POLYHEDRAL HULL ONLINE COMPOSITING SYSTEM: TEXTURING AND REFLECTIONS

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Masters of Science

by

Adam Michael Kravetz

August 2005

ABSTRACT

The Polyhedral Hull Online Compositing System is a prototype system which merges live dynamic video and three dimensional synthetic imagery at interactive frame-rates. This system captures video from convergent cameras and performs geometric reconstruction to generate a three dimensional mesh. Utilizing the background geometry which is generated by an external renderer and the reconstructed mesh, this system adds shadows and reflections to the final composite image. The combination creates a perceptual link between the two otherwise disjoint environments. The computed mesh reconstruction allows this system to be view independent, which is a major advantage over previous state-of-the-art systems. By using modern graphics hardware and a distributed computing model to capture and process live video data with synthetic three dimensional imagery into final composites, we provide an economic alternative to standard commercial systems.

The texturing and reflection processes, two key parts of the Polyhedral Hull Online Construction System, are described in detail in this thesis. The texturing processes explore the problem of providing a visually plausible, view-independent representation with only four video cameras capturing live data. The reflection process calculates the dominant specular highlight of the live object in order to render visually convincing results. These processes add degrees of realism so that final composite image is a visually plausible merger of real and synthetic imagery.

BIOGRAPHICAL SKETCH

Adam Kravetz was born in Rochester, NY. He attended the JCC of Rochester Pre-School, Thornell Road Elementary School, Mendon Center Elementary School, Pittsford Middle School and finally, Pittsford Mendon High School. In 1998 he began his undergraduate career at Cornell University culminating in a computer science degree in the spring of 2002. He joined the PCG in the fall of 2002.

For my dog, Higgins: 1991-2004.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my parents, Jill and Bob Kravetz, for their unfaltering support. I have relied on them for everything from financial to emotional support and they have jumped at the chance to help me in anyway possibly. I would like thank my sister, Lauren, and grandmothers, Adele Stoler and Vera Kravetz, as their understanding and support helped me regain perspective and focus, and for being voices of guidance during tough times. My family is and will continue to be my greatest source of strength in overcoming life's challenges, and I am greatly indebted to them.

My academic family, beginning with my advisor, Don Greenberg, provided insight and support in times when my family was absent. His expertise as an academic, a leader, and a businessman gave me a captivating role model to emulate. My family would most certainly be incomplete without my good friends and academic advisors Kavita Bala and Emin Gün Sirer. Their guidance both personally and professionally shaped me greatly. They provided great advice and assistance in times when I struggled to define my goals.

The late night coding sessions, myriad of shared pizza and endless whiteboarding with fellow student Vikash Goel has been the driving force for my existence since my first day at Cornell seven years ago. Thanks Vikash, you are the brother I never had. A very special thanks goes to Julia Francoise Lowd for pushing, pulling and dragging me through endless edits and providing infinite support in so many ways. Special thanks goes to Elisabeth 'Kiki' Becker for her writing support.

Thanks to all the guys at the PCG:

Henry Letteron: my partner in crime

Ryan Ismert: Mr. Ahead of his time

Jacky Bibliowicz: A great officemate and good friend

Jeremiah Fairbank: For his architect's point of view

Mike Donikian: David D' and Lance in one, what more can you ask

SPF: The keeper of the PCG and the coffee

Jon Moon: You might look like Eminem but you'll always be one of the New Kids On The Block to me.

I would have been lost without the full time staff and researchers at the Program of Computer Graphics. To Hurf Sheldon, Peggy Anderson, Linda Stephenson, Bruce Walter, Jim Ferwerda, Martin Berggren, Steve Westin and Mary. My experiences at the lab were assisted by all of you in inumerable ways. I only hope to find as welcoming a group at my next foray in this world. Thank you all for providing equipment, advice, and help when it was needed most.

Finally I have to thank Cornell Outdoor Education and everyone that I've met, befriended, and worked with there. I could have never predicted the effect that it would have on my life and I'll miss my six years there dearly.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

The entertainment industry's usage of computer generated efffects has increased dramatically in recent years. A particularly popular technique is to merge live and synthetic imagery to replace traditional filmed footage. A computer modeled environment, a "virtual set" eliminates the complexities of filming on location or the difficulty of constructing elaborate sets. Furthermore, computers allow greater flexibility since the virtual environments are not restriced to real life constraints. Unfortunately current virtual set technology is expensive and restrictive in the sense that most systems can not simulate global illumination effects. Our system attempts to overcome these constraints using more economical off-the-shelf components.

The prevalent system for single camera, two-dimensional, real and synthetic compositing is chroma-keying. This process, also known as green-screening, segments an object in the foreground from the background so that it can be stitched to a complex synthetic background to produce a final composite [Sel03]. The live images in these scenes are simple two-dimensional polygons with video textures mapped onto them. These virtual set systems have limited ability to accommodate shadows and reflections.

The goal of the Polyhedral Hull Online Construction System is to utilize three-dimensional information to merge live video with synthetic imagery such that a visually plausible composite can be generated in realtime. Reconstructing a three-dimensional object from a set of planar images and adding texture, shadows, and reflections to this object is both time consuming and mathematically complex.

However, the primary advantage of a three-dimensional reconstruction process is that traditional computer graphics processes, such as ray-tracing can be used. Previous work either generated a two-dimensional object in realtime or reconstructed a three-dimensional object after many minutes of computational work. The Polyhedral Online Construction System captures and reconstructs a three-dimensional representation of a live actor or object and then renders it in a synthetic environment, adding textures, shadows, and reflections at an interactive frame rate.

The system is implemented in stages as shown in Figure 1.1. First, the live object is captured by employing a set of video cameras that view it from different angles. Second, a silhouette is extracted from each individual video image and blurred to remove aliasing. After the contours are found for each image, by applying the rules of epipolar geometry, a polygonal mesh is reconstructed. A texture is then generated for the mesh with information from each of the video cameras. Finally, the compositing step merges the reconstructed and textured mesh with the synthetic image and adds global illumination effects (shadows and reflections) between the two portions of the composite. The addition of these shadows and reflections creates a more coherent image that is both visually pleasing and believable.

The Polyhedral Hull Online Construction System leverages the properties of a three-dimensionional mesh to overcome pitfalls of two-dimensional techniques. While there are other three-dimensional systems, such as the image based visual hulls work [MBM01a], they are either offline or do not offer as much functionality as our system. We chose to build a multi-stage pipeline consisting of image capture, image segmentation, mesh reconstruction, texturing, shadowing, and reflection stages to optimize our system for performance. This pipeline eliminates some of

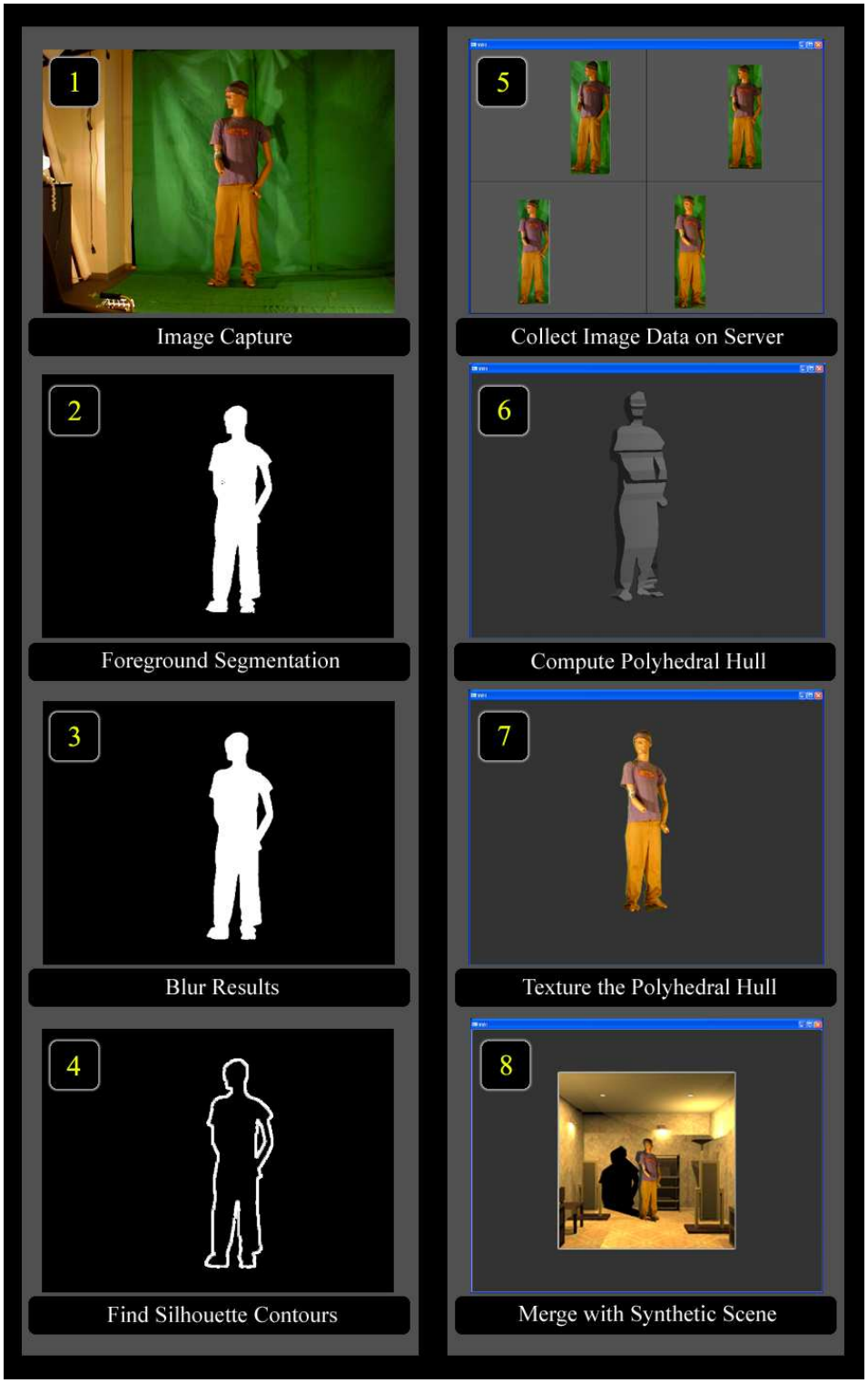

Figure 1.1: *This diagram depicts each stage of the image processing, geometric reconstruction and compositing process.*

the bottlenecks of previous systems by utilizing network communication to enable us to process the necessary data in realtime.

The Polyhedral Hull Online Construction System improves lighting, reconstruction, and efficiency to achieve visually plausible images at interactive frame rates. This allows the end-user to navigate around the composited environment and view both the real and synthetic objects together. Each component of the system adds realism to the final product. First, the visual hull is reconstructed, which produces a three-dimensional mesh to interact with the other objects in the scene. Second, during the texturing process, the mesh is overlaid with information from the group of video cameras used for reconstruction, giving it a pose. Finally, during the global illumination effects stages, shadows and reflections are added to unite the real and synthetic portions, creating a single image in which all the elements of the system interact with one another. The system has two distince advantages: First, the three-dimensional representation, the polyhedral hull, is used to produce more believable shadows. Second, because multiple cameras are used, the final images are not restricted to a single viewpoint.

This thesis focuses on the texturing process and the addition of reflections to the synthetic environment. The remainder of the thesis is organized as follows. Chapter Two is a discussion of previous related work. Chapter Three describes the full Polyhedral Hull Online Compositing System. Chapter Four details the challenges, shortcomings and solutions of the texturing process in this system. Chapter Five explores the reflection generation and composition in our final rendering. Chapter Six concludes with a discussion of future work.

Chapter 2

Related Work

This chapter will review the work that is either motivational or has relevant significance for our research. Motivationally speaking, there are a number of papers that present new work, offering clear, improved results over previous systems, and reinforcing the first principles of Computer Vision and geometric reconstruction. Most of the motivational work comes from the Computer Vision community, in which geometric reconstruction is the core focus of their research. From a standpoint of immediate relevance, we look at image-based visual hulls, view-dependent texture mapping, and related works from recent Siggraph and EuroGraphics proceedings.

2.1 Motivation and Concepts

Our goal is to improve the visual hull reconstruction system, which is a contour-based reconstruction engine. A visual hull is a three dimensional bounding volume created by clipping projections of multiple two dimensional silhouettes against one another. Overall, the system should be able to reconstruct a watertight mesh of significantly high quality, so that it can be composited in a synthetic, globally illuminated scene in a seamless manner. These high standards have led us to carefully study the work of high-power offline computer vision algorithms.

2.2 Computer Vision Techniques

The Computer Vision community is a large, active branch of computer science studies, which addresses the inverse of the computer graphics problem. In computer graphics, we are primarily concerned with generating an image, given the

pose of a camera, as well as the geometry, lighting and materials of a scene. In Computer Vision, there is a notion to take an image as input, and to determine the properties (generally the pose of a camera and the object locations or geometry) of both the scene and the camera that generated it. This work is imperative to our research, since we are utilizing images for geometric reconstruction and compositing them into a typical computer graphics scene. Our work is a natural cross product of these two different bodies of work, of which we generally take a computer graphics slanted approach, yet draw on many first principles adapted from computer vision. The most critical bodies of relevant work are included below.

2.2.1 Epipolar Geometry

Epipolar geometry is the geometric relationship between two cameras. In our example, these camera's are labeled C_1 and C_2. The epipole of a camera is the camera's center, commonly referred to as the pinhole, as viewed from the other camera. In the diagram below, the epipole labeled $C_1's$ *epipole* is the camera center of C_1. An epipolar line is a line through the epipole of one camera and a point on its viewing plane. A stereo-pair is two images of a scene, taken under the same conditions, where the only difference is a change of camera position. In simple stereo vision these epipolar lines run horizontally across imaging planes; due to the position of the stereo pair of images in the same plane, the epipoles of each camera are at infinity.

2.2.2 Stereopsis

Stereo matching is a quintessential problem in computer vision. It has been considered a hard and largely-open problem for many years. Numerous researchers

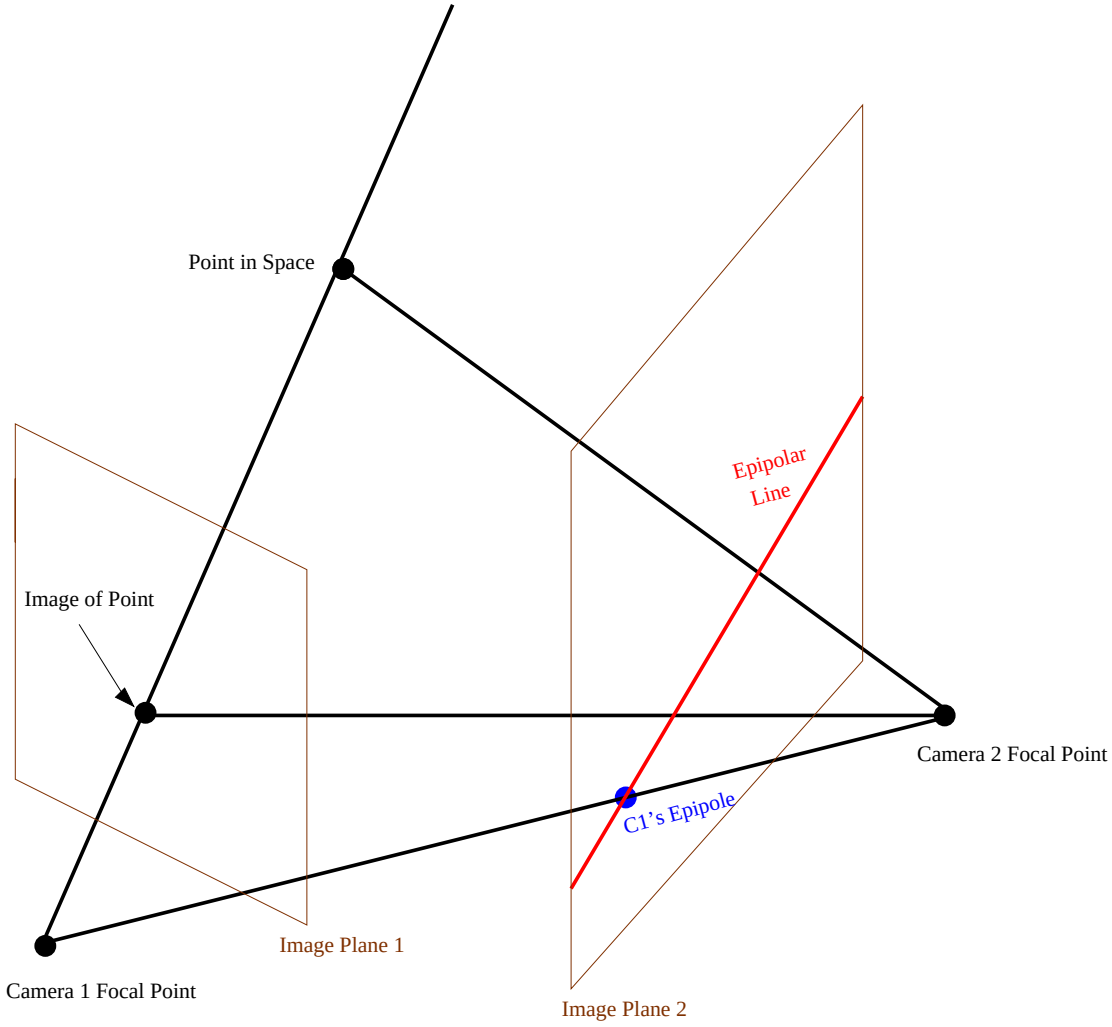

Figure 2.1: *An illustration of epipolar geometry. The image point viewed from the camera*$_1$ *can be located anywhere along the epipolar line shown in camera*$_2$*. Epipole*$_1$ *which is of camera*$_1$ *can be seen in the image plane of the camera*$_2$*.*

have dedicated a great deal of time to understanding it. Essential to the stereo matching problem is the concept of disparity. Disparity is defined as the change in location of a pixel along an epipolar line. The problem can be stated as *"given a pair of images of the same scene,* I_1 *and* I_2*, determine the disparity of each pixel in* I_1 *(change in location) relative to* I_2*, along an epipolar line"*. Once a disparity map is established between cameras, depth of objects in images becomes

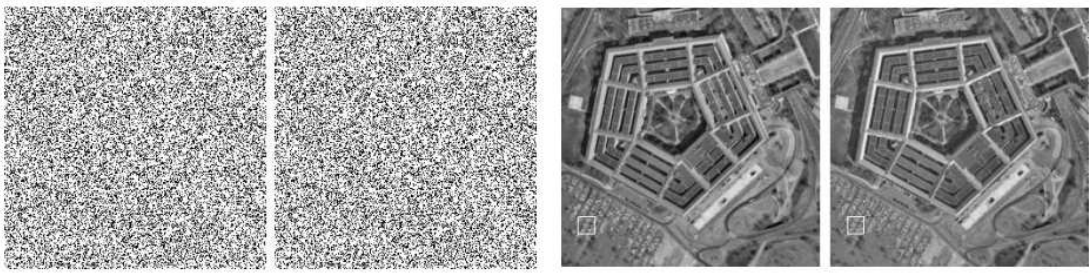

Figure 2.2: *On the left, a typical Marr-Poggio stereo pair. On the right a modern day stereo pair provided by Takeo Kanade of Carnegie Mellon University.*

a trivial calculation. However, vital to note that when working in a discretized environment, the precision of computed depth in these images is directly related to the physical distance between the cameras. The linear distance between the camera centers is referred to as the baseline of a stereo pair. The trade-off in stereo matching is that to have a high precision computed depth there needs to be a large baseline; however, as the baseline grows the correspondences are more difficult to determine. A correspondence is the matching pixel in one image with the other. This obviously motivates work to have an n way solution (using multiple input images) for multi-view stereo, so that a much greater and more accurate range of object location data can be obtained from images. Processing of stereo pairs was originally pioneered by Poggio [MP76] in 1976. It was first developed to be used on black and white noise images and therefore could be based on the assumptions of: continuity, compatibility, uniqueness and epipolar lines. Continuity meant that the disparity changed slow enough to guarantee that an adjacent disparity was a good first guess. Compatibility was an assumption of no illumination difference; in these binary images the black matched black and white matched white. Uniqueness inferred that there was usually, if not always, a unique match between the elements. Finally, the epipolar constraint assumed that all matches were along the

epipolar lines of I_1 and I_2. This iterative method has changed over time to accept more diverse sets of input images and to utilize different assumptions.

Takeo Kanade's work [OK93] offers an n way solution for stereo, which he bases on the assumptions of image intensities. An n way solution involves using two or more images to reconstruct three-dimensional information. This work's assumption is that all images have the same pixel intensities and therefore can be compared in this way, while still assuming the epipolar constraint. The overhead and processing time of this work is very large, due to the number of degrees of freedom allowed by multiple views. Previous works relied on comparing intensities of pixels. Follow-up work suggested that the intensities or other properties of the images be stated in terms of energy. Given a correct energy function, the solution should simply be the minimum of the energy function.

These techniques for solving stereo matching often relied on simulated annealing to find the minimum of this function. Annealing methods are iterative, step-wise gradient-descent functions, which often suffer the problem of being caught in a local minima, since this energy function is non-linear. Through repetition a "best minimum" can be found, yet there is no guarantee that the best minimization is actually the correct one. The computer vision group at Cornell University under the direction of Ramin Zabih has produced state of the art work on this subject [YB01] [KZ01] [DSZ00]. The energy minimization techniques described and implemented by this group provide vastly superior results to the other methods, and accelerate solution time. They use the idea of graph-cuts, known as the min-cut problem in computer science. The min-cut problem comes from algorithmic work on graphs. In a graph, a set of vertices and the edges between them, it can be useful to know the minimum number of edges that need to be removed to seg-

ment the graph into two graphs. This is known as the min-cut. Building off the original Floyd-Fulkerson Min-Cut Max-Flow Algorithm, Zabih's group developed an algorithm for multi-way cuts which segments a graph into multiple graphs. This multi-way cut algorithm can be employed to definitively solve the energy-minimization problem; it does this in less time than simulated annealing (or other methods). Figure 2.3 shows a comparision of these two techniques. Furthermore, it provides a guarantee of validity which other methods cannot. However, all these methods are still largely offline and take on the order of minutes to hours to run, which is inadequate for our purposes. There is recent work from The University of North Carolina [YP03] that provides promising images from real-time stereopsis using commodity graphics hardware, although they are still too rudimentary for our work. In the near future we will hopefully see a fully-functional, real-time stereo system with the potential to be integrated into larger systems.

