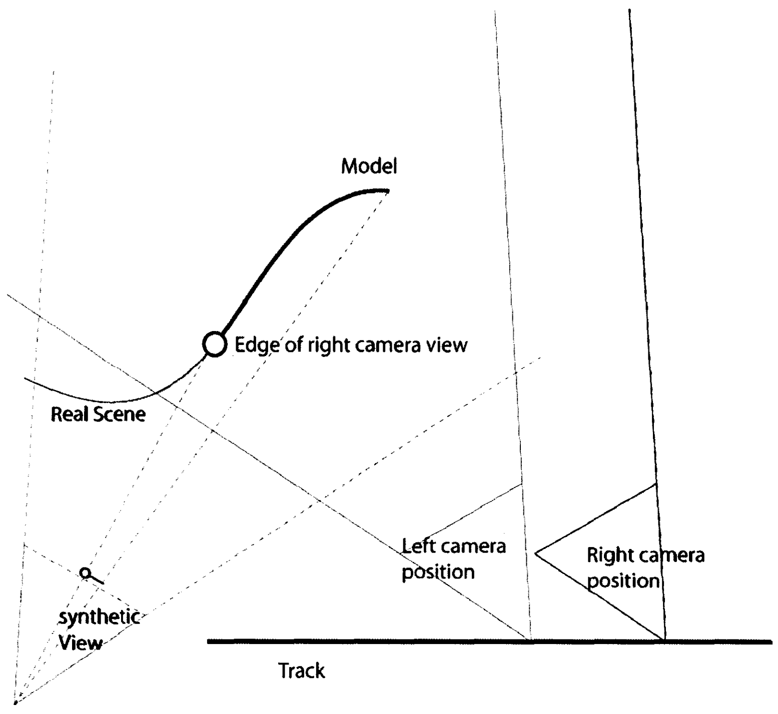


Figure 5-4: Model edge falls at left edge of a view. When this edge falls inside the synthetic view volume, pivoting camera position left will reveal more of the scene.

Chapter 6

Experiments

6.1 CameraNetworkViewer

An application was constructed to visualize state of the Eye Society robots, including their fields of view, relative locations, and ranges of motion. This application was also used to display results of planning experiments.

The CameraNetworkViewer application can connect directly to the software running on the Eye Society Robots, or can be run offline in simulation. When connections to the robots are available, a user can interactively manipulate the pan and tilt and track actuators, and see on screen the camera geometry of the robots in their current positions and orientations. The user can also interactively capture single images or stereo pairs to test the system.

The CameraNetworkViewer can be used to initiate planning by interactively adjusting a set of simulated camera locations to specify the desired views of the scene. After choosing desired views, the application applies the planning algorithm to iteratively compute new camera positions, and then capture images from those positions.

