Novel Volumetric Scene Reconstruction Methods for New View Synthesis

A Thesis
Presented to
The Academic Faculty

by

Gregory G. Slabaugh

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Electrical and Computer Engineering
Georgia Institute of Technology
November 2002

Novel Volumetric Scene Reconstruction Methods for New View Synthesis

Approved by:

Russell M. Mersereau, Committee Chair

Greg Turk

Ronald W. Schafer, Advisor

Mat C. Hans

Anthony J. Yezzi

Date Approved Nov. 12, 2002

This thesis is dedicated to the memory of my mother, Phyllis G. Slabaugh.

ACKNOWLEDGEMENTS

During my graduate career, I have been blessed with many wonderful relationships. I would like to take this opportunity to express my gratitude to those who have helped shape this thesis.

I am most grateful to my advisor, Dr. Schafer, with whom it has been both an honor and a pleasure to work. It was Dr. Schafer who introduced me to the problem of new view synthesis. His open-minded and flexible approach allowed me to forge my own path as a graduate student. I thank him for being an outstanding advisor – insightful, helpful, kind, and patient.

Next I would like to thank Hewlett-Packard Laboratories, which supported and contributed to the research described in this thesis. I thank Fred Kitson, whose unwavering commitment to our work is truly appreciated. I am especially grateful to Tom Malzbender and Bruce Culbertson, with whom I have collaborated throughout my graduate career. Their input is reflected significantly in this thesis. I appreciate all the opportunities that HP has offered me, including two internships and extended collaborations. Mat Hans deserves a great deal of thanks for his assistance in setting up our lab as well as his insights on several projects. I would also like to thank all previous and current HP Labs employees who have collaborated with me on 3D scene reconstruction, particularly Dan Gelb, Mark Livingston, Irwin Sobel, and the rest of the Coliseum team.

I thank my friends and colleagues at the College of Computing, who helped keep me abreast of the latest developments in computer graphics and vision, collaborated with me on projects, and discussed novel research ideas with me. In particular, Greg Turk, Quynh Dinh, Eugene Zhang, Gabe Brostow, and Drew Steedly and deserve special recognition.

I'd also like to thank Leonard McMillan, Wojciech Matusik, and Chris Buehler, who

shared their Image-Based Visual Hulls code and system configuration with us.

Our group, the Center for Signal and Image Processing (CSIP), is at the forefront of teaching and research in digital signal processing. I am grateful to the professors from whom I have taken classes, and/or have opened their doors to me. This includes Dr. McClellan, Dr. Meresereau, Dr. Yezzi, Dr. Smith, Dr. Jackson, and Dr. Zhou.

While a CSIP graduate student I had the opportunity to befriend many fellow graduate students. In particular, I would like to acknowledge Jordan Rosenthal, Amer Abufadel, John Glotzbach, Lisa McCrickard, Gail Rosen, Robert Morris, Ghassan Al-Regib, Sam Li, Raviv Raich, Yen-Chi Lee, Sangkeun Lee, Nik Vasiloglou, Jailin Tain, Arden Huang, and Chris Lanciani.

I owe a sincere thanks to the CSIP staff who provided administrative support, especially Kay Gilstrap, Christy Ellis, and Stacy Shultz. I am also grateful to Sam Smith and Keith May who helped with computer support.

I express much gratitude to and love for my family; my father and his wife, Ronald and Marilyn Slabaugh, my brothers and their wives, Eric and Maria Slabaugh, and Jason Slabaugh and Allison Foulds, my aunt Mary Morris, as well as my extended family. Also thanks to my friends, John and Alina Stevenson, Chuck Chung, Pietro Bonacossa, and Mark Kelley who helped keep me sane by pulling me out of the lab every now and then to enjoy life.

I would especially like to thank my partner Stephen Blanchard, for his alacrity, patience, and love.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

LIST OF TABLES xi

LIST OF FIGURES xii

SUMMARY xvi

I INTRODUCTION 1

1.1 Problem statement 1

1.2 Motivation 1

1.3 Volumetric approaches 2

1.4 Contribution of the thesis 3

1.5 Organization of the thesis 3

II RELATED WORK 5

2.1 Image-based rendering 6

2.2 Stereo matching approaches 8

2.2.1 A brief review of stereo vision 8

2.2.2 Multi-view stereo reconstruction 11

2.3 Structure from motion 13

2.4 Interactive modeling systems 14

2.5 Volumetric approaches 16

2.5.1 Visual hulls 17

2.5.2 Photo hulls 21

2.5.3 Level set methods 25

2.6 Summary 26

III GENERALIZED VOXEL COLORING 28

3.1 Photo-consistency 28

3.1.1 Visibility . . . 28
3.1.2 A single threshold measure . . . 30
3.1.3 An adaptive threshold measure . . . 32
3.2 Voxel coloring - a single planar sweep . . . 33
3.3 Arbitrary camera placement . . . 34
3.4 Space carving: multiple planar sweeps . . . 36
3.5 Generalized voxel coloring . . . 37
3.5.1 GVC-IB . . . 37
3.5.2 GVC-LDI . . . 39
3.5.3 Results . . . 42
3.5.4 Analysis . . . 49
3.5.5 Multi-resolution GVC . . . 50
3.6 Summary . . . 52
IV POST-PROCESSING **54**
4.1 Geometric inaccuracies . . . 54
4.1.1 Theoretical sources . . . 54
4.1.2 Practical sources . . . 55
4.2 Mathematical morphology . . . 62
4.2.1 Basic morphological operations . . . 62
4.2.2 Morphological filtering of reconstructions . . . 63
4.3 Volumetric optimization . . . 64
4.3.1 Optimal reconstructions . . . 66
4.3.2 Reprojection error and standard deviation . . . 67
4.3.3 Optimal scene reconstructions exist but are not unique . . . 68
4.3.4 The search space is large . . . 69
4.3.5 Minimizing the reprojection error . . . 70
4.3.6 Results and Analysis . . . 73
4.4 Other methods . . . 77
4.5 Summary . . . 77

V VOLUMETRIC WARPING **80**

5.1 Background modeling 80

5.2 Volumetric warping 82

5.2.1 Frustum warp 83

5.2.2 Other warping functions 87

5.3 Implementation issues 87

5.3.1 Cameras inside volume 88

5.3.2 Preventing visible holes in the outer shell 89

5.4 Results 89

5.5 Summary 91

VI SPACE CARVING USING LEVEL SET METHODS **94**

6.1 Surface representation 94

6.2 Level set methods 95

6.2.1 Description 95

6.2.2 Evolution of the surface 96

6.2.3 Two common flows 97

6.3 Reconstruction algorithm 98

6.4 Implementation 99

6.4.1 Voxel space 100

6.4.2 Derivatives 100

6.4.3 Stability 101

6.4.4 Narrow band 101

6.4.5 Multi-resolution 101

6.5 Results 102

6.6 Summary 104

VII IMAGE-BASED PHOTO HULLS **109**

7.1 Image-based visual hulls 109

7.1.1 Overview 109

7.1.2 Computing geometry . . . 112
7.1.3 Computing visibility . . . 113
7.1.4 View-dependent texture mapping . . . 115
7.2 Image-based photo hulls . . . 116
7.2.1 Approach . . . 116
7.2.2 Stepping along epipolar lines . . . 117
7.2.3 IBPH visibility . . . 119
7.2.4 Sampling . . . 119
7.2.5 Convergence . . . 121
7.2.6 Spatial coherence . . . 122
7.3 Results . . . 123
7.4 Summary . . . 127

VIII EVALUATION **129**

8.1 3D error evaluation . . . 130
8.1.1 3D error metric . . . 132
8.1.2 Visual hull reconstruction . . . 133
8.1.3 GVC reconstruction . . . 134
8.1.4 Volumetric optimization . . . 137
8.1.5 Level set reconstruction . . . 138
8.1.6 Image-based photo hulls reconstruction . . . 139
8.1.7 Summary of 3D error analysis . . . 141
8.2 New view synthesis evaluation . . . 145
8.2.1 2D mean square error . . . 145
8.2.2 Visual hull . . . 146
8.2.3 Generalized voxel coloring . . . 148
8.2.4 Volumetric optimization . . . 151
8.2.5 Level set method . . . 151
8.2.6 Image-based visual hulls . . . 154
8.2.7 Image-based photo hulls . . . 154

8.2.8 Summary of 2D mean square error analysis 156

IX CONCLUSION **160**

9.1 Review of contributions 160

9.2 Future work 161

APPENDIX A — CAMERA GEOMETRY **163**

APPENDIX B — MULTI-VIEW TRIANGULATION **168**

APPENDIX C — EXPERIMENTAL SETUP **180**

REFERENCES **182**

VITA **193**

LIST OF TABLES

1 Data sets and parameters for GVC reconstructions 47

2 Runtime and memory usage for GVC reconstructions 47

3 Multi-resolution voxel spaces and runtimes 52

4 Results for the Toycar scene 74

5 Results for the Shoes scene 76

6 Volumetric optimization of the SynthPlane data set 138

7 3D error analysis results 143

8 Mean square error analysis results 159

LIST OF FIGURES

1 Voxel classifications 16
2 Visual hull reconstruction 18
3 An example visual hull 20
4 An example photo hull 23
5 Photographs of a cloth toy 24
6 Reconstruction of a cloth toy 25
7 Using photo-consistency to identify scene surfaces 29
8 Visibility is required to correctly compute photo-consistency 29
9 Reconstructing texture and edges 31
10 Voxel coloring via a planar sweep 34
11 Pseudo-code for space carving algorithms. 35
12 GVC data structures 38
13 GVC-IB Pseudo-code 40
14 Voxels that change visibility 42
15 GVC-LDI Pseudo-code 43
16 Two of 17 reference views of a toy model of Ghirardelli Square 44
17 Visualization showing camera placements and reconstruction volume 45
18 An example reconstruction using GVC-LDI 45
19 New view synthesis via rendering of the reconstruction 46
20 Reconstruction results for several data sets 48
21 GVC reconstructs a thin-shelled surface 50
22 Flow diagram for multi-resolution GVC implementation 51
23 Cusps and floating geometry 55
24 Reconstructions exhibiting cusps and floaters 56
25 Photometric differences can occur when moving a camera in a scene 59
26 Vignetting 60
27 The effect of varying photo-consistency threshold 61

28 Morphological filtering of surfaces 65
29 Model refinement by connected components analysis 66
30 The volumetric reconstruction search space 70
31 Toycar data set: two of 17 reference views 74
32 Reprojection error vs. time for Toycar scene 75
33 Refinement of the Toycar reconstruction 76
34 Reprojection error vs. time for Shoes scene 77
35 Close up of cloth toy 78
36 Unknown regions because of reconstruction on a finite domain 81
37 Pre-warped and warped voxel spaces in 2D 82
38 Constant footprint property 84
39 Boundaries and regions used by the warping function 85
40 Warping a point using the warping function 86
41 Pre-carving initialization 89
42 Reconstruction of the marbles scene 91
43 Reconstruction of the Stanford scene 92
44 Surface evolution along surface normal 96
45 Two common flows 97
46 Narrow band implementation 102
47 Initial evolution of the zero level set at the lowest resolution. 103
48 New synthesized views of the broccoli stalk 103
49 Four of eleven reference views of the Camel data set 105
50 New synthetic views of the Camel data set 106
51 New synthetic views of the Tower data set 107
52 Visual hull vs. photo hull for the Pinwheel data set 111
53 View-dependent geometry 112
54 Determining a ray's visual hull intervals 113
55 Visibility computation using hull intervals 114
56 View-dependent texture-mapping 115

57 Pseudo-code for the IBPH algorithm 116
58 Computing the image-based photo hull 117
59 Stepping along an epipolar line 118
60 Multi-resolution IBPH sampling 120
61 Changing the size of the sampling lattice 121
62 Ray classifications vs. iteration 122
63 Varying the spatial coherence parameter 123
64 Using IBPH in a real-time 3D telepresence application 125
65 Comparing IBPH to GVC 126
66 Camera placements and reconstruction volume for the SynthPlane scene . . 131
67 Three of 24 reference views for the SynthPlane scene 131
68 Volume under surface. 132
69 Zooming in on the GVC height map 136
70 Reconstructions of the SynthPlane data set. 142
71 3D error vs. time, SynthPlane scene 144
72 Camera placement used for new view synthesis evaluation 146
73 An image formed by rendering the true scene to viewpoint V 147
74 New view synthesis results for the visual hull algorithm 149
75 New view synthesis results for the GVC algorithm 150
76 New view synthesis results for the volumetric optimization algorithm . . . 152
77 New view synthesis results for the level set algorithm 153
78 New view synthesis results for the IBVH algorithm 155
79 New view synthesis results for the IBPH algorithm 157
80 Mean square error vs. time, SynthPlane scene 158
81 Epipolar geometry 166
82 Correspondence search using epipolar geometry 167
83 A common method for stereo triangulation 169
84 Finding the shortest distance between a point and a ray 170
85 An example showing our multi-view triangulation algorithm 173

86 Triangulation is ill-posed when all rays are parallel or anti-parallel 177

87 Our experimental setup for interactive new view synthesis 181

SUMMARY

This thesis describes methods that generate a digital three-dimensional (3D) model of a visual scene's surfaces, using a set of calibrated photographs taken of the scene. The 3D model is then rendered to produce views of the scene from new viewpoints. In the literature, this is known as 3D scene reconstruction for new view synthesis. This thesis introduces novel approaches that improve upon the quality, efficiency, and applicability of existing methods.

To achieve a high quality reconstruction, it is essential to know which cameras have visibility of local areas on the surface. Accordingly, we present a related pair of techniques for computing visibility during a volumetric reconstruction. We then describe post-processing methods that refine surface reconstructions to improve model fidelity.

We explore different representations for modeling the 3D surface during reconstruction. We introduce a method of warping the 3D space to represent large-scale scenes. We also investigate a level set approach, which embeds the 3D surface as the zero level set of volumetrically sampled function. Finally, we present a view-dependent representation that can be computed at interactive rates.

CHAPTER I

INTRODUCTION

1.1 Problem statement

One of the primary ways we sense the world is through vision. The human visual system acquires light that passes through the lens of the eye and is imaged on the retina. This visual data, rich in information, is used to perceive shape, position, and color.

Mimicking this process, researchers have developed imaging sensors and computer algorithms to analyze visual information. The goal of this thesis is to generate a 3D digital representation of a visual scene using a set of photographs. The photographs are taken from multiple viewpoints of either a static or dynamic scene. As a pre-process, each camera used to photograph the scene is calibrated, yielding the geometry of the image formation process. Given these calibrated photographs, how does one compute a 3D model that can be rendered to synthesize views of the scene at new viewpoints? This is the central problem addressed in this thesis.

1.2 Motivation

A key challenge in computer graphics is photo-realistic image synthesis of real-world scenes. Unfortunately, many real-world scenes consist of geometrically complex shapes that are difficult to recreate in a 3D modeling program. Even after hours of tedious editing, many CAD-based 3D models produce synthetic images that lack a desired level of photo-realism when rendered.

An alternate approach is image-based modeling. The objective in this approach is to create a 3D model using 2D image data. Often this 2D image data consists of photographs, also called *reference views*, taken from multiple viewpoints. In contrast to CAD-based 3D

surface modeling, taking photographs is simple and intuitive. The photographs are input to a computer, which analyzes the photographs to reconstruct a 3D model of surfaces in the scene. This model can the be rendered to produce photo-realistic images from new viewpoints.

3D scene reconstruction is a fundamental problem in computer vision and has many applications, including robot navigation [47], reverse engineering [26], object recognition [7, 77], motion capture [104], model building [27, 126], teleconferencing, and more. Thus, advances in 3D scene reconstruction can significantly impact a variety of applications.

Our application is new view synthesis, the task of generating views of the scene from new viewpoints. Conceptually, new view synthesis endows a user a virtual camera, which the user places about the scene. The goal then is to synthesize the image that a physical camera would see when placed at the same position the virtual camera is placed. We perform 3D scene reconstruction to generate a photo-realistic 3D model of the scene, and then render the 3D model to synthesize an image at the virtual viewpoint.

1.3 Volumetric approaches

In this thesis, we focus on volumetric reconstruction approaches. By *volumetric* we mean that space has been tesselated into discrete volumes called *voxels*. We typically model a voxel as a small cube in 3D space. The voxels are arranged on a 3D sampling lattice, forming a *voxel space*. Within this voxel space, our goal is to find the 3D surfaces that, when rendered, reproduce the photographs taken of the scene

Any surface in the voxel space can be sampled to produce a volumetric representation. Volumetric surface representations provide a computationally convenient and topologically flexible way to characterize a surface in 3D space. When necessary, a voxel-based model can be converted into a polygonal representation for efficient rendering on standard graphics hardware.

1.4 Contribution of the thesis

This thesis examines novel volumetric scene reconstruction approaches, designed for the application of new view synthesis. We advance the state of the art by introducing the following:

1. Two novel methods for computing voxel visibility during scene reconstruction.
2. Methods to post-process the reconstructed geometry, including a volumetric optimization strategy that minimizes reprojection error.
3. A volumetric warping approach for reconstructing large-scale scenes.
4. A reconstruction technique that marries the speed and simplicity of space carving with the advantages of level set methods.
5. An efficient view-dependent reconstruction approach that is capable of interactive reconstructions.

1.5 Organization of the thesis

The rest of this thesis is organized as follows. Chapter 2 presents prior art in the field of 3D scene reconstruction, focussing primarily on related volumetric approaches. Chapter 3 describes our generalized voxel coloring (GVC) algorithm, which introduces two novel approaches for determining visibility during a volumetric reconstruction. In Chapter 4 we extend this work by post-processing the reconstructed 3D model to improve model fidelity.

We then introduce a warping of 3D space in Chapter 5 that efficiently represents a large-scale scene. In Chapter 6, we utilize level set methods to implicitly represent the 3D surface as a zero-level set of a volumetrically sampled function. In Chapter 7, we present a view-dependent algorithm, image-based photo hulls (IBPH), designed for interactive new view synthesis. In Chapter 9 we review the contributions of this thesis and explore directions for future research.

Appendix A offers a brief review of single a multi-view camera geometry, and Appendix B presents a multi-view triangulation algorithm. Appendix C describes our experimental setup for interactive 3D scene reconstruction and new view synthesis.

CHAPTER II

RELATED WORK

The quest for photo-realism in computer graphics has led to very sophisticated modeling of geometry, light, and reflection. Developments in this field have yielded powerful approaches, as demonstrated in many contemporary feature films. However, much skill, artistry, and tedious labor is required to model and render such complex scenes.

Consequently, there has been in recent years much interest in applying computer vision methods to capture real-world scenes from photographs. Photographs are very simple to obtain, and by definition, are photo-realistic. However, one of the limitations of a photograph is its fixed viewpoint. This chapter reviews new view synthesis techniques, which circumvent this limitation to generate photo-realistic synthetic views of the scene at new viewpoints.

One distinguishing characteristic of new view synthesis techniques is that of scene representation, spanning 2D representations for view morphing, 2-$\frac{1}{2}$D representations for view interpolation, 3D model reconstruction, 4D light fields, 7D plenoptic functions, and beyond. In this review, we give special attention to methods that reconstruct 3D models, since that is the approach taken in this thesis.

During the last decade, the topic of new view synthesis has received considerable interest in the computer graphics, computer vision, and image processing domains, and the resulting literature on the subject is vast. To make this review tractable, we limit its scope to methods that are passive with respect to the scene lighting. Active vision methods that project light into the scene, such as structured lighting and laser range imaging techniques are not presented. A goal of many new view synthesis methods is to produce photo-realistic new views using off-the-shelf photographic cameras, which are inexpensive and easy to

use.

2.1 *Image-based rendering*

We begin this review by briefly surveying image-based rendering algorithms. *Image-based rendering* can be defined as a class of techniques that generate synthetic images from photographs rather than geometric primitives. The goal of image-based rendering is new view synthesis. A related class of techniques are *image-based modeling* algorithms, which produce a 3D model of the scene using the photographs. This model can then be rendered for view synthesis. Image-based modeling algorithms will be considered later in this chapter. The algorithms presented in this section do not compute an explicit 3D model of the full scene geometry.

One of the earliest attempts at image-based rendering was the Movie-Map system [61] presented in the early 1980s. This system represents an entire scene as a collection of panoramic images, acquired every 10 feet along several streets of a town. As a user navigates a virtual camera along one of the streets, the system displays an optically corrected view using the image closest to the virtual camera position. While the system is capable of displaying virtual camera rotations, the camera translations are limited to the positions at which the scene was photographed. This approach was recently revisited by Kimber, Foote, and Lertsithichai [53], who photograph the scene much more densely along each path.

In their seminal work, Chen and Williams [20] demonstrated that it was possible to interpolate between viewpoints if dense correspondences are known. The correspondences are used to establish optical flow. Pixels in a reference image are then linearly interpolated along the optical flow vectors to produce intermediate views. While effective, this approach correctly synthesizes new views for only limited virtual camera motions.

In a similar spirit, many other researchers introduced new view synthesis algorithms that warp photographs using correspondence and/or depth information [3, 17, 18, 49, 58,

66, 68, 74, 98, 99, 101, 103]. Notable developments among this class of techniques include the use of epipolar geometry [58], an analysis of the physical validity of view interpolation [99], an occlusion-compatible rendering order that obviates the need for depth comparisons [74], extensions to time-varying scenes [66], and generation of within-scene views without requiring explicit correspondence information [49]. Some of the key challenges to these approaches are establishing correspondences between photographs, synthesizing views from arbitrary viewpoints, and dealing with holes in the synthesized view that result from disocclusions.

A number of researchers have introduced image-based rendering algorithms based on panoramic imagery. Such use of panoramic imaging for view synthesis has its roots in environment mapping [9, 45]. Chen's Quicktime VR [21] approach captures cylindrical panoramas by stitching together photographs taken by a camera rotating 360 degrees about its center of projection. Other researchers extend this idea by stitching images taken from arbitrary camera placements [96, 118]. McMillan and Bishop [74] use cylindrical panoramas in conjunction with stereo correspondence algorithms for view synthesis.

Adelson and Bergen [1] introduce the *plenoptic function*, a 7-dimensional function that characterizes all of the light flowing in 3D space, in all directions, positions, time, and at all wavelengths. As McMillan and Bishop [74] point out, the goal of image-based rendering is to generate a continuous representation of the plenoptic function given a (typically incomplete) set of samples.

While the plenoptic function is a useful theoretical concept, in practice a complete sampling of it would result in an overwhelming amount of data for most scenes. When working with static scenes and sampling the color spectrum in RGB channels, the plenoptic function becomes 5-dimensional. In the lumigraph [44] and light field [60] approaches, it is observed that in the absence of occlusions, the dimensionality can be further reduced to four. These approaches sample the scene as two-dimensional arrays of two-dimensional images. Synthesis of a new view is achieved by resampling the rays captured in this 4D

representation. This work spawned a flurry of subsequent light field research, developed for alternative capture topologies [2, 106], efficient rendering [14, 92] and compression [65]. Surface light fields [22, 75, 129] are a related representation that assign a color to each ray originating from a surface. Surface light fields have shown much promise but do require a 3D model of the surfaces in the scene.

The remainder of the techniques presented in chapter are image-based modeling algorithms. These approaches are more closely related to the techniques developed in this thesis.

2.2 *Stereo matching approaches*

Stereo vision is the ability to see in three dimensions. The binocular human visual system captures light from two slightly different positions. As a result, an imaged object appears slightly displaced in the two retinal images. This displacement is known as *disparity*, and is used by the brain to estimate depth.

Computer vision methods have been developed to simulate this process. Stereo rigs consist of a set of cameras that photograph the scene. Disparity is computed from the photographs by matching corresponding regions between the images. In this section, we review these stereo vision algorithms. We begin with a brief review of two-view approaches to illustrate the basic concepts. We then discuss multi-view stereo algorithms designed for image-based modeling.

2.2.1 A brief review of stereo vision

A standard two-view stereo vision rig consists of two cameras, a left and a right, separated by distance known as the *baseline*. Photographs of a scene are taken simultaneously with the two cameras. Given these two photographs, stereo vision seeks to find the depth of scene surfaces.

Key to stereo vision is the correspondence problem. This problem requires one to

identify, for each scene element (pixel, line, corner, etc.) in one image, a matching scene element in the other. This match is called a *correspondence*. Correspondence algorithms can be classified into correlation-based methods and feature-based methods.

Correlation-based methods attempt to find, for a small patch around a pixel in one image, a match in the other image. A similarity measure that indicates the quality of a prospective match is maximized. A cross correlation similarity measure can be used,

$$\phi_{lr}[n_1,n_2] = \sum_{k_1=-N}^{N} \sum_{k_2=-M}^{M} I_l[n_1+k_1,n_2+k_2] I'_r[k_1,k_2],$$

as well as other correlation-like measures such as the sum of squared differences (SSD),

$$SSD_{lr}[n_1,n_2] = -\sum_{k_1=-N}^{N} \sum_{k_2=-M}^{M} (I_l[n_1+k_1,n_2+k_2] - I'_r[k_1,k_2])^2,$$

where a $(2N+1)$ x $(2M+1)$ patch, $I'_r[n_1,n_2]$, is extracted from the right image, $I_r[n_1,n_2]$, and compared in the left image $I_l[n_1,n_2]$. While the equations above assume grayscale images, they can be easily extended to color images. A correspondence is often assumed to exist for the maximal value of the similarity measure. Disparity is computed for each pixel as the image plane difference in spatial position of the corresponding points.

Feature-based methods match image features, such as lines, edges, and corners. Such features rarely occur at every pixel in an image, and as a result, feature-based methods find only a sparse set of correspondences. The computer vision literature is rife with various feature-based correspondence algorithms for virtually every type of feature.

The correspondence problem is challenging. It is ill-posed, since as a result of occlusions, a match might not exist. In addition, in homogeneously colored surface regions, multiple matches may occur. To make matters worse, correspondences become more difficult to establish when the baseline increases, and when the reflectance of scene surfaces deviates from Lambertian.

However, additional constraints can be imposed to help minimize false matches. Of these, perhaps the most useful is the *epipolar constraint* [36, 123], since it is always valid. For a point in image 1, one can back-project a ray into 3D space that intersects the 3D

feature generating the point. This ray, when projected into image 2, defines an epipolar line, along which the correspondence must lie. Thus, the epipolar constraint reduces the 2D search for a correspondence to a 1D search along an epipolar line. Epipolar geometry is reviewed in Appendix A.

Other constraints used in stereo matching are not applicable to all scenes, but are useful for many scenes of interest. The *uniqueness constraint* [69] requires that there can be no more than one match in image 2 for a pixel in image 1. This constraint has been further used to require that matches are bi-directional. The *continuity constraint* [69] takes advantage of the cohesiveness of matter by requiring the disparity vary smoothly over most of the image. Note that this constraint does not hold at depth discontinuities. The *ordering constraint* [4] requires that the order of neighboring correspondences on the corresponding epipolar lines is preserved. The *disparity limit constraint* specifies a band of allowable depth values for which to search for correspondences. Other constraints are possible. By taking advantage of these constraints, stereo matching algorithms produce much better results.

More recent work on stereo matching has seen the incorporation of color segmentation for textureless regions [120], inclusion of temporal consistency for dynamic scenes [15, 79, 119, 131], use of interline consistency of epipolar lines via graph cuts [94], development of multi- and wide baseline approaches [82, 90, 97, 125], rejection of outliers [8, 117], and efficient implementations for real-time performance.

If the cameras are calibrated, i.e., the geometry of image formation in the reference views is known, the correspondences can be triangulated to reconstruct 3D structure in a metric space. In the two-view case, the 3D information is typically represented as a depth map, $z(x,y)$, for each pixel that has been matched in a reference view. This $2\frac{1}{2}$D representation captures the scene geometry as seen by the reference view. The depth map can then be texture-mapped and rendered to synthesize new views of the scene. However, since the depth map does not capture the full 3D geometry of the scene, the new view synthesis will be effective for only a limited set of virtual viewpoints near the two reference

views. To reconstruct the full scene geometry, a reconstruction must be performed using multiple images taken from cameras surrounding the scene. This is the subject of the next subsection.