2.2.3 Structure from Motion

The "structure from motion" problem can be formally stated as follows: given a set P of feature points in a set F of two-dimensional frames where each point is identified, track the object through new frames.[Ull79] From a solution to this problem it is then possible to recover the camera pose and the three-dimensional structure of the scene. These processes have been studied in computer vision for both orthographic and perspective projection viewing.[LH81] [TT94]. This work is heavily constrained and often assumes non-deforming rigid bodies, in which all tracked points can be seen in all captured frames. This work is a predecessor to the markered tracking methods used in the special effects industry, which will be covered in a subsequent section.

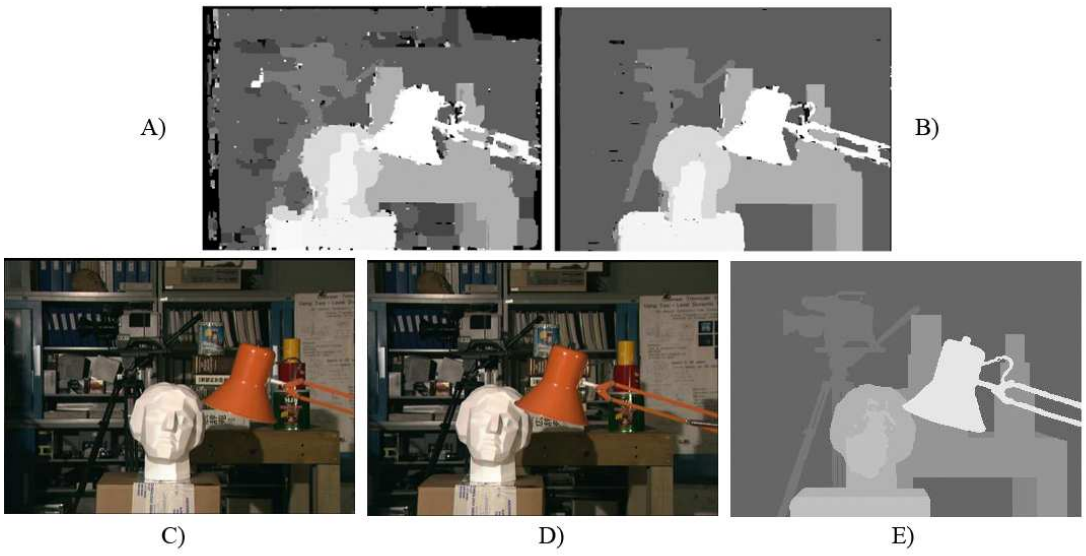

Figure 2.3: *This figure shows the results of two different vision algorithms simulated-annealing and multi-way cuts in A and B respectively. The orginal two images used as input are shown in C and D. The hand-calculated (using physical measuring equipment) ground truth is shown in E. The coloration of the black and white results indicates the depths of each pixel. Notice that in E the depths are consistent and non-infinite (black). While it is clear that the min-cut algorithm is better than simulated annealing in this case it still does not get a perfect solution, especially along the edges of an object.*

2.2.4 Voxel Coloring

Steven Seitz's work [SD99] [SK02] on Voxel Coloring offers a great deal of motivation to our work. It is closely related to stereo matching and structure from motion work and relies on much of the established knowledge from these areas. The Voxel Coloring system considers the problem of reconstructing a photo-realistic 3D model from images obtained from widely distributed viewpoints. They use an offline process to do this; it provides photo integrity; meaning their model and the input

images match when viewed from the input viewpoints. Furthermore, it proffers broad viewport coverage from an input set of images over the entire viewing space. The results are achieved by solving the *color reconstruction problem,* which they formulate in their papers. The broad overview of their technique is to discretize a scene into a set of voxels, then traverse them in a back-to-front order, looking for color correspondence in the input images. This technique provides good results for arbitrary viewing of a reconstructed object. Although offline and only supporting Lambertian-like objects, the high-quality results motivate studying their techniques.

2.3 Immediately Relevant Work

Presented here are the systems and papers that we utilized as a basis for our current implementation. In our discussion of stereopsis we attempted to define a standard for quality to which we will compare our results. However most solutions benefit from a large amount of processing time and we have constrained ourselves to work in real-time. We desire both quality and timeliness and this has led us to explore different concepts.

2.3.1 Visual Hulls

Laurentini [Lau94] describes a visual hull as "a geometric tool to relate the 3D shape of a concave object to its silhouettes or shadows." More formally the definition can be stated as "The Visual Hull of an Object S is the maximal object silhouette-equivalent to S, i.e., which can be substituted for S without affecting any silhouette." The visual hull is a tighter fit to an object than the convex hull of an object, yet still guarantees that the original object lies within its extents.

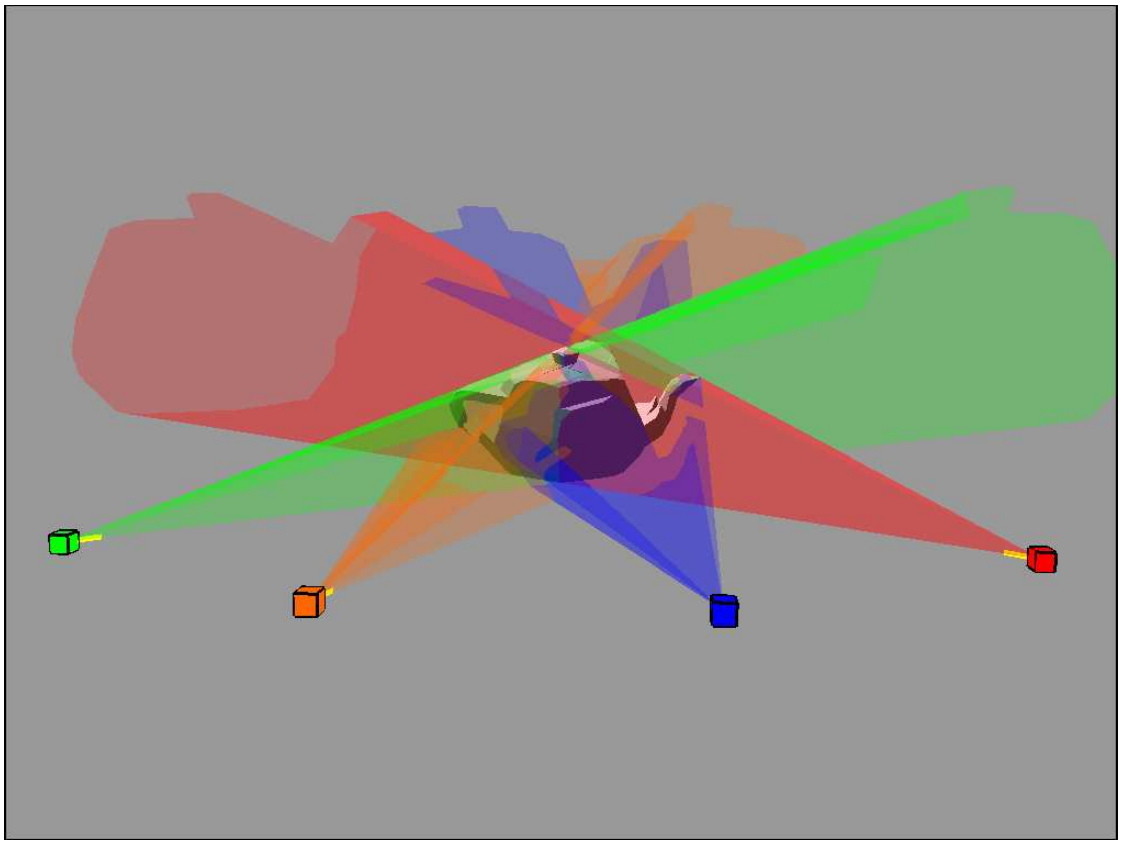

Figure 2.4: *This figure depicts the intersection of four silhouette cones that were used to generate a visual hull of a teapot.*

Figure 2.4 shows an example of a visual hull of a teapot.

2.3.2 Image Based Visual Hulls and Subsequent Work

The first work in the space of Image Based Visual Hulls arose in 2001 from the graphics group at Massachusetts Institute of Technology [MBR+00] [MBM01b] [Mat01]. This work is utilized as a starting point for a cluster of subsequent work at other institutions, including our own work. The first iteration of MIT's work [MBR+00] [Mat01] was not a completed reconstruction system, but in fact one that built a single image for each view based on the visual hull.

The Image Based Visual Hulls system captures video input from a set of four cameras and then uses an algorithm to generate the user-viewport based silhouette and a view-dependent texture map to composite into a synthetic scene. This system does not produce a mesh of a visual hull. Rather it produces a re-projected photograph; in essence this was a two and a half dimension solution. It provides one improvement over Jeremy Selan's [Sel03] work at Cornell; this is arbitrary viewing. Selan's system only allowed for live video that was being merged to be viewed from a fixed viewpoint. Matusik's work leveraged a multi-camera setup to permit viewing of the live video from any arbitrary position. While this improvement offers a novel views of the input scene without having to move the cameras; it offers no more photorealism than a traditional green screening application.

In 2002, the Polyhedral Visual Hulls work [MBM01b] implemented a system based on their previous work, which yields a full 3-dimensional mesh. This was the first fully real-time system to composite three-dimensional objects into synthetic scenes. The clear advantage of having a mesh to represent the captured object is that this work can be easily discretized into two parts: The first part is the image acquisition and generation of a visual hull; the second part is the more traditional computer graphics problem of scene generation given geometry, materials, and lighting.

Matusik's work contributed significantly to the community and has been utilized by a number of other very closely related projects. Followup works have appeared from Max-Planck [MLS03] and University of North Carolina [PSP03], as well as other research institutions.

Figure 2.5: *Real objects illuminated by the Eucalyptus Grove and Grace Cathedral lighting environments captured by Paul Debevec using the Image Based Lighting Techniques. [Deb02]*

2.3.3 Image-based Lighting

Paul Debevec's work [Deb98] [DWT+02] [Deb02] [DTM96] offers a glimpse at the possibilities provided by the utilization of modern computer graphics techniques and hardware while allowing offline processing. Image based lighting [Deb02] incorporates the idea of capturing the complete irradiant light scene at a given point. This work provides an extremely accurate construction of light placement in a scene. It will be critical to the lighting matching step that is covered during the forthcoming system setup chapter. The results that are shown using this method are far superior to other real objects illuminated by captured environments.

In 2002, Debevec's Lighting Reproduction Approach [DWT+02] went a step further in matching the lighting of real scenes on a light stage. He clearly identified the important imaging system properties that provide realism when compositing scenes: brightness response curves, color balance, sharpness, lens flare and noise. He also identified the important lighting elements that need a perceptual match,

these being similar shading, highlights, direct illumination and shadows.

His earliest work [DTM96] provided a great example of a usable, yet minimally taxing, user-driven geometric reconstruction system. This system aimed to render realistic architectural models and was informally known as Façade. This system was very useful for static scenes; however, since it required input from static imagery it did not support dynamic movement.

2.3.4 Video Flashlights

The Video Flashlights work by Sawhney, et. Al. [SAK+02] provides improved texture functionality over Matusik's Polyhedral Visual Hulls System. It harnesses the power of commodity graphics hardware to blend and overlay input from different camera scenes, so that there is a seamless transition between otherwise discontinuous imaging. Their techniques are able to matte textures onto three-dimensional objects, which are more realistic than the direct methods used in previous works. Their work is incorporated into the final system that we have produced, in order to reap the benefits of our graphics hardware.

2.4 Summary

This chapter reviewed the previous work related to producing composite images from synthetic and real imagery. The motivational work comes mainly from the Computer Vision community which uses mathematically time-intensive techniques to produce the best quality three-dimensional reconstructions. The Computer Graphics community's work provides functional examples of systems that attempt to accomplish similar goals.

Chapter 3

System Layout

The goal of our system is to merge live objects captured using video with synthetic imagery, in real-time. We create a high level pipeline capable of delivering, theoretically, 15 frames per second. This system provides the user with an interactive experience as they use our software.

Given the high level of computational complexity involved with geometric reconstruction, along with the goal of real-time results, the performance of our system is a high priority. Throughout this chapter, we focus on the design and implementation of our hardware and software system. We describe the difficulties which arise in trying to engineer a system that both captures geometry interactively and also merges that geometry in a believable manner with a synthetic scene. We justify the design choices that were made to assist in the future development of similar systems.

3.1 System Overview

The chain of events that our system performs can be decomposed into eight broad stages, each of which has its own set of design difficulties that need to be addressed. A graphical illustration of the segmented stages is shown in Figure 3.1. In Figure 3.2 we list some of the challenges associated with each of the respective stages. In stage one, each of the video cameras in our system must capture an image of the physical object that we want composited. Then, in stage two, the frames must be segmented into their foreground and background components. The next step, stage three, is to blur the segmented images. This removes small

holes and smooths out the noise along the foreground/background boundaries. In stage four the segmented image is found using a a contour finding algorithm. The process extracts the foreground silhouettes, which, later serve as input to the hull reconstruction algorithm. The frame and contour data is passed across the network from the client machines to the central server in stage five. This data is used in stage six to construct the polyhedral hull. In stage seven, the hull is textured with the images captured from the cameras. Finally in stage eight, the composite of the virtual environment and this live video is rendered to the screen. Using this approach, we can implement effects such as shadows and surface reflections between the real object and the virtual background.

The remainder of this chapter will be organized as follows. First, in Section 3.2, we will discuss our hardware setup, as well as the computing paradigm used within our system. Then we will briefly describe each of the major system modules referenced in Figure 3.1, with the exception of step six, the geometric reconstruction algorithm, which is covered in detail in Henry Letteron's thesis[Let04].

3.2 Hardware Setup

In this section, we describe the hardware infrastructure of which our system is comprised. Further, we will discuss the client-server paradigm as selected for our computing model, and how this architecture has affected our system. Figure 3.3 shows the hardware configuration of our system, and Figure 3.4 shows both the utilized and theoretical maximum bandwidth between each component.

Figure 3.1: *The diagram above depicts each stage of the image processing, geometric reconstruction and compositing process.*

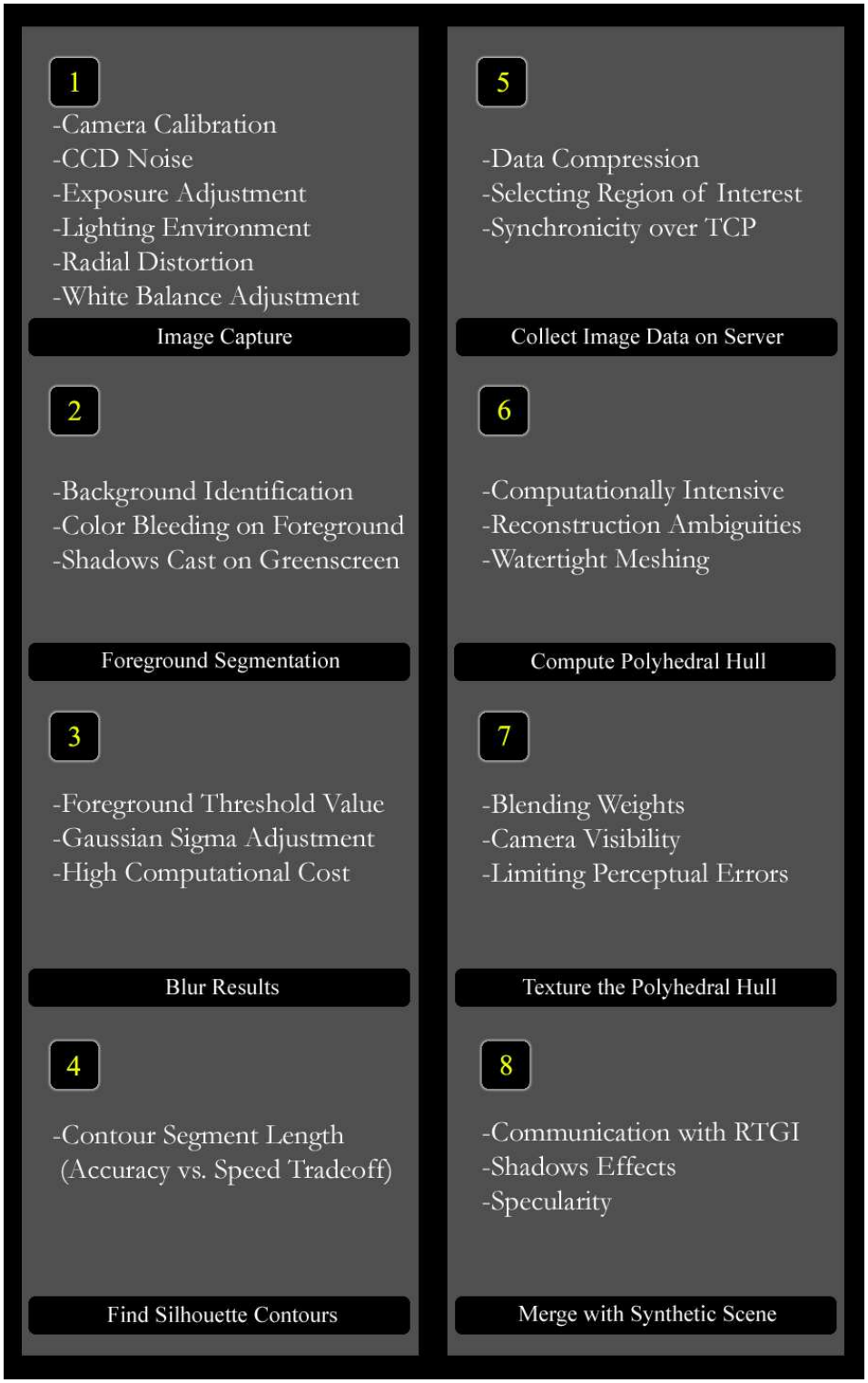

Figure 3.2: *The diagram above displays some of the challenges associated with each stage in the algorithm.*

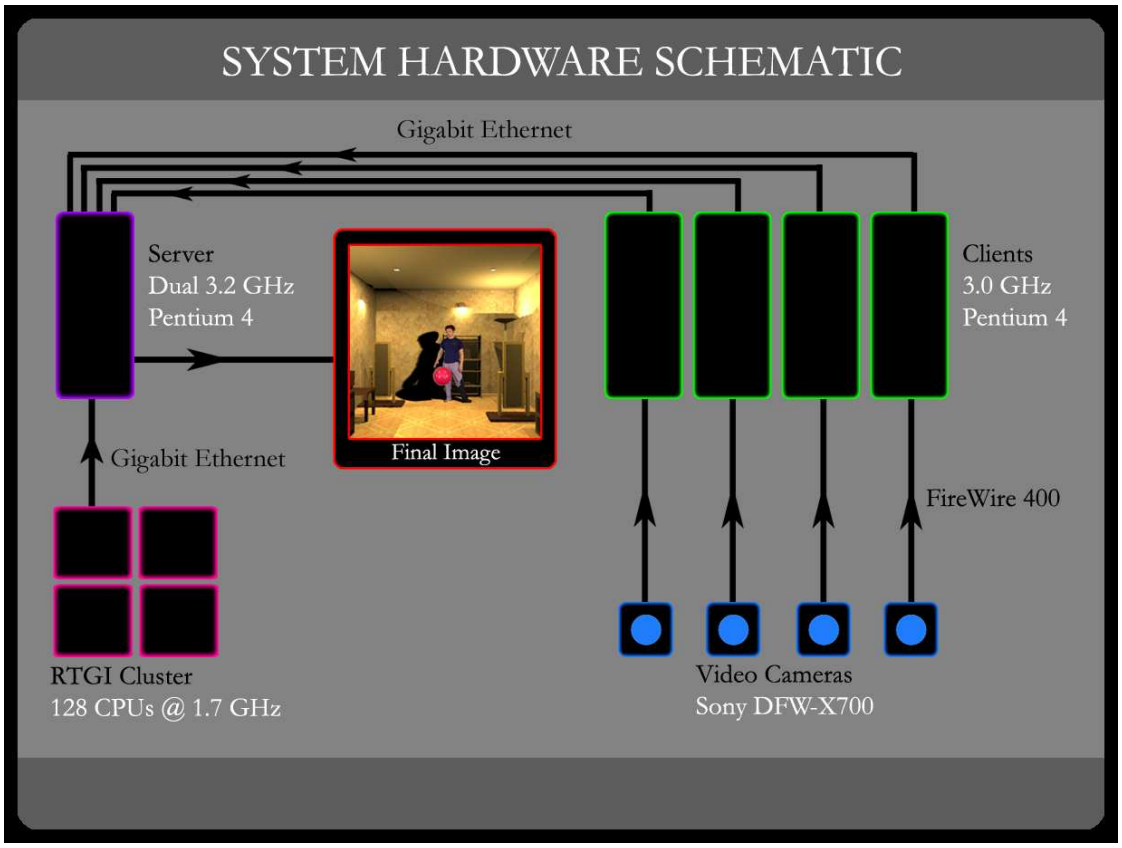

Figure 3.3: *A diagram of the hardware configuration in our system.*

3.2.1 Computer and Camera Specifications

We use four Sony DFW-X700 cameras to capture the foreground object. Figure 3.5 shows one of our cameras. The cameras have a half-inch CCD, and provide progressive scan output at 15 frames per second up to a resolution of 1024x768. Each camera uses standard c-mount lenses, coupled with 7.5mm fixed focal length lenses. At this focal length, we are able to capture a subject of approximately six feet in height from a distance of four meters. Each camera is linked to a separate client computer using the IEEE 1394 (FireWire) interface, supporting transfer speeds of up to 400Mbps. At 15 frames per second and a resolution of 1024x768 (three channel output), the maximum required bandwidth is 33.75 MB/sec, well

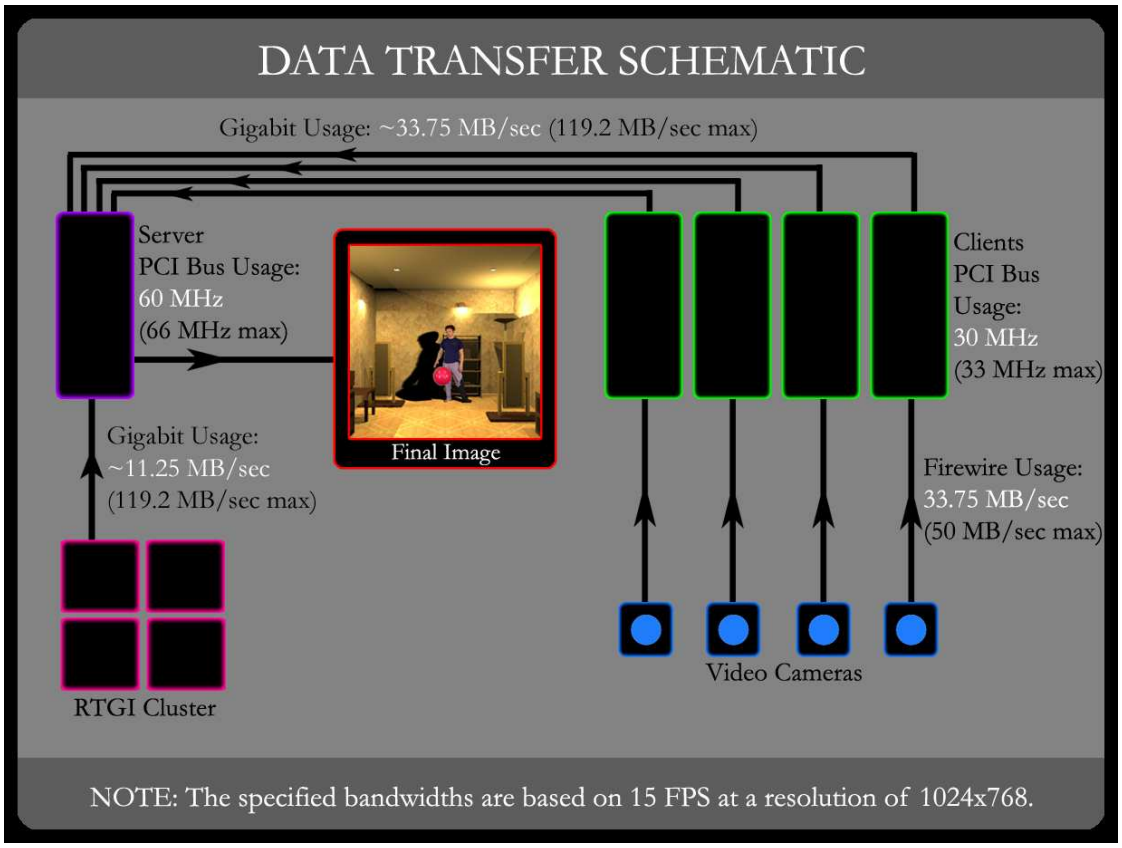

Figure 3.4: *A diagram of the utilized and theoretical data transfer rates in our system.*

below the upper limit of 50 MB/sec, even in the most demanding scenario. Each camera has an external trigger which allows for capture synchronization.