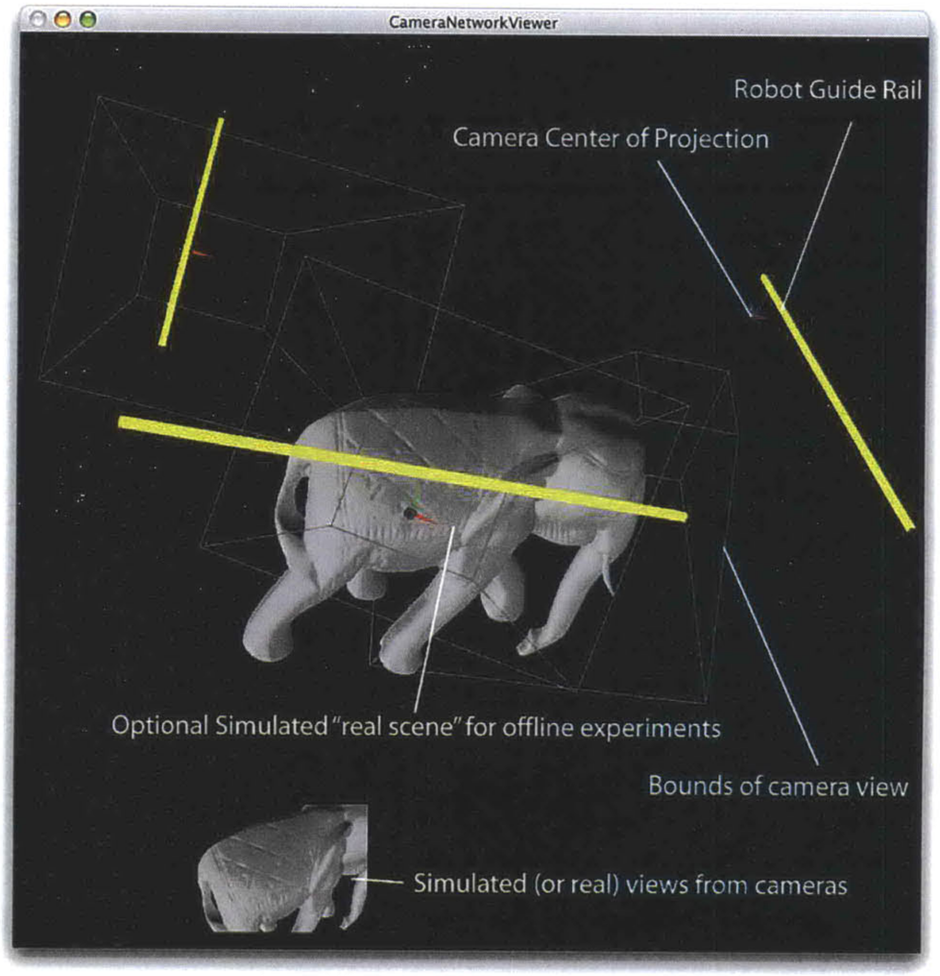

CameraNetworkViewer
Robot Guide Rail
Camera Center of Projection
Optional Simulated "real scene" for offline experiments
Bounds of camera view
Simulated (or real) views from cameras

Figure 6-1: CameraNetworkViewer, an application for interacting with and displaying results from the Eye Society Camera Network

6.1.1 Simulated planning

To quickly test portions of the planning algorithm in isolation, a version of the planning system was implemented to operate on synthetic 3D scenes instead of real physical scenes. Scenes are imported as vrml models, and the model of real camera track locations and ranges of motion is used. Synthetic images can be captured of the synthetic scene, and depth maps may be computed either by the stereo matching procedure described in the Stereo chapter, or by directly reading the z-buffer from the OpenGL pipeline. Directly reading the z-buffer produces depth maps that are unrealistically vivid, but allows for validation of the planning algorithm independent of stereo matching error. Depth maps of the accuracy obtained by reading the z-buffer might be obtained in the future by using alternate stereo matching techniques, higher resolution images, or alternate depth sensing techniques.

6.1.2 Depth-map results

When applied to simulated scenes, the stereo matching algorithm performs reasonably well. Since the exact geometry of images is known, errors in the depth maps can be directly observed.

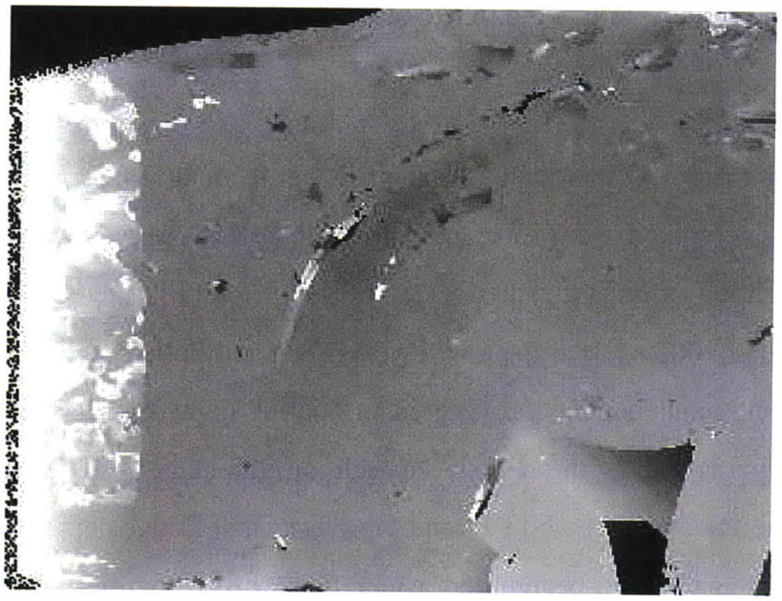


Figure 6-2: Depth image of elephant model, computed with stereo matching shader.