2.2.2 Multi-view stereo reconstruction

We now review multi-view stereo reconstruction methods that execute a stereo correlation-based correspondence algorithm between various subsets of the reference views. These subsets can range from neighboring image pairs [42] to groups of three to six nearby images [78]. This produces multiple depth maps that represent points in 3D space as seen from different viewpoints. For the task of image-based modeling, the goal is then to infer object surfaces from these often noisy points.

In their "virtualized reality" system, Narayanan, Rander, and Kanade [78, 127] reconstruct time-varying events from multi-view video. For data acquisition, the authors initially designed a hemispherical dome consisting of 51 synchronized video cameras that output time-stamped video to a bank of VCRs. Later, they rigged a room using 49 synchronized video cameras, each connected to a computer for direct video digitization. By sampling the scene with many cameras, reconstructions are less likely to have unrecoverable areas that are occluded from all cameras.

For each time instant, a multi-baseline stereo algorithm is executed, resulting in a dense depth map for each camera. Each depth map is embedded into a volumetric space, and fused [26] into a single three-dimensional model. This model is then extracted from the volume using the marching cubes algorithm and texture-mapped by projecting each color image onto the model and accumulating the results using a weighted average based on visibility. New views of the video can be synthesized from arbitrary viewpoints by rendering the polygonal model at each time instant.

Fua and Leclerc [41] have a similar mesh-based approach to the multi-view reconstruction problem. Like [78], the goal is to recover, in world space, a texture-mapped 3D

mesh using all source images. For each stereo pair, their technique performs hierarchical correlation-based matching to find a dense depth map for each image. The depth maps from all source images are combined in world space, resulting in a dense set of unstructured 3D points. These points are fit to quadric patches, clustered into surfaces, and then polygonized into a mesh. The authors then attempt to optimize [41] the mesh to increase the reconstruction accuracy. The cost function landscape contains local minima, so the mesh found prior to optimization must be reasonably close to the true surface for the optimization to be effective. The optimized mesh is then texture-mapped and rendered to synthesize new views. In a similar vein, Isodoro and Sclaroff [50] and Rockwood and Winget [93] demonstrate mesh optimization approaches for 3D scene reconstruction.

The optimization in [41] attempts to find an extremal surface that represents the true surface geometry. An alternate approach formulates stereo matching as an extremal surface extraction problem [19, 94]. In Chen and Medioni's [19] approach, images pairs are first rectified so that correspondence matching occurs over a one-dimensional range of disparity. Then, a correlation coefficient is computed for each pixel $[n1, n2]$ in an image and for each disparity d. Rather than simply pick the best match at each pixel, this method encodes all the correlation coefficients, for each value of d, into a disparity volume $\rho[n1, n2, d]$. The elements of $\rho[n1, n2, d]$ range between zero and one, which indicate low and high probability of a match, respectively. The task is then to extract a maximal surface from $\rho[n1, n2, d]$. This is accomplished picking values close to one as seeds, and then propagating the surface from the seeds by finding locally maximal values. Upon completion, this algorithm finds an accurate disparity surface between two views. This process is repeated on other image pairs, and the results are fused together into a coherent model that is then texture-mapped and rendered to create new views.

Multi-view stereo matching methods are effective in reconstructing scenes using multiple arbitrarily placed cameras. Key issues for these methods are obtaining accurate correspondences, and addressing incorrect matches by using multi-view refinement strategies.

2.3 *Structure from motion*

Structure from motion methods process images over time, observing spatio-temporal changes that are caused by relative motion between a camera and the scene. For image-based modeling, often one is interested in reconstructing a static scene filmed by a moving camera, although other possibilities exist. These structure from motion methods seek to determine the shape, or *structure* of observed objects as well as the relative *motion* of the moving camera from an image sequence. Often, the scene is sampled at video rates, so the spatial differences (disparities) between consecutive frames are much smaller than that of stereo methods, and correspondences are easier to find. Quite a bit of literature on structure from motion exists; Jebara, Azarbayejani, and Pentland [51] provide a nice overview. Here, we focus on methods designed for full perspective cameras, arbitrary camera motion, and scene reconstruction for new view synthesis.

Pollefeys, Koch, Vergauwen, and Van Gool [86] have developed a structure from motion approach that uses video from an uncalibrated camera with a variable focal length. Since 3D structure is poorly estimated for small baselines, the authors track corner features over many views so that the effective baseline is large. This establishes a sparse set of point correspondences. A robust algorithm uses these point correspondences to estimate the fundamental matrix relating consecutive views. Incorporating the epipolar constraint, more correspondences are found, and the epipolar geometry is refined. Points in projective space are computed using triangulation between views. This results in a projective 3D model comprised of sparse points. The authors then show how to self calibrate, which upgrades the reconstruction from projective to metric. The authors rectify the images and then perform dense stereo correlation-based matching between consecutive images. Depth values are refined, and used to build polygonal, texture-mapped model. New views of the scene of the model can then be synthesized by rendering the model.

Zisserman, Fitzgibbon, and Cross [133] [38] have developed a similar structure from motion reconstruction method. Like [86], Zisserman et al. locate corners in the images

and robustly estimate the fundamental matrix between image pairs. Unlike [86], the authors additionally locate line segments in each image and estimate the trifocal tensor using image triplets. The projective location of points and lines are merged among all views, resulting in a sparse set of features in 3D projective space. Adopting Pollefeys et al.'s self-calibration [85] method, these projective features are upgraded to metric. The authors take this sparse metric reconstruction and fit planes to the features using a robust technique. Then, a texture map is extracted from the source image that is most fronto-parallel with each reconstructed plane. Thus, a planar texture-mapped model is recovered.

Other structure from motion work addresses poor [80, 81] and degenerate camera motions [121] as well as dynamic scenes [46].

Perhaps the most significant limitation to structure from motion methods is that they require the images to be closely spaced. Consequently, these methods can become impractical [27] for modeling a larger scale scene, such as a city block, as they could require an unwieldy number of images to capture all surfaces of interest. In comparison, many of the other methods presented in this review have a significant advantage in that they function well with a sparser sampling of the scene.

2.4 Interactive modeling systems

An interactive modeling system is a user-assisted CAD program that utilizes photographs and computer vision methods to produce a photo-realistic 3D model of a scene. By including a human operator in the modeling process, very accurate 3D models can be generated. In fact, some of the most compelling three-dimensional reconstructions from multiple images have been produced using these systems.

Interactive modeling systems are typically designed to reconstruct man-made scenes, often architectural scenes, which are usually composed of basic shapes. Many of these shapes contain parallel and orthogonal lines, a fact often required by the reconstruction program. The input to such a system is a sparse set of photographs. It is important to take

as few photographs as possible, since the user must interact with the images.

Becker and Bove [6], Faugeras et al. [35], and Shum, Hei, and Szeliski [105] present interactive modeling systems. These methods take advantage of parallel and orthogonal lines common to man-made structures. By having a user specify parallel and/or orthogonal lines in image space, the authors show how intrinsic and/or extrinsic camera parameters can be determined. Then, a user identifies features such as lines and planes by clicking on them in the images, and the system solves for the location of the features in 3D space. The reconstructions in these papers contain only simple geometry, and in the case of [6], consist only of planar surfaces. Texture maps are extracted from the images and applied to the reconstructed geometry, producing a model for new view synthesis.

In Debevec, Taylor, and Malik's [27] system, a user instantiates parameterized geometric primitives from a vocabulary of simple shapes, such as boxes, prisms, and surfaces of revolution. Next, the user marks edge features in the images, and marks each corresponding edge on the appropriate geometric primitive. The system then extracts texture maps from the images to apply to each geometric primitive, and computes each primitive's location in 3D space. This typically produces a reasonably accurate model. However, the simple geometric primitives do not capture some of the finer geometric details in the scene. To determine how the actual scene deviates from the approximate model, the authors execute a correlation-based stereo algorithm between pairs of images, producing a dense depth map for each image. This depth information is then used to produce more accurate renderings of the model.

Debevec et al.'s system produces high quality reconstructions that attain a degree of photo-realism unparalleled by most new view synthesis techniques. Their work spurred development of similar systems [28, 88] as well as consumer software packages. However, these systems require a fair amount of user interaction during scene modeling, especially compared to the more automatic scene reconstruction methods presented in this thesis. Another limitation of these systems is that they cannot reconstruct arbitrarily shaped surfaces.

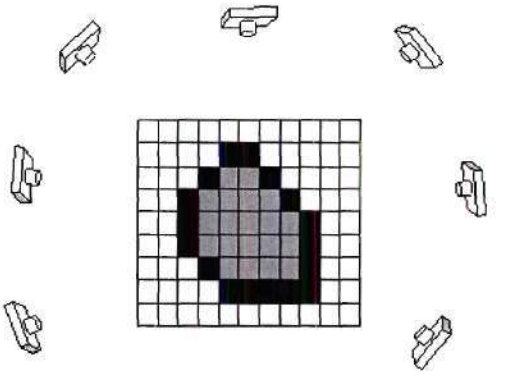


Figure 1: Voxel classifications. For simplicity, this figure shows a 2D voxel space, composed of the three voxel types. Empty space voxels are white, surface voxels are black, and inner voxels are gray.

2.5 Volumetric approaches

Volumetric approaches represent the scene as a collection of voxels, which are typically modeled as small cubes arranged on a 3D lattice. In this context, the goal of 3D scene reconstruction is to determine which voxels represent surfaces in the scene. The reconstruction algorithm classifies each voxel into one of three categories:

Free Space Voxels These voxels represent free space in the scene. These voxels are made transparent.

Surface Voxels These voxels represent the surfaces in the scene. These voxels are rendered opaque.

Inner Voxels These voxels are not visible to any camera, and are assumed to be inside surfaces. These voxels are not typically displayed.

These voxel classifications are shown in Figure 1.

Typical volumetric approaches define a *reconstruction volume*, the region of space that encompasses the scene and in which the reconstruction occurs. The reconstruction volume

is tesselated into voxels, forming a voxel space. The volumetric reconstruction algorithm then removes, or *carves* the free space voxels. Upon completion of the algorithm, the remaining surface and inner voxels represent the 3D volume of objects in the scene. Note that this process is similar to the way a sculptor would chip away at a block of marble to reveal a shape. The surface voxels of the reconstruction model the geometry of the scene surfaces. These voxels can be colored (or texture-mapped) and rendered to produce new views of the scene.

This section reviews volumetric approaches to the 3D scene reconstruction problem. We note that two surveys on the subject appear in the literature [31, 109].

2.5.1 Visual hulls

Perhaps the most straightforward way to reconstruct a 3D scene from multiple calibrated images is to compute the visual hull. However, before we describe the basic approach, we begin our discussion of the visual hull with some definitions. Unfortunately, there is some confusion in the literature regarding the use of the term *visual hull*.

> **Visual hull$_\infty$** The visual hull$_\infty$ is the maximal shape that gives the same silhouette as the actual object for *all* views outside the convex hull of the object.

This definition is given by Laurentini [57]. Reconstruction of the visual hull$_\infty$ could require an infinite number of photographs. We use the term *visual hull* be the approximation of the visual hull$_\infty$ reconstructed using N photographs. Many papers in the literature do not distinguish between the two.

Volumetric approaches [40, 76, 87, 116] are commonly employed to reconstruct the visual hull. Figure 2 gives a visual description of the pipeline. First, photographs of the scene from multiple viewpoints are taken. Then, each photograph is segmented into a binary image consisting of foreground and background, thereby producing silhouettes. This can be achieved by subtracting a known background. Foreground pixels correspond to points to which the 3D object projects. Everything else is background.

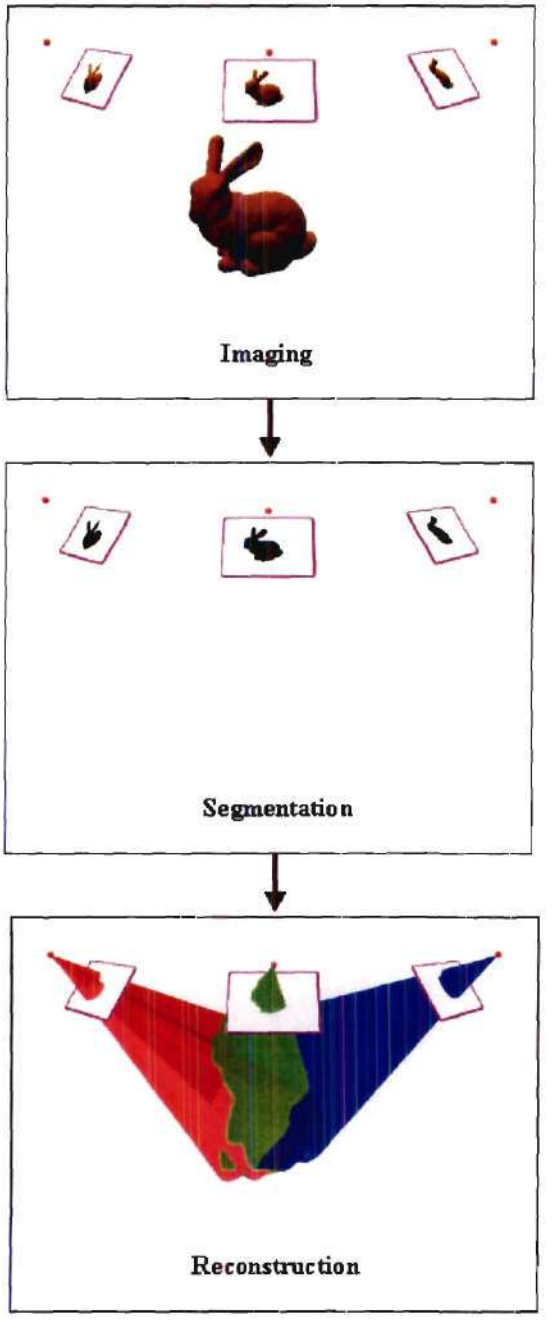


Figure 2: Visual hull reconstruction.

Each silhouette constrains the region of space in which the true surface is located. One can back-project rays from the center of projection of an image, through pixels in the silhouette, and into 3D space. This defines a cone-shaped wedge emanating from a reference view, as shown in the bottom image of Figure 2. If these wedges are intersected from all viewpoints, the resultant shape is the visual hull.

Alternatively, one can project voxels from 3D space into the images. If a voxel projects only to background in a reference view, then it is not in the visual hull and it is carved. The remaining voxels model the surface of the object as well as its interior. For efficiency, the voxel space can be processed in a coarse-to-fine fashion.

The visual hull has several interesting properties. First, although it is only an approximation to the true shape of the object, it is guaranteed to enclose the object. This is shown in Figure 3. Second, in 3D the visual hull of an object can be a better or worse approximation of the object than the convex hull depending on the geometry of the object and the placement of the viewpoints. Third, the size of the visual hull decreases monotonically with the number of images used in the reconstruction. However, even when an infinite number of images are used, not all surface concavities can be modeled with a visual hull. Surface concavities that are not apparent in the silhouettes are not reconstructable using this method.

Work on volumetric reconstruction of visual hulls first appeared in the early 1980s [70]. Since then, techniques have been developed for arbitrary camera placement [87], efficient reconstruction [72, 116], weakly calibrated cameras [43], and dynamic scenes [71, 72, 76]. In Chapter 7, we examine one of these methods [72] in some detail.

Reconstruction of the visual hull has some advantages. First, the visual hull is simple to compute, as one does not need to model visibility of the scene during reconstruction. Second, assuming accurate segmentation of the photographs, visual hull methods are robust to reconstructing surfaces that are non-Lambertian or periodically or homogeneously textured. The disadvantages of visual hull approaches are that background segmentation

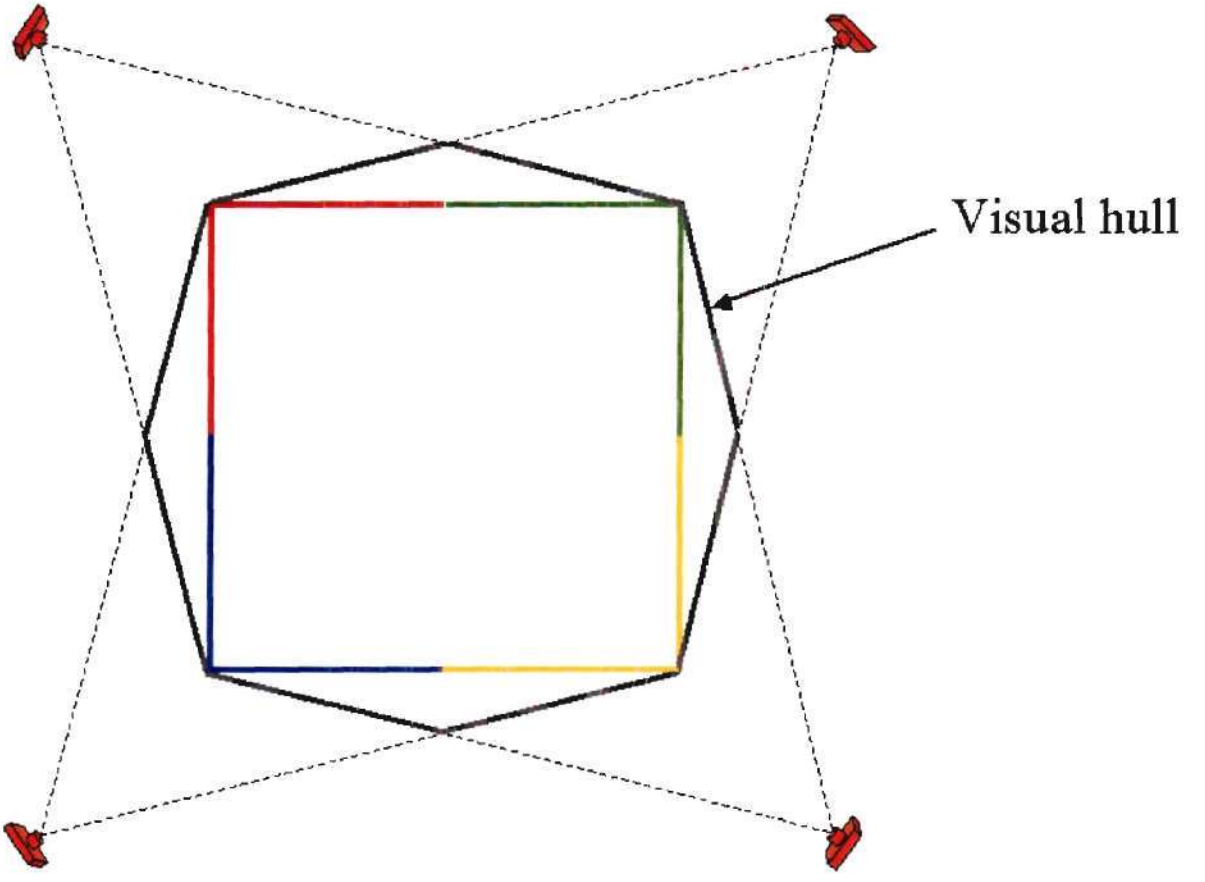


Figure 3: An example visual hull. In this 2D example, a square shape is photographed by four cameras. The visual hull, which contains the true scene, is shown in gray.

can be difficult, and by reconstructing the scene using binary silhouettes, much useful color information that could be used to attain more accurate 3D reconstructions is discarded.

2.5.2 Photo hulls

While a visual hull reconstruction can be rendered to produce new views of the scene, the visual geometry is typically not very accurate. This can diminish the photo-realism of the synthetic new views. To increase the geometric accuracy, more information than silhouette data must be used during reconstruction.

An obvious choice is color (or luminance, for gray-scale images). Many researchers have attempted to reconstruct 3D scenes by analyzing colors across multiple viewpoints, looking for a 3D model that, when projected to reference views, reproduces the photographs. Unfortunately, the problem is ill-posed. For a given set of 2D photographs, multiple 3D models that reproduce the photographs can and often do exist.

In their insightful work, Kutulakos and Seitz [54] introduce the *photo hull*, which is the largest shape that contains all reconstructions in the equivalence class of 3D models that reproduce the photographs. The photo hull is unique, and is itself a reconstruction of the scene. Better yet, it is the tightest possible bound on the shape of the true scene that can be inferred from N photographs, in the absence of a priori geometric or point correspondence information [54].

Critical to the computation of the photo hull is the notion of photo-consistency. A point in 3D space is said to be *photo-consistent* with a photograph if

- The point does not project to background.
- When the point is visible, the light exiting the point (i.e., radiance) in the direction of the camera is equal to the observed color of the point's projection in the photograph.

Since we model points with voxels, the photo hull is found by identifying the spatially largest set of voxels that are photo-consistent with all the photographs taken of the scene.

To compute the photo hull, we require a method to determine photo-consistency. Assume that a point (or voxel) is visible to K of the N reference views. As described in [54], a *photo-consistency check* is an algorithm that that takes as input at least K colors from the photographs, K 3D vectors going from the point (or voxel) toward each camera, and all light source positions (for the non-Lambertian case), and determines if it is possible for the point to reflect light of the observed color in each photograph. The photo-consistency check is assumed to be monotonic in that if a point (or voxel) is found to be inconsistent when visible to M cameras, it will still be inconsistent when visible to $K > M$ cameras.

Such a photo-consistency check is limited to the class of radiance models that are locally computable - that is, the radiance of any point in the scene is independent of the radiance of any other point in the scene [54]. Thus, scenes with transparency would not be reconstructable using such a photo-consistency check. However, a wide class of parameterized radiance models are valid. An important special case is the Lambertian model, which reflects light with equal intensity in all directions [39]. For this model, the photo-consistency check simply measures the similarity of the colors in the projection of the point (or voxel) into each photograph. If the colors are similar, the point (or voxel) is photo-consistent. Otherwise, it is inconsistent. We will describe measures for computing the similarity of colors across viewpoints in Section 3.1.

By taking advantage of these additional color constraints, the photo hull geometry is often a tighter fit to the true scene geometry than the visual hull. That is,

$$\text{True Scene} \subseteq \text{Photo Hull} \subseteq \text{Visual Hull}$$

Figure 4 shows an example of the photo hull for a square shape photographed by four cameras. This is the same shape that appears in Figure 3. We note that the photo hull can contain concavities that are not possible to model with the visual hull.

The standard approach for computing a photo hull is space carving [32, 54, 100]. This class of volumetric algorithms reconstructs the scene by removing (carving) voxels that are not photo-consistent with the reference views. These approaches begin with a voxel space

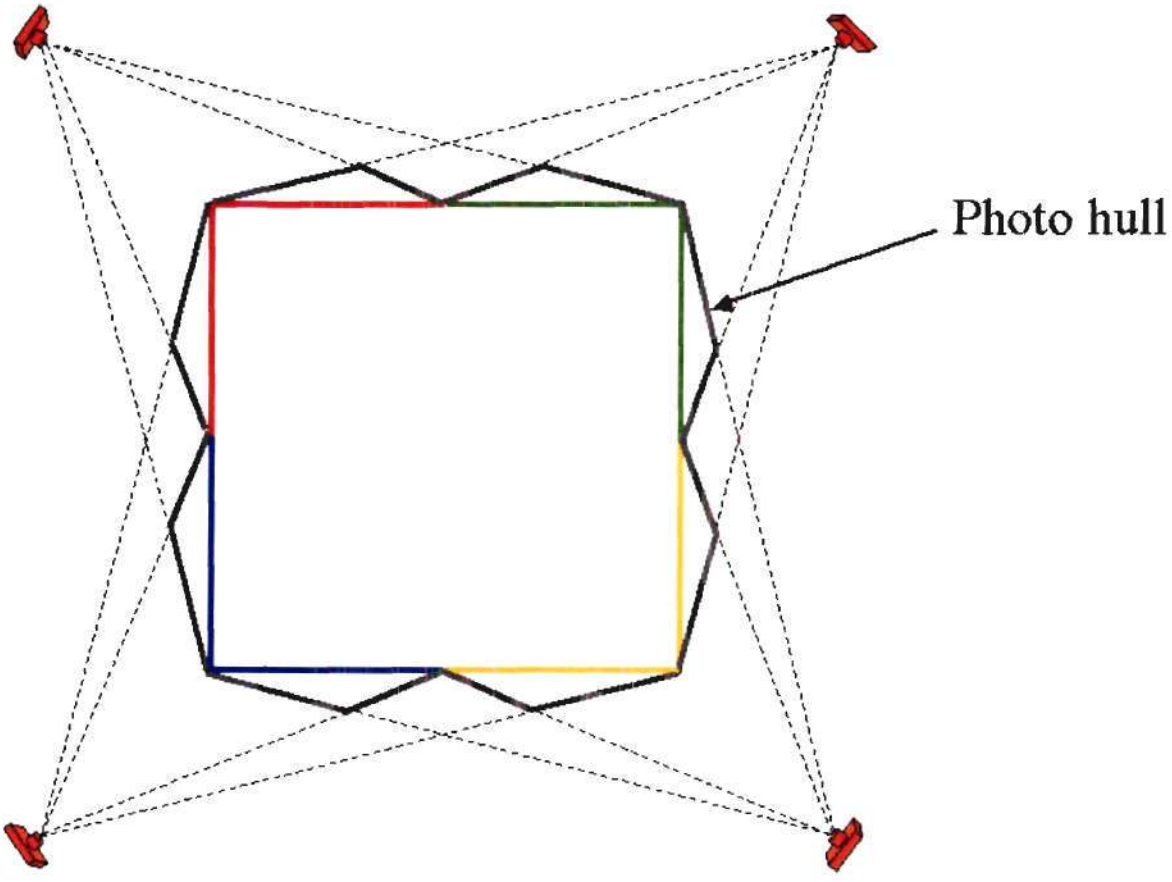


Figure 4: An example photo hull, taken from [54]. In this 2D example, a square shape is photographed by four cameras. The photo hull is shown in gray. Note this shape is a tighter fit than the visual hull shown in Figure 3.

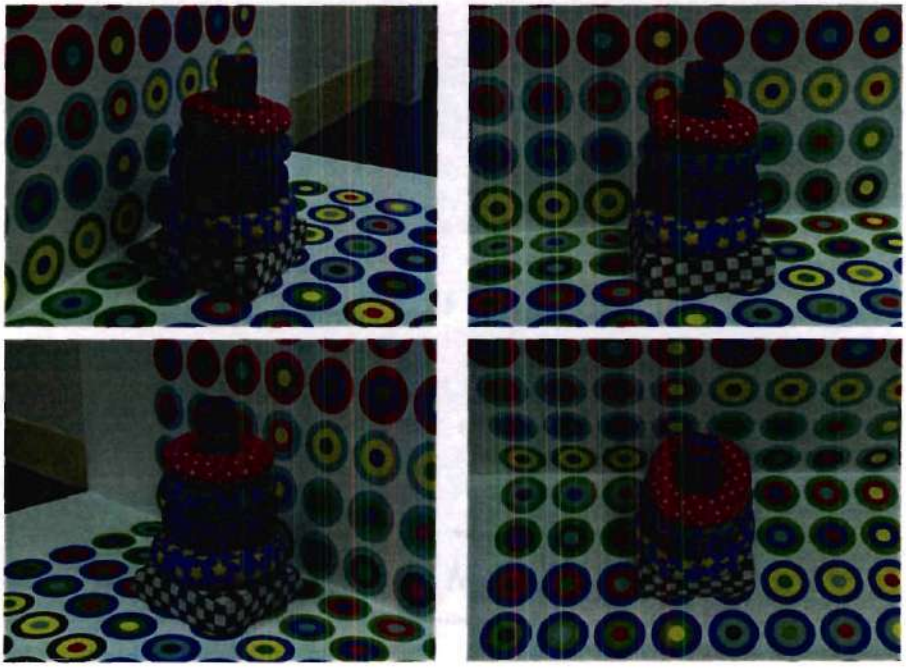
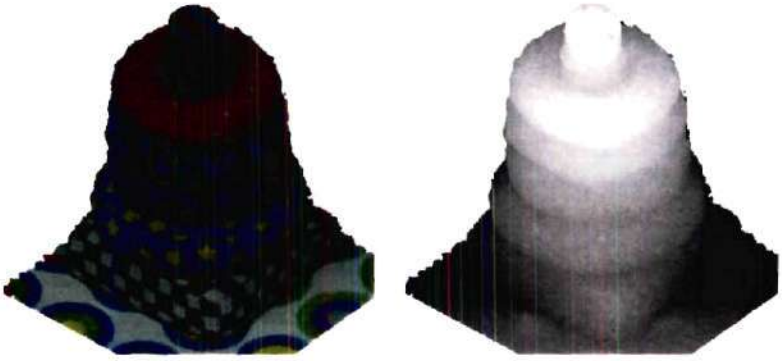


Figure 5: Four of 36 photographs of a cloth toy. This data set is provided courtesy of Tom Malzbender and Bruce Culbertson.

composed of opaque voxels. Voxels that are visible to the cameras are tested for photo-consistency. The inconsistent voxels are carved (made transparent), which changes the visibility of other voxels in voxel space. The algorithm continues to iterate until all visible voxels are photo-consistent. When these remaining photo-consistent voxels are assigned the colors they project to in the photographs, they form a model that closely resembles the scene. We will describe two of these approaches [54, 100] in more detail in Chapter 3.