As previously mentioned, each camera is tethered to a separate client computer that performs the necessary image processing operations on the captured video frames. The computers are single processor 3.0 Ghz Pentium 4 Machines with 512MB of RAM and Gigabit Ethernet cards. These machines perform one of the parallelizable steps in our processing pipeline. Specifically they extract the foreground object from the image and find the line segments which define its silhouette. These tasks are highly computational in nature, thus making the CPU

Figure 3.5: *A DFW-X700 camera and the trigger circuit box.*

the most critical component.

The client computers are networked to the server using Cat 5E crossover cables which support Gigabit transfer speeds. The server has two two-port Intel 1000MT Gigabit server cards which allow the four client machines to each have a direct connection. The central server consists of dual 3.2 Ghz Pentium 4 Processors with 2GB of shared RAM and an NVIDIA Quaddro FX3000 graphics card. The server is responsible for the hull reconstruction, as well as the final hardware rendering and compositing, including shadow generation using advanced vertex and fragment shaders. In order to perform these tasks interactively, the server is required to have a high performance graphics board in addition to raw processing power. A representative image of our hardware configuration can be seen in Figure 3.6.

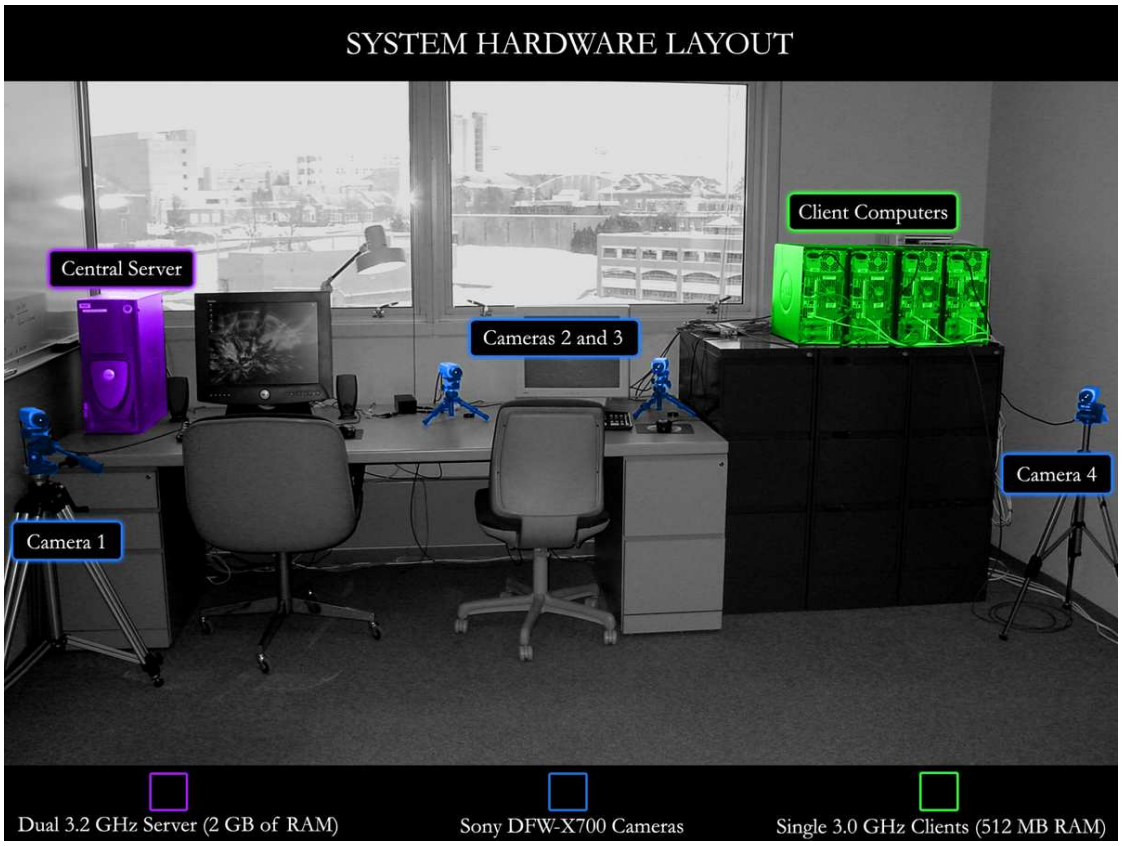

Figure 3.6: *The configuration of our system in one of the rooms we tested in. This diagram intends to show each of the individual components as well as the entire system.*

3.2.2 Client-Server Model

Many of the fundamental algorithms associated with geometric reconstruction and image synthesis are highly computational in nature. The goal of our project is to integrate these two areas into one encompassing system which is able to maintain interactive frame rates. This requires that we collimate as much of the work as possible. In order to achieve this parallelization, we implement a client-server model that distributes the workload across a networked array of computers. Each of the video cameras is attached to a separate client computer that performs the neces-

sary image segmentation and contour finding operations. After this processing is performed, the client computers send the silhouette data and video textures across the network to the server. The geometric reconstruction occurs on the server. At the same time as the client computers are sampling the cameras and matting out the foreground object from the greenscreen, the Real Time Global Illumination (RTGI) system is running on a separate cluster of computers for background image generation, and the server is generating the polyhedral model from the previous frame. By pipelining these three disparate actions, we attempt to maximize the throughput in our system, and minimize the time spent waiting for other processes to complete their task. Since this is a pipelined system there is an associated startup cost in processing, which results in a one frame latency.

3.3 Image Capture

As shown in Figure 3.1, the first step in our system is to capture an image of the foreground object from each of the video cameras. This section covers camera calibration, the synchronization of the cameras, and the control of the cameras' internal and external parameters to achieve the best image quality and cleanest foreground segmentation. We also describe the techniques used to maintain a consistent lighting environment for capturing foreground geometry.

3.3.1 Camera Calibration

A fundamental requirement of the reconstruction algorithm is knowing both the position and the orientation of the cameras in relation to each other within a globally defined reference frame. These are known as the extrinsic parameters of the camera. The intrinsic parameters, namely the principal point and focal length,

must be discovered through calibration as well. The principal point is the location where the principal axis intersects the image plane. The principal axis is the line passing through the camera center that is perpendicular to the image plane. To calibrate the cameras, we used the "Camera Calibration Toolbox for MATLAB"[1]. This is illustrated in Figure 3.7 which shows both our captured checkerboard images and an example of the Matlab routines we use to process them. To begin the calibration process, we placed a checkerboard pattern within the scene such that each of the four cameras had a clear view of the entire surface. We then captured a frame from each camera, and used those as input to the calibration routine. The calibration procedure, which uses Zhengyou Zhang's technique [Zha00], returns the grid reference frame with respect to each of the four cameras' reference frames. Since we desire the relationship that exists between the cameras, we convert each camera frame to being dependent on the grid's frame, so that the grid contains our global orthonormal basis. More details on these procedures can be found in the camera calibration section of the appendix, Appendix A.

3.3.2 Trigger Synchronization

It is imperative that each of the input images used for hull generation is captured at the exact same instant in time because the reconstruction algorithm operates by taking the intersection of the "silhouette cones" (Figure 2.4). A silhouette cone is defined by an apex, which is located at the camera center, and the extrusion of the polygonal object silhouette away from the camera center to infinity. If the images are captured at even slightly varying instances in time, then the silhouettes

[1]The Camera Calibration Toolbox for MATLAB can be downloaded at the URL: http://www.vision.caltech.edu/bouguetj/calib_doc/

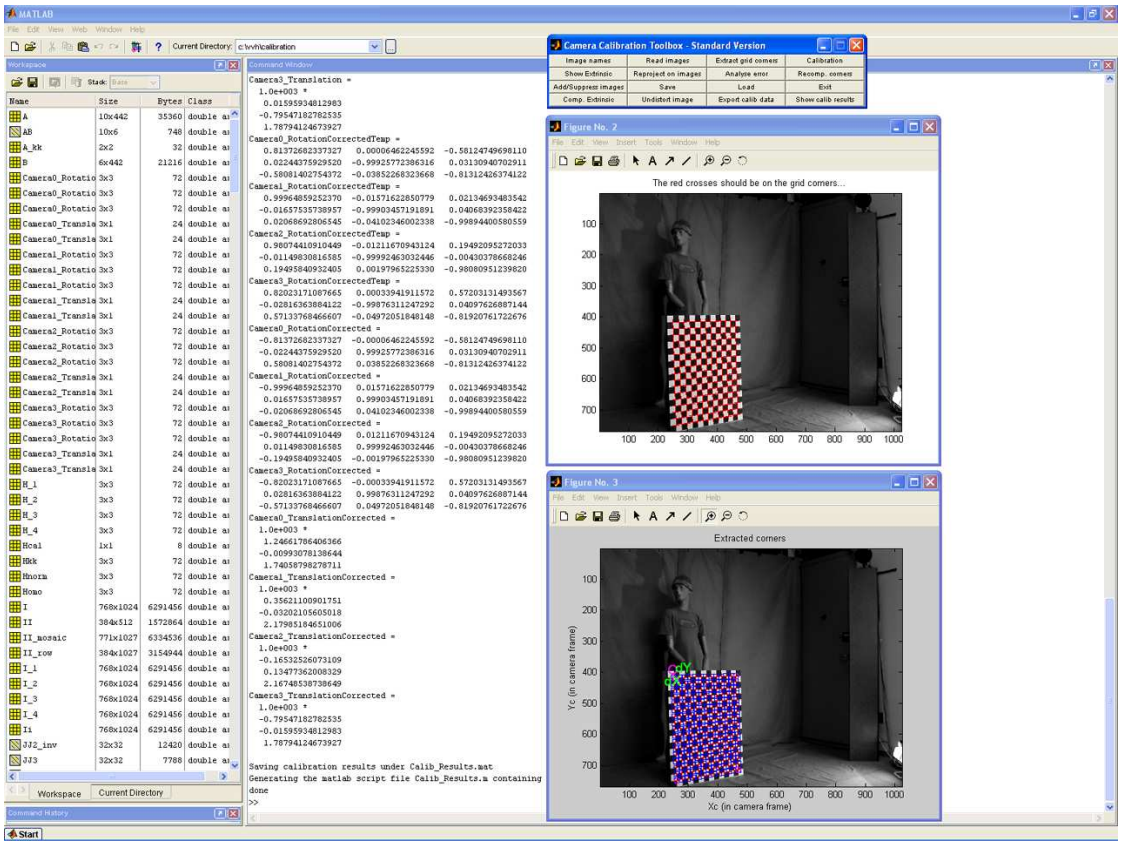

Figure 3.7: *An example screenshot of the MATLAB camera calibration toolbox interface.*

could potentially represent very different poses of the foreground object (assuming the object is moving). The intersection of these extruded silhouette cones would not accurately reproduce the original model at any of the different time steps, but would most likely appear as indiscernible noise, containing geometry only where the poses happened to overlap in space. To prevent this degenerate scenario from occurring, the server is responsible for triggering each of the cameras simultaneously in the main loop of the program, ensuring that the sampling is consistent at each time step. The server drives the trigger circuit by outputting signals on the serial port. For implementation details concerning our use of the serial port as

well as our trigger circuit design, refer to Appendix B.

3.3.3 Use of DirectShow for Camera Control

In order to establish software communication with the video cameras, we took advantage of the DirectShow interface, a component of Microsoft's DirectX 9 multimedia suite. DirectShow allows for the configuration of a filter graph to manage the flow of data from the camera capture device to the user's program. DirectShow also provides an interface for setting such camera parameters as white balance, exposure, video capture format, and the active state of the external trigger. Our camera control software, which runs on each of the four client computers, uses the DirectShow interface to adjust the color balance of the incoming image in order to achieve optimal greenscreen results.

3.3.4 Whitebalance Control

To improve the results of the image segmentation process the whitebalance on the camera should be adjusted. By slightly strengthening the red and blue channels it is possible to remove the excess color bleeding from the greenscreen that occurs on light-colored surfaces. Alternately, by decreasing the red and blue channels it is possible to compensate for shadows on the backdrop which leave parts of the greenscreen marked as foreground. Decreasing the red and blue channels will make the greenscreen surface appear more green, and thus can lead to less noise and better results. Figure 3.8 illustrates these effects. Making this trade-off in color balance is a subtle art, as it is important that the captured image appear as natural as possible while remaining easy to segment cleanly. To maintain consistency, the program saves a configuration file that documents the last used whitebalance

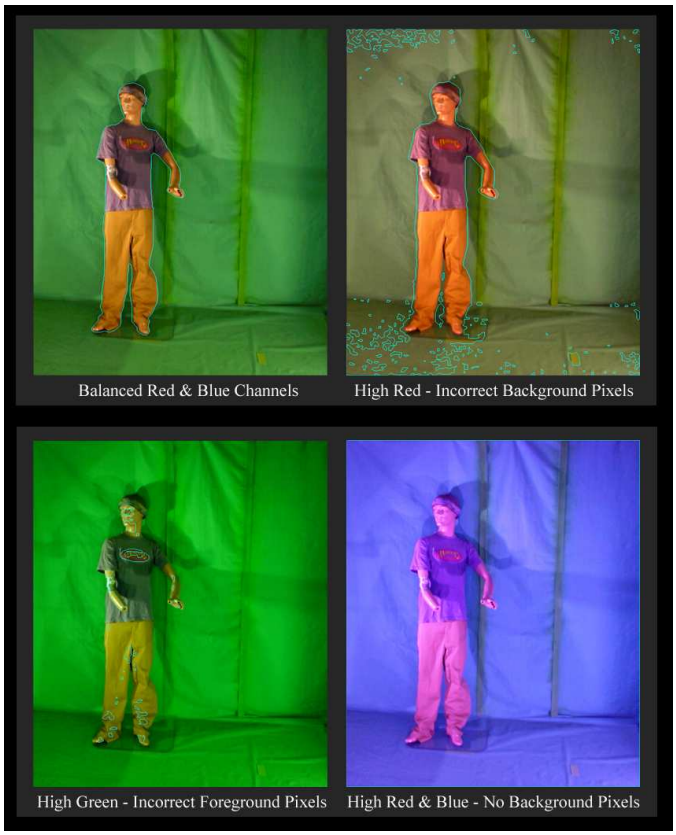

Figure 3.8: *Examples of color-balance adjustment. Clockwise from the upper-left we see examples of common adjustments and their effects on the scene. Each picture illustrates the effect which it is labeled.*

values. These settings are then re-loaded the next time the program is started. Similarly, values for the matte strength, blur threshold, and other camera-related parameters are also stored and loaded.

3.3.5 Scene Lighting

The lighting of the foreground object and the greenscreen is an important factor in determining the quality of the image segmentation and the resulting silhouette contours. Ideally the greenscreen would be placed at infinity, so that there would

be no interactions between it and the foreground subject. Allowing the subject to be too close to the greenscreen often results in color bleeding, where parts of the foreground object, most notable the edges, take on a greenish tint due to light that is first been reflected off the screen and then bounced off the subject. Another situation that can lead to problems is when the foreground object casts shadows on to the surface of the greenscreen. This can be problematic because the darker shadowed regions of the background are often misinterpreted as foreground. The way we addressed these issues was to place several 500 watt halogen lamps in a semi-circle configuration around the object acquisition area. Our goal was to simulate a diffuse environment where light was coming from all directions, and thus eliminate any hard shadows.

3.4 Image Segmentation

After an image has been captured from each of the video cameras, the next step is to segment the foreground from the background. In this section we will cover the initial foreground matting, as well as the subsequent Gaussian filtering operation that we use to eliminate high frequency noise.

3.4.1 Foreground Matting

After the cameras have been triggered and a frame returned, the next step is to segment the image into its foreground and background components. We borrow the method employed by Selan [Sel03] for performing this operation. If a pixel's green channel value is higher than the maximum of the red and blue channels by a preset margin then the pixel is marked background. This amount, termed the "matte strength", is a variable that can be adjusted in the software. If, conversely,

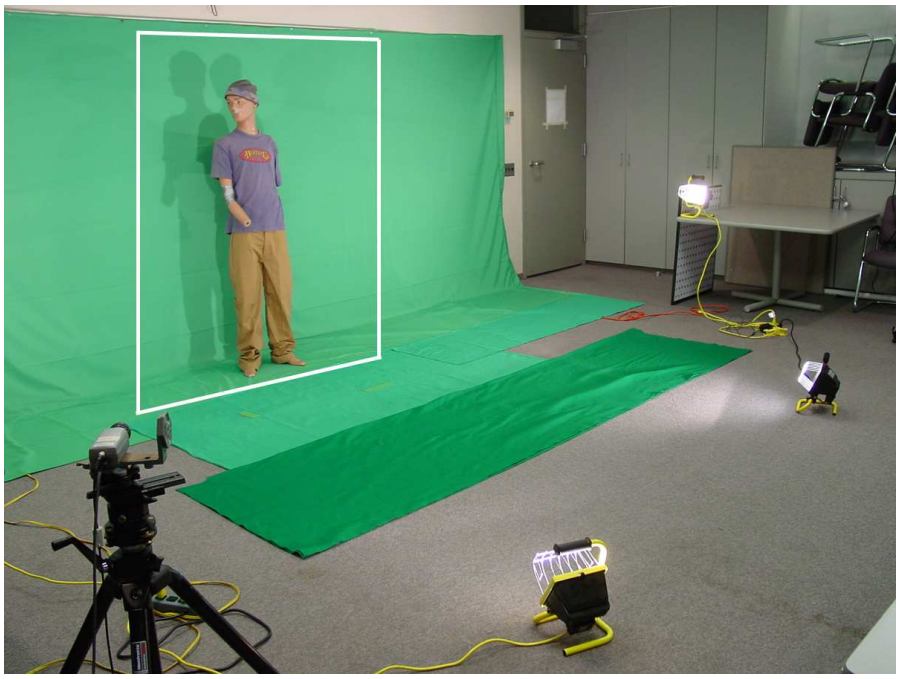

Figure 3.9: *The figure above shows the greenscreen that was used for segmenting out foreground geometry.*

the green channel is not significantly larger than both the red and blue channels, or if the pixel is extremely bright, with a combined channel value over some predefined threshold, then it is marked as foreground. When performing segmentation, we are only concerned with the region of the image that corresponds to the greenscreen (a subset of the entire image). The bounding box which defines this region of interest, or ROI, can be set in our software. Any pixel which falls outside this area is automatically marked as background and set to black in the segmented image. Figure 3.9 shows the greenscreen backdrop for our image acquisition area.

3.4.2 Gaussian Blurring and Thresholding

The segmented foreground image is often plagued by noise, due to the imprecise nature of the scene lighting and the image sensing device, as well as the non-

homogeneous color of the greenscreen. As discussed in Section 3.3.5, areas of the foreground that suffer from heavy color bleeding often have a greenish tint, and are therefore erroneously marked as background. Conversely, areas of the background that receive heavy shadowing are often not recognizably green, and are thus deemed foreground pixels. Inaccurately labeled pixels are also common where the greenscreen meets the floor as the corner is usually dark, and around perturbations in the greenscreen fabric or areas where the lighting undergoes sharp transitions. We attempt to minimize these problem areas through physical means but it is difficult to eliminate them completely. We also rely on software techniques to aid in the image segmentation process. One way we alleviate noise is by convolving with a Gaussian filter over the segmented binary image. Our Gaussian filter implementation, which takes advantage of dynamic programming techniques, makes multiple passes over the image in both the horizontal and vertical directions, each time convolving the image with a box filter. We have found empirically that a Gaussian sigma value between two and four works relatively well for removing the noise in our captured images.