6.1.3 Reprojection results

To further test the stereo matching system, the above depth maps were re-projected into the simulated scene and triangulated to create 3D meshes. Here, meshes were rendered omitting faces that fell at the near or far planes, and also omitted faces that had area greater than

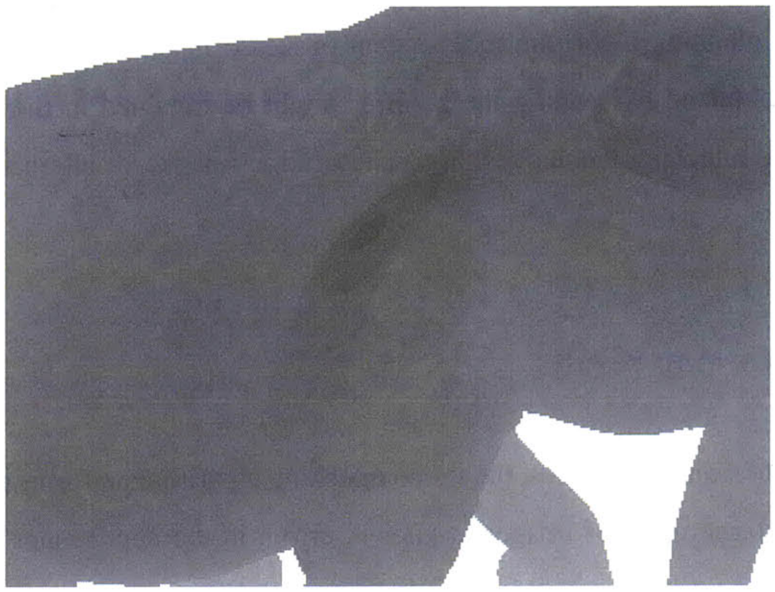


Figure 6-3: Ground-truth depth image of elephant model, from OpenGL z-buffer.

one square centimeter (At this scale, elephant model is about 2 meters long). The depth map shown was computed using 160 candidate depth levels, and even with this many, clear artifacts are visible from quantization of depth levels. Realtime stereo matching systems generally consider 100 or fewer disparity levels [Yang and Pollefeys 2003], yet still, compared to the continuous depth scores stored in floating point z-buffers, this quantization impairs the visual appeal of the results. In addition, the strategy of removing faces that have large area, which works well for removing faces that bridge depth discontinuities in the scene, works against the system here where depth discontinuities also result from the quantization of depth values.

When texturing and back-face culling is performed on the mesh facets used for reprojected object rendering, the depth quantization becomes much less apparent. Clear artifacts can still be seen from areas near the edges of the camera images used to compute the depth map, as well as at locations where no detail is visible on objects or the background.

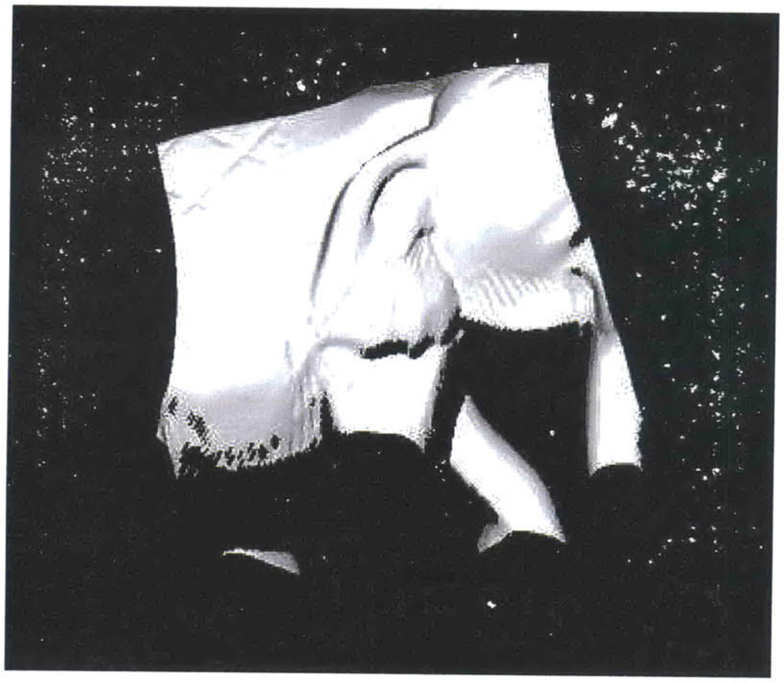


Figure 6-4: Reprojected untextured rendering of OpenGL z-buffer depth map above.