An example of a scene reconstructed using the algorithm from [100] is provided in Figures 5 and 6. This reconstruction was computed of a cloth toy using 36 photographs, four of which are shown in Figure 5. After reconstructing the scene, the 3D model was rendered to a synthetic viewpoint. The new synthesized view is shown in Figure 6 (a), along with the corresponding depth map (b).

Space carving approaches are very powerful, and have captured the interest of many researchers who have proposed extensions to or reformulations of the basic approach. Prock



(a) (b)

Figure 6: Reconstruction of a cloth toy using a space carving algorithm. We show a new synthesized view (a) and the corresponding depth map (b).

and Dyer [91] present a multi-resolution approach as well as hardware implementations for improved efficiency. De Bonet and Viola [11] address the problem of reconstructing scenes that have opacity. Researchers have performed space carving using intrinsically calibrated [33] and weakly calibrated [63, 95] cameras. Vedula et al. [126] link two time-consecutive voxel spaces together for reconstructing shape and motion of time-varying scenes. Space carving was recast in a probabilistic framework by Broadhurst et al. [12]. Carceroni and Kutulakos [15] propose a surfel-based approach for reconstructing time-varying scenes with various reflectances.

Indeed, the goal of this thesis is to introduce novel approaches that improve upon the quality, efficiency, and applicability of these space carving methods.

2.5.3 Level set methods

Another class of volumetric reconstruction algorithms is those that employ level set methods [83, 102]. Level set theory was originally developed to model the evolution of propagating interfaces, but has since been applied to a wide array of problems.

For 3D surface evolution, these methods usually embed a surface as the zero-level set of a volumetric function. This function is sampled on a discrete 3D lattice, forming a voxel space. The surface moves along its surface normal, subject to intrinsic, data-driven, and

independent forces. Level set theory provides an accurate and stable numerical scheme that solves the partial differential equations (PDEs) that characterize the motion for the surface. Topological changes are naturally accommodated in this framework.

Faugeras and Keriven [34] present a level set algorithm for 3D scene reconstruction. Their method begins with an initial surface that encompasses the scene being reconstructed. The surface then flows inward, with a speed on the surface proportional to the mismatch of textures determined by projecting a small patch on the surface into the reference views. The surface slows at regions that match, yielding a reconstruction that models the 3D scene. A polygonal surface representing the zero-level set can be extracted from the volume using the marching cubes algorithm [62]. This technique was extended by Colosimo [24] et al. to support multi-resolution reconstruction.

Yezzi and Soatto [130] introduce a stereoscopic segmentation approach designed for radiance functions with smooth statistics. Their region-based method simultaneously segments the N photographs by evolving a 3D surface using level set methods. They demonstrate that their approach is resistant to errors in camera calibration and allows for a bi-directional flow. In a later paper, this work is extended to address scenes with specularities [52].

2.6 Summary

This chapter reviewed new view synthesis techniques. The goal of these methods is to generate photo-realistic synthetic views given a collection of photographs taken of a scene. We presented image-based rendering as well as image-based modeling methods. Image-based rendering algorithms typically warp or resample the photographs to generate new views. Image-based modeling approaches first reconstruct a 3D model of the scene geometry, and then render the model to produce new views of the scene.

While there has been much work on the problem of new view synthesis, the problem has yet to be fully solved. In this thesis, we introduce methods that extend the state of the

art in volumetric image-based modeling methods. Volumetric approaches have significant advantages over competing methods.

Comparing to image-based rendering methods, volumetric scene reconstruction approaches compute a 3D model of the scene. A 3D model is a more complete representation that has wider application than simply new view synthesis. Reconstructed 3D models have been used in robot navigation, reverse engineering [26], object recognition [7, 77], motion capture [104], model building [27, 126], teleconferencing, and more. Furthermore, this 3D representation is often much more compact than light field approaches, which rely on a dense sampling of the plenoptic function.

Multi-view stereo matching and structure from motion methods can achieve good quality reconstructions. However, volumetric approaches like space carving have a key advantage in that they explicitly model visibility when matching colors across viewpoints. Reasoning about occlusion given two or more views of a scene is difficult in the 2D space of the images, but is relatively simple in 3D world space. Related to this, in volumetric approaches, the cameras can be separated by larger baselines without degrading accuracy or runtime. Another advantage is that volumetric approaches inherently integrate a number of reference views to yield a dense reconstruction. However, in multi-view stereo matching and structure from motion methods, formation of a coherent 3D model from triangulated correspondences is an additional step that is error-prone.

For these reasons, we argue that volumetric approaches are well suited for reconstructing 3D models of scenes for new view synthesis. The remaining chapters of this thesis present methods that extend the state of the art in volumetric scene reconstruction. We introduce techniques that improve the quality, applicability, and efficiency of existing methods. The next chapter presents a volumetric reconstruction algorithm that generalizes visibility to support arbitrary camera placement while using the full visibility of the scene.

CHAPTER III

GENERALIZED VOXEL COLORING

This chapter describes generalized voxel coloring (GVC), which encompasses two new approaches to computing visibility during a volumetric 3D scene reconstruction. Visibility is an essential, yet subtle aspect to this class of algorithms and several interesting variations have been proposed.

Before describing GVC, we discuss the computation of photo-consistency, and describe two previous volumetric algorithms that use photo-consistency to reconstruct a scene. We then describe how our generalized voxel coloring approach extends these earlier approaches.

3.1 Photo-consistency

Computation of the photo hull relies upon establishing the photo-consistency of voxels. Throughout this thesis we will assume that the scene is or is nearly Lambertian. In this case, a voxel that models a scene surface will project to a similar color in each of the reference views, as shown in Figure 7. This fact is used as a constraint for identifying scene geometry. We note that the use of other reflectance models is possible [23], but may require calibration of light sources and computation of surface normals.

3.1.1 Visibility

Visibility is required to correctly compute photo-consistency, as shown in Figure 8. In this figure, a voxel that does model a scene surface could erroneously be declared inconsistent if visibility is not taken into consideration. The voxel is not visible to the rightmost camera, which observes a blue color resulting from occluding geometry in the scene. Only the viewpoints that have visibility of a voxel should contribute to the photo-consistency check.

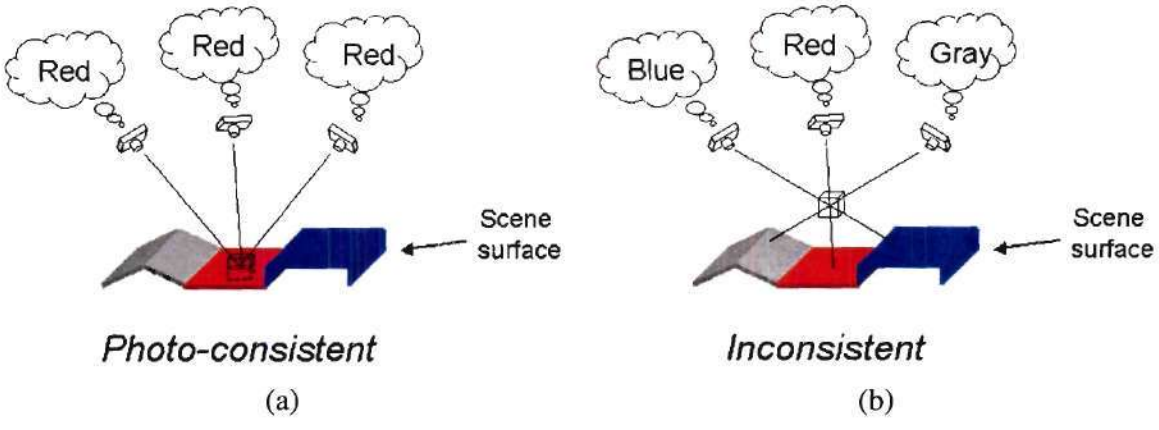


Figure 7: Using photo-consistency to identify scene surfaces (Lambertian example). In (a), a voxel that models a scene surface projects to consistent colors in the reference views. In (b), a voxel that does not model a scene surface projects to different, inconsistent colors.

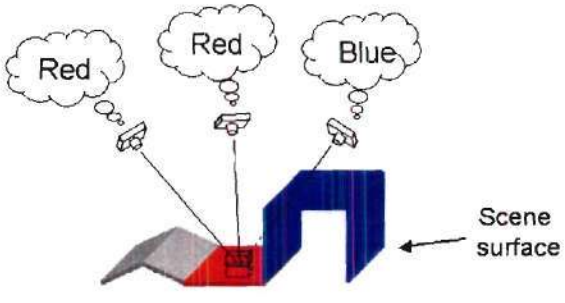


Figure 8: Visibility is required to correctly compute photo-consistency.

Many different approaches to computing photo-consistency have been developed and analyzed [13, 84]. Below we present two simple photo-consistency measures that are used to perform the reconstructions in this thesis.

3.1.2 A single threshold measure

Let $\text{vis}(V,i)$ denote the set of visible pixels to which a voxel V projects in reference view i. The number of pixels in $\text{vis}(V,i)$ varies depending on the resolution of the voxel space, the resolution of the reference view, the imaging geometry, the relative position of the camera and the voxel, and occlusions in the scene. Let $\text{vis}(V)$ be the set of pixels to which a voxel V projects in all the images. That is, $\text{vis}(V) = \bigcup_i \text{vis}(V,i)$.

For RGB images, one can compute the mean in each color channel $(\overline{r},\overline{g},\overline{b})$ and a standard deviation σ for all the pixels in $\text{vis}(V)$. Specifically, we compute

$$\overline{r} = \frac{1}{N}\sum_{j=1}^{N} r_j \tag{1}$$

$$\overline{g} = \frac{1}{N}\sum_{j=1}^{N} g_j \tag{2}$$

$$\overline{b} = \frac{1}{N}\sum_{j=1}^{N} b_j \tag{3}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(r_j^2+g_j^2+b_j^2) - \overline{r}^2 - \overline{g}^2 - \overline{b}^2} \tag{4}$$

where (r_j,g_j,b_j) is the jth of N colors in $\text{vis}(V)$.

Intuitively, the pixel colors match when σ is small. A simple approach to determine photo-consistency, proposed in [100], is to threshold the standard deviation σ. That is,

$$\text{consist}(\text{vis}(V)) = \begin{cases} \text{True, if } \sigma < T_1 \\ \text{False, otherwise} \end{cases} \tag{5}$$

where T_1 is user specified or computed from the noise in the reference views. We note that this approach of thresholding the standard deviation also works with color spaces other than RGB.

This measure of photo-consistency is capable of adequately reconstructing many scenes, especially those that have a mostly homogenous color, such as the one shown in Figure 9 (a). However, its drawback is that it performs poorly for voxels that project to consistent, yet highly varying colors in the images, such as those in (b) and (c) of Figure 9. In these cases, multiple different colors will be observed for the voxel in the images. Consequently, the standard deviation will be high, the photo-consistency measure will incorrectly return false, and the voxel will erroneously be carved.

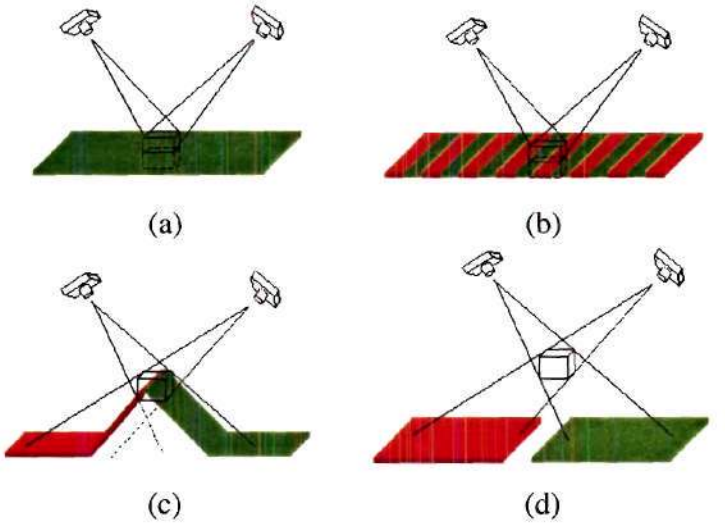


Figure 9: Handling texture and edges. In (a), a voxel represents a homogeneous region, for which both σ and $\overline{\sigma}$ are small. In (b) and (c), a voxel represents a textured region and an edge, respectively, for which both σ and $\overline{\sigma}$ are large. In (d), a voxel representing free space has a large σ and small $\overline{\sigma}$. As discussed in Section 3.1.3, $\overline{\sigma}$ is the standard deviation computed per reference view and averaged across the viewpoints.

Propagating errors can be caused if the photo-consistency measure incorrectly causes voxels to be carved. The removal of a voxel that should be photo-consistent incorrectly changes the visibility of other photo-consistent voxels, which may then incorrectly become inconsistent. This process can iterate, propagating the errors, and in the worst case, no voxels will remain in the volume upon completion of the reconstruction algorithm.

3.1.3 An adaptive threshold measure

Voxels that represent textured surfaces and edges, as shown in Figure 9 (b) and (c), will project to pixels with a high standard deviation in *each* image. We use this fact to modify the consistency measure described above to handle such surfaces. Let the standard deviation of vis(V,i), summed over each color channel, be σ_i. Our new photo-consistency measure is then

$$\text{consist}(\text{vis}(V)) = \begin{cases} \text{True, if } \sigma < T_1 + \overline{\sigma} T_2 \\ \text{False, otherwise} \end{cases} \tag{6}$$

where $\overline{\sigma}$ is the average of σ_i and T_2 is a second user-defined threshold.

This consistency measure simply adds an additional term $\overline{\sigma} T_2$ to Equation 5. This term spatially adapts the consistency measure based on the colors observed in the voxel's projection. The value of $\overline{\sigma}$ will be small when a voxel projects to homogenous colors in each image. In this case, there will be little difference between the two consistency measures 5 and 6. If these colors are similar (as in Figure 9 (a)), the voxel will be declared consistent. If these colors are dissimilar, (as in Figure 9 (d)), the voxel will be declared inconsistent. When the voxel projects to highly varying pixels in each image, the $\overline{\sigma}$ term will increase the maximum value of σ allowable for the voxel to be declared consistent. This allows for textured surfaces, as well as edges, to be correctly reconstructed. The threshold T_2 allows one to weight the contribution of this term to the photo-consistency measure.

The two-threshold consistency measure described above is effective at reconstructing Lambertian scenes. It will be used to produce all the reconstructions presented in this thesis. Its main drawback is that it requires two parameters, T_1 and T_2, to be specified prior to reconstruction. These thresholds are not typically known in advance, and to find them, one sweeps the parameter space until values are found that achieve good results. Other researchers have investigated alternative measures that require little or no parameter tuning [13, 107].

3.2 *Voxel coloring - a single planar sweep*

We now describe Seitz and Dyer's voxel coloring algorithm [100], which was the first volumetric approach to use photo-consistency to reconstruct a color 3D scene.

The voxel coloring algorithm begins with a reconstruction volume of initially opaque voxels that encompasses the scene to be reconstructed. As the algorithm runs, opaque voxels are tested for photo-consistency and those that are found to be inconsistent are carved. The algorithm stops when all the remaining opaque voxels are photo-consistent. When these final voxels are assigned the colors they project to in the input images, they form a model that closely resembles the scene.

As voxel coloring progresses, opaque voxels occlude each other from the input images in a complex and constantly changing pattern. To test the photo-consistency of a voxel, its visibility must first be determined. Since this is done many times during a reconstruction, it must be performed efficiently.

To simplify the computation of voxel visibility and to allow a scene to be reconstructed in a single scan of the voxels, Seitz and Dyer imposed what they called the *ordinal visibility constraint* on the camera locations. It requires that the cameras be placed such that all the voxels are visited in a single scan in near-to-far order relative to every camera. Typically, this condition is met by placing all the cameras on one side of the scene and scanning voxels in planes that are successively further from the cameras. Thus, the transparency of all voxels that might occlude a voxel V is determined before V is checked for photo-consistency. This insures that the visibility of a voxel stops changing before it needs to be computed, which is important since every voxel is visited just once. An occlusion bitmap, with one bit per input camera pixel, is used to account for occlusion. These bits are initially clear. When a voxel is found to be consistent, meaning it will remain opaque, all the occlusion bits in the voxel's projection are set, as shown in Figure 10. The visibility set of a voxel is simply the pixels in the voxel's projection whose occlusion bits are clear.

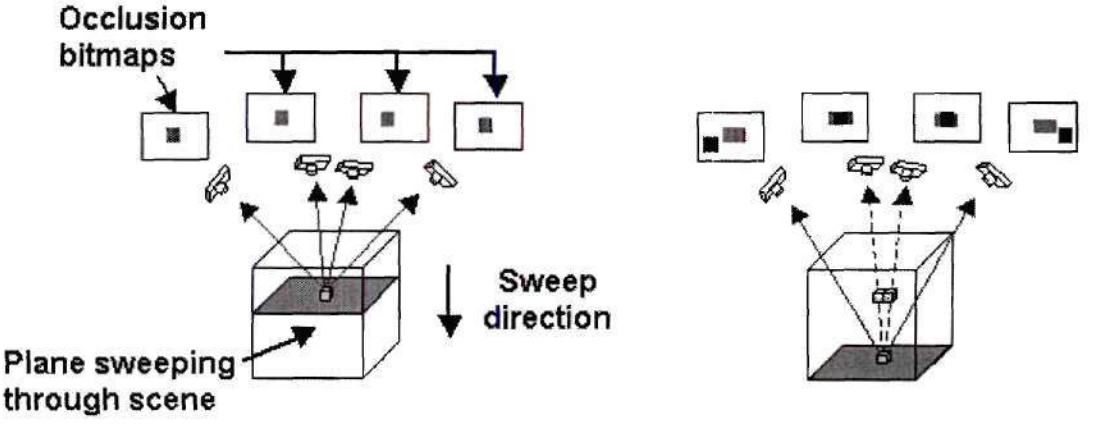


Figure 10: Voxel coloring via a planar sweep. On the left, a voxel is found to be consistent, and a bit in the occlusion bitmap is set for each pixel in the projection of a consistent voxel into each image, indicated by the gray pixels in the occlusion bitmaps. On the right, visibility of the lowest voxel is established by examining the pixels to which the voxel projects. These pixels are shown in black. If the occlusion bits have been set for these pixels, then the voxel is occluded, as is the case for the two middle cameras.

3.3 Arbitrary camera placement

The voxel coloring algorithm described in the previous section is elegant and efficient. However, the ordinal visibility constraint is a significant limitation. Since the voxels must be ordered from near to far relative to all the cameras, the cameras cannot surround the scene. Thus, some surfaces will not be visible in any image and hence cannot be reconstructed. Because it is often desirable to obtain a model that resembles the scene from every direction, several variations of voxel coloring have been developed to circumvent this limitation. If we surround the scene with cameras, we give up the ordinal visibility constraint. Without the constraint, there is no order in which to scan voxels that guarantees their visibility will not change after we check their photo-consistency. Hence, algorithms that allow arbitrary camera placement must test voxels repeatedly for photo-consistency until their visibility stabilizes.

```
set all voxels opaque
loop {
   AllVoxelsConsistent = TRUE
   for every opaque voxel V {
      determine vis(V)
      if vis(V) has consistent color {
         assign V the average color of all pixels in vis(V)
      } else {
         AllVoxelsConsistent = FALSE
         set V to be transparent
      }
   }
   if AllVoxelsConsistent = TRUE
      quit
}
```

Figure 11: Pseudo-code for space carving algorithms.

Figure 11 gives the general approach for voxel coloring algorithms that allow arbitrary camera placement. In the inner loop, the visibility of voxels is found, their photo-consistency is checked, and they are carved if they are found to be inconsistent. If one voxel is carved, the visibility of other voxels potentially changes, invalidating any photo-consistency tests they may have passed. Hence, there is an outer loop that repeats the photo-consistency checking until no carving occurs in the inner loop. No carving occurs on the final iteration of the outer loop so no testing is invalidated and the final set of opaque voxels is guaranteed to be photo-consistent.

When the algorithm in Figure 11 begins to run, the model bears little resemblance to the scene. Yet, the algorithm computes the visibility for voxels, and carves those found to be inconsistent, based on this model. It is reasonable to wonder if the algorithm might fail because of carving voxels early on that would be photo-consistent in the final model. As described in [25], this cannot happen if a monotonic photo-consistency measure is used. For such a photo-consistency measure, if a set of pixels is inconsistent, any superset of those pixels is also inconsistent. Since the algorithm changes only opaque voxels to transparent and never vice versa, remaining opaque voxels can only become more visible as

the algorithm runs and the pixels that can see a voxel at one point in time will be a subset of those that see the voxel at any later time. Thus, if the photo-consistency measure ever finds a voxel to be inconsistent, the voxel will also be inconsistent in the final model. Therefore, when using a monotonic photo-consistency measure, the algorithm never carves a voxel that would be consistent in the final model. In this sense we say that carving is conservative.

We note that in this thesis, we use the term *space carving* to refer to the class of volumetric reconstruction algorithms that allow for arbitrary camera placement and also use photo-consistency to carve away voxels that do not model scene surfaces. These space carving algorithms include [25, 32, 54], and are grouped together by some authors writing papers in the computer vision literature. Below, we refer to the algorithm in [54] as Kutulakos and Seitz's space carving approach.

3.4 Space carving: multiple planar sweeps

Kutulakos and Seitz [54] present a space carving algorithm that implements Figure 11. It always scans voxels for photo-consistency by evaluating a plane of voxels at a time, as is done with voxel coloring. Unlike voxel coloring, their approach performs multiple scans, typically along the positive and negative directions of each of the three axes. The algorithm forces the scans to be near-to-far, relative to the cameras, by using only images whose cameras have already been passed by the moving plane. Thus, when a voxel is evaluated, the transparency is already known of other voxels that might occlude it from the cameras currently being used. Because carving is conservative, the set of uncarved voxels is a shrinking superset of the desired color-consistent model as the algorithm runs.

Kutulakos and Seitz's approach achieves the goal of allowing arbitrary camera placement. For simplicity, the original formulation of the algorithm [56] used only a subset of the cameras that have visibility of a voxel when computing photo-consistency. Such an approach can produce a model that includes some inconsistent voxels. A later enhancement

to the algorithm [54] describes some additional bookkeeping that enables the method to compute visibility exactly.

3.5 *Generalized voxel coloring*

Now that we have reviewed voxel coloring and Kutulakos and Seitz's space carving algorithm, we are ready to present our new algorithm, generalized voxel coloring (GVC). The goal of GVC is to reconstruct the scene using the exact visibility while supporting arbitrary camera placement. In this spirit, it is similar to the enhanced space carving algorithm described in [54]. However, GVC has several advantages that will be described below.

GVC encompasses two different methods, GVC-IB and GVC-LDI, for computing scene visibility during a volumetric reconstruction. GVC-LDI is an enhancement of GVC-IB, our basic algorithm. The carving of one voxel potentially changes the visibility of other voxels. When an uncarved voxel's visibility changes, its photo-consistency should be reevaluated and it, too, should be carved if it is then found to be inconsistent. GVC-LDI uses layered depth images (LDIs) [73, 103] to determine exactly which voxels have their visibility changed when another voxel is carved and thus can reevaluate exactly the right voxels. In the same situation, GVC-IB does not know which voxels need to be reevaluated and so reevaluates all voxels in the current model. Therefore, GVC-LDI performs significantly fewer photo-consistency evaluations than GVC-IB during a reconstruction. However, GVC-IB uses considerably less memory than GVC-LDI. Like Kutulakos and Seitz's space carving approach, both GVC-IB and GVC-LDI initially assume all voxels are opaque, i.e., uncarved. They carve inconsistent voxels until all those that remain project into consistent colors in the images from which they are visible.

3.5.1 GVC-IB

GVC-IB determines visibility as follows. First, every voxel is assigned a unique ID. Then, an item buffer [128] is constructed for each image. An item buffer, shown in Figure 12

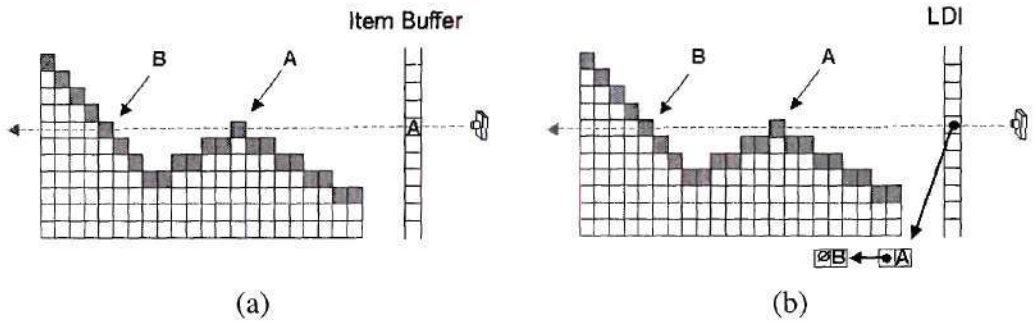


Figure 12: GVC data structures. An item buffer is shown in (a), and a layered depth image is shown in (b). For simplicity, we show a 2D scene captured by 1D cameras.

(a), contains a voxel ID for every pixel in the reference view. To compose an item buffer, each voxel is rendered to a camera using z-buffering, however, instead of storing a color, the voxel ID is stored. Each pixel in the item buffer will then contain the ID of the closest voxel that projects onto the pixel. This is exactly the visibility information we need when computing photo-consistency.

Once valid item buffers have been computed for the images, it is then possible to compute the set vis(V) of all pixels from which the voxel V is visible. To compute vis(V), we first project V into each image. Then, for every pixel P in the projection of V, if P's item buffer value equals V's ID, then P is added to vis(V). To check the photo-consistency of a voxel V, we apply a photo-consistency function consist() to vis(V).

Since carving a voxel changes the visibility of the remaining uncarved voxels, and since we use item buffers to maintain visibility information, the item buffers need to be updated periodically. GVC-IB does this by recomputing the item buffers from scratch. Since this is time consuming, we allow GVC-IB to carve many voxels between updates. As a result, the item buffers are out-of-date much of the time and the computed set vis(V) is guaranteed only to be a subset of all the pixels from which a voxel V is visible. However, since carving is conservative, no voxels will be carved that should not be carved. During the final iteration of GVC-IB, no carving occurs so the visibility information stays up-to-date. Every voxel is checked for photo-consistency on the final iteration so it follows that the final model is

photo-consistent.

As carving progresses, each voxel is in one of the three categories described in Section 2.5 of this thesis:

- *Free space voxel:* It has been found to be inconsistent and has been carved.
- *Surface voxel:* It is on the surface of the set of uncarved voxels and has been found to be consistent whenever it has been evaluated.
- *Inner voxel:* It is not visible to any of the reference views and its photo-consistency is undefined.

We use an array of bits, one per voxel, to record which voxels have been carved. This data structure is called 'carved' in the pseudo-code Figure 13 and is initially set to false for every voxel. We maintain a data structure called the surface voxel list (SVL) to identify the second category of voxels. The SVL is initialized to be the set of voxels that are not surrounded by other voxels. The item buffers are computed by rendering all the voxels on the SVL to each reference view. Though inner voxels are uncarved, they do not need to be rendered into the item buffers because they are not visible from any images. When a voxel is carved, adjacent inner voxels become surface voxels and are added to the SVL. To avoid adding a voxel to the SVL more than once, we need a rapid means of determining if the voxel is already on the SVL; hence we implement the SVL as a hash table. When GVC-IB has finished, the final set of uncarved voxels may be recorded by saving the function carved() or the SVL.