After the image has been blurred, each pixel has a gray-scale value between 0 (black) and 255 (white). The next step is to perform a threshold operation, such that pixels which contain a value above the threshold are marked foreground, or white, and those pixels with a value below the threshold are differentiated as background, or black. This threshold value is a variable in software, and can be adjusted as necessary for optimal segmentation.

3.5 Contour Finding

The fourth stage in our system is to find the contours that approximate the object's shape. This section will discuss how we recover the silhouettes from the segmented images, and a noise elimination approach that we use to prune away unwanted contours.

3.5.1 Polygon Approximation

After the image has been segmented into foreground and background regions, the silhouette contours that define the foreground object are extracted. This is done using Intel's open source computer vision library, OpenCV. We use the routine *cvFindContours()* , which, uses an algorithm similar to that of marching squares, in order to find and return the contours in a binary image. The contours that the algorithm recovers are initially very fine in resolution, such that each edge only spans neighboring pixels. To adjust the granularity of the contours, stringing together short edges in order to produce more representative line segments, one can leverage the function *cvApproxPoly()*, which, takes as input the original contours and a constant that denotes the desired level of contour resolution. The higher the constant, the more coarse the final contours will be. A value of zero returns the original contours with no change in granularity.

There is a trade-off to be made when using this routine. The higher the value, the fewer the line segments there will be. This will result in less data, lower network transfer times, and a quicker hull reconstruction process. However, the disadvantage is that the geometry is less refined. For example, when finding the contours of a hand, if the polygon approximation factor is set too high, the individual fingers

will be lost and the hand will appear as a single polygon. If the approximation factor is set too low, then the contour definitions will be more detailed but the running time of the hull intersection routine will be much higher. This is a direct result of the fact that there are now many more surfaces on which to perform polygon intersections. The goal is to find a balance such that the contours are of a high enough resolution to generate visually pleasing models and yet low enough to maintain interactive frame rates. One proposed solution, employed by Matusik et Al [MBM01a], is to have the approximation factor adjusted on the fly in software. The contour resolution is decreased automatically when the program slows down, and increased when there are extra cycles to devote to the mesh generation process. In our system, the variable is user driven, as opposed to software controlled, and can be adjusted on the fly to match the user's current desire.

3.5.2 Noise Elimination

Despite our best efforts to eliminate noise at the segmentation stage, there are still occasionally incorrectly marked pixels that bleed into the contour finding routine. To counter this, we automatically throw away any contour that consists of three or less edges. No reasonable foreground object would be so nondescript, and thus we have found this to be a relatively successful way of removing background noise from the contour detection process.

3.6 Data Transfer to Server

In order for the server to reconstruct, and later texture, a model of the foreground object, it must acquire the necessary contour and image data from each of the client machines. This is the fifth step in our system overview diagram, Figure 3.1,

and will be the topic of this section.

3.6.1 Network Protocol and Region of Interest

After the silhouettes have been extracted from the segmented image, a buffer is constructed that stores all the relevant data the server will need to compute the final polyhedral hull. Instead of sending the entire video frame to the server, which is later used for texturing the constructed hull, we only send the region that contains the foreground object. By minimizing the amount of data transferred between the client and server, the transfer times and the bandwidth consumed are also optimized. To determine the minimum bounding extent of the foreground object, we iterate through the line segments which define the contours and keep track of the minimum and maximum (x,y) points. These two points constitute a bounding box that encompasses any pixels potentially required during the texturing stage. The pixels inside this region of interest, along with the contour definitions, are then sent across the network. All our network communication is done over sockets, using the winsock library that comes standard with the Windows operating system.

3.6.2 Use of Compression

In order to reduce the transfer times between the client computers and the server, we experimented with compressing the data before sending it over the socket. We used the zlib compression library for our testing [2]. Unfortunately, the time required to compress and decompress the data was more than the time saved in transmission. This may not be true for all forms of compression (for example the JPEG algorithm might prove faster), or for all data set sizes. However, we have

[2]The zlib library can be downloaded at the URL: http://www.gzip.org/zlib/

currently settled on sending the data in its raw form.

3.7 Texturing the Polyhedral Hull

After the hull has been reconstructed the next step is to texture the model using the original video frames. In this section, we briefly introduce our initial naive texturing approach. A full discussion of the the naive approach and the more advanced texture blending technique, is included in Chapter 4.

3.7.1 Naive Texturing Method

The source textures for the polyhedral hull are always comprised of the original video streams. In the simplest case, the process of selecting the optimal video image for texturing a particular hull surface reduces to a question of visibility and orientation. In the best case scenario, each polygon to be textured corresponds directly to a pixel or region of pixels from the set of video images. It is unlikely, however that this will be the case and so we therefore attempt to find the nearest match. We quantify the correctness of a match in the simplest case as smallest dot product between the viewing vector and the surface normal. The dot product indicates the cosine of the angle between the these two vectors. In the ideal case, the surface normal of the polygon has a direct match, and the vectors are anti-parallel resulting in a dot product of zero. We apply a simple visibility test for each camera, once this has been done, we can then select the camera that has the best viewing angle, using the aforementioned technique. Figure 3.10 shows an example hull surface and the camera that this naive algorithm would select for texturing that face.

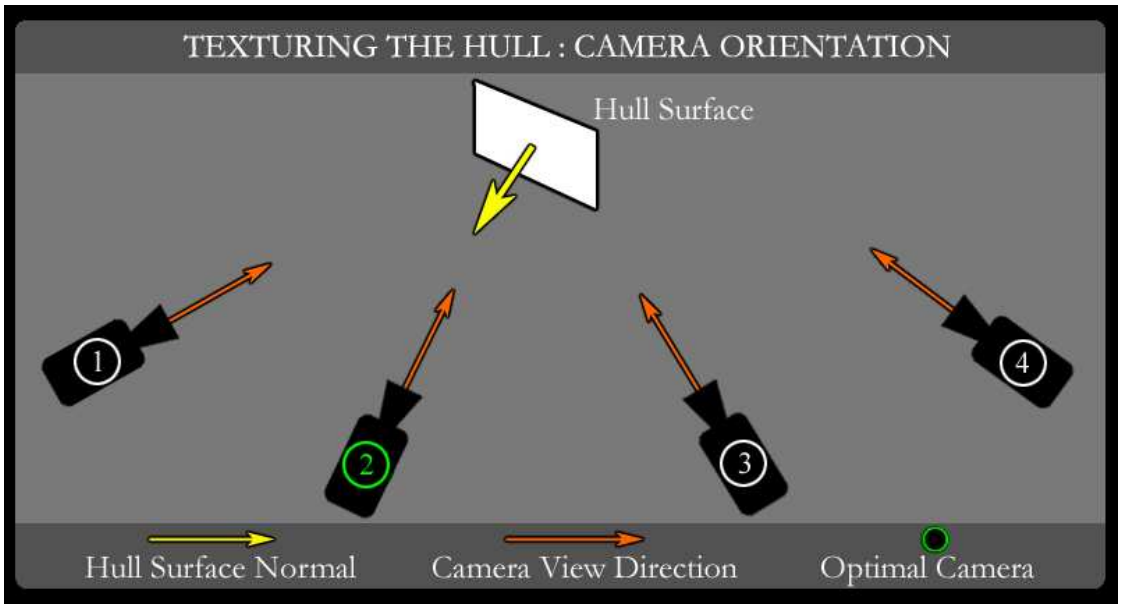

Figure 3.10: *The optimal camera for texturing a hull surface is the one whose viewing vector has the most opposing angle to the surface normal. In the diagram above, Camera 2 would be the best choice, as it has the most direct view of the face.*

3.8 Foreground and Background Compositing

The final stage in our system is to merge the reconstructed foreground object with the background environment. It is at this point that we add in global illumination effects, such as shadows and reflections, to enhance the plausibility of the composite image. This section will focus on the method by which we generate our background images, as well as the compositing process. The shadowing technique that we developed for our system is discussed in detail in Henry Letteron's thesis[Let04], and the implementation of reflections is covered in Chapter 5.

3.8.1 Background Image Generation

RTGI is the Program of Computer Graphics at Cornell's "Real Time Global Illumination" system. It is a ray tracer that is designed for both interactive walkthroughs and the rendering of static scenes. The RTGI system can render in many different modes, each having varying levels of quality and performance; however, these options are all transparent to our system. As far as our software is concerned, RTGI is a "black box" that feeds it background images for compositing with our captured geometry. RTGI can be run in one of two modes, on a single client computer or in walk-through mode, which takes advantage of the PCG's 128 CPU cluster[2]. When running in walk-through mode, the pixels are distributed among each of the machines, so that you can add secondary effects such as indirect lighting while still achieving interactivity. Although the RTGI system can handle an arbitrary number of light sources, our compositing software currently only supports hardware shadow generation for the primary light source in the scene. This allows us to maintain user interactivity, as implementing hardware shadows from multiple light sources requires additional rendering passes and texture lookups during the shadow generation stage. As a result, we primarily deal with background environments that have a relatively small number of lights.

3.8.2 Information Sharing

There are several parameters which need to be synchronized between the RTGI system and our software. These include the camera position and orientation, the primary light source location, and the current background model used for composit-

[2]The Cluster is 64 Computers with Dual 1.7Ghz Processors and a 1GB of RAM per machine

ing. It is important to keep the cameras synchronized between the two systems, so that if the user moves the camera within the RTGI framework, thus changing the orientation of the background model with respect to the viewer, the captured foreground geometry will be viewed from an identical pose. If this consistency is not maintained, the alignment of the foreground object with respect to the background scene will be constantly changing – a highly disconcerting visual effect. Therefore, in order to guarantee compatibility, the RTGI camera position, view direction, up vector, field of view, and aspect ratio are transmitted to our software each frame. Similarly, the main light source in the scene must also be in the same position on both ends of the system. The RTGI environment allows the user to modify the light positions on the fly, and consequently our software needs to be made aware of such changes. This ensures that the hardware shadows cast by the foreground object will match the rest of the shadows in the background image. Unfortunately, as it is beyond the scope of this project to attempt to re-light the captured video frames, any changes to the scene lighting will not be incorporated in the textures used to shade the foreground geometry. Thus the user must maintain a plausible lighting environment in which to insert the captured model. RTGI is also responsible for which background model is currently in use, and it needs to alert our software when a model transition is being made.

3.8.3 Hardware Compositing Process

When a background image is received from the RTGI system, it is loaded into the color buffer on the graphics board. Our software stores a local copy of the scene model, and the next step is to render this geometry into the depth buffer. While the scene geometry is being rendered, the color buffer on the graphics card

is disabled for writing. This prevents the ray-traced background image, which we previously loaded into the color buffer, from being overwritten with a hardware generated version. Then the color buffer is re-enabled for writing, and the captured foreground geometry is drawn. Because the depth buffer already has the scene geometry at this point, the depths are composited correctly.

In lieu of having our program render the scene into the depth buffer, we also experimented with RTGI forwarding along the depth data for the background image. Since the ray tracer has already generated a depth value for each pixel in the scene, it would seem logical to try and reuse this information by passing it over the network to our reconstruction server. However, we found that the time required to transfer the depth data across the network and load it into the z-buffer on the graphics board was significantly longer than it took to render the scene in hardware. Furthermore, a copy of the scene model is required on the reconstruction server for rendering shadows.

When performing shadow generation in our system, one needs to be able to render the scene from the point of view of the light source, thus requiring a local copy of the scene geometry. This might prove to be computationally prohibitive when dealing with highly complex scenes, for example when there are multiple polygons that exist within the space of a single pixel. Should this scenario ever arise, it may prove advantageous to revert back to a system where RTGI passes over the depth data for the generated background image, and then use a different approach for shadowing. This would eliminate the need for the compositing software to maintain a local copy of the scene model and would achieve a greater separation of complexity.

3.8.4 Object Positioning

The software which we have developed allows the user to rotate, translate, and scale the captured geometry so that it fits correctly within the scene. Exact positioning is necessary in order to create a convincing composite. Furthermore, the scenes in which we are inserting the subject are often modeled using arbitrary scales, uncorrelated to any physical units. In order to scale the model without also affecting its positioning, we first determine the center of the object. We implemented two different methods for computing the center of the object. The first approach is to use the average position of all the points which define the object's hull, which approximates the centroid of the object. The second approach is to use the center of the bounding box which surrounds the geometry. The latter method has proven to be the better of the two, as the exact positioning of the vertices which define the hull are constantly changing due to slight variations in image intensity and numerical imprecision during the reconstruction processes. Using the former method, the centroid shifts slightly each frame, and consequently the model skips around within the scene. This motion is highly disconcerting, and detracts from the realism. The bounding box method is less affected by the inherent noise which plagues the individual vertex positions. The temporal variation is much less, and consequently the jittering of the object within the scene is also drastically reduced.

Finally, to position the foreground object within the scene, it is first translated such that its center is situated at the origin. The object is then scaled up or down to the user-specified size and then translated back to its original location.

Chapter 4

Texturing

4.1 Introduction

In pursuit of our overall goal – to merge live video with synthetic imagery – we add a texture to the polyhedral hull. This chapter addresses the process by which this texturing is performed. We select the best data from the video streams to achieve results that look close to perceptually correct from any viewing angle.

The texturing of the polyhedral hull is the penultimate stage of the reconstruction system. At this point we have segmented the foreground from the background and used our knowledge of epipolar geometry to reconstruct a polygonal mesh (Figure 3.1). The resulting polyhedral hull is a three-dimensional shape defined by a set of triangular polygons. By adding a texture, we transform this set of polygons into a fully-defined textured surface representation, so that it can be composited into the scene. Figure 4.1 shows how textures add substantial realism to the final composite image.

4.2 Basic Texture Mapping

In computer graphics, textures are often added to geometry to give the impression of high-definition detail in a final image. By adding a texture to a simple mesh, it is possible to obscure tessellation boundaries and other visually disturbing artifacts that arise from simplistic modeling. Furthermore, it provides an alternative to tedious or often impossible modeling tasks. The basic texture mapping process is illustrated in Figure 4.2.

Figure 4.1: *An example composite with and without texturing. (A) Without texturing, the polygonal hull can only be displayed as a set of polygonal surfaces, flat-shaded polygons in this case. (B) With texturing, the coarse geometry of the polyhedral hull is masked yielding a believable composite image.*

The first step in this process is the capture of a texture. This can be accomplished in a number of ways, including photographing real-world materials, scanning previous art or documentation, or mathematically generating a procedural texture, such as a fractal. For photographic textures, to preserve the highest quality throughout the entire process, the texture should be captured with the viewing direction orthogonal to the overall surface of the texture. Furthermore the source textures should be captured from planar surfaces as in the brick example shown in Figure 4.2.

In order to ensure uniform sample density across a captured texture from a planar surface the distance D from the camera to the surface must be large. This ensures that the angle between the viewing vector of the camera and the surface

normal of the texture sample is constant. If the distance is too small the sampling is not uniform. In the case where the texture is built utilizing a mathematical procedure, the resulting texture should reflect this orthogonal capture process. We refer to this captured or generated texture as the *source texture*.

The second step in the standard texturing process is to specify the relationship of the source texture to the surface of the object. This relationship defines a mapping between all the polygons on the three-dimensional object's surface in the XYZ coordinate system and the UV coordinate space of the source textures. Once this relationship is established, a viewing matrix can be applied to transform our textured object definitions to a representation in image space.

In state-of-the-art compositing systems there is a single video camera which is utilized. This camera captures the object as a two-dimensional "billboard" to which the texture is applied [Sel03][1]. Since the virtual camera view is restricted to the fixed position and orientation of the physical video capture camera and hardware, it can accommodate precisely this type of texture-mapping process and the resulting composite image is easily generated.

Our system is different in the sense that it is view-independent and three dimensional. We replace the simplistic billboarding with a texturing process to accommodate the three dimensional mesh generated during the reconstruction stage of our system. The texture or raw-video data is mapped onto the reconstructed object during this process, which is detailed during the remainder of this chapter. Because the reconstruction process is split from the texturing process we have two different types of cameras in our system: a *reconstruction camera* which captures

[1]In Selan's '03 work the silhouette of the object was super-imposed on a translucent quadrilateral. This quadrilateral defined the region of display and computation, not the silhouette.

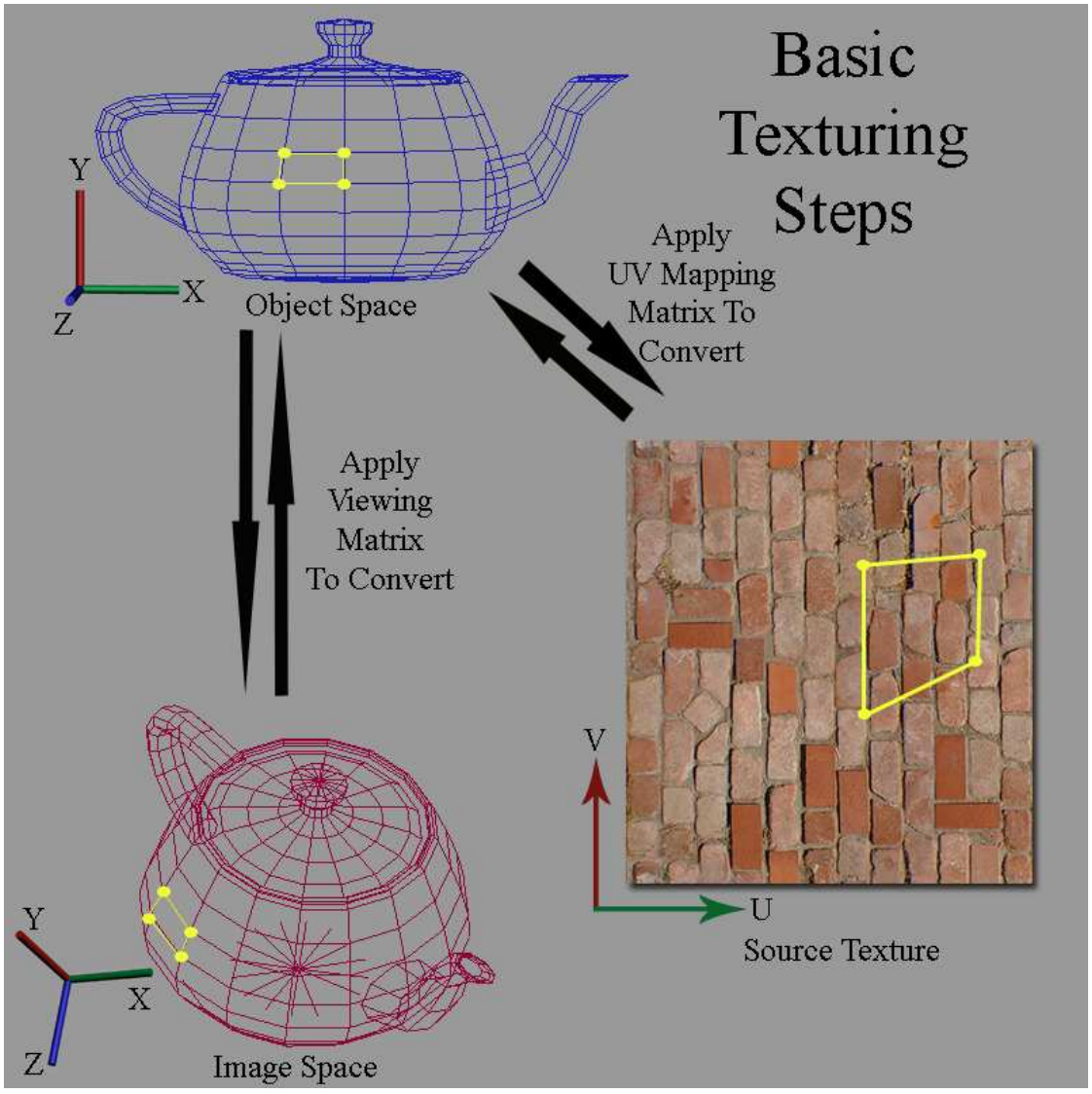

Figure 4.2: *An example of basic texture mapping. A source texture has been captured from a camera that lies orthogonal to the texture surface. When mapping from the UV coordinate space of the source texture to the object space coordinates a UV mapping matrix is applied. The Viewing matrix is applied to translate, rotate, and scale the object to its viewing position in world space. Both the UV Mapping and Viewing Matrices can be inverted to perform these operations in the opposite directions. In practice both matrices are concatenated and the texture is then sampled using standard mip-mapping hardware.*

raw video and a view-independent end user camera, the *viewport camera*, which is used to view the final reconstructed mesh with textures applied.

4.3 Our Texturing Problem

A major difference between our system and previous state-of-the-art compositing systems is that we obtain three-dimensional geometry so that the resulting textured polyhedral hull can be placed in a virtual three-dimensional environment, interacting with the virtual geometry and lighting by casting believable shadows and reflections. Furthermore, this system exhibits view-independence, allowing the end-user total flexibility to move the virtual camera to arbitrary positions within the synthetic environment. By using our polyhedral hull and by offering an unrestricted choice of viewing angles to the end-user, we confront a unique set of problems which must be addressed to successfully achieve our goals.

First, the reconstructed hull is larger than the original object, and our reconstruction cameras do not cover every viewpoint. This results in missing texture information. Second, the source textures captured from the video cameras are generally poor. They are neither orthogonally captured, nor are the source textures' surfaces planar. To alleviate these problems we utilize multiple captured source textures, one from each of the n reconstruction cameras. Obviously some camera positions will provide better source textures than others for each reconstructed polygon of our polyhedral hull. This however presents another dilemma: how to choose which of the source textures to use and how to interpolate smoothly between the textures for arbitrary viewing positions. Each of these problems and our implemented solutions will be discussed in detail below.