6.1.4 Real image results

Stereo matching results for interior scenes in the room where the Eye Society Robots are deployed have been poor. The environment contains numerous reflective and untextured surfaces, and although small patches of the scene are reprojected accurately, still frames of these reprojected scenes are not compelling. In depth maps from interior scenes, and also when interactively viewing the reprojections, it can be seen that correct depth estimates are being made for the borders of the LCD monitors in the room, but fail to match their untextured centers. Stereo matching algorithms have been developed to improve performance in untextured areas [Yang and Pollefeys 2003] although demonstration images provided usually only contain textured surfaces. Future work will include exploration of refinements to the stereo matching algorithm presented here.

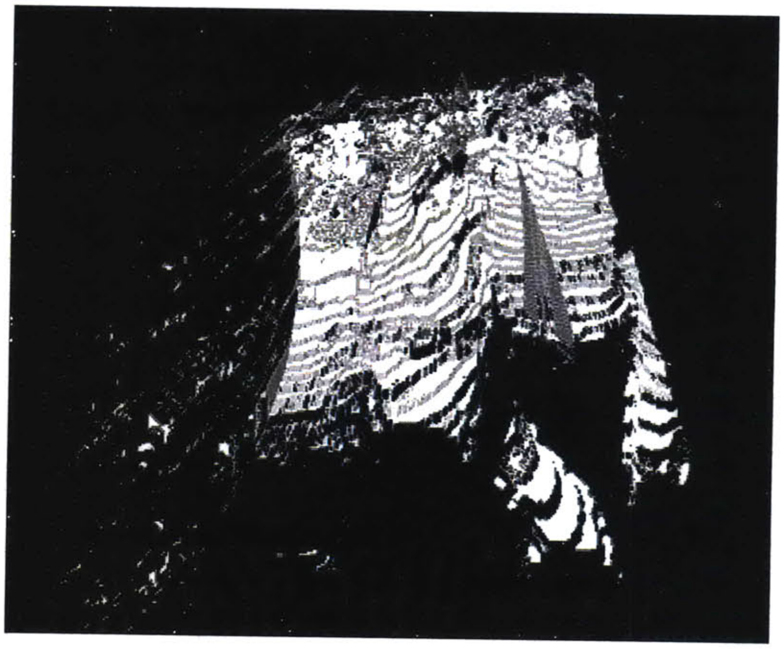


Figure 6-5: Reprojected untextured rendering of stereo matched depth map above.

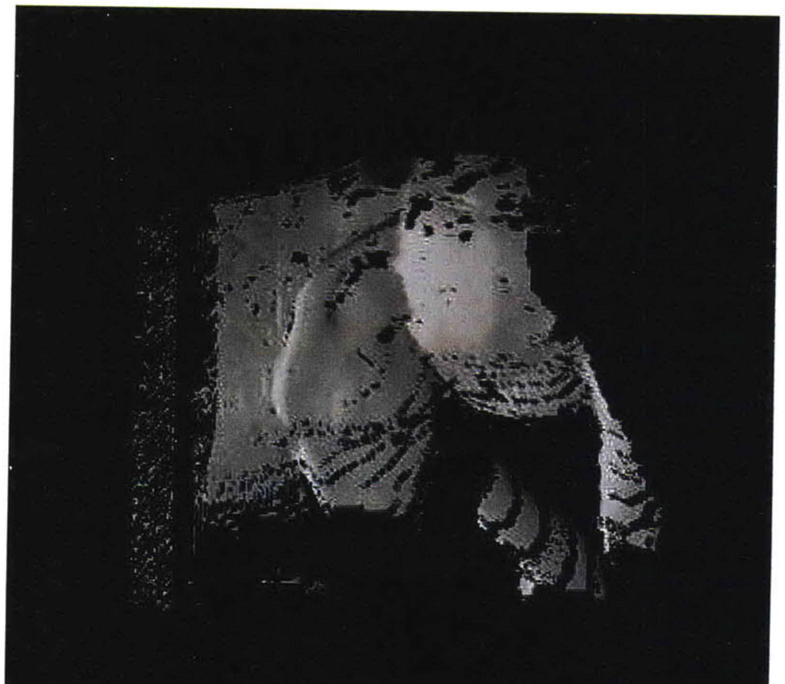


Figure 6-6: Reprojected textured rendering of stereo matched depth map.