3.5.2 GVC-LDI

GVC-IB computes visibility in a relatively simple manner that makes efficient use of memory. However, the visibility information is time consuming to update. Hence, GVC-IB updates it infrequently and it is out-of-date much of the time. This does not lead to incorrect results but it does result in inefficiency because a voxel that would be evaluated as

```
initialize SVL
for every voxel V
   carved(V) = false
loop {
   visibilityChanged = false
   compute item buffers by rendering voxels on SVL
   for every voxel V ∈ SVL {
      compute vis(V)
      if (consist(vis(V)) = false) {
         visibilityChanged = true
         carved(V) = true
         remove V from SVL
         for all voxels N that are adjacent to V
            if (carved(N) = false and N ∉ SVL)
               add N to SVL
      }
   }
   if (visibilityChanged = false) {
      save voxel space
      quit
   }
}
```

Figure 13: GVC-IB pseudo-code.

inconsistent using all the visibility information might be evaluated as consistent using a subset of the information. Ultimately, all the information is collected but, in the meantime, voxels can remain uncarved longer than necessary and can therefore require more than an ideal number of photo-consistency evaluations. Furthermore, GVC-IB reevaluates the photo-consistency of voxels on the SVL even when their visibility (and hence their photo-consistency) has not changed since their last evaluation. By using layered depth images instead of item buffers, GVC-LDI can efficiently and immediately update the visibility information when a voxel is carved and also can precisely determine the voxels whose visibility has changed.

The item buffers used by the GVC-IB method record at each pixel P just the closest voxel that projects onto P. In contrast, the LDIs store at each pixel a list of *all* the surface voxels that project onto P as shown in Figure 12 (b). These lists, which in the pseudo-code are called LDI(P), are depth sorted according to the distance of the voxel to the image's camera. The head of LDI(P) stores the voxel closest to P, which is the same voxel an item buffer would store. Since the information stored in an item buffer is also available in an LDI, vis(V) can be computed in the same way as in GVC-IB. The LDIs are initialized by rendering the SVL voxels.

The uncarved voxels whose visibility changes when another voxel is carved come from two sources, as shown in Figure 14:

- They are inner voxels adjacent to the carved voxel and become surface voxels when the carved voxel becomes transparent.
- They are already surface voxels (hence they are in the SVL and LDIs) and are often distant from the carved voxel.

Voxels in the first category are trivial to identify since they are next to the carved voxel. Voxels in the second category are impossible to identify efficiently in the GVC-IB method; hence, that method must repeatedly evaluate the entire SVL for photo-consistency. In

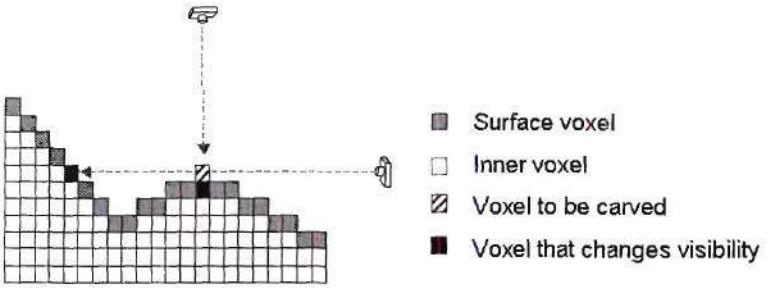


Figure 14: Voxels that change visibility.

GVC-LDI, voxels in the second category can be found easily with the aid of the LDIs; they will be the second voxel on LDI(P) for some pixel P in the projection of the carved voxel. GVC-LDI keeps a list of the SVL voxels whose visibility has changed, called the changed visibility SVL (CVSVL in the pseudo-code in Figure 15). These are the only voxels whose photo-consistency must be checked. Carving is finished when the CVSVL is empty.

When a voxel is carved, the LDIs (and hence the visibility information) can be updated immediately and efficiently. The carved voxel can be deleted easily from LDI(P) for every pixel P in its projection. The same process automatically updates the visibility information for the second category of uncarved voxels whose visibility has changed; these voxels move to the head of LDI lists from which the carved voxel has been removed and they are also added to the CVSVL. Inner voxels adjacent to the carved voxel are pushed onto the LDI lists for pixels they project onto. As a byproduct of this process, we learn if the voxel is visible. If so, we put it on the CVSVL. Pseudo-code for GVC-LDI appears in Figure 15.

3.5.3 Results

We have reconstructed a number of scenes using the GVC-IB and GVC-LDI algorithms. Figures 16, 17, 18, and 19 demonstrate the process for a typical example. In Figure 16, we show two of 17 reference views of a toy model of Ghirardelli Square. These images were taken with a resolution of 1152 x 872 pixels. The object was placed on a black

```
initialize SVL
render SVL to LDIs
for every voxel V
   carved(V) = false
copy SVL to CVSVL
while (CVSVL is not empty) {
   delete V from CVSVL
   compute vis(V)
   if (consist(vis(V)) = false) {
      carved(V) = true
      remove V from SVL
      for every pixel P in projection of V into all images {
         if (V is head of LDI(P))
            add next voxel on LDI(P) (if any) to CVSVL
         delete V from LDI(P)
      }
      for every voxel N adjacent to V with N ∉ SVL {
         N_is_visible = false
         for every pixel P in projection of N to all images {
            add N to LDI(P)
            if (N is head of LDI(P))
               N_is_visible = true
         }
         add N to SVL
         if (N_is_visible)
            add N to CVSVL
      }
   }
}
save voxel space
```

Figure 15: GVC-LDI Pseudo-code.

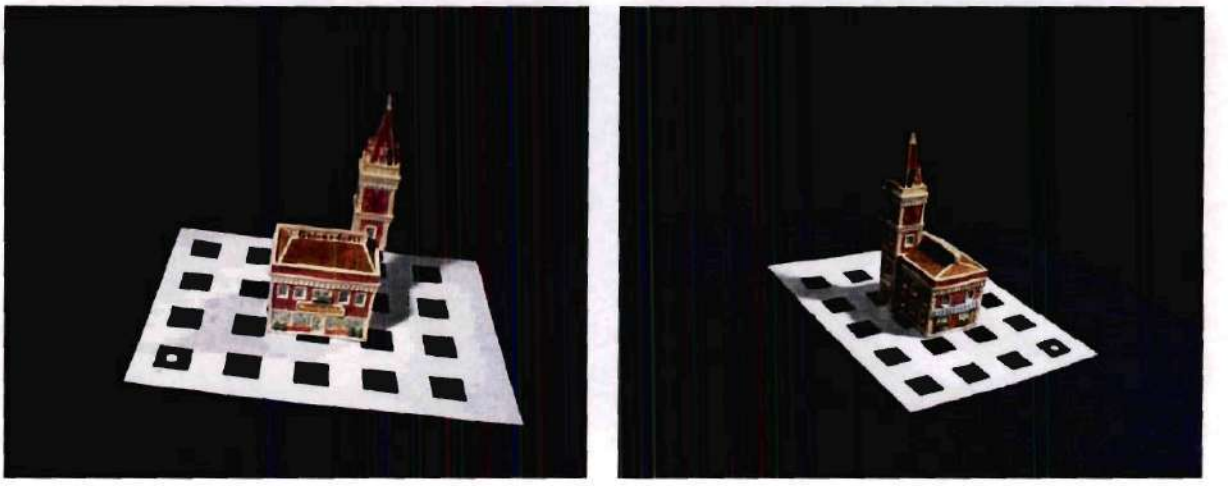


Figure 16: Two of 17 reference views of a toy model of Ghirardelli Square.

and white checkerboard pattern so that the cameras used to photograph the scene could be calibrated. Calibration was performed using Tsai's method [124]. For each camera, the calibration gives the pose, shown in Figure 17. The green polyhedron in the figure is the reconstruction volume.

The reconstruction volume was tesselated into 168 x 104 x 256 voxels. At this resolution, each voxel had a physical size of 0.7 mm per side. Figure 18 shows a GVC-LDI reconstruction in progress. In the figure, the green voxels are the voxels on the CVSVL. Initially, all voxels visible to the reference views are placed on the CVSVL, as shown in the left-most image of the figure. The algorithm carves the inconsistent voxels, which exposes new surface voxels that must be processed. Each voxel that is photo-consistent is given a color equal to its mean $(\overline{r}, \overline{g}, \overline{b})$ of vis(V), and moved to the surface voxels list. As the algorithm iterates, more photo-consistent voxels are found and the scene surfaces emerge. The algorithm converges when all visible voxels are photo-consistent. The figure shows that GVC processes the voxels in a more general order than the plane sweeping approach used in voxel coloring and Kutulakos and Seitz's space carving approach. Once the reconstruction is complete, new views of the scene can be synthesized by rendering the reconstructed surface to new viewpoints, as shown in Figure 19.

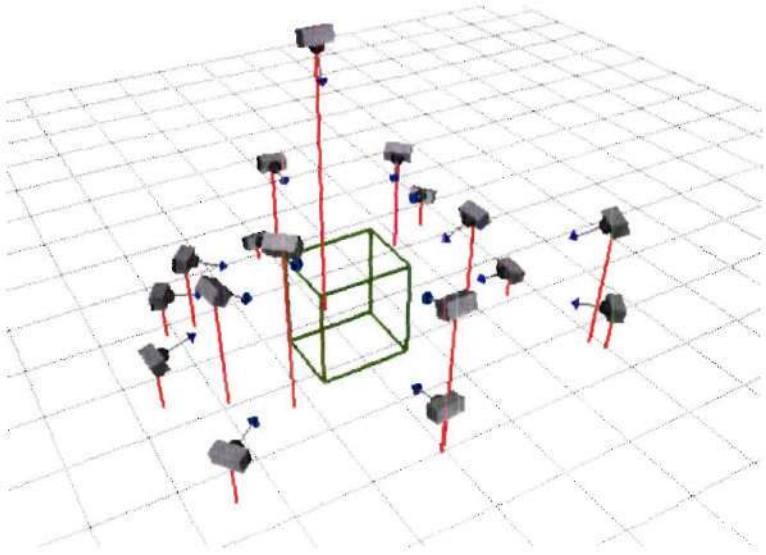


Figure 17: Visualization showing camera placements and reconstruction volume for the Ghirardelli data set.

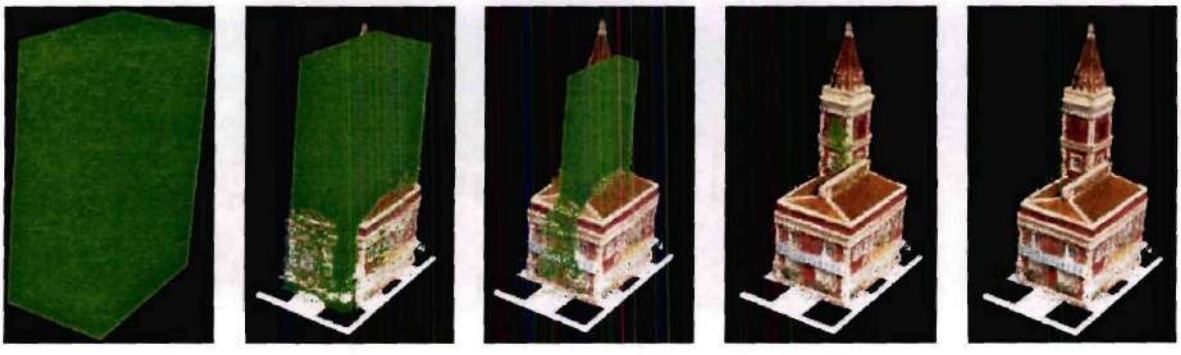


Figure 18: An example reconstruction using GVC-LDI.

Figure 19: New view synthesis via rendering of the reconstruction.

Reconstruction results for multiple data sets appear in Figure 20. In this figure, we show one reference view, one new view synthesized from the reconstruction, and the corresponding depth map for each data set. Details regarding the number and resolution of the images as well as the resolution of the voxel space for these data sets appear in Table 1.

Table 2 presents the runtime and memory requirements for GVC reconstructions of these data sets. Reconstructions were performed on a 2.0 GHz Pentium 4 machine with 1 GB of RAM. The algorithm ran completely in memory so there was no paging to the hard disk.

GVC-IB and GVC-LDI differ significantly in their memory usage. Storage of the reference views and the item buffers accounts for the majority of memory consumed by GVC-IB. In contrast, in GVC-LDI, the LDIs dominate the memory usage. Each LDI consumes an amount of memory roughly proportional to the number of image pixels times the depth complexity of the scene. Consequently, GVC-LDI uses considerably more memory than GVC-IB, as shown in Table 2. Memory consumed by the carve and SVL data structures is relatively insignificant, and therefore the voxel resolution has little bearing on the memory requirements for GVC-IB and GVC-LDI.

GVC-LDI executes significantly faster that GVC-IB. For the Ghirardelli, Broccoli, and

Data Set	Number of Images	Image Resolution	Voxel Resolution
Ghirardelli	17	1152 x 872	168 x 104 x 256
Broccoli	17	576 x 436	168 x 136 x 184
Tower	36	1536 x 1024	141 x 141 x 176

Table 1: Data sets and parameters for GVC reconstructions.

Data Set	Algorithm	Runtime (h:m:s)	Memory Usage
Ghirardelli	GVC-IB	1:28:39	157.2 MB
Ghirardelli	GVC-LDI	0:37:16	284.6 MB
Broccoli	GVC-IB	1:22:39	45.5 MB
Broccoli	GVC-LDI	0:39:00	177.9 MB
Tower	GVC-IB	2:19:29	472.1 MB
Tower	GVC-LDI	1:04:24	724.5 MB

Table 2: Runtime and memory usage for GVC reconstructions.

Tower data sets, GVC-LDI ran 2.4, 2.1, and 2.2 times faster than GVC-IB. There are two reasons for this improvement. First, GVC-LDI always uses the full visibility of the scene when computing photo-consistency. GVC-IB, in contrast, often uses out-of-date item buffers, resulting in some inconsistent voxels being declared photo-consistent on the Nth iteration of the algorithm. Once the item buffers are updated on the $N+1$st iteration, these inconsistent voxels are carved.

Second, GVC-LDI processes only the voxels that change visibility. Near convergence, only a small number of voxels are changing visibility so the algorithm is exceptionally fast. GVC-IB processes all voxels on the SVL, regardless of whether or not they have changed visibility. Medium to high resolution scenes will have a large number of voxels on the SVL, so GVC-IB will be significantly slower than GVC-LDI. By processing only the voxels that change visibility, GVC-LDI minimizes the number of photo-consistency checks required to reconstruct the scene.

In this thesis we do not compare results from the generalized voxel coloring algorithm

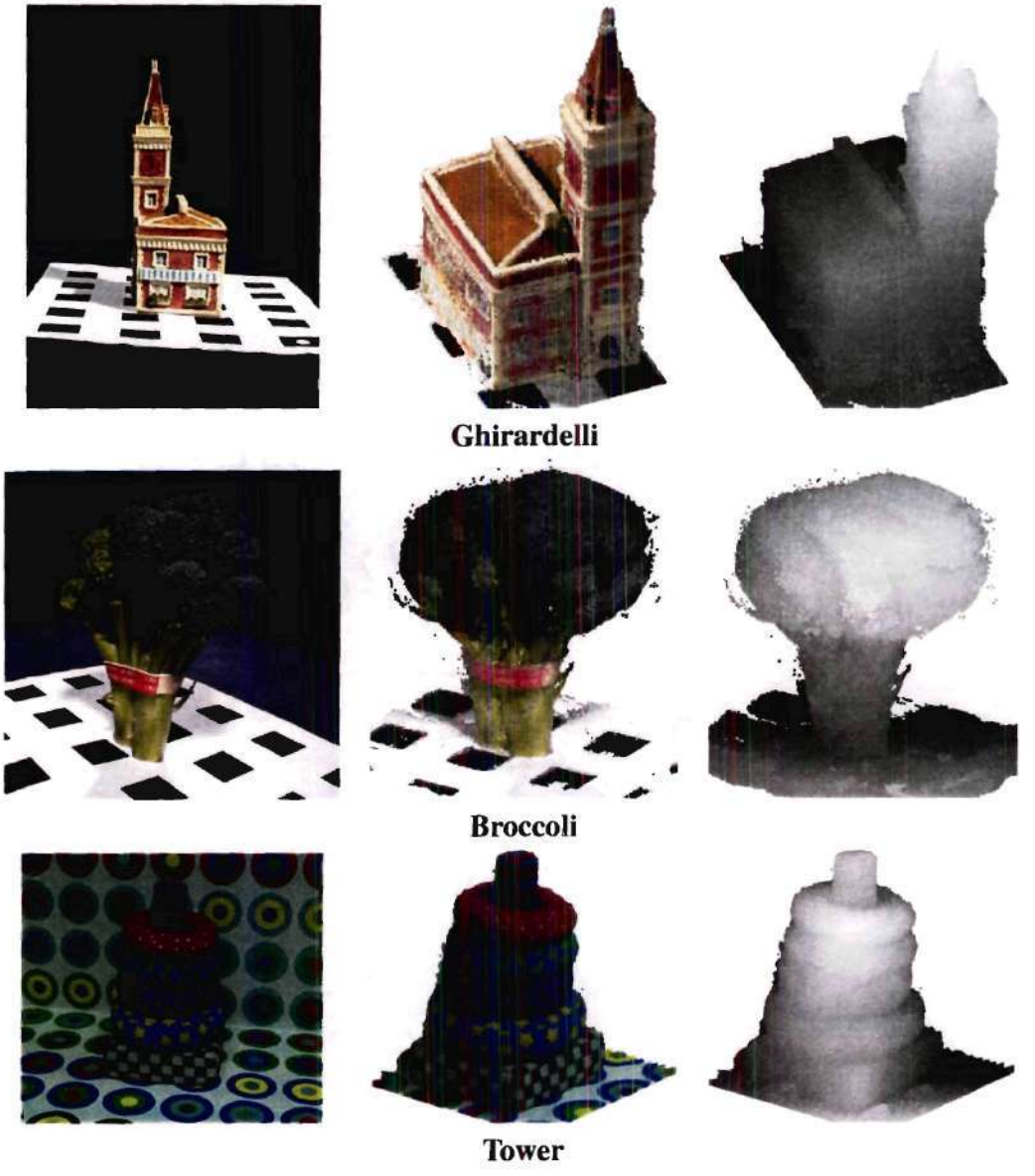


Figure 20: Reconstruction results for several data sets.

to those computed using the voxel coloring or Kutulakos and Seitz's space carving approach. The voxel coloring algorithm would not be able to reconstruct many of our scenes, like the Broccoli or Ghirardelli data sets, because it restricts the placement of cameras that are used to photograph the scene so that they satisfy the ordinal visibility constraint. We often photograph an object by placing cameras all around the object in a manner that does not satisfy the ordinal visibility constraint. We do not compare to Kutulakos and Seitz's space carving approach because there are two versions of their algorithm. One version does not use full scene visibility when reconstructing the scene. We performed a comparison to that version of their algorithm in our generalized voxel coloring paper [25], and found GVC to produce superior results. In response to GVC, the Kutulakos and Seitz's space carving algorithm was enhanced to use full scene visibility. We have not implemented this algorithm, but believe it would yield results similar to GVC.

3.5.4 Analysis

GVC produces a thin-shelled voxel surface, as demonstrated in Figure 21. This figure plots the voxels that lie on a plane that slices through the reconstruction of the Ghirardelli model. The region of space *inside* a surface is not visible to any camera, and is not processed by GVC because it never becomes exposed. In contrast, algorithms like voxel coloring and Kutulakos and Seitz's space carving approach visit every voxel in the voxel space.

Since GVC-IB and GVC-LDI do not stop carving until the remaining uncarved voxels are all color-consistent and since they never carve consistent voxels, we expect them to produce identical results when used with a monotonic photo-consistency function. However, in practice, monotonic photo-consistency functions can be hard to construct. An obvious monotonic approach to computing photo-consistency of a set of pixels P would threshold the maximum difference between the colors of any two pixels in P. However, using distance in the RGB cube as a difference measure, this function is $O(n^2)$ on the size of P and

Figure 21: GVC reconstructs a thin-shelled surface, as demonstrated in this slice running through a reconstruction.

has poor immunity to noise and high frequency color variation. The standard deviation-based photo-consistency measure we use is not monotonic. Consequently, GVC-IB and GVC-LDI generally produce models that are different but similar in quality.

We note in passing that GVC-IB can be parallelized, as well as hardware accelerated using standard graphics hardware. However, we do not describe these approaches in this thesis.

3.5.5 Multi-resolution GVC

To decrease runtime, GVC can be implemented in a coarse-to-fine fashion [91], as shown in Figure 22. We first perform a reconstruction at lower resolution using coarse voxels. At this lower resolution, we are able to carve away a large part of space that would require a lot of computation at a higher resolution. Once the reconstruction at the lower resolution is complete, we dilate the surface. After the dilation, we tessellate each voxel into eight sub-voxels, which doubles the resolution in each dimension. We then re-execute the algorithm at the higher resolution. This process continues until a desired resolution is obtained. The voxels in the final reconstructed model all have the same size. However, the size of carved voxels varies depending on the resolution of the voxel space when the carving occurred.

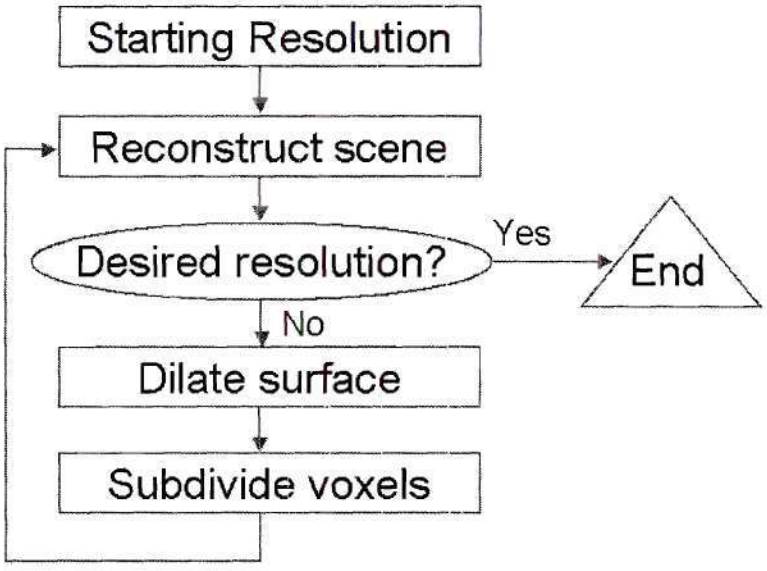


Figure 22: Flow diagram for multi-resolution GVC implementation.

Dilation of the surface prior to the resolution increase is necessary to prevent the evolving surface S from passing though fine details that cannot be properly modeled at a lower resolution. For example, consider a large voxel that contains a small patch of the true 3D surface T being reconstructed. If the voxel is projected into the reference views, the majority of the pixels in the voxel's projection will not represent T. Such a voxel could be inconsistent. At coarse resolutions this can cause some parts of T to go undetected. A simple solution, presented in [91], is to dilate S before the resolution increase. Doing so provides a mechanism for S to back up, and then reevaluate the skipped part of T at a higher resolution where it can properly be reconstructed.

We implemented this coarse-to-fine strategy in the GVC-LDI algorithm. Table 3 presents runtimes demonstrating the faster results produced by this multi-resolution approach. Compared with Table 2, we note that the multi-resolution reconstruction runs 1.9 times and 1.5 times faster than the single resolution implementation for the Ghirardelli and Broccoli data sets, respectively. The multi-resolution reconstruction of the Tower data set did not fit into main memory so we do not present runtime results in the table for this data set.

Data Set	Initial Resolution	Final Resolution	Runtime (h:m:s)
Ghirardelli	21 x 13 x 32	168 x 104 x 256	0:19:28
Broccoli	21 x 17 x 23	168 x 136 x 184	0:25:32

Table 3: Multi-resolution voxel spaces and runtimes.

In Chapter 8 we will compare GVC to other reconstruction approaches presented in this thesis.

3.6 Summary

In this chapter, we presented a new volumetric scene reconstruction algorithm called generalized voxel coloring. GVC uses color analysis to identify geometry in the scene that is consistent with the photographs taken of the scene. To properly compute photo-consistency, the visibility of voxels in the scene must be known. GVC-IB and GVC-LDI use item buffers and layered depth images, respectively, to compute this visibility information. The GVC-LDI approach is time efficient, while the GVC-IB method uses less memory.

GVC has a number of advantages over competing space carving methods, summarized below.

- GVC-IB and GVC-LDI are capable of reconstructing scenes photographed from arbitrarily placed cameras.
- GVC-IB and GVC-LDI do not process inner voxels.
- GVC-LDI uses a flexible data structure that minimizes photo-consistency checks. We will show in Chapter 4 that the layered depth images are useful in optimizing the reconstructed surface.

While GVC is a useful algorithm for reconstructing 3D scenes using multiple photographs, it does have some limitations. As will be shown in Chapter 4, the surface that is reconstructed can have a number geometric artifacts, some of which can be mitigated using

simple 3D morphological filtering. Additionally, smoothing the surface during reconstruction, a topic discussed in Chapter 6, helps reduce such artifacts.

GVC finds the largest 3D surface that projects to pixels that match based on the threshold(s) used in the photo-consistency measure. This approach does not necessarily find the photo hull, which optimally reproduces the photographs. To get closer to the photo hull, we present a volumetric optimization approach in Chapter 4 that refines a GVC reconstruction to better reproduce the photographs.

GVC is well suited to reconstructing small-scale scenes, such as those presented in this chapter. However, reconstructing large-scale scenes can become difficult as such a scene may require a prohibitively large number of voxels that must be processed. We address this issue in Chapter 5.

The amount of time required for a reconstruction can vary based on number of factors, including image resolution, voxel resolution, the number of images. Typically reconstructions take a few minutes to hours for the data sets used in this thesis. In Chapter 7, we introduce a new algorithm that can perform reconstructions in a fraction of a second. To do this, we trade-off geometric accuracy for speed, as well as move as many of the computations in the 2D space of the images instead of the 3D space of the scene.

In fact, the rest of this thesis can be viewed as extensions to GVC. These extensions are designed to improve model fidelity, increase applicability to a wider class of scenes, and improve reconstruction speed. Some open issues we do not address in this thesis include incorporation of motion constraints when reconstructing temporally varying scenes [15, 126], alternative photo-consistency measures for reconstructing non-Lambertian scenes [23, 52], and reconstruction using uncalibrated or weakly calibrated cameras [33, 95].

CHAPTER IV

POST-PROCESSING

Chapter 3 of this thesis presented new volumetric reconstruction algorithms that use multiple calibrated photographs of a visual scene to produce a 3D voxel-based model of the scene surfaces. While these algorithms often produce high quality models that can be rendered to produce new views of the scene, geometric inaccuracies can diminish the photo-realism of the synthesized views.

In this chapter we introduce some post-processing methods for improving the fidelity of the reconstructed model. We begin by enumerating the sources of geometric inaccuracies in multi-view reconstructions.

4.1 Geometric inaccuracies

The geometric inaccuracies in a reconstruction come from both theoretical and practical sources.

4.1.1 Theoretical sources

A point on the true scene geometry projects to photo-consistent colors in the reference views. However, the converse is not necessarily true: points that project to photo-consistent colors in the reference views may or may not lie on the true scene geometry.