4.3.1 Larger Reconstructed Object

During the reconstruction phase of our system we use a shape-from-silhouette algorithm to generate our polyhedral hull. This hull represents the extents of our object. It is a tighter fit than a bounding box, and even a convex hull, but it does not capture the full complexities of the original object. The result is a coarse geometry that has a slightly larger volume than the original object. Theoretically, the best case scenario yields a polyhedral hull that is the exact volume of the original but it has been demonstrated in our work that this theoretical ideal is not achievable in real-life scenes [Let04].

The inequality between the reconstructed and original object's volume causes problems when we texture our reconstructed object. The source textures used during the texturing process are taken from the reconstruction cameras. Each texture is a two-dimensional representation of a portion of the original three-dimensional object's surface. However, our reconstructed mesh has a larger volume and therefore a larger projected surface area than the actual object. Consequently, the source textures do not map directly to the surfaces of the reconstructed polyhedral hull (Figure 4.3).

A second reason why our reconstructed object is slightly larger is the manner in which we prepare our data for reconstruction. Through a chroma-keying process we segment the foreground object from the background image. During this segmentation process a Gaussian filter is applied to remove noise from the boundaries of the object; this process however slightly enlarges the boundaries of the silhouette. Since these silhouettes are slightly bigger than the original object's silhouette the resulting reconstructed object is slightly larger. In order to provide a perceptually accurate final composite we must compensate for the differences

between the larger reconstructed hull and the smaller textures.

4.3.2 Missing Texture Information

There are two cases where texture information is not available for our polyhedral hull. First, since we do not have full coverage of the original object from our reconstruction cameras there are triangles on the reconstructed mesh that do not correspond to any of the available textures. This case arises when the hull is viewed from outside the extreme bounds of the reconstruction cameras. When this is the case the triangle in question receives a medium gray color and is left untextured. The system uses gray to represent the untextured regions since it is a neutral and inoffensive color, but does not misrepresent the results of our system. This gives the end-user the visual cue that the current camera configuration cannot completely represent their desired viewpoint. The most commonly untextured regions are the back-facing polygons of the mesh, ones which are not seen by any reconstruction camera.

The second case where texture data is missing is when the mesh is not watertight. With the reconstruction algorithm utilized, a non-watertight mesh is generated. To approximate a watertight mesh we iterate over the vertices of the entire mesh, combining vertices that are within a very small range of one another. The watertight process is described in Henry Letteron's work [Let04]. This mostly alleviates the problem, but occasionally a gap in the mesh is seen. The texture mapping process can only add a texture to polygons. If there are areas which are not watertight, there are no polygons to be textured and the synthetic background of the composite shows through.

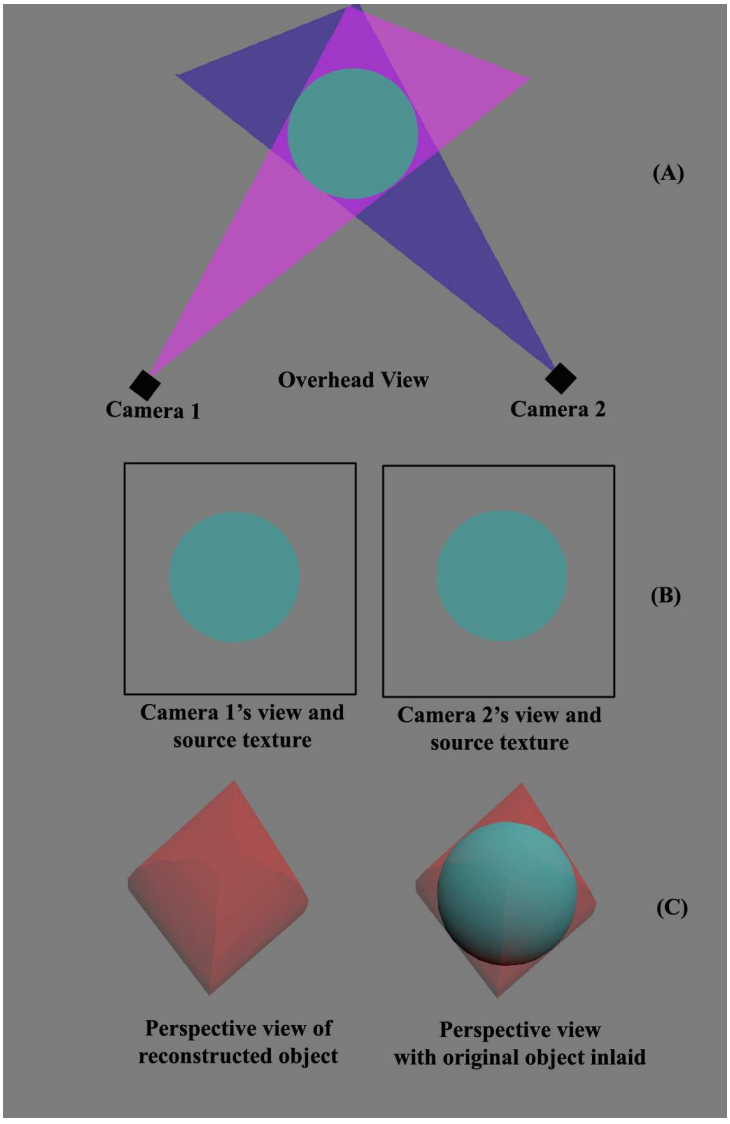

Figure 4.3: *This diagram shows the reconstruction of a blue sphere. (A) Overhead view of two reconstruction cameras and their silhouette cones, projected onto the top view of a sphere. (B) Each camera's views. These are used as source textures in the texturing processes. (C) The reconstructed hull from a perspective view physically above the midpoint of the two reconstruction cameras. This is the polyhedral hull of our computed sphere, using two-reconstruction cameras. The original object is shown overlaid on the hull, illustrating the difference between the reconstruction and original. The challenge is how to fill in the missing textures.*

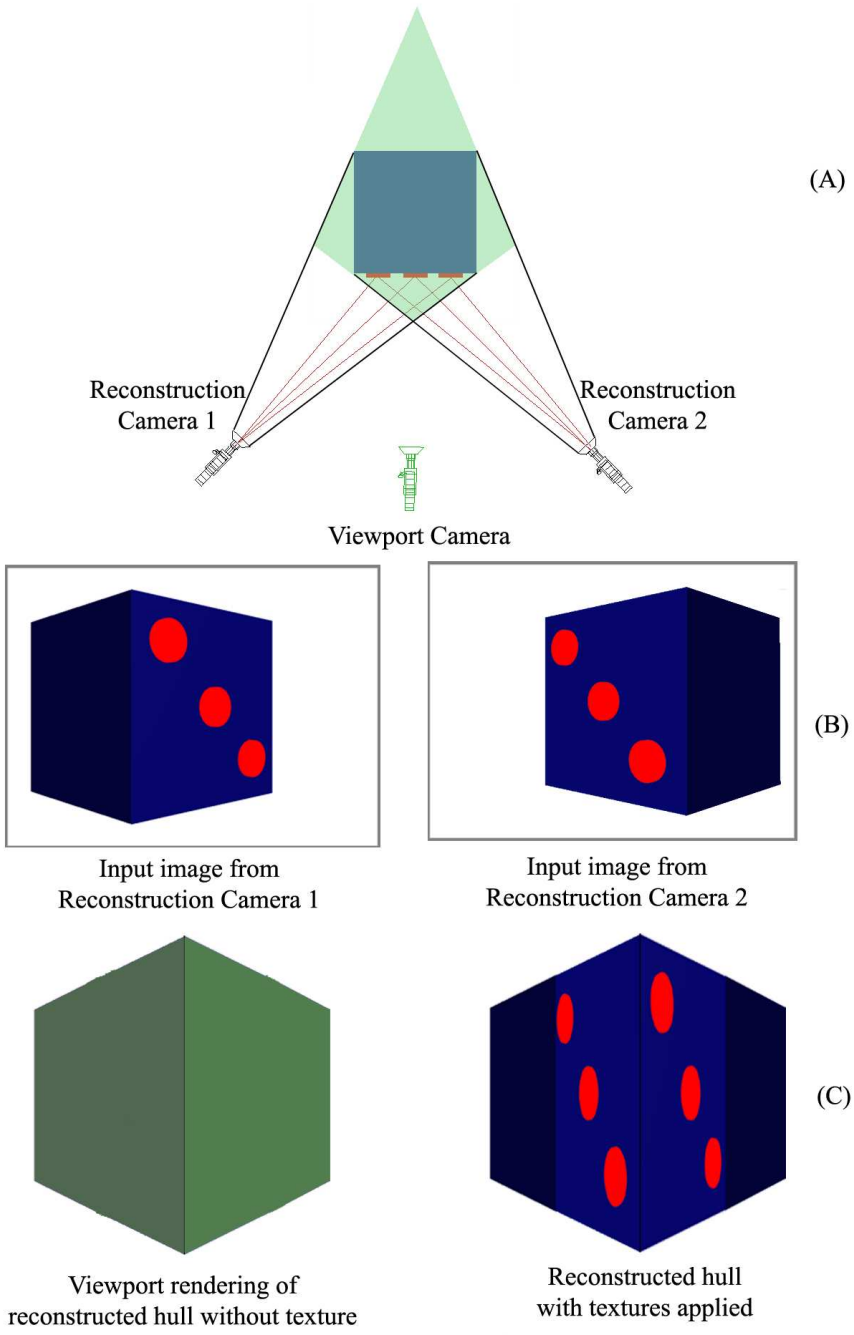

Figure 4.4: *(A) The cube is reconstructed as a diamond-like shape by the two reconstruction views. (B) View from each of the reconstruction cameras. (C) The viewport cameras view of the non textured image, alongside the textured version. Notice that the texture is repeated. Two reconstructed faces in (A) would map to the same region on the original object.*

4.3.3 Non-orthogonal Texture Capture and Choice of Source Texture

Since we only have n fixed reconstruction cameras we do not necessarily view the object from the particular arbitrary viewing angle the end-user desires. As a result the source textures could be poorly applied to the reconstructed hull as is seen in Figure 4.4. This also means that we do not have an even sampling of our source textures over the entire surface. A texel is an atomic element of a texture, just as a pixel is of a digital picture. When evaluating the appropriateness of a texture for rendering it is preferred to have a higher texel density than pixel density.

As mentioned earlier in this chapter, during the standard texture mapping process the ideal source textures would be captured using a camera whose image-plane is orthogonal to the surface on which the texture exists. In our system the reconstruction cameras view the live object at angles that are often far from orthogonal. This causes the sampling frequency of the texture to change over the entire image. (Figure 4.5)

The sample frequency of a texture depends on two factors:

1. The distance D between the reconstruction camera and the small area from which the texture is sampled, called a patch.

2. The cosine of the angle between the reconstruction camera's viewing vector and the estimated surface normal of the patch.

If either of these two dimensions changes then the sampling frequency of a texture changes. There are three reasons that these dimensions might change in a system such as ours.

Since camera images are taken in perspective, the aforementioned cosine changes over the surface of a sampled polygon. This problem is neglible, in our system, because the distance D is sufficiently larger than the size of any particular patch.

Second our source texture is not planar. Our system is designed to use a human being as a reconstruction subject. Humans are certainly not polyhedral objects so our system has the problem that at any given point the surface normal is different from adjacent points. In our system since we do not know the original geometry of our reconstructed object we approximate its surface normals by using ones from the reconstructed mesh, which consists of planar polygons.

Finally since the source textures do not lie orthogonal to the camera at every point. In fact there are some polygons for which the aforementioned angle is very large. This problem is aggravated as the surfaces are farther and farther from orthogonal to the camera. Our texturing solution focuses on solving this problem first, since it is the most visually displeasing.

4.4 Our Texturing Solutions

In the previous section we described the problems encountered when attempting to apply textures at the end of the polyhedral hull reconstruction process. The remainder of this chapter details the texturing procedures we developed to solve these problems, completing the polyhedral visual hull system. By using information from multiple source textures intelligently many of the problems that we encounter can be solved.

We implemented two texturing methods. Our initial work to provide visually correct texturing, the direct texturing method, uses a single source texture for each region of the polyhedral hull. However, after implementing the direct texturing

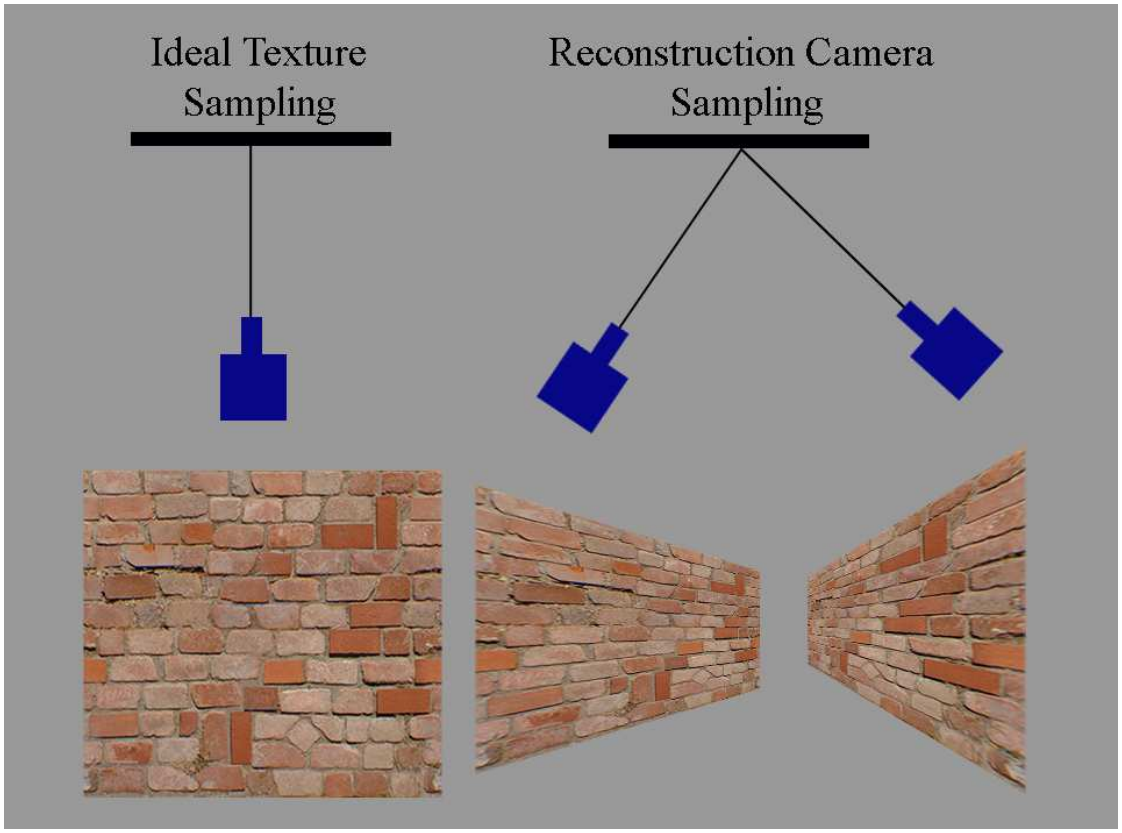

Figure 4.5: *This figure shows the traditional texturing method juxtaposed with the texture samples that are generated using the polyhedral hull system. On the left is a orthogonally viewed texture. The texture has the same sample frequency across the entire image. We assume the distance from the object to the camera is much larger than the size of the object, and therefore the spatial frequency does not change over an orthogonally viewed image. On the right are two cameras that represent the texturing process in our system. The textures that these cameras capture have varying sample frequencies relative to the perspective viewing angle. We design our method to favor textures with high sample frequency.*

method, we discovered that it introduced new problems: we see them manifested as visual discontinuities and a problem we call view-skew. We then developed a more sophisticated texturing method called the blending method which provides better results. The blending method addresses both the texturing problems outlined in the previous section and those which arise when direct texturing is employed. For the remainder of this chapter we define the angle between the surface normal of a polygon and a reconstruction camera's viewing vector as the *camera-normal* angle.

4.4.1 The Direct Texturing Method

This direct texturing method is the simplest texturing method and is effective in some cases. At a high level this process can be thought of as importance-based source texture sampling from the viewport camera location. The multi-staged process selects the portions from each of the reconstruction cameras source textures which contribute the most realistic textures for that viewpoint. This process is shown in Figure 4.6.

4.4.2 Stage One: Texture Capture

During the capture phase, each reconstruction camera in the system transmits its image to the server. The cameras are synchronized with one another. For each frame, the raw video image data is sent to the server along with the contour of the object that has been found. We utilize the raw video in the highest resolution possible from each of our cameras. We do not filter any of the raw data from the cameras. There is some noise and color bleeding from the video capture and chroma-keying process, which can be seen in the final composites.

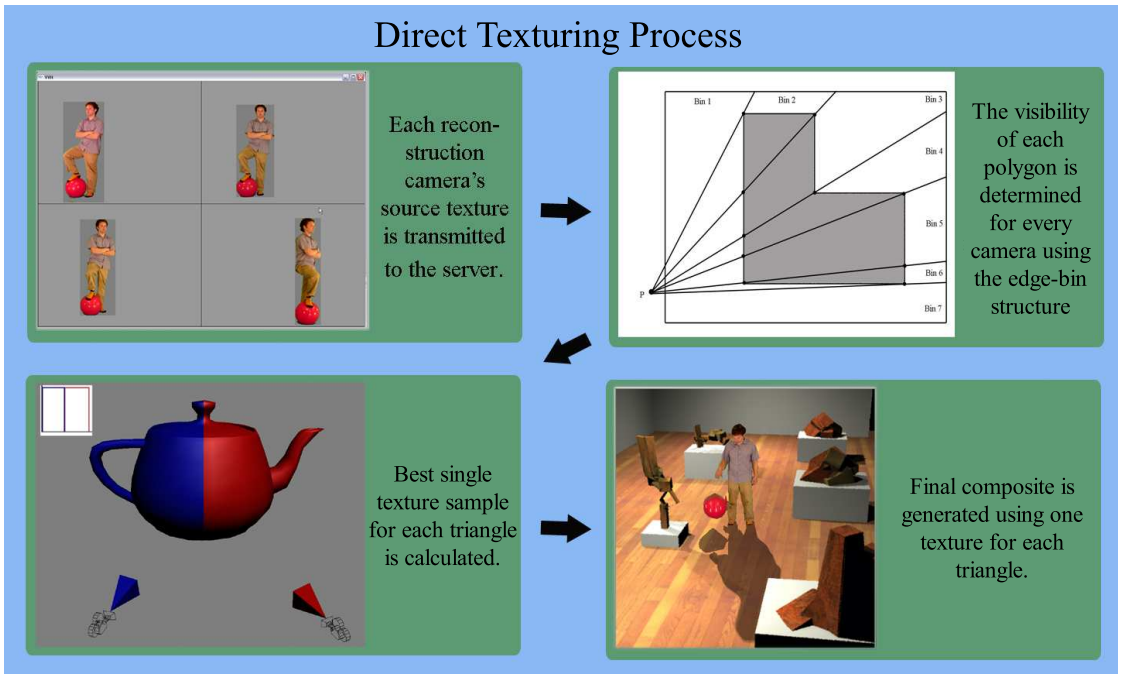

Figure 4.6: *A stage diagram of the direct texturing method implemented in our system.*

4.4.3 Stage Two: Visibility Determination

In the second stage of the texturing process we compute per-polygon reconstruction visibility so that we can ensure that texture data is not inappropriately applied to polygons which do not correspond to any source texture region. We iterate over all the polygons in the reconstructed mesh and compute a boolean visibility value indicating whether or not it can be seen by a reconstruction camera. This is easily done because we reuse computations performed in the geometric reconstruction phase of our system.

Each reconstruction has a set of edge-bins associated with it. An edge-bin is best thought of as a view-dependent z-buffer which lists each of the polygons which could be visible, from nearest to farthest. Utilizing this structure it is easy to determine which polygons are seen by a camera and can therefore be textured

by it. We start by marking every polygon as not visible. For each camera we iterate over all of its bins, marking the first polygon in the bin as visible. All other polygons in the bin are not marked visible and will not receive any texture data from this reconstruction camera. Please refer to [MBM01a] or [Let04] for details on the edge-bin data algorithm and data structure.

In the case where more than one camera can see a particular polygon the system must decide which camera's texture contribution to utilize. This is the third stage of our process. For any polygons that are not seen by a reconstruction camera or are only partially visible the texture is set to neutral grey, which indicates that this camera configuration should not contribute texture information to that region.

4.4.4 Stage Three: Texture Contribution Calculation

Once we have determined the visible polygons, the next task is to select the best source texture from our reconstruction cameras, so that we can generate a visually plausible composite image. To do this we must define a metric with which we can delineate between the multiple source textures we have available for use. Using this metric, we choose a source texture for each polygon in the reconstructed mesh.

The metric we chose to implement is based on the camera-normal angle. The premise is that the best source texture would have been captured with the reconstruction camera's view direction orthogonal to the original object's surface normal. We estimate how closely any given reconstruction camera's source texture approximates this ideal by computing the dot product of the camera's viewing vector and the surface normal of the polygon in the reconstructed mesh. This dot product is used as a positively-correlated metric, and the source texture with the highest value is chosen. Again, this is computed per-polygon.

4.4.5 Stage Four: Final Rendering

Finally, we render each polygon with its associated texture, using the graphics hardware of the server. An example of the resulting composite is shown in Figure 4.7.

4.4.6 Direct Texturing Analysis

This direct texturing method addresses each of the problems identified in the previous section. By using the mathematical calculation for the closest aligned camera, we address the problems of non-orthogonal source texture capture and our choice of source texture. Choosing the most closely aligned camera to the viewport camera minimizes the effects of non-orthogonal source textures. Because the use of this metric maximizes its cosine value of the camera-normal angle, and sample density is proportional to its cosine, the selected texture inherently has the best spatial frequency of the samples available.

The coarse geometry is addressed by providing a mapping between the source textures from the original object and the reconstructed mesh. In all regions where the cameras can view the original object, we have a mapping from the source-textures to the reconstructed mesh. It should be noted that in back-facing and partially visible regions these areas are colored grey to indicate the lack of information unobtrusively.

The direct texturing process is simple to understand and can provide visually acceptable results. We discovered, however, that the results were not visually convincing outside of certain special cases. This method seems to be completely acceptable only for cases when the viewport camera is closely aligned with one of the reconstruction cameras. This severely limits view-independence.