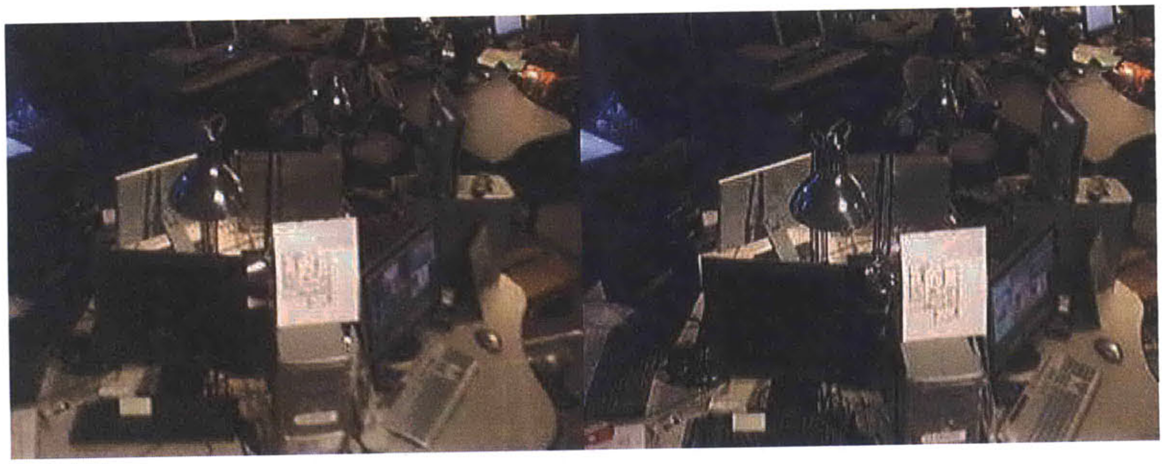


Figure 6-7: Two images taken by an Eye Society camera, used for depth reconstruction below.

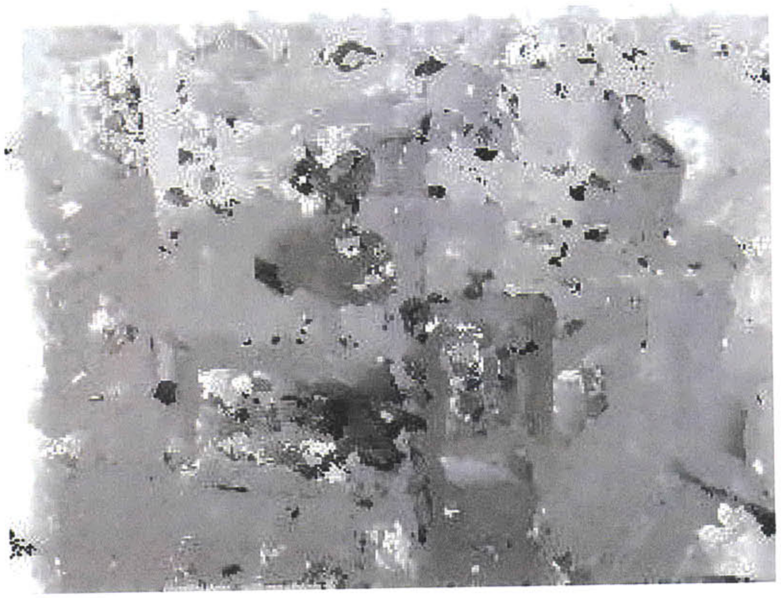


Figure 6-8: Depth map computed from the stereo pair above.

Figure 6-9: Reprojection of above depth map.

Chapter 7

Discussion

This thesis presented algorithms and software tools developed to allow automated capture of real scenes. While experiments show that the tools are capable of addressing all of the intermediate steps towards this goal, there is still much room for refinement of these tools before compelling synthetic scene views can be produced. This section outlines implications of the results for the individual parts of the planning system and robotic camera network, and identifies possible directions for future work in these areas.