As a result, the photo hull deviates from the actual geometry of the scene. This geometrical deviation appears in one of two related forms: *floaters* and *cusps*, both of which are shown in Figure 23. Both examples in the figure illustrate a multi-colored flat surface photographed by two cameras. In the left image, a floater appears as a small disconnected

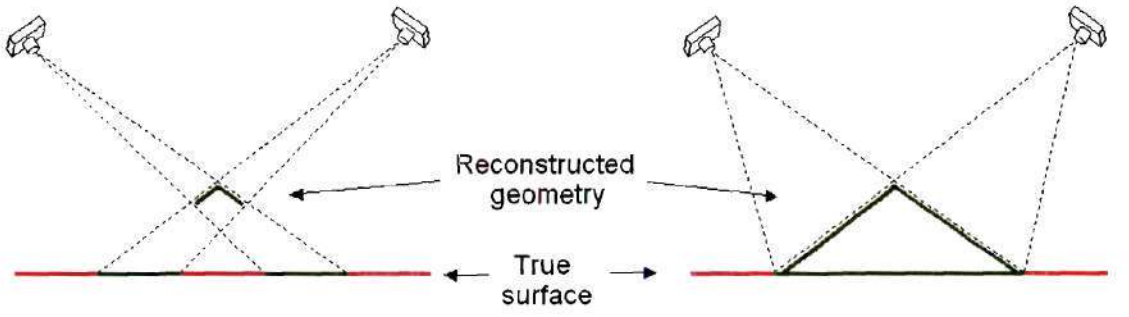


Figure 23: A multi-colored flat surface is photographed by two cameras in both examples. Due to floaters and cusps, the reconstructed geometry can deviate from the true scene.

piece of geometry, and typically is found floating off of the main surface. In the right image, a cusp appears as a wedge-shaped area that points towards the cameras. Any point on the cusp or floater projects to photo-consistent colors in the reference views, even though there might not be any scene geometry at that location.

Figure 24 shows the presence of cusps and floaters in GVC reconstructions. Cusps appear at locally homogeneously colored parts of a surface, which is the case for the white areas on the calibration paper in the tower model shown in part (a) of the figure. Floaters typically appear as a single or a small group of voxels floating near the main reconstructed surface.

The theory of the photo hull states that we cannot expect to get a better reconstruction unless we have some a priori knowledge about the scene. Just based on the photographs, the cusps and floaters in Figure 23 *could* be the true scene geometry, and appear because the photo hull is the largest photo-consistent reconstruction. However, if we knew a priori that the scene was was flat (or smooth), or was one connected piece, we could use this information to mitigate the effects of the cusps and floaters.

4.1.2 Practical sources

While one strives to reconstruct the photo hull when using a space carving algorithm like GVC, in practice, one often falls short of this goal. A number of sources of error prevent

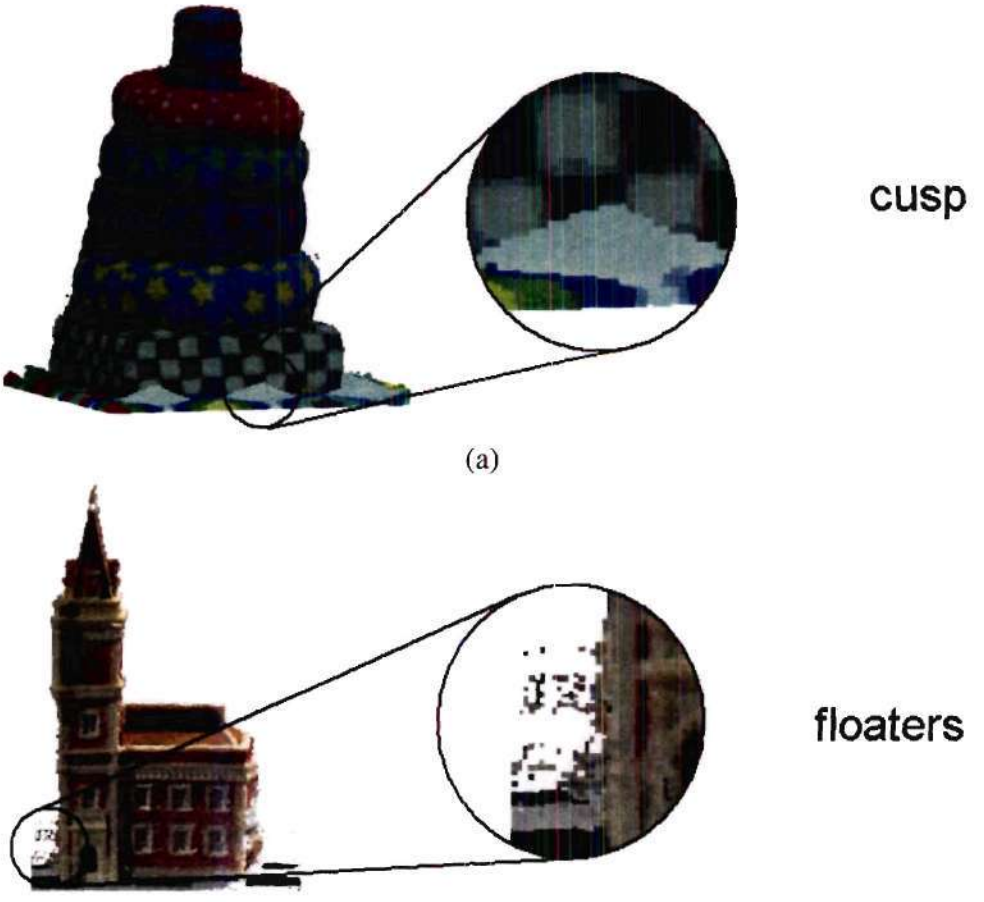


Figure 24: Reconstructions exhibiting cusps and floaters.

the algorithm from achieving the best possible reconstruction that can be inferred from N photographs. Below we enumerate the most significant sources of error.

4.1.2.1 Camera calibration errors

Camera calibration errors come in two forms: *geometric* and *photometric*. As a pre-process before reconstruction, the cameras used to photograph the scene are calibrated. The geometric calibration computes the intrinsic and extrinsic parameters of the camera so that points in 3D space can be projected into the reference views. Single and multiple camera geometry is described in Appendix A. Despite much progress in camera calibration research in the computer vision community, the process is still error prone. Typical errors are on the order of sub-pixel to a few pixels of reprojection error for the data sets used in this thesis.

Consequently, the projection of a voxel into a reference view is not exact. This results in incorrect pixels being used in the photo-consistency check, which in turn can yield misclassifications. Voxels that should be photo-consistent can erroneously be declared inconsistent, and voxels that should be inconsistent can be declared photo-consistent.

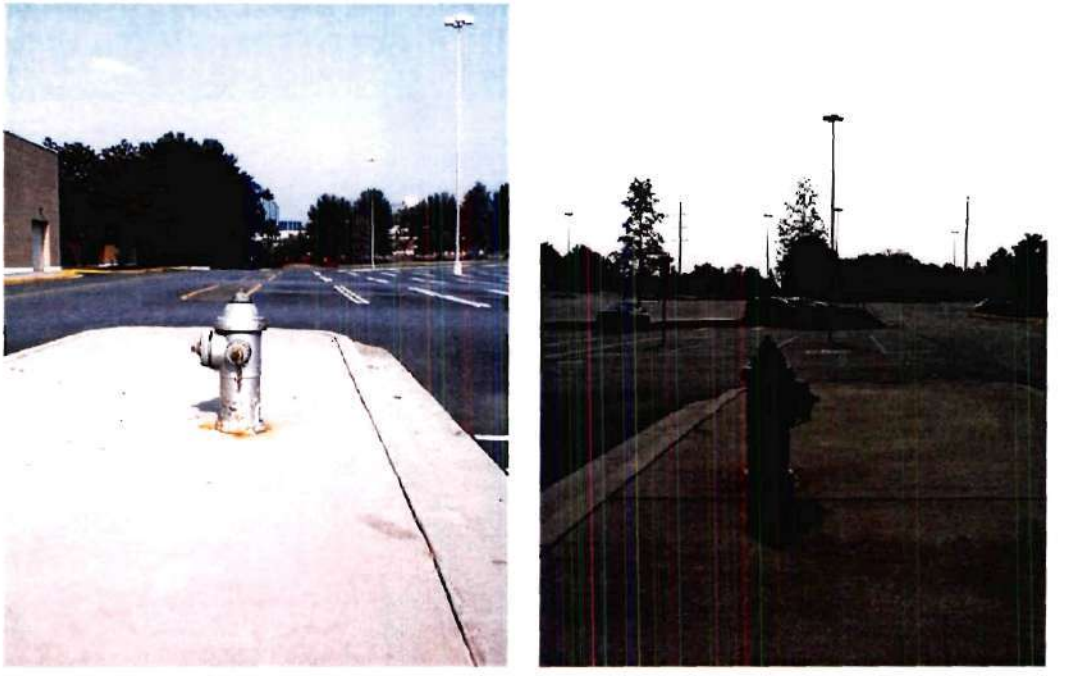
Intuitively, geometric calibration errors limit the resolution of the voxel space. Consider a small voxel that models the true surface. In each image, the voxel's footprint, shifted from its true location, might not contain any photo-consistent colors across the reference views, and the voxel would be carved. Using larger voxels helps mitigate this problem, since the footprint of the voxel's projection into each image will be contain a larger area. Kutulakos [55] presents a related method that helps address geometric calibration errors during reconstruction by looking for a matching pixel in a variably sized footprint in each of the reference views when determining photo-consistency. He shows that such a technique can reconstruct a scene photographed by cameras that significant calibration errors. However, the surface that is reconstructed is much larger and lower quality than the photo hull.

Photometric calibration errors are colorimetric variations that result from color misalignment. A variety of sources can cause these calibration errors; we describe a few here. As a simple example, consider the case of multiple cameras photographing a Lambertian object of a single color. Each camera has different controls, such as brightness and color balance that affect image quality. If each camera has different settings, errors in the photometric calibration can result in different RGB values to be recorded by each camera for the object. Large differences can cause problems when photo-consistency is computed.

Photometric differences can occur as the camera is moved about the scene, as shown in Figure 25. In this example, we photographed a scene on a sunny day using a digital camera. The two photos were taken within a minute of each other. With the sun to the photographer's back, the surfaces appear bright as shown in (a). However, when the photographer takes a photo from the other side of the scene, there is much more incident light on the camera, particularly from the sky. When the photo is taken, the camera's shutter is open for less time. The resulting image has a saturated sky and darker foreground objects. Corresponding regions in the two photos do not project to similar colors.

Photometric errors can even occur *within* a photo. Vignetting is a light falloff that is a characteristic of the camera lens, is present in nearly all photographic images, and is often modeled using a 4th order cosine. An example of vignetting appears in Figure 26. In this example, a cylindrical panorama was formed by rotating a 1D scanner around its optical axis. Vignetting appears vertically as a darkening of the pixels as one moves away from the vertical center of the image, as shown in part (a) of the figure. Part (b) shows the results of a photometric compensation for the vignetting. The colors of bricks on the ground in the compensated image appear much more uniform.

Without proper photometric calibration, the photo-consistency check is unreliable, since the true scene geometry will not project to photo-consistent colors in the reference views. Therefore, it is often necessary to perform a photometric adjustment prior to reconstruction.



(a) (b)

Figure 25: Photometric differences can occur when moving a camera in a scene.

4.1.2.2 Invalid photo-consistency measures

Reconstruction errors occur when the photo-consistency measure does not accurately account for the radiance in the scene. For example, when one assumes a Lambertian scene, the photo-consistency measure tries to match colors across the different viewpoints. However, if the scene contains specular surfaces, some viewpoints may observe specularities for a true surface point while others might not. These varied colors can result in a misclassification by the photo-consistency check.

4.1.2.3 Thresholds

Standard photo-consistency measures threshold a color matching metric. The threshold is typically found through experimentation by sweeping through the parameter space of the threshold until the best reconstruction is found. As Figure 27 indicates, if too low a threshold is used, few voxels will be declared photo-consistent. However, using a threshold

(a)

(b)

Figure 26: A photograph exhibiting vignetting appears in (a). The compensated image appears in (b).

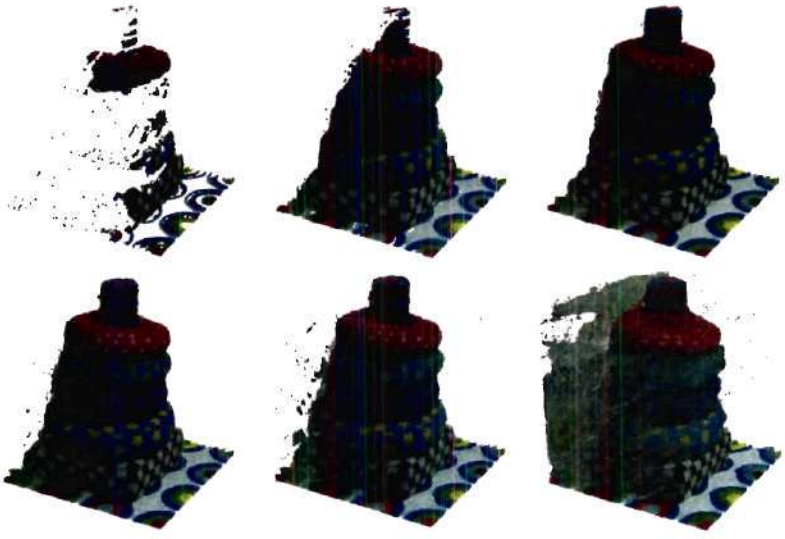


Figure 27: The effect of varying the photo-consistency threshold, from low (upper left) to high (lower right). The best reconstruction occurs for the image in the upper right.

that is too high results in a fattened model. Typically one uses the lowest threshold that reconstructs all scene surfaces.

Unfortunately, in general there is not a single threshold that is ideal for reconstructing *all* surfaces in the scene. For a given threshold, some surfaces could likely be more accurately reconstructed with a lower threshold. But lowering the global threshold can cause other surfaces to become carved that should be in the final reconstruction. Ideally, one would like a spatially adaptive threshold that is locally optimized. In Section 4.3 we will present a technique that achieves such a threshold.

4.1.2.4 Poor photographic sampling

Poor photographic sampling takes on several forms. For a local area on a surface to be properly reconstructed, it must be visible to at least two reference views. Thus, the resolution of the photographs must be sufficiently high so that a visible surface to be reconstructed projects to at least one pixel in two of the reference views. Additionally, one must carefully consider the camera placements when photographing the scene, particularly when dealing with complex self-occluding surfaces. Finally, the images should be in focus, as

reconstructing the scene using blurry photographs can result in fattened models.

4.1.2.5 *Insufficient voxel resolution*

The voxel space should have sufficient resolution to model the scene surfaces. Typically one represents a voxel as a cube in space with a single color. Many thin objects cannot be modeled with this representation. For example, consider a thin plane that is black on one side and white on another. A voxel occupying the space of the plane may project to different colors in the reference views even though the voxel models a scene surface. Increasing the resolution of the voxel space so that the white side of the plane is modeled with voxels distinct from the ones modeling the black side solves this problem. Alternatively, one could allow for a different color on each of the six sides of the voxel cube.

4.2 *Mathematical morphology*

One very straightforward way to improve model fidelity is to apply a 3D morphological processing to the reconstructed geometry. Simple operations such as openings and closings [16, 67] can remove floaters and fill holes in the model.

4.2.1 Basic morphological operations

We apply morphological operations to the carved array of Sections 3.5.1 and 3.5.2. This 3D binary array indicates if a voxel has been carved. The binary surface and its interior, denoted as C, are the set of voxels that have not been carved. Our morphological filters also employ a 3D binary structuring element, denoted as SE below.

The basic functions in mathematical morphology are erosion and dilation. Let SE_{xyz} be the structuring element after it has been translated so that its origin is located at the point (x,y,z). The *erosion* of C by the structuring element SE is defined as

$$C \ominus SE = \{x,y,z | SE_{xyz} \subseteq C\}. \tag{7}$$

This equation states that the binary volume that results from eroding C by SE is the set

of points (x, y, z) such that if SE is translated to (x, y, z), then it is completely contained within the uncarved region of C.

The *dilation* of C by SE is defined as

$$C \oplus SE = \{x, y, z | SE_{xyz} \cap C \neq \emptyset\}. \tag{8}$$

Thus, the binary volume that results from dilating C by SE is the set of points (x, y, z) such that if SE is translated to (x, y, z), then its intersection with the uncarved region of C is non-empty.

Erosion and dilation can be cascaded to produce other operations. An *opening* is an erosion followed by a dilation. An opening excels at removing small volumes such as floaters, as well as smoothing the surface without significantly changing its volume. Opening is defined as

$$C \circ SE = (C \ominus SE) \oplus SE. \tag{9}$$

A *closing* is a dilation followed by an erosion. A closing has the effect of filling holes in the surface and connecting nearby points, also without significantly changing its volume. Closing is defined as

$$C \bullet SE = (C \oplus SE) \ominus SE. \tag{10}$$

Openings and closings can be cascaded as well, to remove floating voxels and fill small holes if desired.

4.2.2 Morphological filtering of reconstructions

We have applied morphological filters to GVC reconstructions, illustrated in Figure 28. We use a cubical structuring element that is 3 x 3 x 3 voxels large. Part (a) shows a reconstruction of the Tower data set. There are small holes in on the plane of the paper in the reconstructed surface. While not very large, these holes are visually distracting when viewing animations of the reconstructed surface. The holes are filled by applying a closing to the surface, shown in (b). Part (c) of the figure shows a reconstruction of the Broccoli

data set. An opening operation removes the floaters seen hovering near the main surface, as demonstrated in part (d) of the figure.

Another simple way to remove the floaters is to perform a connected components analysis. This approach clusters the voxels into groups of connected pieces. Many of the scenes reconstructed in this thesis consist of one connected component. Therefore, any piece of geometry not connected to the largest component can be removed from the voxel space. Figure 29 shows the results of applying this filter to a reconstruction of the Ghirardelli data set. Note that the filtered surface contains no floaters disconnected from the main surface.

4.3 Volumetric optimization

The morphological approaches to model refinement presented in the previous section can be effective at filling small holes and removing floaters from the reconstructed geometry. However, these approaches have the disadvantage that they do not take photo-consistency into account when adding or removing voxels to the model. Consequently, they might make modifications to the surface that decrease visual quality. For this reason, we use a small structuring element when applying morphological operations to the reconstructed surface.

Recall that space carving algorithms find a set of voxels whose photo-inconsistency falls below a threshold. It would be preferable to have an algorithm that minimizes inconsistency to find the model that is most photo-consistent with the input images.

In this section, we present a method that post-processes a GVC reconstruction to minimize reprojection error, which measures how well projections of the reconstructed scene reproduce the photographs. The reprojection error, defined in image space, guides the refinement of the scene reconstruction in 3D object space. This refinement approach makes better use of all color information from all viewpoints, and thereby produces a model closer in quality to the photo hull.

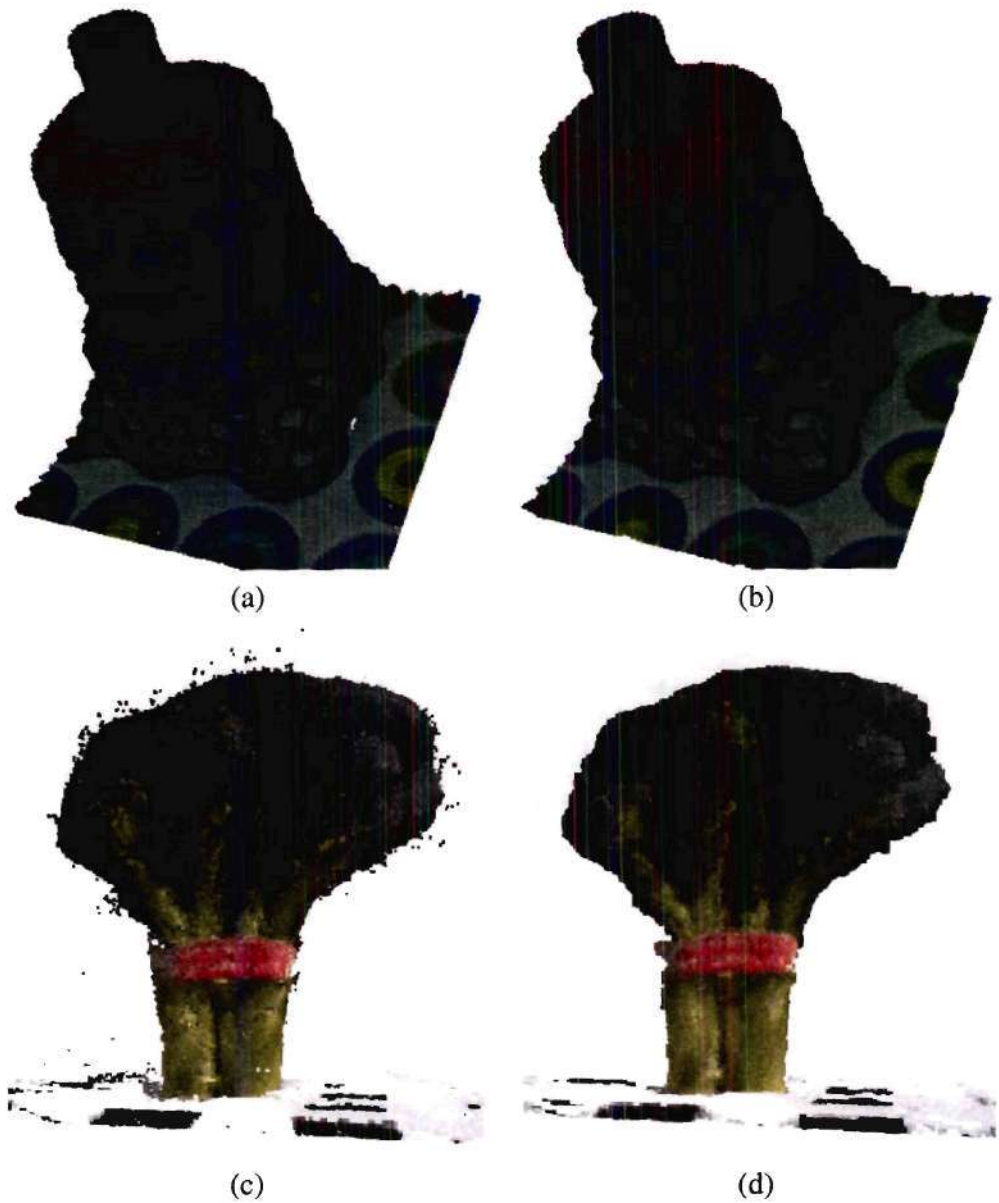
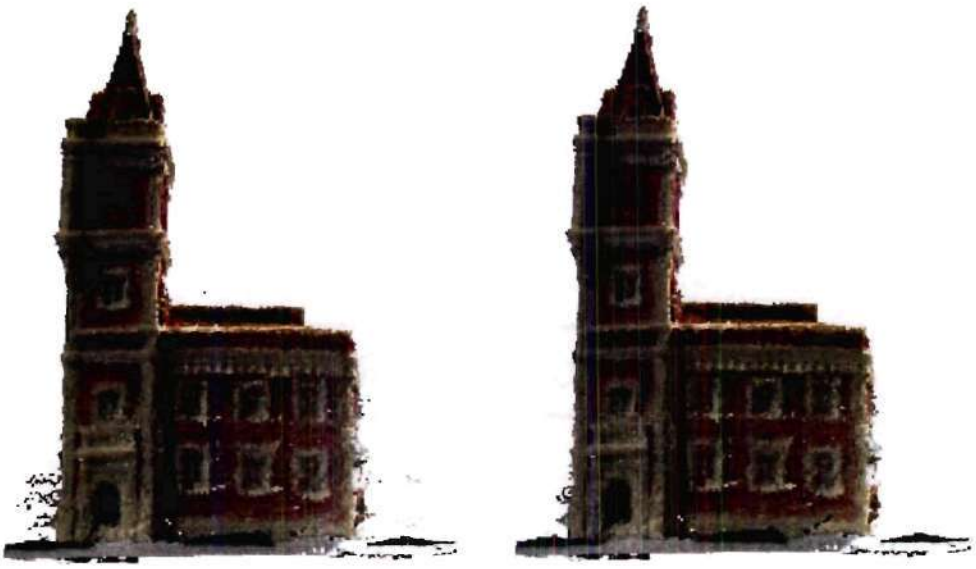


Figure 28: Morphological filtering of surfaces.



(a) (b)

Figure 29: Model refinement by connected components analysis.

4.3.1 Optimal reconstructions

Our goal is to find a 3D surface that optimally reproduces the photographs. To be more specific, we seek a volumetric surface S_o, which, when projected to each camera, minimizes a multi-view image space error function $E(S)$, whose argument is a volumetric reconstruction. Borrowing from [115], characteristics that $E(S)$ should possess include:

1. The function should incorporate information from all available viewpoints.

2. The function should weight the error so that viewpoints that have more visibility have a greater contribution to the error.

3. The function should be minimized by the true scene.

4. The function should provide a relative ordering of solutions, so as to rank the quality of reconstructed surfaces. If the volumetric surface S_1 is a better solution than S_2, then $E(S_1) < E(S_2)$.

5. The function should relatively simple to compute.

Let the reprojection of a reconstruction S into the ith camera be denoted as R_i. Also, let the original photograph at the ith camera be denoted as P_i. Since R_i and P_i are color images, they have pixels with r, g, and b components. Let these color components in the jth pixel of R_i be referenced as $R_i(j).r$, $R_i(j).g$, and $R_i(j).b$, respectively. Similarly, we denote the r, g, and b components of the jth pixel of P_i as $P_i(j).r$, $P_i(j).g$, and $P_i(j).b$.

Reprojection error computes the dissimilarity of R_i and P_i. We define it to be

$$E(S) = \frac{\begin{array}{r}(P_i(j).r - R_i(j).r)^2 + \\ \sum_{i=1}^{N}\sum_{j=1}^{M_i} \quad (P_i(j).g - R_i(j).g)^2 + \\ (P_i(j).b - R_i(j).b)^2\end{array}}{\sum_{i=1}^{N} M_i},$$

where N is the number of images, M_i is the number of pixels used in the comparison for the ith image, and R_i and P_i are images as described above. The reprojection error is the average squared difference between pixels in R_i and P_i, taken over the pixels M_i.

This function possesses the necessary characteristics mentioned above. Specifically, it incorporates information from all views, as the sum over i is over all viewpoints. The variable M_i weights the reprojection error so that viewpoints with greater visibility have a larger contribution. The function reaches its minimal value of zero for the true scene, for which R_i is identical to P_i for the pixels indexed by j. The function does provide a relative ordering of solutions, so that one reconstructed scene can be objectively ranked relative to another. Consequently, the reprojection error can guide changes to the surface during reconstruction. We will show that the difference in reprojection error can be computed incrementally, i.e., for only the pixels in R_i that change, resulting in an efficient computation.

4.3.2 Reprojection error and standard deviation

A comparison of the reprojection error function with that of the single threshold standard deviation measure described in Section 3.1.2 is useful in relating this new approach to standard space carving methods. The main difference between the two is that the standard deviation measure is defined on a per voxel basis. In contrast, the reprojection error

is defined over the entire projection of the reconstructed scene. A per scene basis for the reprojection error is necessary so that we can globally measure the effect of a surface modification. If however, only one voxel changes visibility, then the reprojection error is simply the square of the standard deviation used in voxel coloring. To see this, recall that during reconstruction, a voxel is projected into each image, and the set of visible pixels in all images, previously denoted as vis(V), is found. For simplicity, we change notation and now denote the pixels in vis(V) as X. If the voxel is photo-consistent, its color is set to equal the mean of X. Thus, for a given voxel, each color channel of $R_i(j)$ will be the expected value of the corresponding color channel in P_i, over all images i. We denote this expected value as

$$\begin{aligned} \mu.r &= E[X.r], \\ \mu.g &= E[X.g], \\ \mu.b &= E[X.b]. \end{aligned}$$

Let k index over pixels in X, and suppose the cardinality of the set is L. Then, the reprojection error has the form,

$$E(V) = \frac{\begin{array}{c}(X(k).r - \mu.r)^2 + \\ \Sigma_{k=1}^{L}\ (X(k).g - \mu.g)^2 + \\ (X(k).b - \mu.b)^2\end{array}}{L} = \sigma^2.$$

This is simply a formula for the sample variance [59] of X, equivalent to the square of the standard deviation used in voxel coloring, assuming a biased variance measure is used.