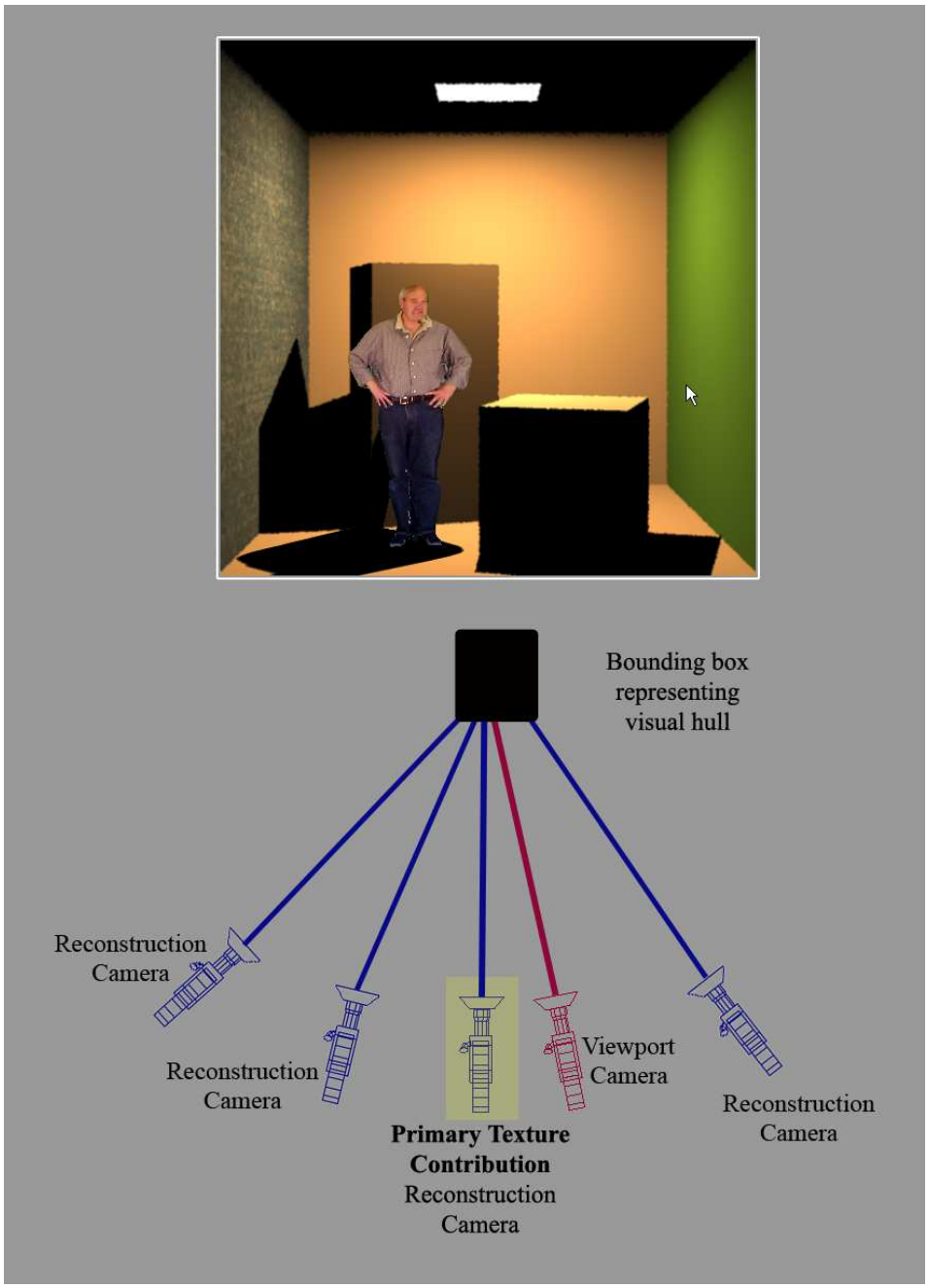

Figure 4.7: *An example of a well textured polyhedral hull using the direct method. These results look very realistic due to the relatively small angle between the viewing directions of the viewport camera and reconstruction camera.*

There are two types of visually disturbing artifacts which can be seen as a result of this texturing process. The most obvious are the texture discontinuities which occur when the choice of source texture changes abruptly along the surface of the reconstructed mesh. The second type of artifact is more subtle and is caused by view skew. Since we only have a relatively small number of cameras there are many virtual camera positions which share the same direct texturing solution. Thus, although the object may move, the same texture is used, masking what should be a perceivable change in object geometry.

4.4.7 Discontinuities

Since any given polygon only uses one source texture, in cases where the origin of the source texture switches from one camera to another we see a discontinuity, signaling the end of one camera's viewing dominance and the beginning of another's. This is illustrated in Figure 4.8.

By using this approach we are treating camera relevance as a step function over the viewing space. At the point where the dot products between two adjacent reconstruction cameras are the same the boolean check for camera-dominance switches from one camera to another. We see this offensive visual artifact in our results (Figure 4.8). The step function description is important to understand here, as it illustrates the way which the direct method treats source textures. In both the mathematical description and the reconstructed texture we see rapid changes in the graph of data which are represented as visual discontinuities. Figure 4.9 illustrates this.

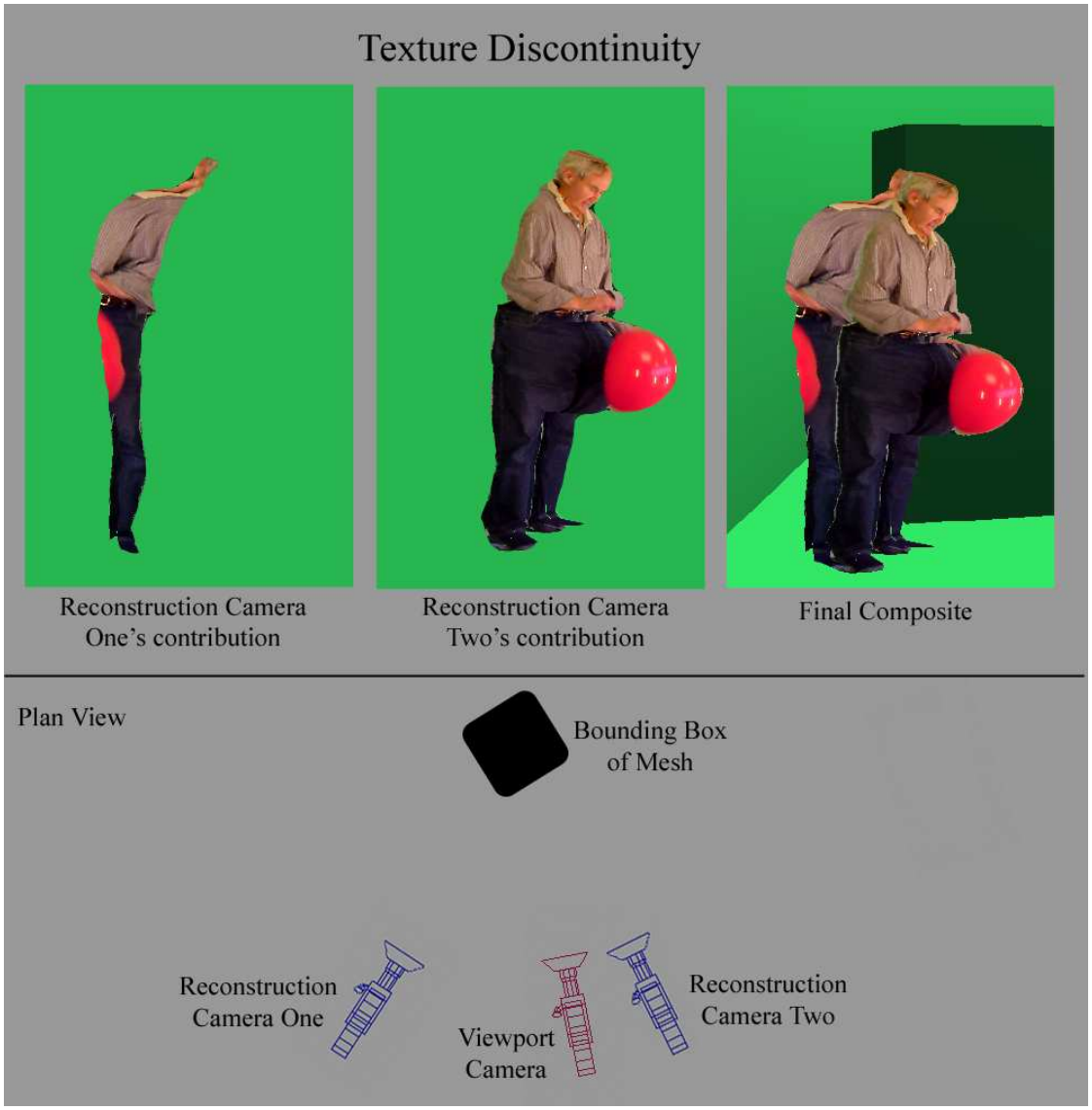

Figure 4.8: *On the far left we see the portion of the texture that was contributed by one camera. In the middle we see the portion of the texture from the second camera. On the right we see the poorly textured polyhedral mesh, using the direct method. At the bottom of the diagram is a plan view of the approximate camera positions that generated this composite image.*

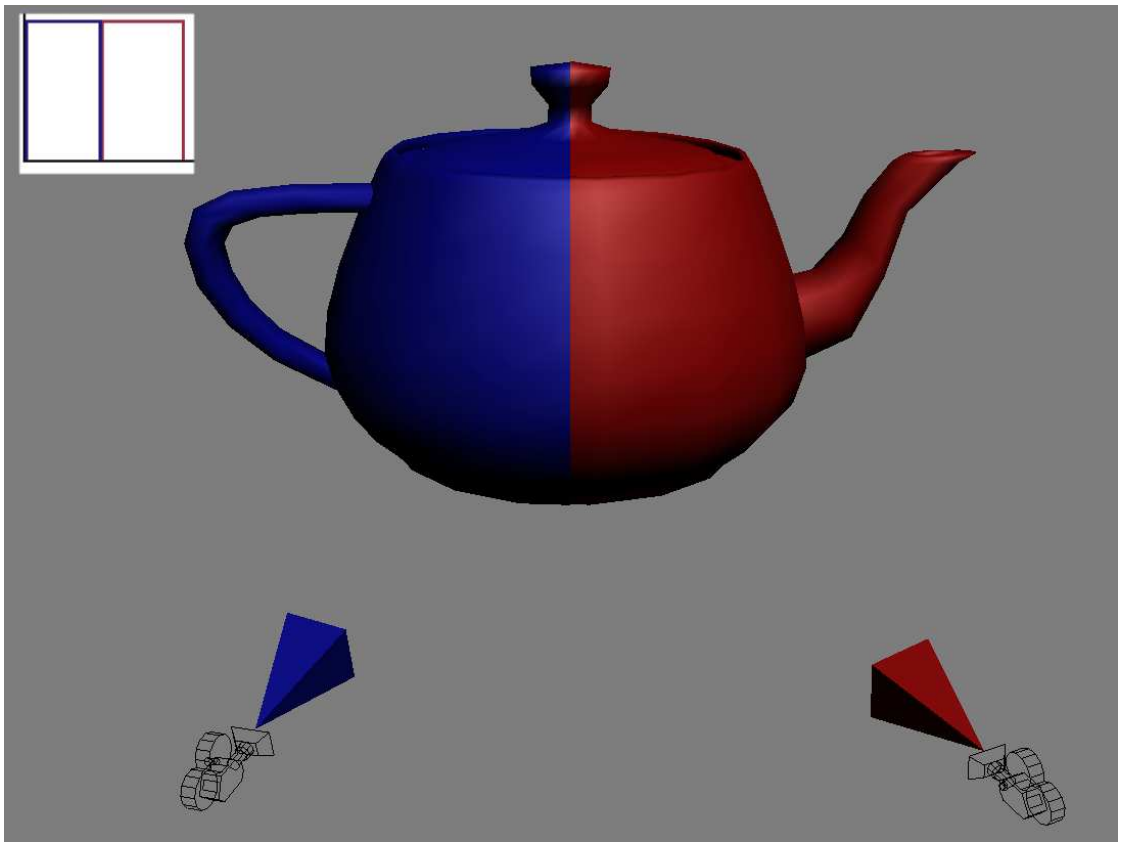

Figure 4.9: *Here is an example of reconstruction camera contribution for a region of the reconstructed teapot using the direct texturing method. The camera on the left has a smaller camera-normal angle in the blue region; where the color shifts to red the camera on the right has a smaller angle. There is an obvious and abrupt shift from the blue camera's contribution to the red camera's as is expected. The icon in the upper left hand corner shows a step function which is the mathematical equivalent of the direct texturing method.*

4.4.8 View-Skew

The view-skew problem is a less visually detectable problem, since it can only be seen in the live operation of the system or in video, but it is still quite visually displeasing. Direct texturing often yields the same texturing solution for two distinct but similar viewport camera locations. The effect is that the user sees the same texture repetitively despite the fact that the underlying geometry changes.

The view-skew problem is illustrated in Figure 4.10 in which the viewport camera pans around our static model, Howie. One of the four reconstruction cameras contributes the dominant portion of the texture. As we pan, the majority of the texture solution to the composite remains unchanged; the only detectable change to the texture occurs around the silhouette edges. Thus, the rotation of the underlying geometry is completely unnoticeable.

To address these two new problems we designed a blending approach. This approach was derived from the direct texturing method but introduces a greater level of sophistication in the texturing process.

4.4.9 The Blending Method

The blending method represents an advance over the direct method; it addresses the ambiguity in and subsequent difficulty of mapping source textures to polygons by using multiple textures simultaneously. The direct texturing process often yields inadequate composites due to discontinuities and view-skew. The goal of our blending method is to reduce the problems introduced by direct texturing. Texture blending provides true view independence since it addresses these problems in addition to the original texturing issues.

In this method, in contrast to the direct method, there are a series of additional

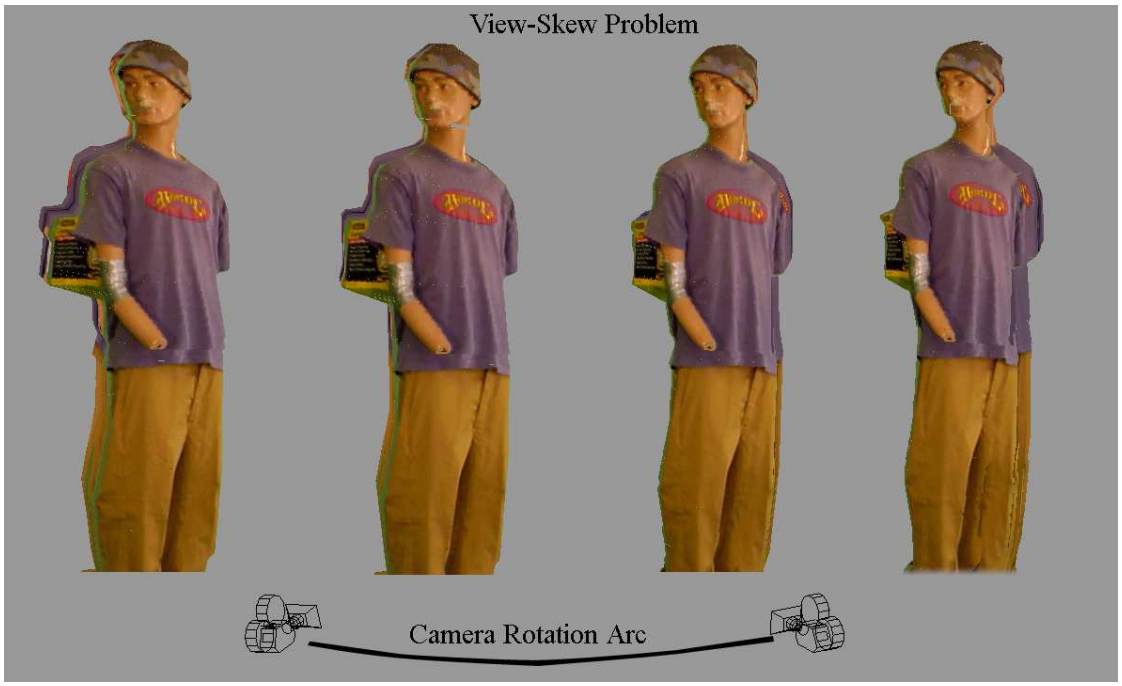

Figure 4.10: *An example of the view-skew problem. Notice that as the viewport camera is rotated around the object only the edges of the texture change. Ninety percent of the texture remains unchanged. The changes in the geometry are masked so this is disconcerting. We expect to see changes in the textures and geometry; instead we see nothing but a sudden shift as a discontinuity arises.*

calculations during the the camera contribution stage to utilize textures judiciously. The stage diagram for the blending process is illustrated in Figure 4.11.

4.5 Overview

The direct method implementation of texturing causes visual problems, such as discontinuity or view-skew. This method discards data and therefore relies on only a small subset of the original non-orthogonally sampled source textures. Direct texturing only picks the single best source texture. It misses any potentially useful

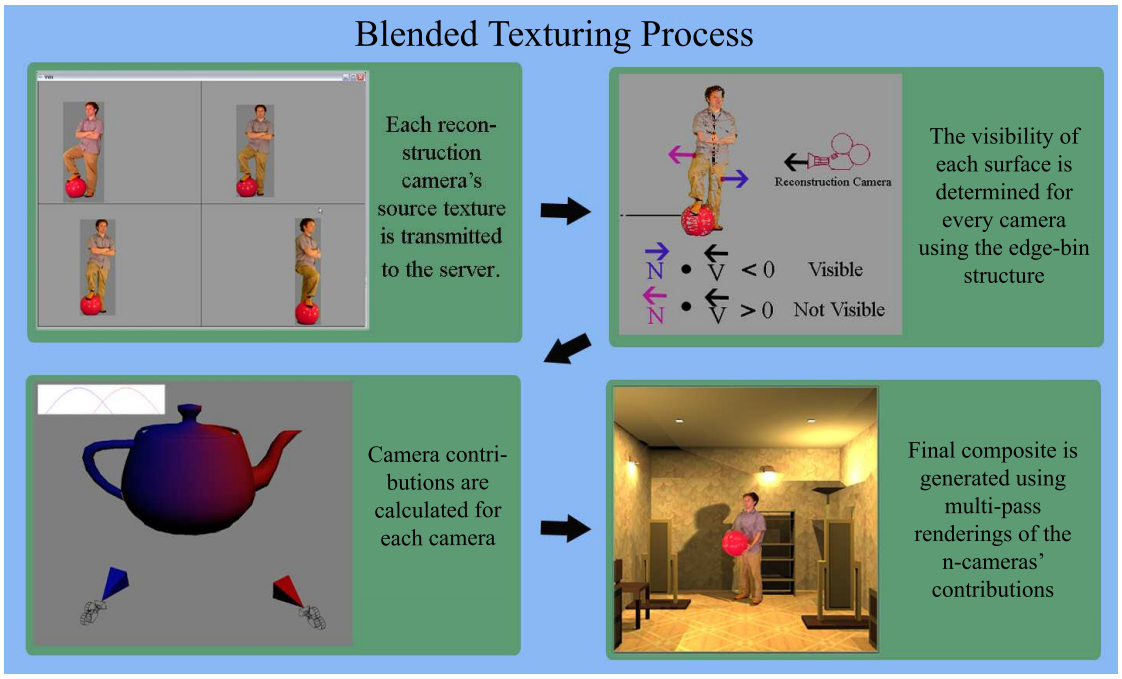

Figure 4.11: *A stage diagram of the blending process implemented in our system.*

information that the remaining $n-1$ textures might contain.

To better texture the polyhedral mesh we use multiple source textures to provide the most realistic composites at each polygon. We assume that the relevance of each texture is a continuously varying function over a surface. In order to generate textures in areas where we do not have an exact match from one of our reconstruction cameras, we sample each of the source textures and use a linear combination of these to provide a visually plausible texture. This effectively eliminates the discontinuity problem that is prevelant in the direct method implementation. Because it allows for smoothly varying textures from multiple sources this also reduces the view-skew. This is illustrated in Figure 4.12 where we see the two methods juxtaposed.