7.1 Calibration

Calibration of camera properties enables correction of lens distortion present in the Eye Society cameras. While the minimization procedure implemented in the Matlab Camera Calibration Toolbox [Bouguet 2000] performs reasonably well, an improved system could use image comparison across movement of the robot actuators. The current system relies on a target with known geometry, which does have the advantage of fixing absolute scale of images, does not make use of much of the area of the calibration images, as only image locations containing the calibration chessboard are used. This limitation requires the capture of more images than would an automatic system. Likewise, simultaneous calibration of the camera intrinsic parameters and the actuator ranges of motion could enable more precise location of scene points, and could improve matching and modeling performance.

7.2 Depth map generation

Depth maps computed by the GPU shader described in this thesis perform well on textured images, or images where geometry is precisely known. Performance in situations with more pose misestimation error is often poor. Search for matching image patches is currently constrained to locations falling along a single line. When pose of cameras is estimated incorrectly, matching image locations may fall outside the area that is searched. Incorrect pose estimation resulting from the open-loop nature of the track actuator could be improved by adding an optical or resistive sensor on the track to better monitor robot position. Physical construction of the robots could also modified to improve stiffness, so that the weight of the power cables, and tension on the camera USB cable would be less likely to move the physical camera in ways that are not measured by the servo control loops.

7.3 Mesh generation

Construction of meshes from stereo depth maps produces reasonable facsimiles of synthetic scenes. The appearance of meshes that have visible errors when viewed without texture can improve substantially when scene-derived texture is applied. Raw meshes computed by tessellating depth maps contain bridging membranes across discontinuities even in the best case, and contain holes and spikes where errors appear in the source depth map. Meshes might be improved by applying operators, such as a low-pass filter or a median operator, but these are likely to degrade accuracy for scenes containing high-frequency detail. The mesh implementation used in the CameraNetworkViewer application contains an implementation of the half-edge data structure used in geometric algorithms such as complex hull. These data structures maintain information about connections between facets in the mesh, and could be used to detect and fill holes resulting from missing information.

Chapter 8

Acknowledgments

I would like to thank my advisor, Mike Bove, and readers Wendy Plesniak and Professor Pentland for their time in reviewing this thesis, and for their comments on how to improve it. Thanks also to the US Postal Service, DHL, and Deutsche Post for enabling the signing and submission of this document across many borders.

Bibliography

ABRAMS, S., ALLEN, P. K., , AND TARABANIS, K. A. 1993. Dynamic sensor planning. In *Proceedings 1993 IEEE International Conference on Robotics and Automation.*

ADELSON, E. H., AND WANG, J. Y. A. 1992. Single lens stereo with a plenoptic camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2, 99–106.

AGAMANOLIS, S. 2001. *Isis, Cabbage, and Viper: New Tools and Strategies for Designing Responsive Media.* PhD thesis, MIT.

AZARBAYEJANI, A., GALYEAN, T., HOROWITZ, B., AND PENTLAND, A. 1994. Recursive estimation for cad model recovery. In *Proceedings of the 1994 Second CAD-Based Vision Workshop*, 90–07.

BECKER, S., AND BOVE, JR, V. M. 1995. Semiautomatic 3-d model extraction from uncalibrated 2-d camera views. In *Proc. SPIE Image Synthesis*, vol. 2410, 447–461.

BELHUMEUR, P. 1993. A binocular stereo algorithm for reconstructing sloping, creased,and broken surfaces in the presence of half-occlusion. In *Computer Vision, 1993. Proceedings., Fourth International Conference on*, 431–438.

BOUGUET, J. 2000. Matlab camera calibration toolbox.

BOVE, JR, V. M., PLESNIAK, W. J., QUENTMEYER, T., AND BARABAS, J. 2005. Real-time holographic video images with commodity pc hardware. In *Proc. SPIE Stereoscopic Displays and Applications*, v. 5664A.

BOVE, JR, V. M. 1989. *Synthetic Movies Derived from Multi-Dimensional Image Sensors*. PhD thesis, MIT.

BROOKS, R. A. 1982. Symbolic error analysis and robot planning. *International Journal of Robotics Research 1*, 4 (December), 29–68.