4.3.3 Optimal scene reconstructions exist but are not unique

An optimal scene reconstruction must exist. However, optimal reconstructions are not necessarily unique because of the ill-posed nature of the 3D reconstruction problem; many 3D surfaces can exist that globally minimize the reprojection error. Finding an optimal

scene reconstruction can be challenging, since the error landscape can contain numerous local minima far away from a global minimum.

4.3.4 The search space is large

Performing a combinatorial analysis can shed some light on the size of the search space. The search space for the optimal reconstruction is defined on a six-dimensional (x, y, z, r, g, b) domain for standard color scenes. Voxel opacity accounts for three of these dimensions, as it consists of a bit for the spatial location, defined on a three-dimensional (x, y, z) domain, of each voxel in the voxel space. Each opaque voxel can then have a unique color, defined on another three-dimensional (r, g, b) domain. Consider a voxel space containing v voxels. Since a voxel can either be opaque or transparent, there are 2^v possible voxel opacity configurations.

Now let us consider the number of different colorings of the 2^v possible voxel opacity configurations. Let the variable p be number of opaque voxels in a candidate configuration. Then, there will a total of

$$\begin{pmatrix} v \\ p \end{pmatrix}$$

configurations with p opaque voxels. If the number of possible colors for the voxel is c, then there are c^p possible colorings of the voxel configurations with p opaque voxels. The total number of candidate reconstructions, n, is then the sum of the possible colorings over all possible voxel geometries,

$$n = \sum_{p=0}^{v} \begin{pmatrix} v \\ p \end{pmatrix} c^p.$$

This sum can be evaluated using the binomial theorem, yielding

$$n = \sum_{p=0}^{v} \begin{pmatrix} v \\ p \end{pmatrix} c^p = (c+1)^v.$$

A simple example of this is provided in figure 30.

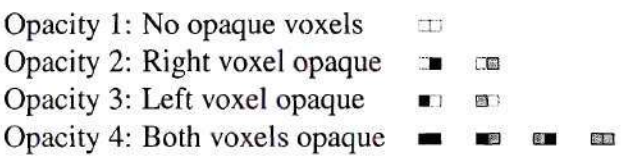


Figure 30: Example voxel opacity configurations and colorings for $v = 2$, $c = 2$. Suppose that we have a reconstruction volume that contains two voxels ($v = 2$), and each opaque voxel can take on one of two possible colors, ($c = 2$), which are gray and black. Then, there are $2^v = 4$ unique voxel opacity configurations, and $(c+1)^v = 9$ possible reconstructions to check. These nine reconstructions, grouped into the four voxel opacity configurations, are shown above.

When working with real world scenes, both the values of v and c are quite large. If we use 24-bit images, a voxel can have one of $c = 2^{24}$ unique colors, so

$$n = (2^{24}+1)^v \approx 2^{24v}.$$

Since v is typically large, the number of possible reconstructions defined on this six-dimensional domain is enormous. Even if one ignores color from this analysis, the 2^v possible voxel opacity configurations is unsearchably large for typical scenes. Thus, we will require effective algorithms to limit our searching in this huge space.

4.3.5 Minimizing the reprojection error

We have developed two different methods to minimize the reprojection error. One method performs a greedy optimization. This *greedy approach* is discussed in Section 4.3.5.1. The other method uses simulated annealing to minimize the reprojection error. This *simulated annealing* approach is discussed in Section 4.3.5.2. But first, we present what these two approaches have in common.

The general methodology for both approaches is the same. First, a space carving algorithm such as GVC is executed. This produces a reconstruction that can be quickly computed, and should be reasonably close to a global minimum in the reprojection error. Then, this volumetric surface is refined by either removing surface voxels, or adding voxels that are adjacent to surface voxels. We call these adjacent voxels *neighbor* voxels. By

attempting to carve surface voxels, we provide a mechanism for the reconstructed surface to thin. We exploit the spatial coherency of surfaces by attempting to add neighbor voxels. This provides a mechanism for the reconstructed surface to grow. After modifying the surface, the methodology incrementally evaluates the reprojection error, upon which the decision to accept or undo the modification is made.

One of the key aspects underlying our two algorithms is the incremental update of visibility, the reprojection error, and the reprojected images. During a surface modification, often just a few voxels change visibility. Rather than re-render the entire scene reconstruction to determine visibility from scratch, we use the layered depth images of Section 3.5.2 to incrementally update visibility for only the voxels whose visibility changes. Using the LDIs, our algorithms efficiently compute the change in reprojection error only over the pixels that change visibility. If the modification is accepted, the reprojected images R_i are updated for only those pixels as well. Incremental updates of visibility, reprojection error, and reprojected images provide raw ingredients for optimization: efficient ways to modify the model and compute an error function.

In our methodology, modifications to the surface are made tentatively, for if it is determined that a modification results in a less favorable surface, the modification is undone. In our implementation, we make tentative changes to the LDIs. For each pixel that changes on an LDI, we store undo information that is used only if we undo the surface modification. All other changes are stored in temporary data structures, whose elements are copied to permanent data structures upon acceptance, or discarded upon rejection, of the surface modification.

One of the flexible features of space carving algorithms is their ability to reconstruct a scene using either segmented or unsegmented photographs. A photograph P_i is segmented if each pixel is labeled as foreground or background. Foreground pixels correspond to objects that should be reconstructed, while background pixels correspond to objects that should not be reconstructed. If the algorithm is given such information, it computes the

reprojection error over the union of the foreground pixels, i.e., P_i, with the pixels in the reprojection of the reconstructed scene, i.e., the pixels in R_i. However, not all scenes have easily segmentable photographs. For these latter scenes, we compute the reprojection error over the pixels in R_i.

See [108] for pseudo-code and in-depth implementation details.

4.3.5.1 Greedy method

In the greedy method, voxels are added or carved only if they reduce the reprojection error. The reprojection error is computed for the surface before modification, yielding the quantity E_{old}, and after modification, yielding the quantity E_{new}. If the change in reprojection error,

$$\Delta E = E_{new} - E_{old}$$

is negative, the reprojection error has decreased and the surface modification is accepted. If this quantity is nonnegative, the surface modification is undone. Since scenes reconstructed by space carving algorithms tend to be larger than the true scene, our implementation first performs a greedy carving pass that thins the reconstruction, and then performs a greedy adding pass. By executing these two sequential passes, we avoid attempting to add voxels to the fattened model, as such voxels are unlikely to decrease the reprojection error. One application of the greedy method usually results in much improved surfaces. Additional applications of the greedy method can further reduce the reprojection error, with diminishing rewards each time the algorithm is applied. In practice, we typically apply the algorithm once to a reconstruction.

4.3.5.2 Simulated annealing method

The reprojection error can contain numerous local minima. Since the greedy method only accepts changes that decrease the reprojection error, it has the disadvantage that it could get caught in a local minimum of the error, possibly far from a global minimum. To address this issue, we developed a simulated annealing [89] method that accepts some changes

that increase the reprojection error. This method introduces a few new variables, namely, k, the Boltzmann constant, and T, the temperature. Surface modifications that lower the reprojection error are always accepted. However, surface modifications that increase the reprojection error are accepted with probability

$$p = e^{-\Delta E/kT}.$$

Initially, the temperature is high, and it is more likely that unfavorable changes will be accepted. Accepting such an unfavorable change can dislodge the algorithm if it is caught at a local minimum. As the program runs, the temperature is slowly decreased to zero. At zero, the simulated annealing method no longer accepts unfavorable changes.

Since this method can make unfavorable surface modifications, it simultaneously attempts to add and carve voxels, unlike the sequential nature (carving pass, then an adding pass) of the greedy algorithm. This prevents the algorithm from adding (carving) many unfavorable voxels without providing an opportunity to carve (add) them.

4.3.6 Results and Analysis

We have executed our volumetric optimization algorithms on several data sets, both real and synthetic. For experimental runs we used an HP J5000 workstation, with a 440 MHz processor and 2 GB of RAM.

4.3.6.1 Toycar scene

We executed our algorithms on a data set we call Toycar, which consists of seventeen 800 x 600 images of a synthetic scene that is relatively easy to reconstruct, as the scene is colorful, the cameras are perfectly calibrated, and the background has been subtracted. Three reference views are shown in Figure 31. We first performed a reconstruction in a 168 x 120 x 104 volume using the GVC-LDI algorithm, which took 39 minutes to complete. Then the reprojection error was minimized using the greedy and simulated annealing algorithms.

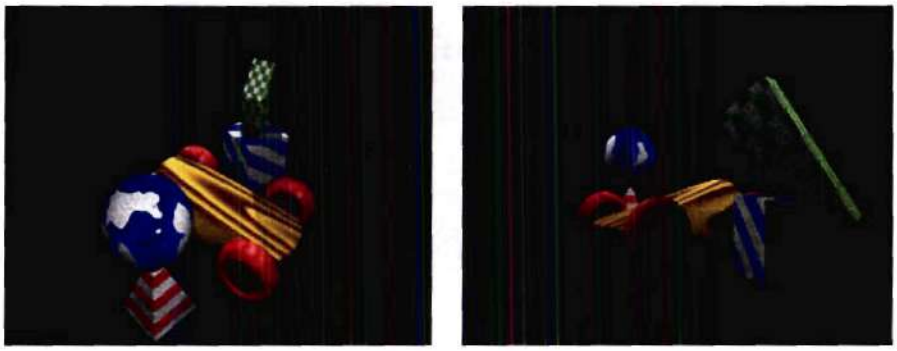


Figure 31: Toycar data set: two of 17 reference views

Method	Greedy	Sim. annealing
Execution time (min)	134	269
Memory usage (MB)	233	270
E_{start}	4373	4373
E_{end}	1747	1727
Improvement	60.1%	60.5%

Table 4: Results for the Toycar scene

We define the improvement, I, of the reprojection error to be

$$I = (E_{start} - E_{end}) / E_{start},$$

where E_{start} and E_{end} are the reprojection errors at the start and end of the reprojective error minimization algorithm, respectively.

The execution time, memory usage, and percentage improvement for the two methods are reported in Table 4. Since the simulated annealing method performs surface modifications that increase the error, many of which are later undone, it has a longer execution time. The memory usage of both algorithms was dominated by the LDIs, and was found to be comparable. Both methods significantly reduced the reprojection error for this scene, offering an improvement of about 60%. Figure 32 plots the reprojection error as a function of time. The simulated annealing approach ultimately gets to a slightly lower reprojection error, but at the expense of taking longer to complete.

Sections of input and reprojected images are shown in Figure 33. We show sections of

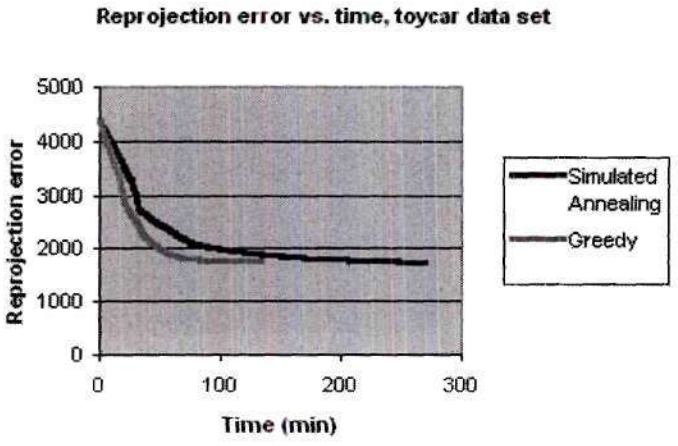


Figure 32: Reprojection error vs. time for Toycar scene

the images in order to zoom in so that details in the images are more apparent. The primary benefit that reprojective optimization offers is the thinning of the fattened reconstruction output by GVC. Reprojections of the refined reconstructions output by the optimization algorithms more closely match the input photographs. In particular, notice how the globe in (c) and (d) has a more reasonable size compared with that of (b). This thinning effect is quite apparent when one makes a simple animation that cycles between these images. Since the thinned surface geometry more closely matches that of the true scene, voxels receive a more accurate coloring. For example, in (b) notice how the small island near the center of the globe's projection is barely visible, and the striped pyramid has a noisy appearance. In (c) and (d), the island is much more discernable, and the pyramid has a better coloring. The refined scene is significantly improved, but is not perfect. For example, the simulated annealing approach has a few mis-colored voxels in the striped pyramid.

4.3.6.2 Shoes scene

Our Shoes data set consists of twelve 1536 x 1024 photographs of a cloth toy and two pairs of shoes on multi-colored paper. This data set is more challenging to reconstruct, as it is not segmented and has camera calibration errors. We first performed a reconstruction in a

(a) (b) (c) (d)

Figure 33: Refinement of the Toycar reconstruction. A section of one of the input photographs of the scene is shown in (a), with the silhouette of the globe outlined in white. The GVC-LDI algorithm was used to find a scene reconstruction, which was reprojected to the same viewpoint as that of (a), yielding (b). The reconstructed scene was refined using the greedy algorithm (c) and the simulated annealing algorithm (d). In (b) through (d), the the silhouette of the globe from the input photograph is added to demonstrate that the fattened model of (b) is refined in (c) and (d).

Method	Greedy	Sim. annealing
Execution time (min)	491	709
Memory usage (MB)	809	818
E_{start}	892	892
E_{end}	595	557
Improvement	33.3%	37.6%

Table 5: Results for the Shoes scene

144 x 128 x 80 volume using the GVC-LDI algorithm, which took 71 minutes to complete. Then, our optimization algorithms were executed. Table 5 presents the execution time, memory usage, and percentage improvement for the two methods. As for the Toycar scene, the simulated annealing method took considerably more time to finish, but had a slightly better improvement. Figure 34 shows the reprojection error as a function of time for the two methods.

A section of an input photograph and reprojected images are shown in Figure 35. The refined scenes are both objectively and subjectively improved. For this scene, notice that the simulated annealing algorithm does a better job than the greedy algorithm in reconstructing

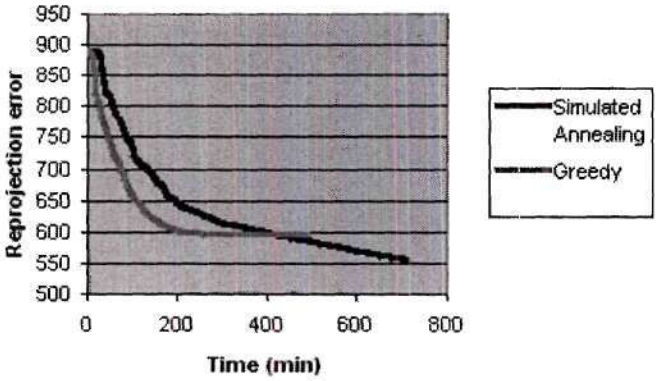


Figure 34: Reprojection error vs. time for Shoes scene

the black and white checkerboard cushion.

In Chapter 8 we will compare the volumetric optimization algorithm to other reconstruction approaches presented in this thesis.

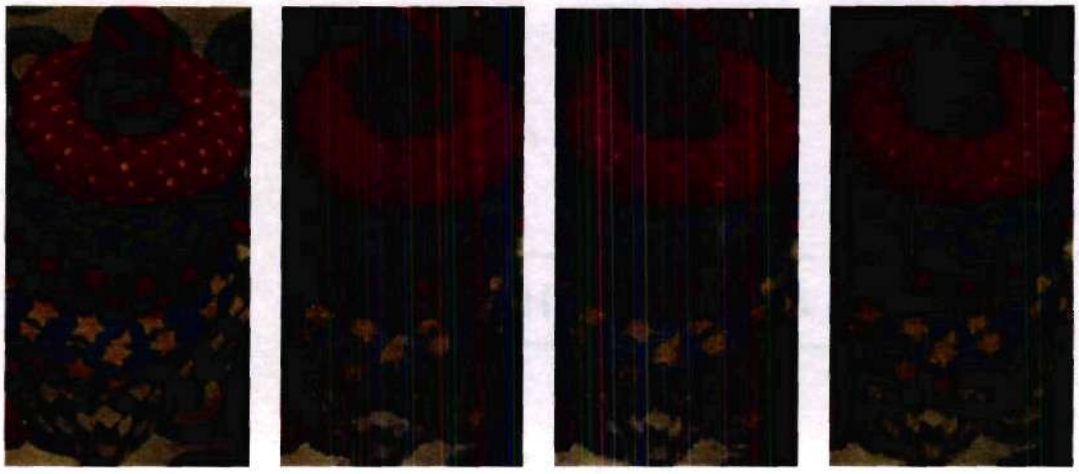
4.4 Other methods

Other methods for refining the reconstructed model have been developed. Dinh, Turk, and Slabaugh [29, 30] describe a method that fits an implicit surface to the reconstructed geometry. The approach places either radially symmetric [29] or anisotropically distorted [30] radial basis functions at scanned surface locations. In the presence of noise, direct interpolation of the scanned surface points results in a noisy model. Instead, this approach finds a reasonable surface that approximates the geometrical data. This method additionally converts the voxel-based model to a polygonal model well suited for rendering with standard graphics hardware.

4.5 Summary

This chapter presented post-processing methods that take a volumetric reconstruction as input and then refine the surface geometry to improve model fidelity.

After describing the major sources of geometric error exhibited in volumetric reconstructions, we demonstrated that simple morphological filtering of the surface can fill small



(a) (b) (c) (d)

Figure 35: Close up of cloth toy. As before, (a) is from the input photograph, (b) is a reconstruction using the GVC-LDI algorithm, and (c) and (d) are refined scenes using the OVC greedy and simulated annealing algorithms, respectively. Notice how more detail is visible in (c) and (d) than in (b); in particular, the white speckles in the topmost cushion, and the black and white checkerboard pattern on the lowest cushion.

holes and remove floaters. While morphological filtering can be effective, it does not take photo-consistency into account when making changes to the surface. Consequently, we use a small structuring element when applying morphological operations to the reconstructed surface to prevent extensive filtering of the surface.

We then described a volumetric optimization approach that computes a volumetric surface that strives to optimally reproduce the photographs. This approach defines reprojection error, which measures how well a volumetric reconstruction reproduces the photographs. We then use reprojection error to guide surface refinement. We developed greedy and simulated annealing approaches to minimizing the reprojection error. We demonstrated that modifications to the surface that minimize reprojection error visually improve model fidelity.

The volumetric optimization approach described in this thesis is brute-force in the sense that it tries removing every voxel from surface, and tries adding neighbors to every voxel on the surface. While this approach works, an alternate, possibly more efficient approach

could use a reprojection error gradient to indicate which way the surface should be deformed in order to most effectively minimize reprojection error. We leave this as future work.

The resolution of the voxel space will have an effect on the value of the reprojection error. Recall that each voxel is given a single color in our representation. Even if we were able to find the photo hull, which optimally reproduces the photographs, depending on the resolution of the voxel space, the reprojection error would be non-zero. A coarse voxel that projects to many differently colored pixels in a reference view would contribute a non-zero amount to the reprojection error. At higher resolutions of the voxel space, this quantization error component of the reprojection error decreases. When the voxel resolution is sufficiently high so that each voxel projects to one pixel in each of the reference views, this quantization error term becomes zero.

To decrease runtime of our optimization method, it would be possible to execute the algorithm in coarse-to-fine fashion similar to the multi-resolution implementation of GVC described in Section 3.5.5. This method would start with a coarse resolution, find an optimal reconstruction, and then increase resolution. The algorithm could iterative in this fashion until a desired resolution is reached. Such an approach would be less susceptible to local minima at higher resolutions.

In conclusion, post-processing a volumetric reconstruction computed using a space carving algorithm like GVC can significantly improve model fidelity. In Chapter 6, we will revisit this issue of mitigating photo hull geometric distortions like cusps and floaters, but instead of post-processing a voxel-based surface, we will incorporate smoothness constraints on an implicit surface during reconstruction.

CHAPTER V

VOLUMETRIC WARPING

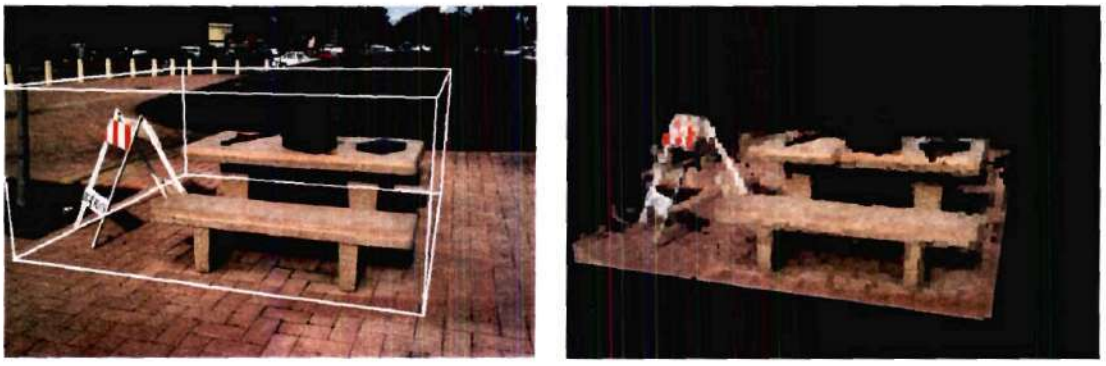
The space carving algorithms discussed so far in this thesis are often successful at reconstructing small-scale scenes defined on a limited spatial domain. Applying them to large-scale scenes can become challenging, since one must use a large reconstruction volume to contain the scene. Such a large reconstruction volume can consist of an unwieldy number of voxels that becomes prohibitive to process.

In this chapter, we present a method that warps the voxel space, so that the domain of the reconstruction extends to an infinite or semi-infinite volume [110]. Doing so enables the reconstruction of objects far away from the cameras, as well as reconstruction of a background environment. As we will show, our warping approach produces a spatially adaptive voxel size that allows for foreground objects to be reconstructed at higher resolution, while objects farther away from the cameras are reconstructed at lower resolution as they project to fewer pixels in the reference views.

5.1 Background modeling

Space carving algorithms are not well suited to capturing the environment (sky, background objects, etc.) of a scene. Typical reconstructions are photo-realistic in the foreground, which is modeled, but empty in the background, which is unmodeled. As a result, synthesized new views can have large "unknown" regions, as shown in black in Figure 36. For some scenes, such as an outdoor scene, we might like to reconstruct the background as well, yielding a more photo-realistic reconstruction.

In the computer graphics domain, infinite scenes have been modeled and rendered using environment mapping. This method projects the background onto the interior of a sphere



(a) (b)

Figure 36: Unknown regions because of reconstruction on a finite domain. A photograph of our Bench scene is shown in (a), with the reconstruction volume superimposed. Only the voxels within the reconstruction volume are considered during reconstruction. The scene contains many objects outside of the reconstruction volume that are not reconstructed, resulting in unknown regions that appear as black in a projection of the reconstruction, shown in (b).

or cube that surrounds the foreground scene. Blinn and Newell [9] use such an approach to synthesize reflections of the environment off of shiny foreground surfaces, a procedure known as reflection mapping. Greene [45] additionally renders the environment map directly to generate views of the background. This approach is quite effective at producing convincing synthetic images. However, since the foreground and background are modeled differently, separate mechanisms must be provided to create and render each. Furthermore, the three-dimensionality of the environment is lost, as the background is represented as a texture-map. Like environment mapping, the techniques described in this chapter seek an efficient mechanism to represent the background scene. Our warped volumetric space provides this in a single framework that can more easily accommodate surfaces that appear both in the foreground and background. We reconstruct the background scene three-dimensionally using computer vision methods.

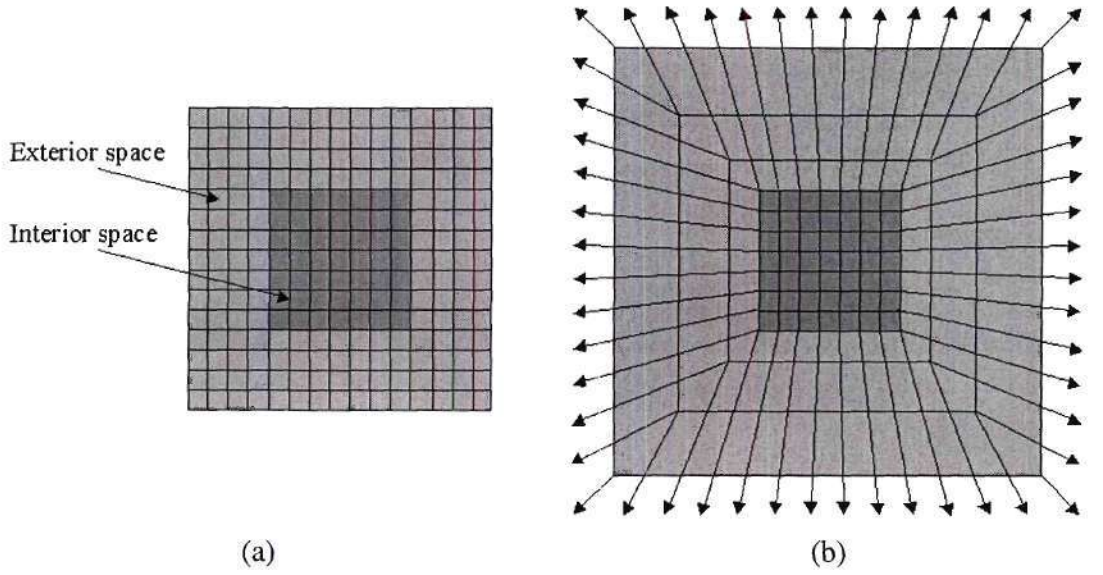


Figure 37: Pre-warped (a) and warped (b) voxel spaces shown in two dimensions. In (a), the voxel space is divided into two regions; an interior space shown with dark gray voxels, and an exterior space shown with light gray voxels. Both regions consist of voxels of uniform size. The warped voxel space is shown in (b). The warping does not affect the voxels in the interior space, while the voxels in the exterior space increase in size further from the interior space. The outer shell of voxels in (b) are warped to infinity, and are represented with arrows in the figure.

5.2 Volumetric warping

The goal of a volumetric warping function is to represent an infinite or semi-infinite volume with a finite number of voxels, while satisfying the requirement that no voxels overlap and no gaps exist between voxels. There are many possible ways to achieve this goal. In this section, we use the term *pre-warped* to refer to the volume before the volumetric warping function is applied.

The volumetric warping method presented here separates the voxel space into an interior space used to model foreground surfaces, and an exterior space used to model background surfaces, as shown in Figure 37 (a). The volumetric warp does not affect the voxels in the interior space, providing backward compatibility with previous space carving algorithms, and allowing reconstruction of objects in the foreground at a fixed voxel resolution.

Voxels in the exterior space are warped according to a warping function that changes the size of the voxel based on its distance from the interior space. The further a voxel in the exterior space is located from the interior space, the larger its size, as shown in Figure 37 (b). Voxels on the outer shell of the exterior space have coordinates warped to infinity, and have infinite volume. Note that while the voxels in the warped space have a variable size, the voxel space still has a regular 3D lattice topology.

To help further limit the class of possible warping functions, we introduce the following desirable property of a warped voxel space:

Constant footprint property: For each reference view, voxels project to the same number of pixels, independent of depth.

Figure 38 shows an example of a voxel space that satisfies the constant footprint property for two cameras. Assuming perspective projection, a voxel space that satisfies this property has a spatially adaptive voxel size that increases away from the cameras, in a manner perfectly matched with the images. While a useful conceptual construct, the constant footprint property cannot in general be satisfied when more than N cameras are present in R^N space. Thus, for three-dimensional scenes, a voxel space cannot be constructed that satisfies the property for general camera placement when there are more than three cameras. Since reconstruction using three or less cameras is limiting, we instead design our volumetric warping function to approximate the constant footprint property for an arbitrary number of images.

5.2.1 Frustum warp

We now describe a frustum warp function that is used to warp the exterior space. We develop the equations and figures in two dimensions for simplicity; the idea easily extends to three dimensions.