We blend textures by rendering each of the reconstruction cameras' views as a variable-opacity texture-map. The following process is performed for every polygon

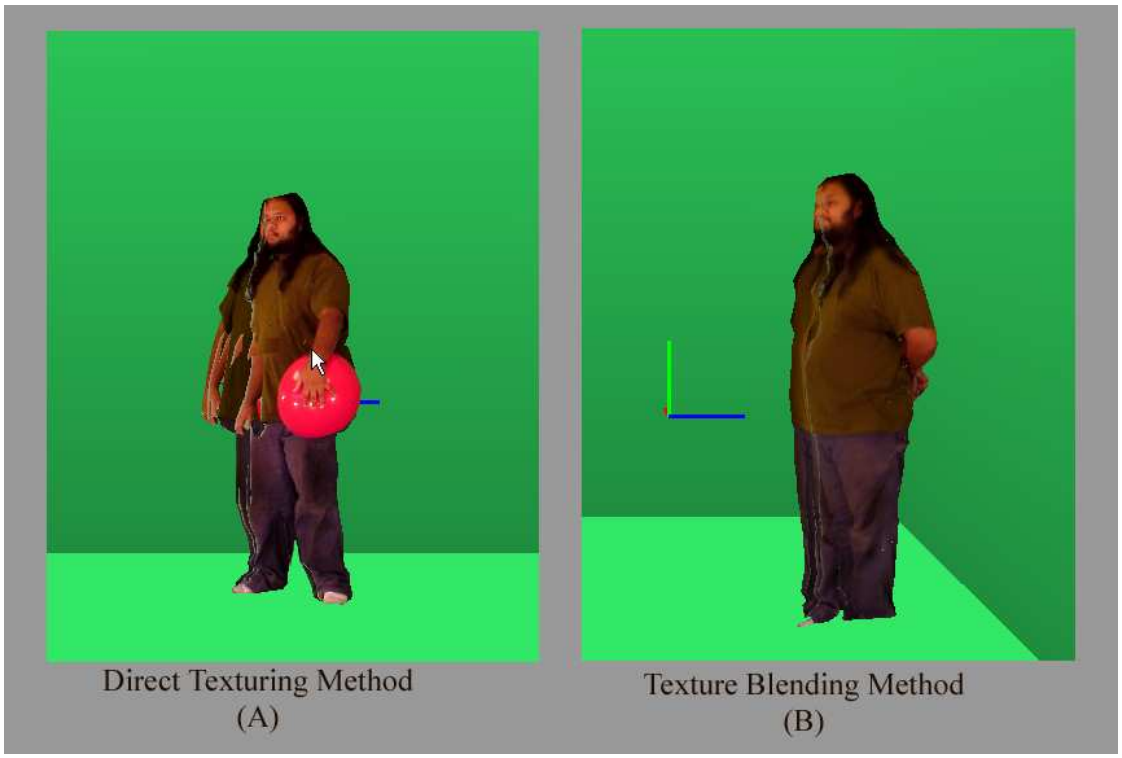

Figure 4.12: *An example of the two different texturing processes. On the left (A) is the basic, direct texturing method and on the right is the blended texturing blending. Notice on the left the repetition of arms. On the right (B), there is a single representation of the entire model. The slight line visible in the texture blended example is due to chroma-key noise.*

in the reconstructed mesh. First, we calculate the texture coordinates in each of the n source textures, using our knowledge of the projection matrices. Once these values are calculated, they are stored in n different texture maps, one for each source texture. Next, we evaluate a metric which represents how closely the pose of the reconstruction and viewport cameras match for each reconstruction camera. The value of the metric for each camera is used as the blending weight for the corresponding source texture, after appropriate normalization. The last step renders each of the texture maps in a separate pass, using the alpha buffer to

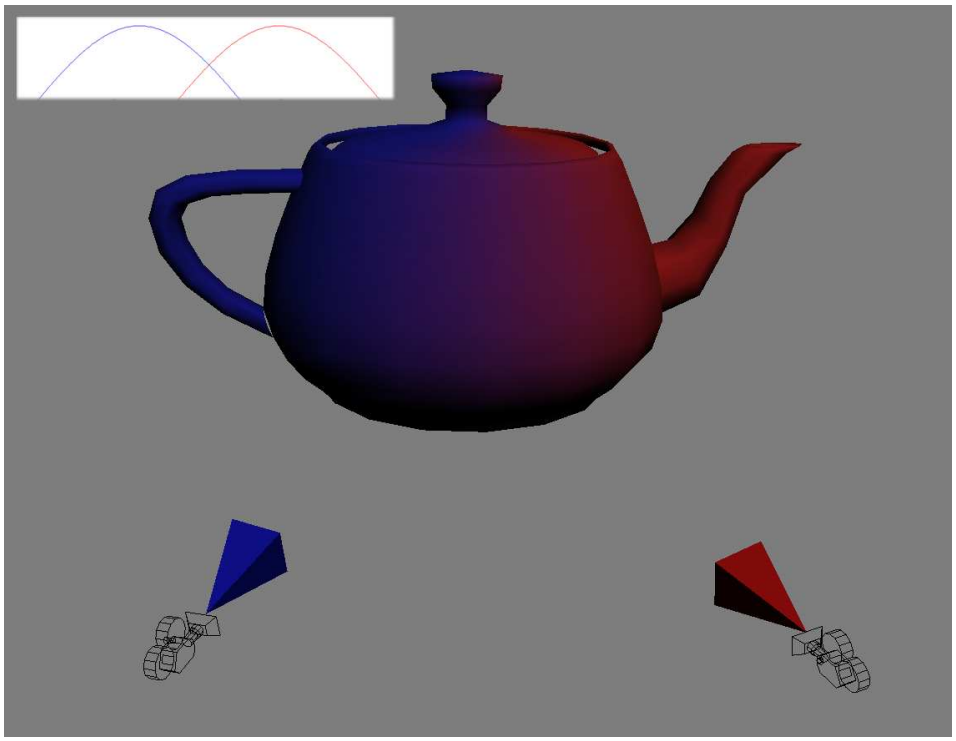

Figure 4.13: *Here we see blending from two cameras. The red and blue regions overlap and are blended into purple with varying intensities. In the upper left is the mathematical equivalent of the blending function, showing overlapping cosine curves.*

compute the linear combination of textures (Figure 4.13).

4.5.1 Stage One: Capture

The capture step is the same as in the direct texturing method. No special processing for the texturing method is done. All that is accomplished in this step is the capture of the source textures from each of the reconstruction cameras and the transmission of the textures from the client computers to the server.

4.5.2 Stage Two: Visibility

Unlike the direct texturing method which determines visibility using the edge-bin data structure, the blending method tests the normal at each polygon against each camera's viewing vector to determine the visibility of each polygon for each reconstruction camera. In this way we compute which cameras are able to view the polygon and should have a camera relevance and blending weight calculated. Figure 4.14 shows how we use the dot product as a boolean indicator of visibility. This visibility calculation eliminates all of the back-facing polygons for each reconstruction camera, but does not eliminate cases where a polygon occludes another. These occlusion errors are less perceptible because of the blending process. Since we are constrained to produce real-time results no additional occlusion removal calculations are executed.

4.5.3 Stage Three: Blending

One of the crucial parts of texture blending is the ability to identify the relevance of the reconstruction camera to the overall blended texture. This involves the use of a metric to demarcate each camera's effectiveness. We experimented with two methods, which we will call the *viewing-angle* and *camera-normal* metrics (Figure 4.15). The viewing-angle metric assigns a value to the angle between the viewport camera and the reconstruction camera. The camera-normal metric compares the normal of the surface to be textured to the viewing-vector of the reconstruction camera. This method is the same as that which is used in the direct texturing process. While both of these metrics produce a blended texture, the results can be vastly different. We discuss our choice of one method over the other later in this section.

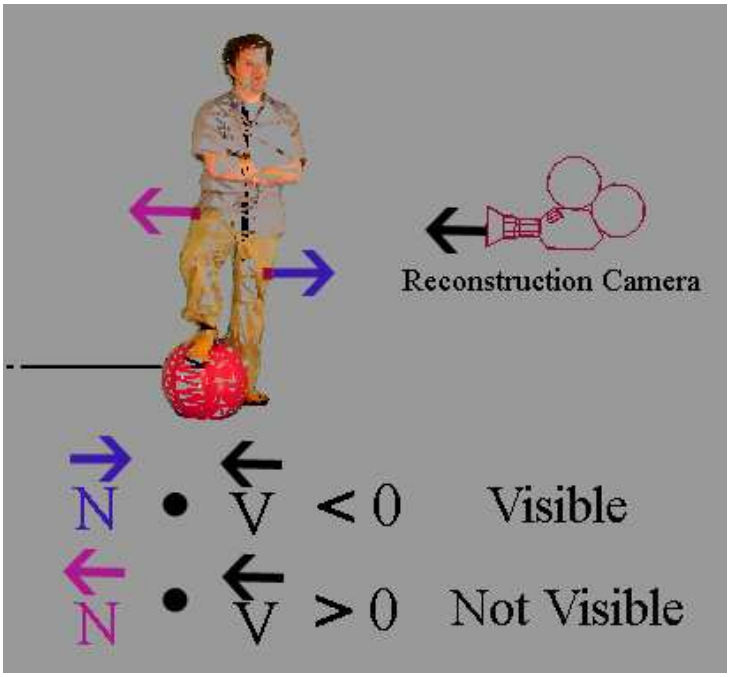

Figure 4.14: *Here we see a scenario of both visible, and non-visible texture areas from a particular reconstruction camera's point of view. The camera's normal is colored black and is clearly anti-parallel with the nearer polygon's normal, seen in blue. The dot product is also obviously negative since two vectors pointed in opposite directions always result in a negative dot product. Further we see that this reconstruction camera will have no information for the vertex with the pink normal, as it is back-facing to this particular camera. As is expected, the dot product of this normal and the camera normal yields a positive value. This visibility process removes back-facing polygons only. It does not handle occlusion by other polygons.*

The viewing-angle metric is computed by comparing the cosine of the angle of the viewing rays of the reconstruction cameras and the viewport camera. We construct rays from the center of the polygon to be textured to each of the camera centers and compute the dot-products which corresponds to the cosine of the angle between these vectors. In this case the largest dot product represents the most closely aligned viewing angle with the viewport camera. In our testing, results of this method were visually poor, so we developed another metric called the camera-normal method. Unfortunately we have no comparison between these two methods since the viewing-angle method was completely removed from the final system.

The camera-normal metric is calculated by comparison of the surface normal of the triangular polygon with the reconstruction camera's viewing vector. We remember from the visibility test that the dot product of these two vectors is negative, indicating that the camera can see this polygon if it is the nearest polygon to the reconstruction camera. During the visibility test, if the dot product is negative the value is recorded so it can be used later. This method gives importance to the cameras that most directly view each polygon we texture. The camera which has a value nearest to negative one indicates that its viewing ray is most nearly orthogonal to the polygon. Upon visual inspection we found that this metric yields better results than the viewing-angle metric.

We chose to utilize the camera-normal method in our final system. The camera-normal method is our best effort to choose the minimum angle of any reconstruction camera's viewing ray and the normal of the source texture sample, and yielded good results.

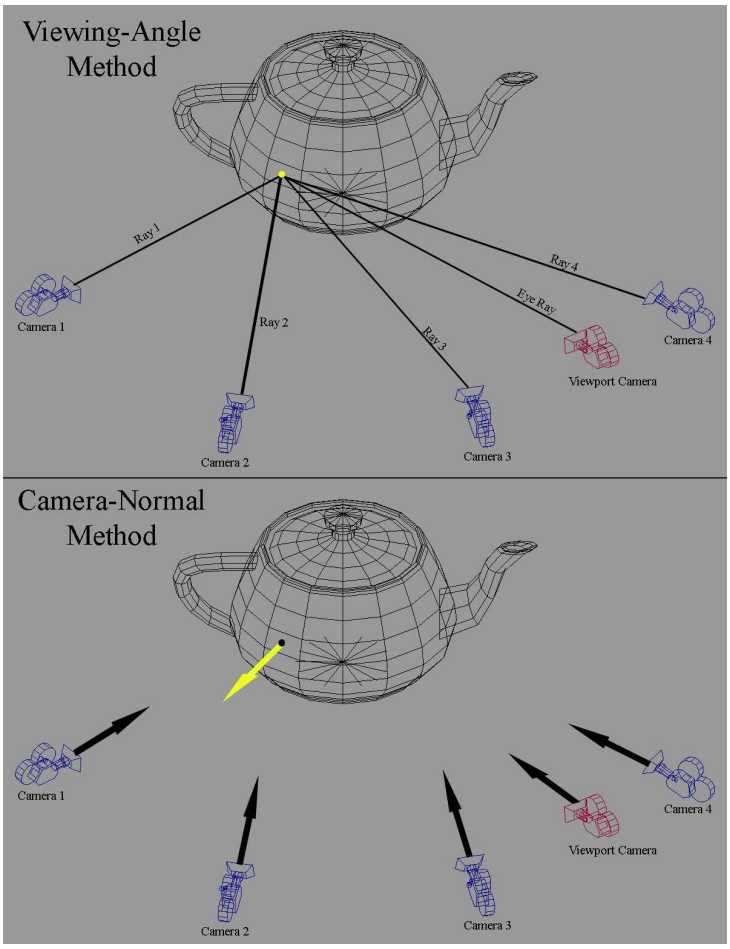

Figure 4.15: *On the top half of the diagram we see the viewing-angle method. It shows that the minimal angles between the viewing rays of the camera would be obtained at cameras three and four. However, these cameras have an extremely skewed interpretation of the texture at the polygon in question. This results in more coherence to the viewport camera but yields bad texture samples. In the bottom half of the diagram we see the camera-normal method. Using this method the cameras with the most direct view are those which are chosen to have the greatest texture importance. In this scenario the yellow surface normal and associated polygon is visible by all cameras. The dot-product values are recorded for each of these cameras and used as relative relevance values. Camera one will have the most negative dot-product followed by cameras two, three and four in that order.*

4.5.4 Stage Four: Compositing

Once we have determined the relevance of each camera we are then ready to use a blending function to modulate between each of the four textures. After the application of the blending function we can use the graphics hardware to render the mesh using each of the variable opacity texture maps. The overall idea is to take a relative contribution from each of the reconstruction cameras as if they were control points in a parametric curve. This contribution is calculated at every polygon for every reconstruction camera. We then modulate our reconstruction camera textures into a final composite image. We normalize the calculated blending weights so they sum to one. Once this is done these values can be used as opacity values for multi-pass rendering of our final composite. Since we have a relevance for each polygon we are able to use the alpha channel of the texture maps to render our composite image as a linear combination of the inputs. Each texture map has different alpha values, equal to the normalized blending weight, for each polygon in the reconstructed mesh.

We experimented with blending all n cameras, as well as using just the two with the highest metric values. Our final system utilizes the two highest metric values since it is significantly faster without any perceptible loss in quality.

4.5.5 Blending Analysis

It is important to note that many other blending functions could be utilized to provide either equivalent or different results. In our system, we opted to use a simple blending function since it provided adequate results while minimizing processing time. In future work, especially with different configurations or number of reconstruction cameras it would behoove the designers to test a wide array of

blending functions. In order to minimize processing, we avoided using the arc cosine function or any unnecessary divisions since these operations are expensive and texture blending already adds a significant overhead to the system.

As previously stated, we also empirically found the best results using just the two most coherent cameras instead of blending between all the texture samples from n reconstruction cameras. The reason for this is probably due to the evenly-spaced reconstruction camera distribution. In our camera configuration, it was nearly impossible to find a novel viewpoint where three or more cameras are able to view the polygon, or the camera relevance is significant for each camera. In future systems, given ample space, it would be interesting to try different configurations resulting in the use of different blending functions.

Chapter 5

Reflection Effects

5.1 Introduction

The polyhedral hull system provides a visually plausible composite of live video and computer generated synthetic imagery in a real-time environment. To achieve our goal of visually acceptable composites we utilized global illumination effects to provide visual cues for the interaction between the real and synthetic components. This chapter presents a novel implementation that generates and composites specular reflections of a reconstructed object into a synthetic scene on a single planar surface. Shadowing, another global illumination component also supported by the polyhedral hull system is described in Henry Letteron's thesis [Let04]. Our algorithm is designed with both rendering speed and image quality in mind; it is a process that has extremely low overhead and does not affect the overall frame-rate while also yielding convincing composite images.

There is a virtual disconnect that exists between the live and synthetic objects in traditional chroma-keying and compositing work. The different portions of the composite do not interact with one another. In order to achieve our overall goal of merging live video in a synthetic environment, we must break the virtual barrier between the two environments. In traditional chroma-keying systems and advanced computer generated dynamic scenes [Deb98] there is a clear division between the foreground and background. Previous work from the Program of Computer Graphics [Sel03] covered some interaction between the billboarded representation of the live object and the synthetic surrounding scene; however, this work stopped short of complex interactions since there was no captured geometry

to embrace for further visual cues.

Our system for polyhedral hull reconstruction and composition provides us with a number of advantages over previous work. First, we construct a three dimensional vertex mesh that has all the properties of a standard modeled one. This mesh is key in the pursuit of our goal – a visually plausible composite image. Without it we would not be able to interact optically with the synthetic background. Another important advance over previous work is the integration with the Realtime Global Illumination System (RTGI) [PTG02]. This technology provides a stable and mature system for background generation.

These advantages allow us to bridge the foreground and background divide. Utilizing RTGI and the reconstructed three-dimensional geometry our system adds dynamic shadowing to the initial polyhedral system [Let04]. Taking this work a step further and adding a specular reflection from the object into the scene adds another layer of realism to the final composites (Figure 5.1). In this chapter we present our implementation of specular reflections. The forthcoming sections detail the implementation as well as the results of our work.

5.2 Unique Reflection Environment

The polyhedral hull reconstruction system merges dynamic live objects and three-dimensional synthetic scenes. To accomplish this task we must overcome a unique set of obstacles. The most relevant with respect to global illumination effects, such as specular reflections, is determining the position and orientation of the lighting in both environments. Our dynamic object is filmed in a studio environment and the lighting properties of that studio are an inherent part of the video images used for texturing. In order to provide a visually plausible composite we need to

Figure 5.1: *(A) On the left a scene without a reflection. Notice that the object seems to be floating and does not interact optically with the surrounding environment. (B) On the right a scene with a reflection. The object appears to be grounded and part of the scene.*

understand the position and orientation of the lighting in the studio as well as the synthetic three-dimensional environment. There are two possible solutions to this complication: perform inverse global illumination or assume matched lighting.

Inverse global illumination, or de-lighting and re-lighting the object, is an open research question. It requires that all the material properties of the object and both the environment and lighting of our dynamic real-environment be known. Because of the flexibility given to the user in terms of object selection, it is impractical to know the material properties of every object that could possibly be reconstructed. It is not yet well understood how cloth, hair and skin – the three common materials encountered when imaging human beings – behave optically, and even if it were possible to understand the properties of every material, our reconstructed mesh

would need to be segmented into sub-meshes, each composed of a simple material. This image-space segmentation is still considered an open research question within the computer vision community. For all of these reasons we did not pursue the option of using inverse global illumination.

We chose the second option: to assume a matched lighting environment between the video studio where dynamic live video is captured and the synthetic three-dimensional background environment. By utilizing a matched lighting assumption our system is able to provide plausible composites without the significant delighting and re-lighting computations that inverse global illumination would require.

Another unique problem our system must address is the identification of reflective surfaces in the synthetic three-dimensional environment. In order to represent every possible specular reflection in the final composite, our system would have to identify every polygon with a specular component and compute a reflection on that polygon. Computing specular reflections is normally accomplished by ray-tracing, a rather slow and largely offline process. We employ a fast method to render a reflection on a polygonal patch; however, even with this speed-up it would be impractical to compute the specular reflection for every polygon.

Our implementation allows the user to see reflections of our dynamic object in a manner which neglibly affects the overall frame-rate. In order to maintain interactivity we make some simplifying assumptions. First we assume that the reflected surface is planar and unoccluded. This allows us to use a simplified math to project a reflection on a planar surface. Second, because the reflective surface is unoccluded there is no need to calculate where the reflection is occluded, a process that is computationally expensive. Third, we restrict our attention to the single most important reflection. By limiting our system to one reflection there is

no need to account for interreflections which again is computationally expensive. Even with these restrictive assumptions however our system produces believable images.

5.3 Hardware Rendering Methods

In order to generate realistic looking images faster than traditional ray-tracing methods hardware solutions were explored. Since the late 1970's hardware methods have been used to approximate ray-traced images.

5.3.1 Reflection Mapping

Synthetically rendered reflection and environment maps were first introduced in 1976 by Blinn [BN76]. The terms *environment mapping* and *reflection mapping* are used interchangeably throughout the literature, and since our system generates reflections based on this technique, the latter term is adopted in this text. The purpose of environment maps is to add complexity to simple objects so that they appear to interact with their surroundings. Blinn introduced the idea of computing lighting contributions in a scene separately from the rasterization of an object(Figure 5.2).

If the lighting contribution from all the surfaces including every light is computed at each pixel in the scene a "reflection map" can be generated. This pixel-based key for light contribution represents a component of a final image. The other component of the final image is the pixel-based rasterization of each three-dimensional object in the scene. Taking a linear combination of these two components produces a final composite image. With the advent of specialized graphics hardware the rasterization process can be done very fast, leaving only the pre-

Figure 5.2: *The first reflection mapped scene. (A) A map of a room with windows (B) The reflection map composited on a teapot model. [BN76]*

computation of the reflection map for the general purpose processor. The pre-computation presumes a static environment and does not include self-occlusion. Once the objects are rasterized to a frame-buffer the shading of each pixel in the buffer can be modified by the corresponding value in the reflection map. This adds environmental lighting properties to the final image. This is clearly a gross simplification of global illumination. Without occlusion information and material properties, shadows and reflections between objects in the scene could not be calculated properly; they are approximated. Reflection mapping, however, presented a large time improvement over ray-traced solutions, and despite its limited abilities, it provides visually pleasing results.

Since the late 1970s both general purpose computing hardware and graphics specific hardware has improved in speed and complexity. In order to generate a reflection for an object, a reflection map is generated by rendering the scene from that object's point of view. After this is done for every reflective object in the

scene, the reflection maps can be successively applied using the graphics hardware to add realism and complexity. Applying multiple maps (i.e. reflection maps, shadow maps, texture maps) in a hardware rendering process is commonplace using today's graphics hardware.

In the polyhedral hull system, shadows are generated for a single light source using penumbra and shadow maps. First, the shadowing process computes a depth image of the composited live object and synthetic environment from the point of view of the light. After this is done, the depth image is used to generate shadow regions in the composited scene. Despite the hardware accelerations this process is still computationally expensive. The shadowing process takes approximately a quarter of the total rendering time for any frame in the system. Adding another stage in our process that takes this amount of time would sacrifice our real-time goal [Let04].

In an interactive system in which geometry is known in advance, such as a gaming environment, adding both shadows and reflections using mapping techniques would be more feasible. But, our system has to allow for video capture, data transfer and reconstruction, which consume the remaining time in the per-frame computations. Therefore our implementation of reflections had to add minimal frame-rate delay.

5.3.2 Physically Based Reflections

The method that our system uses provides identical results to reflection mapping, but utilizes previous data and computations from earlier stages of the system. Reflection mapping generates a map by rendering all of the objects from the point of view of the reflective surface. The physically based reflection process leverages

the fact that we are compositing a live object into a three-dimensional synthetic environment. By the time the system enters the final compositing stages, the live object has been reconstructed, positioned in the virtual environment relative to the viewport camera, and textured. Since these calculations have already been executed it is possible to use them to determine where the reflections in the scene will appear. Furthermore, since our system only handles reflections of the dynamic object in the synthetic scene it is possible to make more assumptions that simplify this process.

The most important observation about specular reflections is that the angle of incidence from the eye is equal to the angle of reflection (Figure 5.3). This allows us to avoid computing a reflection map since we also already know the reflective surface in the synthetic model, which in the parallel shadowing structure takes up 90% of the shadowing time [Let04]. Figure 5.5 shows the difference in computation time between the shadowing and reflection steps.