BROWN, M. Z., BURSCHKA, D., AND HAGER, G. D. 2003. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25*, 8, 993–1008.

DAVIS, J., AND CHEN, X. 2003. Calibrating pan-tilt cameras in wide-area surveillance networks. In *Proceedings. Ninth IEEE International Conference on Computer Vision*, vol. 1, 144–150.

DEBEVEC, P. E., YU, Y., AND BORSHUKOV, G. D. 1998. Efficient view-dependent image-based rendering with projective texture-mapping. In *Eurographics Rendering Workshop 1998*, 105–116.

FUA, P., AND LECLERC, Y. 1995. Object-centered surface reconstruction: combining multi-image stereo shading. *International Journal on Computer Vision 16*, 1, 35–56.

KANADE, T., RANDER, P., AND P.J.NARRAYANAN. 1997. Virtualized reality: Constructing virtual worlds from real scenes. In *IEEE Multimieda*, vol. 4(1), 34–47.

KARTCH, D. 2000. *Efficient Rendering and Compression for Full-Parallax Computer-Generated Holographic Stereograms*. PhD thesis, Cornell University.

LEHEL, P., HEMAYED, E. E., AND FARAG, A. A. 1999. Sensor planning for a trinocular active vision system. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, vol. 2, 2306.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2 (Nov), 91.

MALLETT, J., AND BOVE, JR, V. M. 2003. Eye society. In *Proc. ICME 2003 (special session on Smart Cameras).*

MALLETT, J. 2005. *The Role of Groups in Smart Camera Networks.* PhD thesis, MIT.

MATUSIK, W., PFISTER, H., ZIEGLER, R., NGAN, A., AND MCMILLAN, L. 2002. Acquisition and rendering of transparent and refractive objects. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 267–278.

PILPRE, A. 2005. *Self-* Properties of Multi Sensing Entities in Smart Environments.* Master's thesis, MIT.

PINHANEZ, C., AND BOBICK, A. 1995. Intelligent studios: Using computer vision to control tv cameras. In *Proc. of IJCAI'95 Workshop on Entertainment and AI/Alife*, 69–76.

POLLEFEYS, M., GOOL, L. V., VERGAUWEN, M., VERBIEST, F., CORNELIS, K., TOPS, J., AND KOCH, R. 2004. Visual modeling with a hand-held camera. *Int. J. Comput. Vision 59*, 3, 207–232.

ROST, R. 2004. *Open GL shading language.* Addison-Wesley.

SCHARSTEIN, D., AND SZELISKI, R. 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision 47*, 1/2/3 (April-June), 7–42.

SE, S., AND JASIOBEDZKI, P. 2005. Instant scene modeler for crime scene reconstruction. *cvprw 0*, 123.

SHI, J., AND TOMASI, C. 1994. Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition*, 593–600.

SINHA, S., AND POLLEFEYS, M. 2004. Towards calibrating a pan-tilt-zoom cameras network. In *ECCV Conference Workshop CD-rom proceedings*.

SUN, J., ZHENG, N.-N., AND SHUM, H.-Y. 2003. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25*, 7, 787–800.

TELLER, S. J., ANTONE, M. E., ZACHARY BODNAR, M. B., COORG, S. R., JETHWA, M., AND MASTER, N. 2005. Calibrated, registered images of an extended urban area. *International Journal of Computer Vision 53*, 1, 93–107.

TRIVEDI, M. M., HUANG, K. S., AND MIKIC, I. 2005. Dynamic context capture and distributed video arrays for intelligent spaces. In *IEEE Transactions on Systems:Man and Cybernetics - Part A: Systems and Humans*, vol. 35.

UNGER, J., WENGER, A., HAWKINS, T., GARDNER, A., AND DEBEVEC, P. 2003. Capturing and rendering with incident light fields. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 141–149.

WILLSON, R. 1994. *Modeling and Calibration of Automated Zoom Lenses*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

WREN, C., AND RAO, S. 2003. Self-configuring lightweight sensor networks for ubiquitous computing. In *Proc. Int. Conf. Ubiquitous Computing*, 205–206.

YANG, R., AND POLLEFEYS, M. 2003. Multi-resolution real-time stereo on commodity graphics hardware. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 211–218.

[illegible] 78 - [illegible]