The frustum warp assumes that both the interior space and the pre-warped exterior space have rectangular-shaped outer boundaries, as shown in Figure 39. The pre-warped exterior

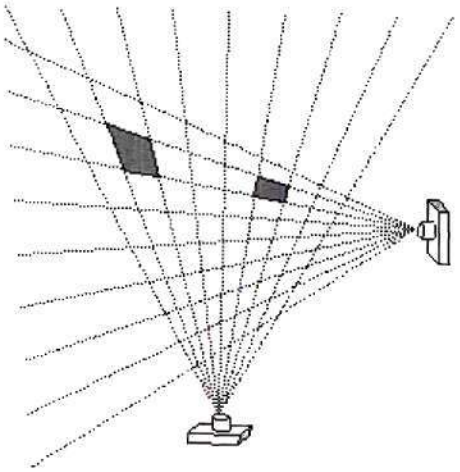


Figure 38: Example of a 2D voxel space that satisfies the constant footprint property for two images. Notice that the two filled in voxels project to the same number of pixels in the right image, regardless of their respective distance from the camera. Note that this figure is solely used to illustrate the constant footprint property; the warped voxel space developed and used in this thesis actually looks like that of Figure 37 (b).

space is divided into four trapezoidal regions, bounded by (1) lines l connecting the four corners of the interior space to their respective corners of the exterior pre-warped space, (2) the boundary of the interior space, and (3) the boundary of the pre-warped exterior space. We denote these trapezoidal regions as $\pm x$, and $\pm y$, based on the region's relative position to the center of the interior space. These regions are also shown in Figure 39.

Let (x,y) be a pre-warped point in the exterior space, and let (x_w,y_w) be the point after warping. To warp (x,y), we first apply a warping function based on the region in which the point is located. This warping function is applied to only one coordinate of (x,y). For example, suppose that the point is located in the $+x$ region, as depicted in Figure 40. Points in the $+x$ and $-x$ regions are warped using the x-warping function,

$$x_w = x\frac{x_e - x_i}{x_e - |x|}, \tag{11}$$

where x_e is the distance along the x-axis from the center of the interior space to the outer boundary of the exterior space, and x_i is the distance along the x-axis from the center of

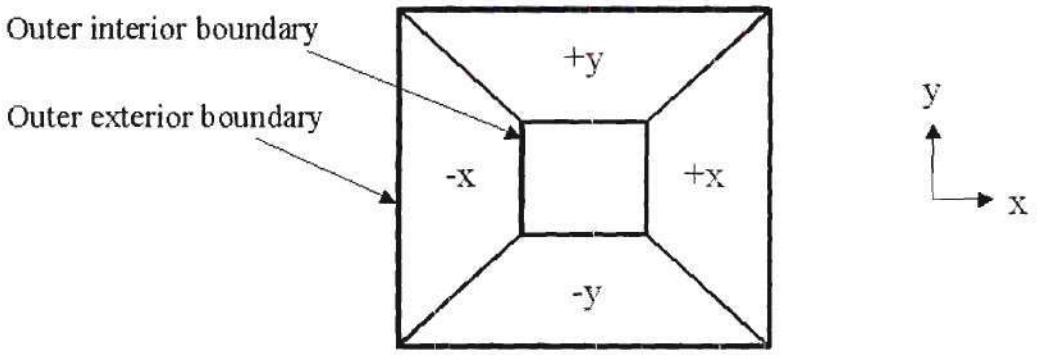


Figure 39: Boundaries and regions. The outer boundaries of both the interior and exterior space are shown in the figure. The four trapezoidal regions, $\pm x$ and $\pm y$ are also shown.

the interior space to the outer boundary of the interior space, shown in (a) of Figure 40. A quick inspection of this warping equation reveals its behavior. For a point on the boundary of the interior space, $x = x_i$, and thus $x_w = x_i$, so the point does not move. However, points outside of the boundary get warped according to their proximity to the boundary of the exterior space. For a point on the boundary of the exterior space, $x = x_e$, and so $x_w = \infty$.

Continuing with the above example, once x_w is computed, we find the other coordinate y_w by solving a line equation,

$$y_w = y + m(x_w - x), \tag{12}$$

where m is the slope of the line connecting the point (x, y) with the point a, shown in (b) of Figure 40. Point a is located at the intersection of the line parallel to the x-axis and running through the center of the interior space, with the nearest line l, as shown in the figure. Note that in general, point a is not equal to the center of the interior space.

As shown above, the exterior space is divided into four trapezoidal regions for the two-dimensional case. In three dimensions, this generalizes to six frustum-shaped regions, $\pm x$, $\pm y$, $\pm z$; hence the term *frustum warp*. There are three warping functions, namely the

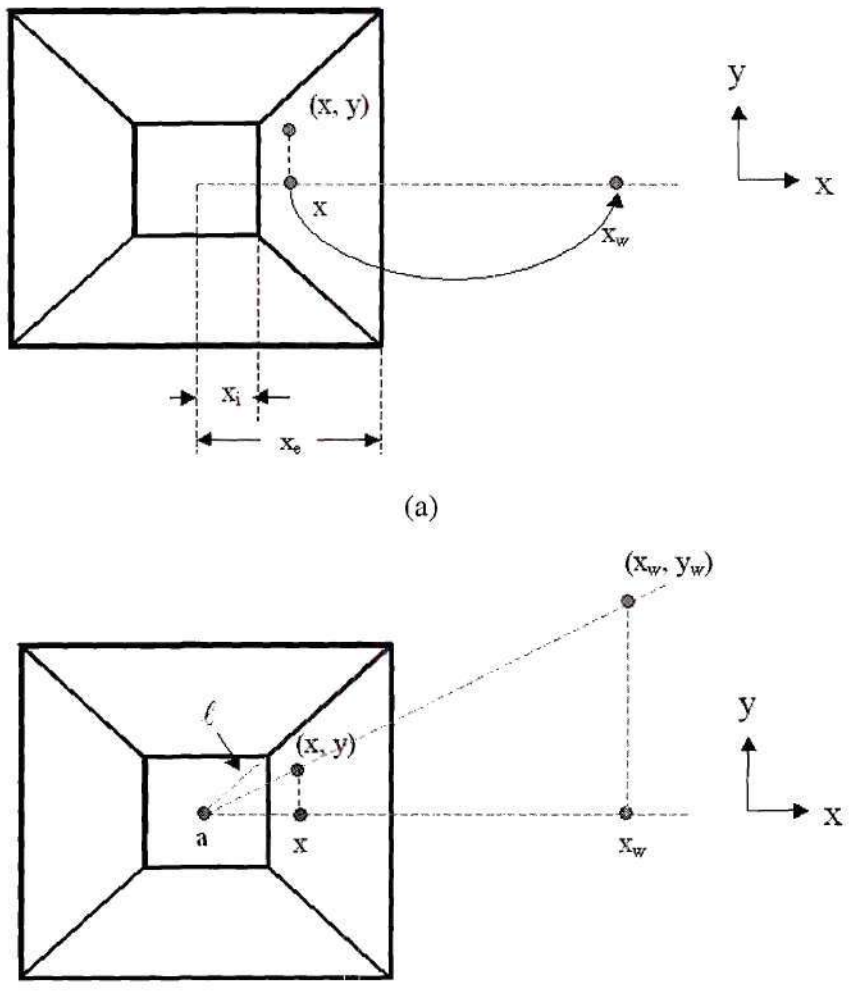


Figure 40: Finding the warped point. The x-warping function is applied to the x-coordinate of the point (x,y), as the point is located in the $+x$ region. This yields the coordinate x_w, shown in (a). In (b), the other coordinate y_w is found by solving the line equation using the coordinate x_w found in (a).

x-warping function as given above, and y- and z-warping functions,

$$y_w = y\frac{y_e - y_i}{y_e - |y|} \tag{13}$$

$$z_w = z\frac{z_e - z_i}{z_e - |z|}, \tag{14}$$

In general, the procedure to warp a point in the pre-warped exterior space is as follows.

1. Determine in which frustum-shaped region the point is located.

2. Apply the appropriate warping function to one of the coordinates. If the point is the in $\pm x$ region, apply the x-warping function, if the point is in the $\pm y$ region, apply the y-warping function, and if the point is the $\pm z$ region, apply the z-warping function.

3. Find the other two coordinates by solving line equations using the warped coordinate.

After reconstruction, we intend the model to be viewed from near or within the interior space. For such viewpoints, voxels will project to approximately the same footprint in each image.

5.2.2 Other warping functions

The frustum warp presented above is not the only possible warp. Any warp that does not move the outer boundary of the interior space, and warps the outer boundary of the pre-warped exterior space to infinity, while satisfying the criteria that no gaps form between voxels, and that no voxels overlap, is valid. Furthermore, it is desirable to choose a warping function that approximates the constant footprint property for the cameras used in the reconstruction as well as the camera placements during new view synthesis. An example of an alternative warping function is one that warps radially with distance from the center of the reconstruction volume.

5.3 Implementation issues

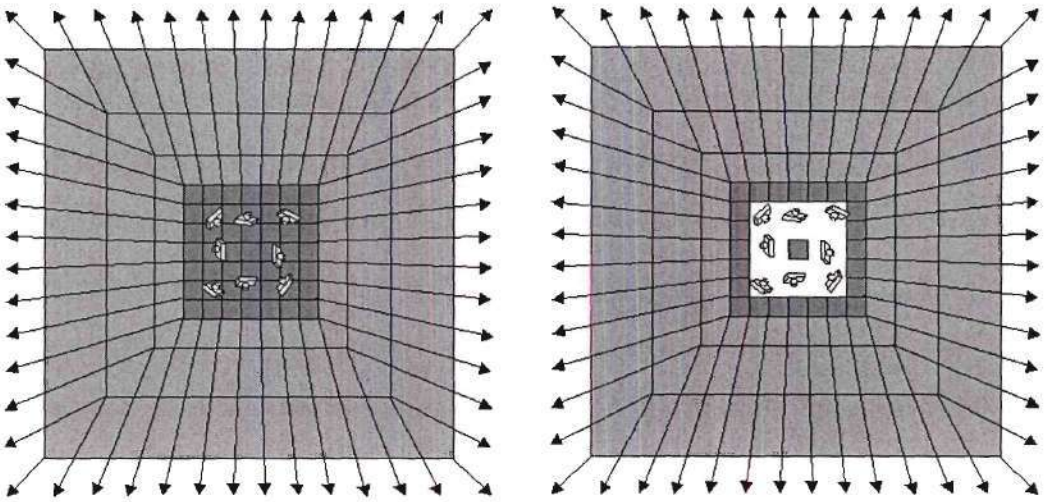
Reconstructing a scene using a warped reconstruction volume poses some new challenges, described in this section.

5.3.1 Cameras inside volume

Perhaps the most difficult challenge is that of having the cameras embedded inside the reconstruction volume. Typically, when one uses a standard space carving algorithm, the cameras used to take the photographs of the scene are placed outside of the reconstruction volume, so that at least two cameras have visibility of each voxel. The photo-consistency measure used in space carving algorithms, qualitatively, determines if all the cameras that can see a voxel agree on its color. This photo-consistency is poorly defined when a voxel is visible from only one camera.

Since the warped reconstruction volume can occupy all space, cameras get embedded inside the voxel space, as shown in (a) of Figure 41. Our reconstruction algorithm initially assumes that all voxels are opaque. Therefore, camera views are obscured, and the cameras cannot work together to carve the volume. This poses a problem, since to be properly defined, the photo-consistency measure requires that at least two cameras have visibility of a voxel. Consequently, the space carving algorithm cannot proceed, and terminates without removing any voxels from the volume.

To address this issue, we must remove (pre-carve) a section of the voxel space so that initially, each surface voxel is observed by at least two cameras, validating the photo-consistency measure, as shown in (b) of Figure 41. There are a variety of possible methods to achieve this result. A generic method is to have a user identify regions of the voxel space to pre-carve. Obviously, the pre-carved regions must consist only of empty space, i.e., not contain any scene surfaces to be reconstructed. While effective, this method precludes a fully automatic reconstruction. Alternatively, one can pre-carve the volume using a heuristic. For example, if appropriate, one could require that the cameras have visibility of the boundary between the interior space and the exterior space. Other heuristics are possible. Once the pre-carving is complete, we execute a standard space carving algorithm using the warped voxel space.



(a) (b)

Figure 41: Pre-carving operation. Reconstruction in the warped space causes the cameras to be embedded in the voxel space, as shown in (a). For many camera placements, it would be impossible to carve any voxels, since no voxel is visible to more than one camera. We execute a pre-carving step in (b) so that cameras can work together to carve the volume.

5.3.2 Preventing visible holes in the outer shell

Because of errors in camera calibration, image noise, inaccurate color threshold etc., space carving sometimes removes voxels that should remain in the volume. Thus, it is possible that voxels on the outer shell of the voxel space will be deemed inconsistent. Removing such voxels can result in unknown black regions similar to those in Figure 36 during new view synthesis, as no voxel would project onto the camera for some pixels in the image plane. Since one cannot see beyond infinity, we do not carve voxels on the outer shell of the voxel space, independent of the photo-consistency measure.

5.4 Results

We have modified the GVC algorithm to utilize the warped voxel space. We created a synthetic data set, called Marbles, consisting of twelve 320 x 240 images of five small texture-mapped spheres inside a much larger sphere textured with a rainbow-like image. We reconstructed the scene using a voxel space that consisted of 48 x 48 x 48 voxels, of

which the inner 32 x 32 x 32 were in the interior space and unwarped. The voxel space was set up so that the five small texture-mapped spheres were reconstructed in the interior space, while the larger sphere, making up the background, was reconstructed in the exterior warped space. Sample images from the data set are shown in (a) and (b) of Figure 42. A reconstruction was performed using the warped voxel space. The reconstruction was projected to the viewpoints of (a) and (b), yielding (c) and (d). Note that the background environment was reconstructed using our warped voxel space.

Next, we took a series of ten panoramic (360 degree field of view) photographs of a quadrangle at Stanford University, using a digital camera that produces cylindrical images. These photographs had resolution of about 2502 x 884 pixels. One photograph from the set is shown in Figure 43 (a). We have found that when reconstructing an environment, it is preferable to use large field of view images, as objects far from the cameras are visible in many photographs. This achieves a sufficient sampling of the scene with fewer photographs. A voxel space of resolution 300 x 300 x 200 voxels, of which the inner 200 x 200 x 100 were interior voxels, was pre-carved manually by removing part of the voxel space that containing the cameras. Then, the GVC algorithm was used to reconstruct the scene. Figure 43 (b) shows the reconstructed model reprojected to the same viewpoint as in (a). Note that objects far away from the cameras, such as many of the buildings and trees, have been reconstructed with reasonable accuracy for new view synthesis. New views are shown in (c) and (d) of the figure.

Despite the successes of this reconstruction, it is not perfect. The sky is very far away from the cameras (for practical purposes, at infinity), and should therefore be represented with voxels on the outer shell of the voxel space. However, since the sky is nearly textureless, cusping occurs, resulting in inaccurate computed geometry, apparent in an animated sequence of new views of the reconstruction. Reconstruction of outdoor scenes is challenging, as surfaces often do not satisfy the Lambertian assumption. To compensate, we used a higher consistency threshold, also resulting in some inaccurate geometry. On the whole,

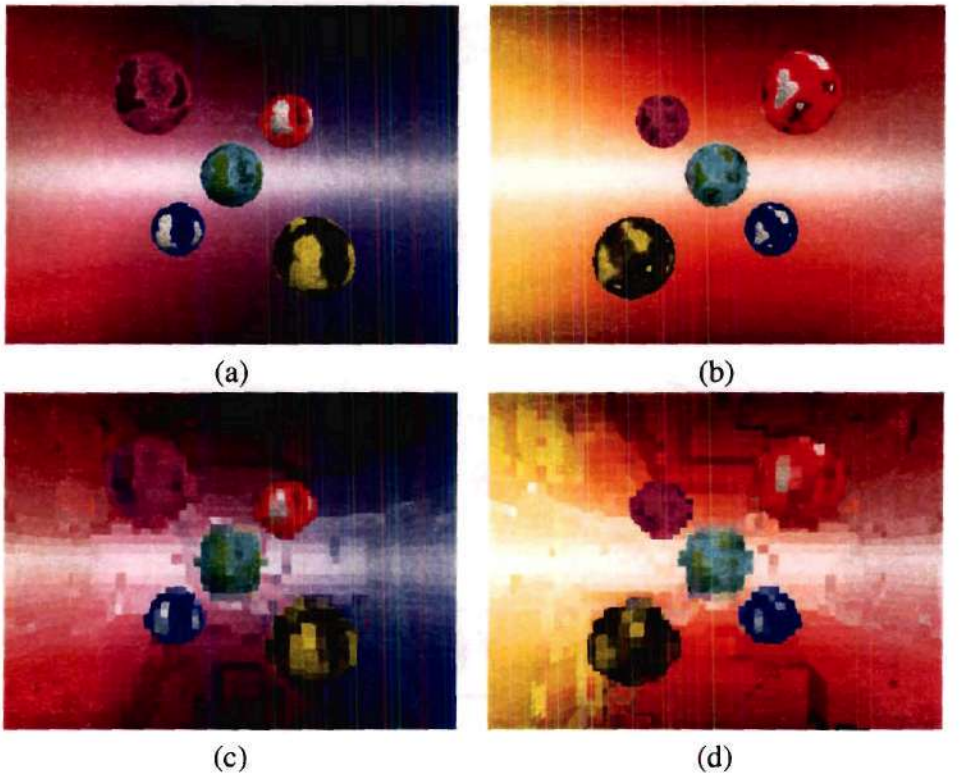


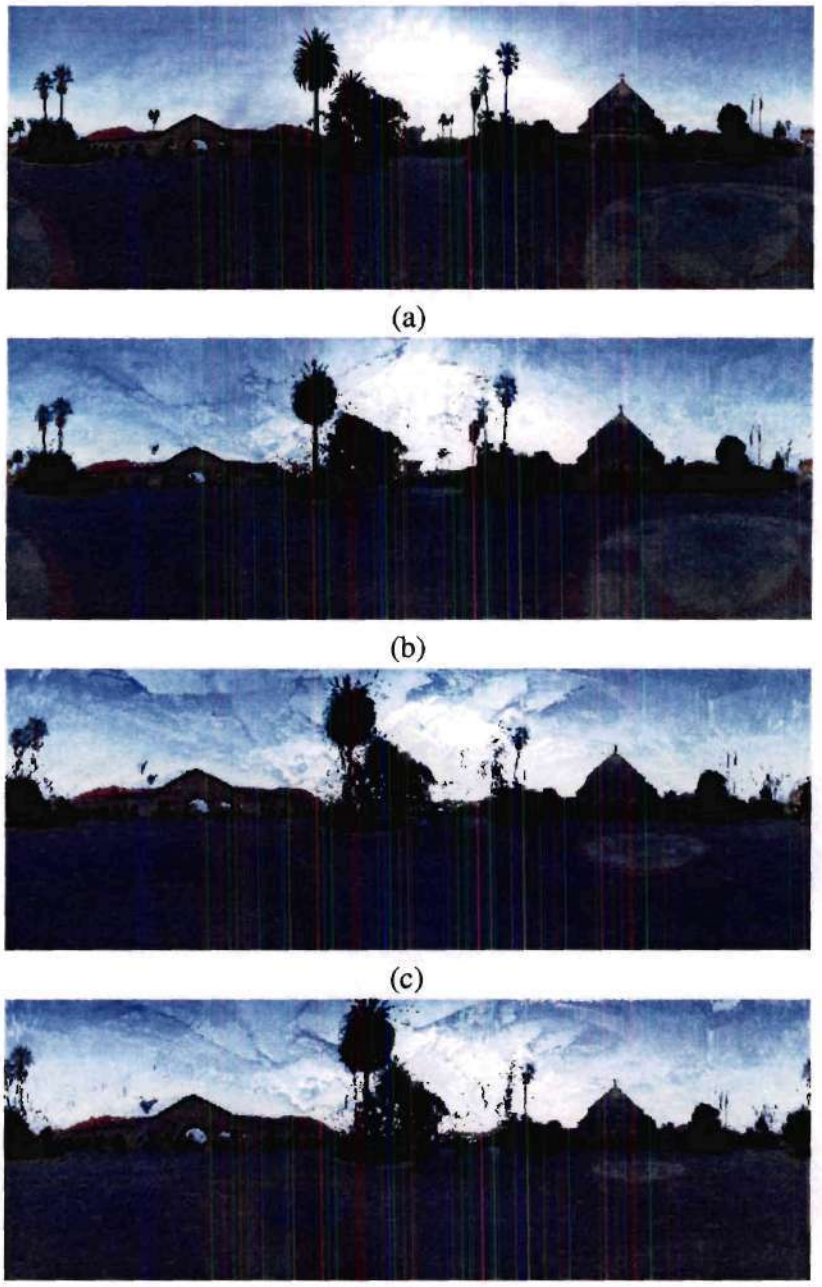
Figure 42: Original images of the Marbles data set are shown in (a) and (b), and a reconstruction projected to the same viewpoints of (a) and (b) is shown in (c) and (d), respectively.

though, the reconstruction is reasonably accurate and produces convincing new views.

5.5 Summary

In this chapter we introduced a volumetric warping approach for modeling infinitely large scenes with a finite number of voxels. We modified our GVC algorithm to reconstruct the scene using this warped voxel space, and demonstrated results for a synthetic scene as well as a large-scale real-world scene. Our approach is capable of reconstructing a background environment in addition to a foreground scene.

In this approach we intend to synthesize new views in or near the interior space. When the virtual camera moves into the exterior space, the frustum shape of the exterior voxels



(a)

(b)

(c)

(d)

Figure 43: Results for the Stanford scene. One of the ten panoramic photographs is shown in (a). The reconstructed model, projected to the same viewpoint as that of (a) is shown in (b). New synthesized panoramic views are shown in (c) and (d).

becomes apparent. Also, when synthesizing new views placed in the exterior space, exterior voxels near the virtual camera can be excessively large. Therefore, this approach only has a limited region of space for which new view synthesis is effective. To increase the range of effective viewpoints for new view synthesis, one could combine multiple interior / exterior spaces together. Such work remains an open problem. Another open problem is the development of a multi-resolution voxel space that is customized to the scene content as observed in the reference views. For example, large voxels could model far away surfaces or nearby homogeneously colored surfaces, while smaller voxels could model finer nearby surfaces that exhibit more color variation. Such an approach might employ on octree representation of 3D space.

As a practical concern, an implementation of our volumetric warping approach can have difficulties in representing a large dynamic range of depth values using finite-precision values on a computer. This becomes especially significant when performing z-buffering during rendering. The bit depth of the z-buffer should be sufficiently high to avoid rendering errors. Alternatively, one could rewarp the depth values so that they are constrained to a limited range.

In Chapter 7, we will present a view-dependent representation of 3D space that satisfies the constant footprint property at all virtual camera positions.

CHAPTER VI

SPACE CARVING USING LEVEL SET METHODS

While space carving approaches like GVC are effective, they can produce ragged and irregular surfaces, especially where there are high curvature features such as cusps and floaters described in Chapter 4. Rather than post-process the reconstructed geometry, one can require surface smoothness during the reconstruction. Doing so mitigates cusps and penalizes floaters.

In this chapter we present a multi-resolution space carving approach that uses level set methods [113]. Unlike most standard space carving techniques, this approach produces a smooth reconstruction composed of manifold surfaces. The method outputs a polygonal mesh, which is more suitable for rendering on standard graphics hardware than a collection of voxels. We texture-map the reconstructed polygonal mesh using the photographs, and then render it to produce photo-realistic new views of the scene.

We begin this chapter with a brief description of surface evolution using level set methods, and describe our reconstruction algorithm.

6.1 Surface representation

Surface evolution techniques characterize the motion of a surface changing its shape over space and time. In this class of methods, two different surface representations are commonly used: *explicit* and *implicit*. Explicit surface representations directly specify the location of points on the 3D surface, often as a polygonal mesh or as a parametric surface.

The points on an implicit surface, in contrast, are implied by the evaluation of a function ψ. The surface exists at a point (x,y,z) if $\psi(x,y,z) = c$, where c is a constant. Implicit surfaces are very useful for describing manifolds, as $\psi(x,y,z)$ is negative for points inside

the surface, and positive for points outside. Additionally, useful surface properties can be computed directly from the implicit surface representation. For example, the general technique for finding the unit surface normal **N** to an implicit surface is to take partial derivatives of $\psi(x,y,z)$ as follows [10]:

$$\mathbf{N} = \frac{\nabla\psi}{|\nabla\psi|}, \tag{15}$$

where

$$\nabla\psi = \left(\frac{\partial\psi}{\partial x}, \frac{\partial\psi}{\partial y}, \frac{\partial\psi}{\partial z}\right). \tag{16}$$

6.2 *Level set methods*

Level set theory was developed by Osher and Sethian [83, 102] to model the evolution of propagating interfaces. Early work was applied to computational fluid dynamics and crystal growth; however, in recent years there has been much interest in applying the theory to problems in computer vision, graphics, and image processing.

6.2.1 Description

Level set methods represent an $N-1$ dimensional surface S implicitly as the zero-level set of an N-dimensional function ψ. This implicit representation has a number of advantages over explicit surface representations, including natural support of topological changes, concise descriptions of differential structure, and no need for reparameterization.

The surface moves with a spatially and temporally variable speed F along its surface normal, subject to intrinsic, independent, and data-driven forces, as shown in Figure 44. Intrinsic forces are determined by local properties of the surface such as curvature. Independent forces are independent of the surface, such as a underlying velocity that passively transports the surface in a direction. Data-driven terms are used to guide the evolving surface to a desired shape that solves a particular problem.

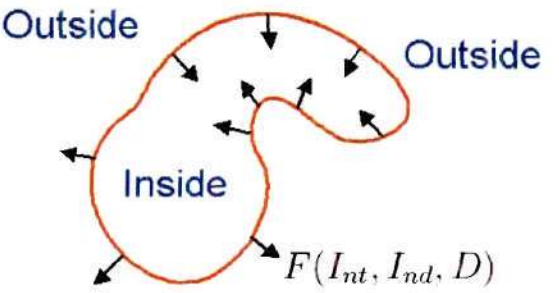


Figure 44: Surface evolution along surface normal, with a speed F composed of intrinsic, independent, and data-driven terms.

6.2.2 Evolution of the surface

In this section we derive the equation of motion that describes evolution of the surface S embedded as the zero-level set of ψ. Since we will be moving the surface temporally as well as spatially, we include an additional temporal variable t, giving $\psi(x,y,z,t)$.

Let $\mathbf{x} = (x,y,z)$ be a point on the surface. The velocity of a point on the surface is the temporal derivative of $\mathbf{x}$,

$$\frac{\partial \mathbf{x}}{\partial t} = \left(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t}, \frac{\partial z}{\partial t}\right). \tag{17}$$

Since the evolving surface moves along its unit normal $\mathbf{N}$ with a speed F,

$$\frac{\partial \mathbf{x}}{\partial t} \cdot \mathbf{N} = F. \tag{18}$$

Using Equation 15, we can rewrite this expression as

$$\frac{\partial \mathbf{x}}{\partial t} \cdot \nabla \psi = F |\nabla \psi|. \tag{19}$$

Now consider the function ψ. For points on the zero-level set,

$$\psi(x,y,z,t) = 0. \tag{20}$$

Differentiation of Equation 20 with respect to time gives

$$\frac{\partial \psi}{\partial t} + \frac{\partial \psi}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial \psi}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial \psi}{\partial z}\frac{\partial z}{\partial t} = 0. \tag{21}$$

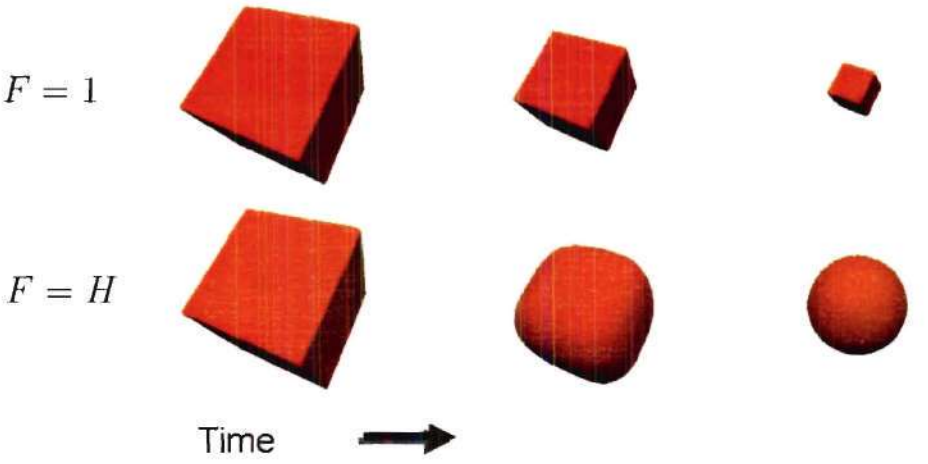


Figure 45: Two common flows: normal flow (top) and curvature flow (bottom).