If a reflection falls on a specular floor, then it is simple to calculate. To render a reflection, a transformation about the reflection line is performed by the graphics hardware. This transformation places the object where the virtual image belongs. The mesh is then rasterized using a de-saturated alpha value to give a plausibly lit reflective surface (Figure 5.3). The de-saturated alpha value is a user controlled variable that changes the appearance of the reflection.

When a reflection does not fall on the floor another simple approximation is utilized. In order to construct a reflection at the appropriate location in an image, an arbitrary *reflection line* is established. The reflection line is calculated as follows. First, the centroid of the reconstruction center and the center of the reflected surface are found. The midpoint between these two centers is one point

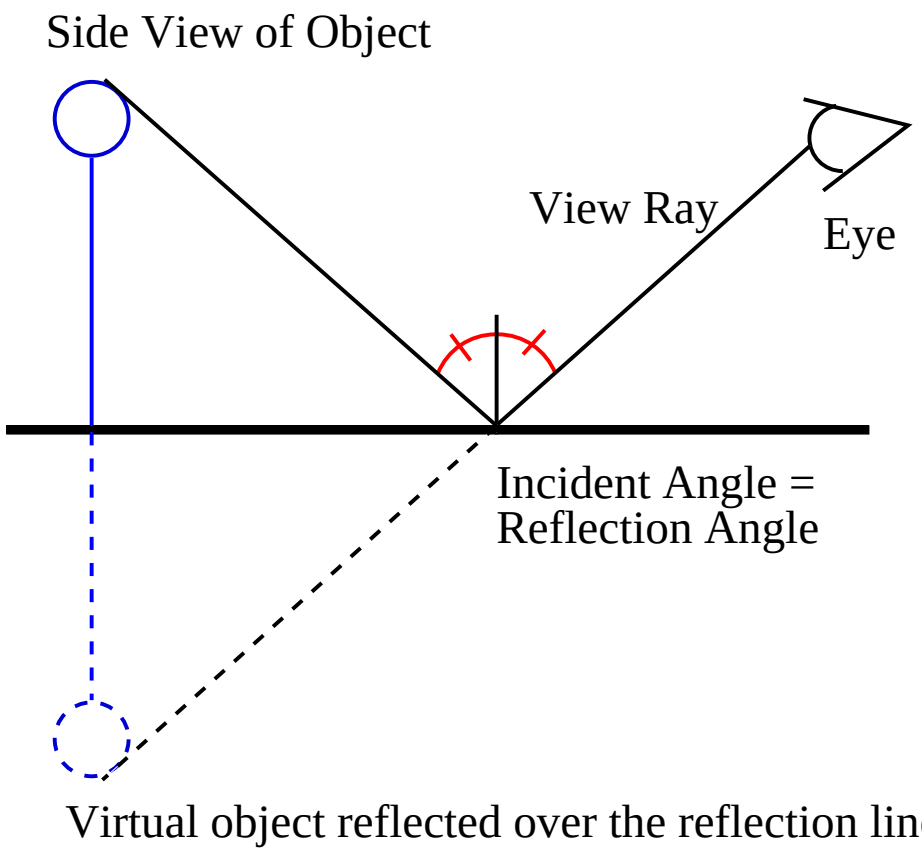

Figure 5.3: *This diagram shows that the incidence angle on the reflective surface is the same as the angle of reflection on the same surface. The reflection line lies in the plane of the surfaces perpendicular to the viewing ray. This observation allows us to simplify our reflection rendering.*

on the reflection line. The reflection line is constructed orthogonally to the line formed between the centers. Figure 5.4 illustrates this calculation. This line is then used as described in the previous paragraph.

Our program uses a default setting which gives the appearance of the reflection on a polished wood surface. Our reconstructed mesh has only a few thousand polygons that can be rasterized extremely quickly by the graphics hardware, and since little extra computation is needed by the already taxed general purpose

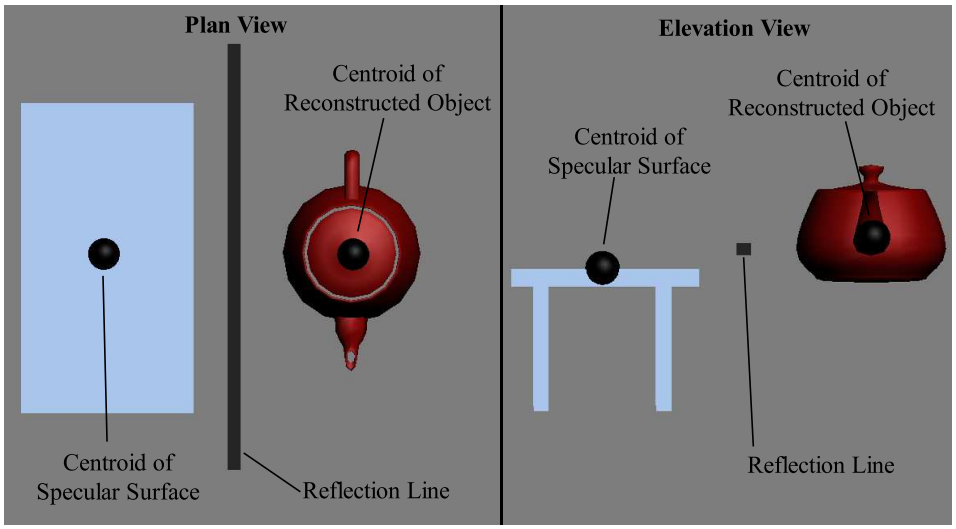

Figure 5.4: *This diagram shows the calculation of a reflection line. The reconstructed teapot is to be reflected onto the table top surface in blue. The teapot's reflection is mirrored over the reflection line in order to place the reflection correctly.*

processor, there is a miniscule cost associated with rendering reflections.

5.4 Analysis

The reflections in our system offer yet another visual cue that establishes the relationship between the real and synthetic parts of our composite image. We add a reflection by utilizing calculations from previous stages of the system to minimize processing time. The results are within the bounds of visual plausibility and dynamically respond to changes in the system. Finally, the reflective surfaces in the synthetic images can be selected so that the strongest reflective object, whether it is on the floor, wall, table, or mirror can be shown. Our system cannot handle reflections on partially occluded planar surfaces or on curved surfaces.

Although our reflection process provides adequate results, it is far from perfect

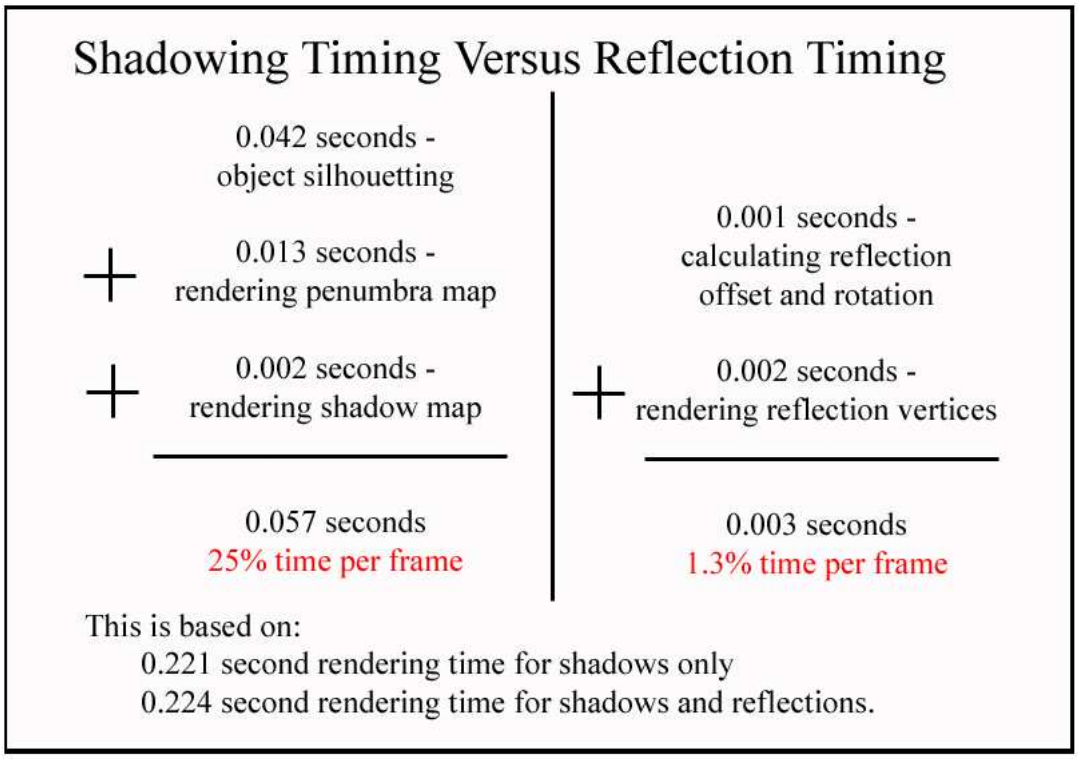

Figure 5.5: *The total processing time of the shadowing process is a significant portion of the total rendering time per-frame. For this reason the reflection implementation must be extremely efficient in order to ensure real-time results.*

and it certainly could be improved. First, to increase the visual cues and to be able to correctly demonstrate compositing in a complex scene, such as a hall of mirrors, this system should be extended to offer specular reflection on multiple surfaces. This addition to the system would offer significant overhead since the stencil buffer on the graphics card would be need to utilized to provide reflection clipping. Secondly, even though this process scales in a purely linear fashion and is very time efficient to begin with, the cumulative overhead of rendering our reconstructed mesh many times would begin to be evident. The conclusion presents an analysis of how this work should be improved in the long-term.

Figure 5.6: *Captured output from the Polyhedral Hull Online Compositing System showing a reconstructed mesh with texture blending, shadowing and a specular reflection on the floor. The captured video object is fully merged into the RTGI background environment.*

Chapter 6

Conclusion

This thesis has presented the Polyhedral Hull Online Compositing System, a three-dimensional reconstruction system that merges live video imagery with computer-generated synthetic environments. The improvement that the Polyhedral Hull Online Compositing System presents over previous systems is the real-time three-dimensional reconstruction which allows the incorporation of limited but realistic shadows and reflections for added realism. This system offers significant improvement over traditional two-dimensional processes such as chroma-keying, since three-dimensional information can be exploited to correctly generate physically-based global illumination effects. Furthermore, the Polyhedral Hull Online Compositing System presents techniques to merge texture information captured from multiple sources so that they appear coherent. Because we have a three-dimensional reconstruction obtained from multiple cameras, we also offer view- independence to the end-user as there is a limited set of constraints placed on viewing the final composite image.

The reconstruction system is built on widely available, consumer-grade hardware in order to minimize cost and to provide an alternative to higher priced options while maintaining visually plausible results. This hardware is used to build a pipeline to efficiently reconstruct, texture, composite, and add global illumination effects to the final resulting image.

This thesis described two specific aspects of the pipeline: texturing and reflections. The texturing portion of the pipeline utilizes weighted importances of captured images to determine the relevant portions of each reconstruction cam-

era's view. Once these portions are determined it is then possible to blend them together to produce a final texture that looks coherent. During the reflections stage of the pipeline, the live object is reflected onto a mirrored or polished surface in order to visually link the disjoint parts of the composite [Let04] [PTG02].

6.1 Prototype System

This system served as an experimental model that demonstrated that the concepts and design ideas are plausible. Overall the prototype is functional with some portions of the system working better than others. One initial goal of this system was to recreate a three-dimensional mesh from two-dimensional live video imagery using multiple cameras. The visual hull reconstruction methods provided a reasonable approximation of the three-dimensional geometry at interactive (15 fps) rates. Unfortunately the reconstructed geometry suffered from two problems. First, the extents of the reconstructed object did not match the physical extents; this caused subsequent problems in the system related to texturing. Second, the reconstructed geometry is not watertight, and although this is masked in some cases, it is a problem that will need to be addressed in the future. Once a mesh was generated, the goal was to merge the view-independent mesh into a synthetic scene.

Merging this view-independent mesh into the synthetic scene was more difficult than the reconstruction process. The biggest issues arose with the application of the view-dependent textures for the mesh. Chapter 4 detailed the entire process of reconstructing a texture utilizing rudimentary techniques that provided passable results in some, but not all, cases. The system was able to capture texture information from multiple cameras, simultaneously allowing for the selection or blending of different textures. However, this did not mask the problems that arose when

texture information was missing. In the end, it did not provide greater realism then what could be provided by a single-texture camera. The texture information was generally the same from frame to frame even though a new geometry was generated each time, and this perceptually overrode the changing geometry. Blending two or more textures is not a straightforward process. In order to blend two images together the registration, or overlap between one another must be known. The blended textures often suffered from discontinuities as in a region where one input image is favored, by mathematical comparision, over another. This can often be seen along edges of occlusion on an object.

The Global Illumination techniques, despite their simple implementations and coarse approximation, provided convincing additional realism. The limited shadows and reflections worked since human perception is generally unable to determine which light a shadow is generated by or how light transport is hitting a surface to cause a reflection.

6.2 Future Work

The Polyhedral Hull Online Compositing System demonstrated that it is possible to merge imagery from real and synthetic scenes with view independence and have have plausible results. Future systems could build off of this research and improve the results to obtain superior quality systems to those available today which still utilize two-dimensional techniques.

The first area of improvement is the reconstruction of the geometry. The most important improvement would be to generate a single watertight mesh with a tight bound on the actual object being reconstructed. More cameras and processing power would aid this process, but the reconstruction algorithm itself may need to

be modified as well.

Another area that needs improvement is the generation of global illumination effects. Currently, the global illumination effects only support a single shadow and a single reflection. A significant improvement could be achieved by utilizing a system which can dynamically update objects on a per-frame basis. This would allow for the support of support more complex global illumination effects including interreflections and multiple shadows.

Finally, the texturing process offers opportunities for vast improvement. Simple improvements such as adding more cameras could help to provide a better prototype, but would not completely solve the problem. Algorithms and techniques need to be developed to resample, interpolate, and re-use texture data in a meaningful way from frame to frame. In order to re-use texture data, correspondences would have to be registered on a per-pixel basis from camera to camera. These correspondences would indicate which pixels are the same in each image. Currently only the relationships between the physical locations of the cameras in a scene are known, not the information that a camera captures. Pixel to pixel correspondences between the cameras could greatly aid in fully view-independent texture reconstruction that looks accurate.

6.3 Summary

This thesis focused on the texturing and reflections process of the Polyhedral Hull Online Construction System. The system utilizes three-dimensional information to merge live video with synthetic imagery such that a visually plausible composite can be generated in real-time, allowing the end-user to navigate around the composited environment and view both the real and synthetic objects together. The

prototype system accomplished the overall goal: to merge the reconstructed three-dimensional objects from live video imagery and the computer-generated synthetic scenes in a globally illuminated environment at interactive rates. Hopefully, in the future, this system will be improved upon by reducing computation times and creating more realistic images with more advanced global illumination integration. The results shown in this thesis are exciting and we hope they motivate new work on this topic.

Appendix A

Camera Calibration

The camera calibration toolbox for MATLAB was used exclusively for determining the intrinsic and extrinsic parameters of the cameras in our system. The input to the toolbox is a sequence of digital images, each containing the surface of a checkerboard. The user selects the four corners which define the outer extents of the checkerboard, and the software can then perform automatic extraction of the internal grid corners. As the toolbox computes the grid corners, it prompts the user to enter a value for the adjustment of radial distortion (if desired). The main calibration routine can then be executed, which utilizes a gradient descent technique to minimize the reprojection error of the grid pixels. This returns the internal and external parameters of the camera.

The format of the returned data is such that each camera is considered its own world reference frame, and the rotation and translation vector to the grid origin are given. This is not ideal for our purposes, as we require a single world reference frame, in terms of which each camera basis and position are given. To make this conversion, we invert each camera's rotation matrix, by taking its transpose, and then we negate each camera's translation vector and multiply it by the camera's new rotation matrix. In MATLAB syntax, this resolves to the following for a single camera:

Camera1_RotationMatrix = Rc_1'

*Camera1_Translation = Camera1_RotationMatrix * (-Tc_1)*

Where "Rc_1" is the rotation matrix returned by the toolbox for Camera 1 and "Tc_1" is the translation vector returned by the toolbox for Camera 1. After we have the location and pose of each camera in terms of a global reference frame, we convert the reference frame such that it is the same right-handed basis that our system uses internally. In our system, the x-axis is positive to the left, the y-axis is positive in the up direction, and the z-axis is positive going into the screen. The MATLAB syntax to make this conversion for a single camera is:

```
Cam1_RotTmp = [Cam1_RotMat(2,:); -Cam1_RotMat(1,:); Cam1_RotMat(3,:)]
Cam1_RotationFixed = [-Cam1_RotTmp(:,1) -Cam1_RotTmp(:,2) Cam1_RotTmp(:,3)]
Cam1_TranslationFixed = [Cam1_Trans(2,:); -Cam1_Trans(1,:); Cam1_Trans(3,:)]
```

The "fixed" rotation matrices and translation vectors, along with the principal point and focal length, are stored in a settings file that is loaded into our program each time it is executed. This provides the necessary information for computing the cameras' projection matrices as well as the fundamental matrices which relate the cameras.

Appendix B

Trigger Circuit Design

Each of the DFW-X700 cameras has an external trigger that can be used to drive the capture process, by sending a low pulse of at least 1MS duration on the "TRIG IN" connector, pin 3. The cameras have an internal pull-up resistor, so in order to trigger the low signal, we simply pull pin 3 to ground when we want an image to be captured. For our trigger design, we take advantage of the serial port, COM1, on the central server. According to the RS232 standard, the serial port transmits a '1', or logic high, as -3 to -25 volts and a '0', or logic low, as +3 to +25 volts. We use the data transmit pin on the DB9 serial port connection to control our trigger circuit. When no data is being actively transmitted, our serial port outputs -11 volts on the data transmit pin. When we want the signal to go high, we write out a packet of zeros, and the data transmit pin goes to +7.5 volts.

The trigger circuit is quite simple and consists of an NPN transistor and a 2.2k resistor. The signal ground (SG) from the serial port, pin 5, is tied to the emitter pin on the 2N3904 transistor, as is the ground from each of the four trigger cables which run to the cameras. The data transmit pin (DT), or pin 3, from the serial port is tied to the base pin on the transistor through the 2.2k resistor, and the collector pin is attached to the positive leads of the trigger cables. When our software, running on the server, writes a string of zeros to the serial port, the DT pin goes high, permitting the flow of electrons from the emitter to the collector, and pulling the "TRIG IN" pins on each of the cameras to ground. This causes the trigger to fire, and an image to be captured.

BIBLIOGRAPHY

[BN76] J.F. Blinn and M.E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, Oct 1976.

[Deb98] Paul Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 189–198, July 1998.

[Deb02] Paul Debevec. Image-based lighting. *IEEE Computer Graphics & Applications*, 22(2):26–34, March 2002.

[DSZ00] Paul Viola Dan Snow and Ramin Zabih. Exact voxel occupancy with graph cuts. *IEEE Conference on Computer Vision and Pattern Recognition*, June 2000.

[DTM96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of SIGGRAPH 1996*, Computer Graphics Proceedings, Annual Conference Series, pages 11–20, August 1996.

[DWT+02] Paull Debevec, Andreas Wenger, Chris Tchou, Andrew Gardner, Jamie Waese, and Tim Hawkins. A lighting reproduction approach to live-action compositing. *ACM Transactions on Graphics*, 21(3):547–556, July 2002.

[KZ01] Vladimir Kolmogorov and Ramin Zabih. Computing visual correspondence with occlusions via graph cuts. In *International Conference on Computer Vision*, pages 508–515, 2001.

[Lau94] Aldo Laurentini. The visual hull concept for silhouette based image understanding. *IEEE Pattern Analysis and Machine Intelligence*, 16:150–162, February 1994.

[Let04] Henry Letteron. Polyhedral hull online compositing system:reconstruction and shadows. Master's thesis, Cornell University, August 2004.

[LH81] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[Mat01] Wojciech Matusik. Image-based visual hulls. Master's thesis, Massachusetts Institute of Technology, 2001.

[MBM01a] Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for real-time rendering. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, pages 115–126, June 2001.

[MBM01b] Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for real-time rendering. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, pages 115–126, June 2001.

[MBR+00] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 369–374, July 2000.

[MLS03] Marcus Magnor Ming Li and Hans-Peter Seidel. Online accelerated rendering of visual hulls in real scenes. In *Journal of WSCG*, volume 11, February 2003.

[MP76] D. Marr and T. Poggio. Cooperative computation of stereo disparity. *Science*, 194(4262):283–287, October 1976.

[OK93] Masatoshi Okutomi and Takeo Kanade. A multiple-baseline stereo. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 15, 1993.

[PSP03] Leonard McMillan Peter Sand and Jovan Popovic. Continous capture of skin deformation. In *ACM Transactions on Graphics*, volume 22, pages 578–586, 2003.

[PTG02] B. Walter P. Tole, F. Pellacini and D. P. Greenberg. Interactive global illumination in dynamic scenes. In *SIGGRAPH 2002 Conference Proceedings*, pages 537–546, 2002.

[SAK+02] H. S. Sawhney, A. Arpa, R. Kumar, S. Samarasekera, M. Aggarwal, S. Hsu, D. Nister, and K. Hanna. Video flashlights - real time rendering of multiple videos for immersive model visualization. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, pages 157–168, June 2002.

[SD99] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):151–173, November 1999.

[Sel03] Jeremy Adam Selan. Merging live video with synthetic imagery. Master's thesis, Cornell University, 2003.

[SK02] Steven M. Seitz and Kiriakos N. Kutulakos. Plenoptic image editing. *International Journal of Computer Vision*, 48(2):115–129, July 2002.

[TT94] C. Tomasi and Kanade T. Shape and motion from image streams under orhtography: A factorization method. *International Journal of Computer Vision*, 9(2):137–154, 1994.

[Ull79] S. Ullman. *The Interpretation of Visual Motion*. MIT Press, 1979.

[YB01] Ramin Zabih Yuri Boykov, Olga Veksler. Fast approximate energy minimization via graph cuts. *IEEE Pattern Anaylsis and Machine Intelligence*, 23(11), November 2001.

[YP03] Ruigang Yang and Marc Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *IEEE Computer Vision and Pattern Recognition*, 2003.

[Zha00] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, 2000.