This expression can be rewritten as

$$\frac{\partial \psi}{\partial t} + \nabla \psi \cdot \frac{\partial \mathbf{x}}{\partial t} = 0. \tag{22}$$

Substituting Equation 19 into Equation 22 yields the level set evolution equation,

$$\frac{\partial \psi}{\partial t} + F\,|\nabla \psi| = 0. \tag{23}$$

Equation 23 describes the time evolution of the level set function ψ so that the zero-level set of this evolving function is always identified with the propagating surface.

6.2.3 Two common flows

Two common flows are the *normal flow* and *curvature flow*. The normal flow is a surface evolution that moves along the surface normal with a constant speed. The surface erodes when evolved along the inward surface normal as shown in the top of Figure 45. Note that the normal flow preserves corners and edges. Curvature flow is a surface evolution with a speed based on H, a measure of curvature. Places where there is high curvature move more quickly, resulting in smoothing over time, shown in the bottom of Figure 45.

6.3 Reconstruction algorithm

Given Equation 23, all that is left to do is to define an initial surface, and a speed function F that drives the evolving surface to the true scene geometry while enforcing smoothness.

We embed the initial surface as the zero-level set of a volumetrically sampled function ψ. The exact shape of the initial surface is not very important, as long as it contains the surfaces being reconstructed. This requirement is necessary since we will shrink the surface during reconstruction. A simple initial surface to use is the surface of the reconstruction volume. This is the same initial surface we used for our GVC reconstructions in Chapter 3.

The speed function F we use is

$$F = \alpha_0 \phi + \alpha_1 \phi H, \tag{24}$$

resulting in a surface evolution of

$$\frac{\partial S}{\partial t} = F\mathbf{N} = \alpha_0 \phi \mathbf{N} + \alpha_1 \phi H \mathbf{N}, \tag{25}$$

where ϕ is a measure of color mismatch related to photo-consistency and H is the mean curvature. This evolution is comprised of two terms: a weighted normal flow and a weighted curvature flow. The constants α_0 and α_1 are used to weight the two flows relative to each other. The weighted curvature keeps the reconstruction smooth, while the weighted normal flow allows the evolving surface to better fit the scene being reconstructed.

The weighting factor ϕ is data dependent, and proportional to the standard deviation used in the photo-consistency measure described in Section 3.1.3. We scale ϕ so that it is within the range [0, 1], using the equation

$$\phi = \begin{cases} 1, & \text{if voxel is inconsistent} \\ \min(\frac{\sigma}{\sigma_0}, 1), & \text{if voxel is consistent} \end{cases} \tag{26}$$

where σ_0 is user-specified parameter. Thus, ϕ is large when a voxel modeling the surface is inconsistent. This will allow the surface to readily propagate along its inwardly pointing normal through regions of space that do not contain object surfaces being reconstructed.

The ϕ term becomes small when a voxel modeling the surface is photo-consistent. Here, the surface propagation slows. At places where the surface is fully photo-consistent, $\phi = 0$ and the surface evolution stops altogether.

The value of σ_0 affects both the amount of smoothing as well as the convergence of the algorithm. A lower value of σ_0 will allow the evolving surface to more readily propagate through photo-consistent space. This results in more smoothing of the model when $\alpha_1 > 0$. However, a value that is too low will allow the evolving surface to pass through the photo-consistent points where one would like the surface to stop. In such a case, the evolution may fail to converge. A larger value of σ_0 slows the evolving surface down when it is in photo-consistent space. In the limit as σ_0 goes to infinity, Equation 26 becomes

$$\phi = \begin{cases} 1, & \text{if voxel is inconsistent} \\ 0, & \text{if voxel is consistent} \end{cases} \tag{27}$$

This is the approach taken by space carving algorithms, which do not carve photo-consistent voxels.

In the level set framework, the surface evolution of Equation 25 becomes

$$\frac{\partial \psi}{\partial t} = (\alpha_0 + \alpha_1 H)\,\phi\,|\nabla \psi|, \tag{28}$$

with

$$H = \nabla \cdot \mathbf{N} = \frac{(\psi_{yy} + \psi_{zz})\psi_x^2 + (\psi_{xx} + \psi_{zz})\psi_y^2 + (\psi_{xx} + \psi_{yy})\psi_z^2 - 2\psi_x\psi_y\psi_{xy} - 2\psi_x\psi_z\psi_{xz} - 2\psi_y\psi_z\psi_{yz}}{(\psi_x^2 + \psi_y^2 + \psi_z^2)^{3/2}}. \tag{29}$$

During evolution, we keep track of which voxels are on the surface. We iterate subject to Equation 28 until the number of surface voxels remains the same for X, where X is a small number. This indicates the surface has slowed down to a point where it the evolution is near or at convergence.

6.4 Implementation

In this section, we discuss the implementation of Equation 28.

6.4.1 Voxel space

We sample ψ on a discrete grid using two 3D voxel arrays that store ψ as a function of (x,y,z) for the current time t and the next time, $t+\Delta t$. This forms a 3D voxel space in which the reconstruction occurs.

6.4.2 Derivatives

We use finite difference approximations for the spatial and temporal partial derivatives of ψ. In our implementation of Equation 23, we employ the first forward difference for the temporal derivative of ψ,

$$\frac{\partial \psi}{\partial t} = \frac{\psi(x,y,z,t+\Delta t) - \psi(x,y,z,t)}{\Delta t}. \tag{30}$$

With this approximation, Equation 23 becomes

$$\psi(x,y,z,t+\Delta t) = \psi(x,y,z,t) - \Delta t F \left|\nabla \psi\right|. \tag{31}$$

We approximate the spatial derivatives with central differences. For the first-order spatial derivative in the x-dimension we use

$$\psi_x = \frac{\partial \psi}{\partial x} = \frac{\psi(x+\Delta x,y,z,t) - \psi(x-\Delta x,y,z,t)}{2\Delta x}. \tag{32}$$

The second-order derivative in the x-dimension is approximated as

$$\psi_{xx} = \frac{\partial^2 \psi}{\partial x^2} = \frac{\psi(x+\Delta x,y,z,t) - 2\psi(x,y,z,t) + \psi(x-\Delta x,y,z,t)}{(\Delta x)^2}. \tag{33}$$

Spatially, we use step size Δx of one voxel. Similar equations exist for the y and z dimensions. The temporal time step Δt is discussed below.

Sethian [102] has shown that normal flow surface evolution can develop singularities known as shocks. Shocks are sharp corners or edges that occur when the evolving surface flows in on itself. To properly model this phenomenon, Sethian proposes the use of weak solutions given by entropy conditions. These solutions require that one take the appropriate forward or backward approximation to the spatial derivatives in the implementation

of Equation 23. However, Sethian also shows that shocks do not form for a combined normal and curvature flow. Thus, as long as $\alpha_1 > 0$, we can safely use central difference approximations for the spatial derivatives.

6.4.3 Stability

In order for the numerical implementation of the surface evolution to stay stable, one must satisfy the Courant-Friedrichs-Lewy (CFL) condition, which gives an upper bound on the time step used in the evolution. For $\alpha_0 + \alpha_1 = 1$, one can derive the CFL condition using Von Neumann analysis [89], which, for the surface evolution of Equation 28, yields a maximum allowable time step of

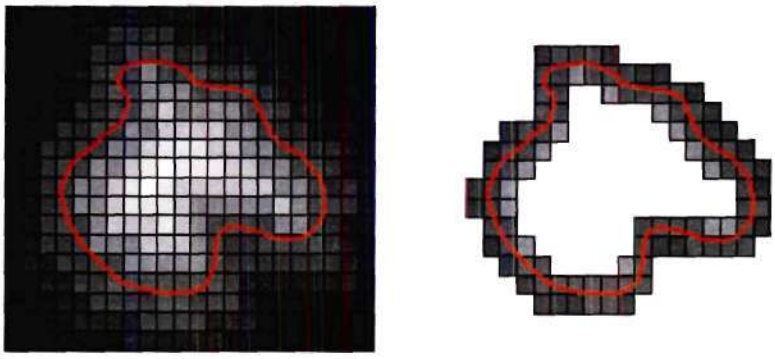
$$\Delta t = \frac{1}{6}. \tag{34}$$

6.4.4 Narrow band

Updating ψ in Equation 28 over the entire voxel space requires $O(N^3)$ operations, where N is the number of voxels in the voxel space along one dimension. However, in our approach, we are interested only in the evolving surface. Therefore, it is possible to update ψ only in a narrow band [102] around the zero-level set, as shown in Figure 46 for a 2D slice of the 3D volume. The narrow band approach lowers the computational cost to $O(kN^2)$, where k is the number of band voxels. The orange curve in the figure indicates the surface embedded in the voxel space. It is interpolated from the voxel array, using the marching cubes [62] algorithm.

6.4.5 Multi-resolution

For efficiency, our algorithm works in a coarse-to-fine fashion similar to the multi-resolution GVC approach discussed in Section 3.5.5. We first perform a reconstruction at resolution R using large voxels. At a lower resolution, we are able to carve away a large part of space that would require a lot of computation at a higher resolution. Once the reconstruction at resolution R is complete, we dilate the surface. After the dilation, we tessellate each voxel



(a) (b)

Figure 46: Updating ψ in the complete array (a) vs. narrow band (b), shown for a 2D slice of the 3D volume.

into eight sub-voxels, which increases the resolution to $R+1$. We perform trilinear interpolation to compute each value of ψ in the higher resolution volume. We then re-execute the algorithm at the higher resolution. This process continues until a desired resolution is obtained, as described in Figure 22

6.5 Results

We performed a multi-resolution reconstruction using the Broccoli data set. We started with a box-shaped initial surface in a coarse 21 x 17 x 23 volume. We executed the algorithm for three resolution increases, resulting in a 168 x 136 x 184 volume. Figure 47 shows the initial evolution of the zero level set from the initial surface. Once the geometry is reconstructed, we extract a polygonal representation of the surface by executing the marching cubes [62] algorithm. Next, we texture-map the polygonal model using the photographs of the scene. For this, we consider the rays between the triangle center and each reference view that has visibility of the triangle. We then compute the angle θ between each ray and the triangle surface normal. We apply pixels onto the triangle from only the reference view that has the smallest θ. New photo-realistic views of the reconstructed broccoli stalk are shown in Figure 48.

Figure 47: Initial evolution of the zero level set at the lowest resolution.

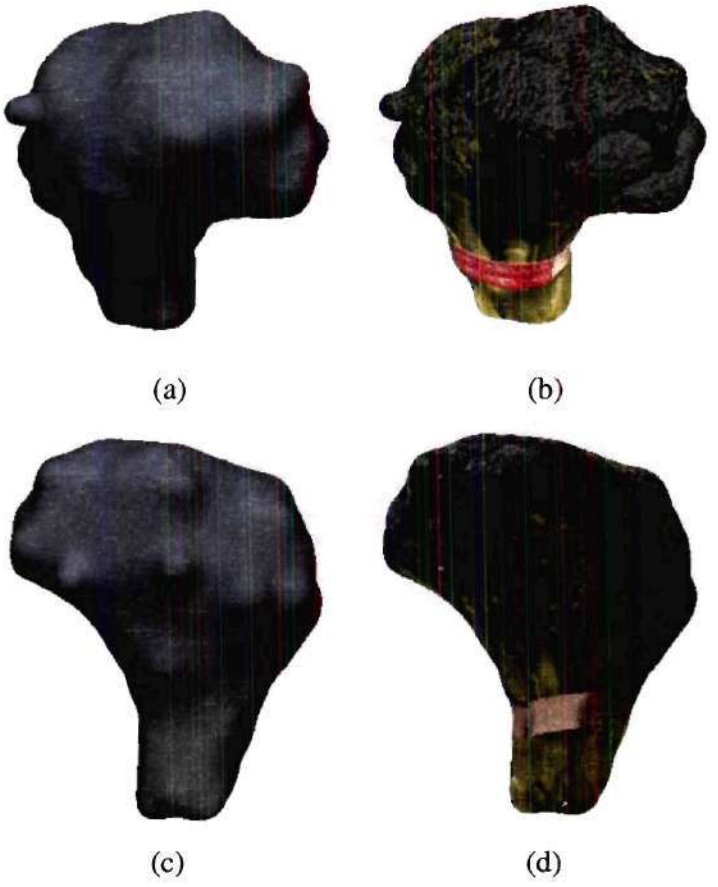


Figure 48: New synthesized views of the broccoli stalk. Untextured surfaces are shown in (a) and (c), textured surfaces are shown in (b) and (d).

We also performed a multi-resolution reconstruction of our Camel data set. This data set, courtesy of Gabe Brostow, consists of eleven 720 x 480 photographs of a toy camel. Four photographs from this data set appear in Figure 49. Our initial surface was again a box surrounding the scene. The resolution was 35 x 35 x 45. We executed the algorithm for two resolution increases, ending with a volume that had resolution 140 x 140 x 180. Some of the photographs contained specular highlights from the scene lighting reflecting from the object surface. Correspondingly, we had to increase the photo-consistency threshold during reconstruction, which resulted in a fatter surface. Also, the specular highlights result in some speckles that appear as texturing artifacts in new views rendered from the reconstruction, as shown in Figure 50.

Finally, we show results from reconstructing the Tower data set. Our initial surface was again a box, this time embedded into a 35 x 35 x 46 volume. We executed the algorithm for one resolution increase, resulting in a 70 x 70 x 92 volume. New views synthesized by rendering the polygonal surface are shown in Figure 51. There are some high curvature locations on the true surface, such as in between the cushions of the object, that are not properly reconstructed by this approach. The smoothed surface does not accurately model the scene geometry in such locations, which results in reconstruction artifacts, apparent in the synthetic new views in the figure. However, the overall geometry of the scene has been reconstructed with reasonable accuracy.

In Chapter 8 we will compare the level set reconstruction algorithm to other reconstruction approaches presented in this thesis.

6.6 Summary

We have presented a multi-resolution space carving algorithm implemented with level set methods. Using a set of photographs taken with calibrated cameras, this approach generates a smooth, texture-mapped 3D polygonal model that can be rendered using standard graphics hardware to produce new views of the scene.

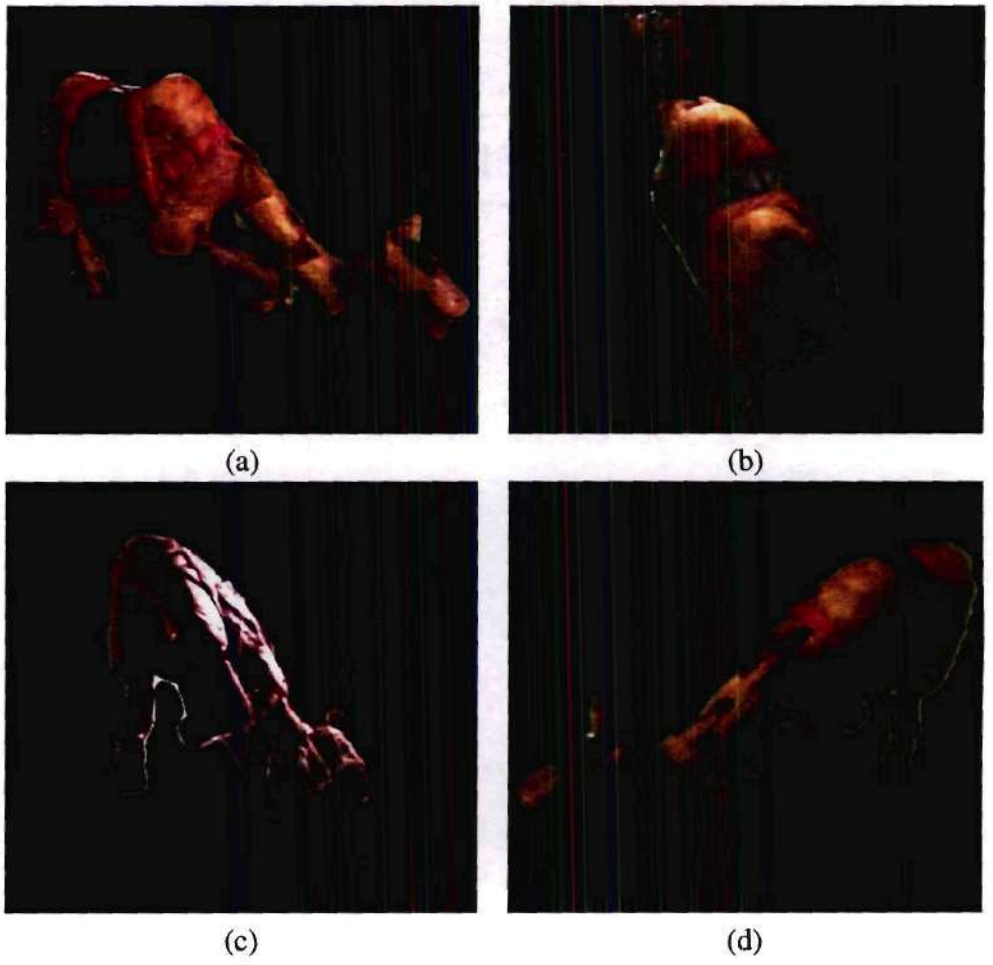


Figure 49: Four of eleven reference views of the Camel data set.

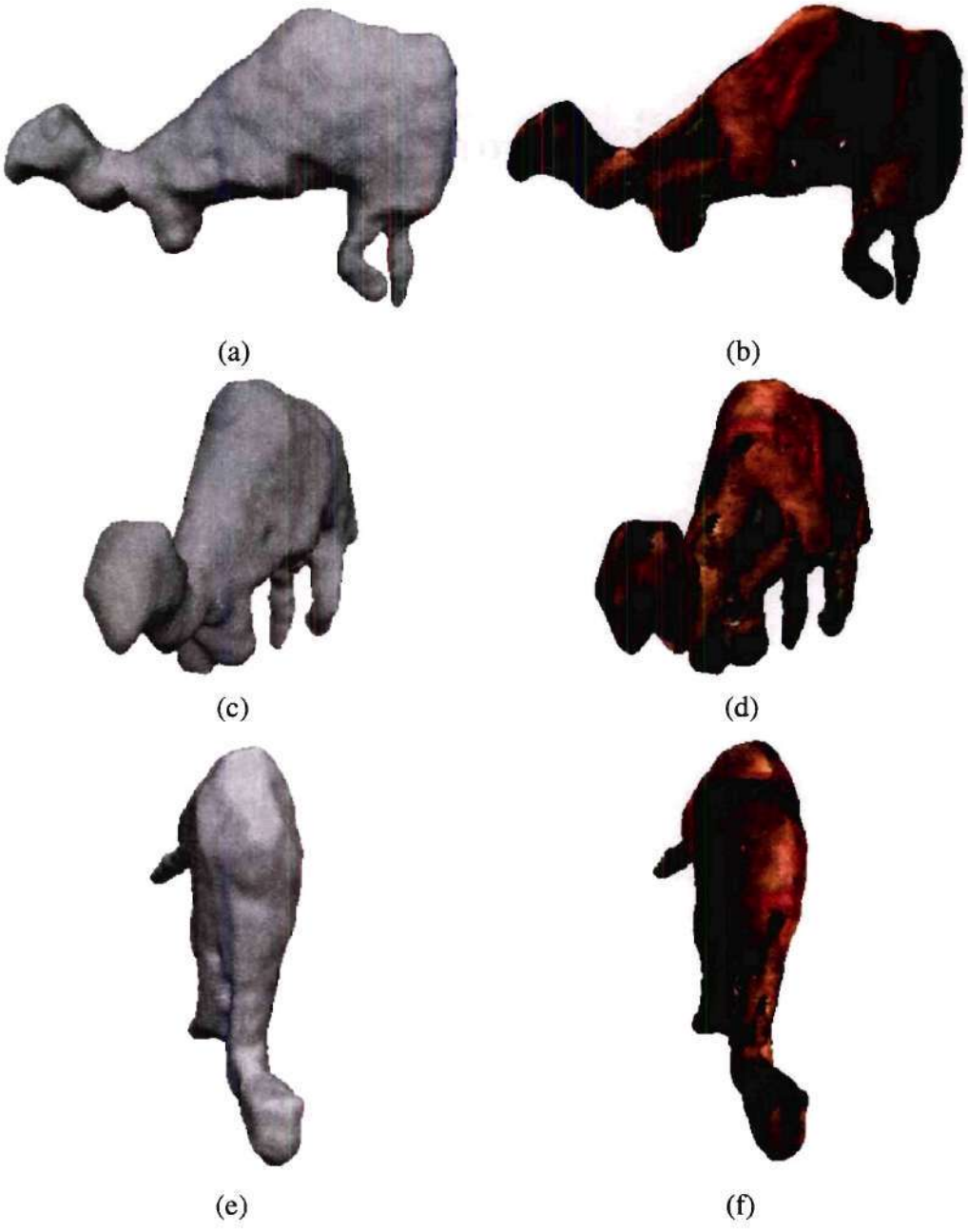


Figure 50: New synthetic views of the Camel data set. Untextured surfaces are shown in (a), (c), and (e), while textured surfaces are shown in (b), (d), and (f).

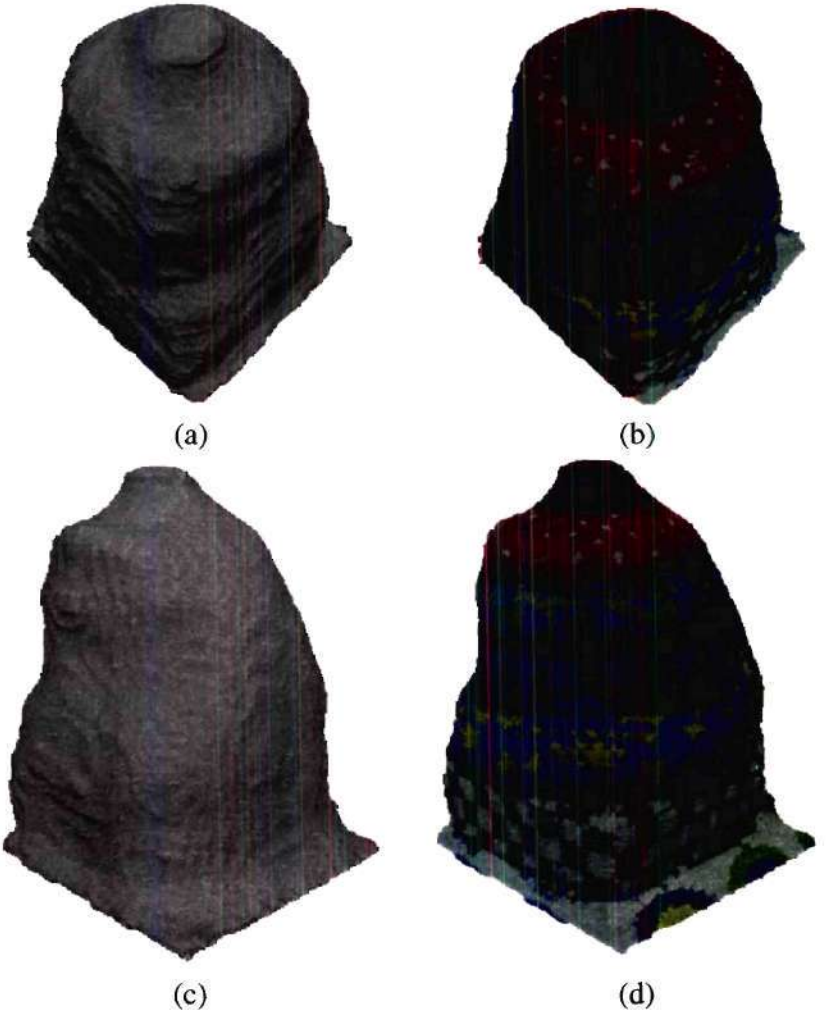


Figure 51: New synthetic views of the Tower data set. Untextured surfaces are shown in (a) and (c), while textured surfaces are shown in (b) and (d).

Our level set approach gives a structured way to incorporate smoothing during reconstruction. The reconstructed surface is smooth and regular, and does not contain extraneous floating geometry, holes, or large cusps that can plague standard space carving approaches. However, due to the smoothing term, however, some sharp edges, corners, and fine geometrical details are not preserved. For example, the broccoli model has some thin small leaves near the stalk that are not present in the final reconstruction.

Our approach does not necessarily find the most photo-consistent surface. A possible extension to this work would be to design a flow that stops at maxima of the photo-consistency function, similar in spirit to the volumetric optimization technique discussed in Section 4.3. Such an optimization-based level set approach might require a bi-directional flow. Also, future work might investigate ways to automatically adapt α_0 and α_1 during reconstruction. In places where fitting is desired, the normal flow could dominate the surface evolution. However, in places where smoothing is desired, the curvature flow term could dominate.

CHAPTER VII

IMAGE-BASED PHOTO HULLS

Depending on the number of photographs and resolution of the voxel space, the time required to reconstruct a scene using a space carving algorithm like GVC on modern computer hardware typically ranges from a few minutes to several hours. While parallelization and hardware acceleration of these algorithms is possible, space carving algorithms have not yet been demonstrated to offer interactive performance for non-trivial resolutions.

In fact, most standard approaches to the 3D scene reconstruction problem such as multi-baseline stereo, structure from motion, and shape from shading were not designed for real-time performance and thus are too slow to process the images online. When working with multi-view video data, most techniques perform the 3D reconstruction offline after the images have been acquired. Once the reconstruction is complete, it is rendered in real-time.

In this chapter we present a new algorithm called *image-based photo hulls* (IBPH) [111, 112] that reconstructs and synthesizes views of a scene's photo hull at interactive rates. Online 3D reconstruction and new view synthesis significantly broadens the applicability of this class of techniques; we demonstrate the use of our algorithm in a 3D video-conferencing application.

7.1 Image-based visual hulls

7.1.1 Overview

Our IBPH algorithm builds upon the *image-based visual hulls* (IBVH) algorithm [72]. The IBVH algorithm is a very efficient approach to computing views of the visual hull. The key to this algorithm's efficiency is its use of epipolar geometry for computing the geometry

and visibility of the scene. By taking advantage of epipolar relationships, all of the steps of the algorithm function in the image space of the reference views. Note that epipolar geometry is reviewed in Appendix A.

While the IBVH algorithm is exceptionally efficient, the geometry it reconstructs is not very accurate. This is because the IBVH algorithm reconstructs only the visual hull of the scene. As described in Chapter 2, the visual hull is a conservative volume reconstructed from silhouettes. When photographed by only a few cameras, the scene's visual hull is much larger than the true scene. Even if photographed by an infinite number of cameras, many scenes with concavities will not be modeled correctly by a visual hull. One can partially compensate for such geometric inaccuracies by view-dependent texture-mapping (VDTM), as done in the IBVH approach [72].

However, artifacts resulting from the inaccurate geometry are still apparent in new synthesized views, as shown in Figure 52. This figure demonstrates a reconstruction of a pinwheel photographed from five viewpoints. Two of the reference views, after background subtraction, are shown in the top row of the figure. A new view of the scene, placed halfway between two reference views, is rendered from the reconstruction. The middle row shows the visual hull reconstruction. At this viewpoint, the right side of the reconstructed pinwheel is texture-mapped with one reference image, while the left side of the pinwheel is texture-mapped with another. Because of the geometric inaccuracy of the visual hull, there is a salient seam along the pinwheel where there is a transition between the two images used to texture-map the surface. In particular, the center of the pinwheel is not present in the synthetic view. The improved geometry of the photo hull corrects this problem, as shown in the bottom row of the figure.

In this chapter we adapt the IBVH algorithm to reconstruct views of the photo hull, by utilizing the color information of the images to identify scene geometry. These additional color constraints result in more accurately reconstructed geometry, which often projects to better synthesized virtual views of the scene. Our approach combines the efficiency of the

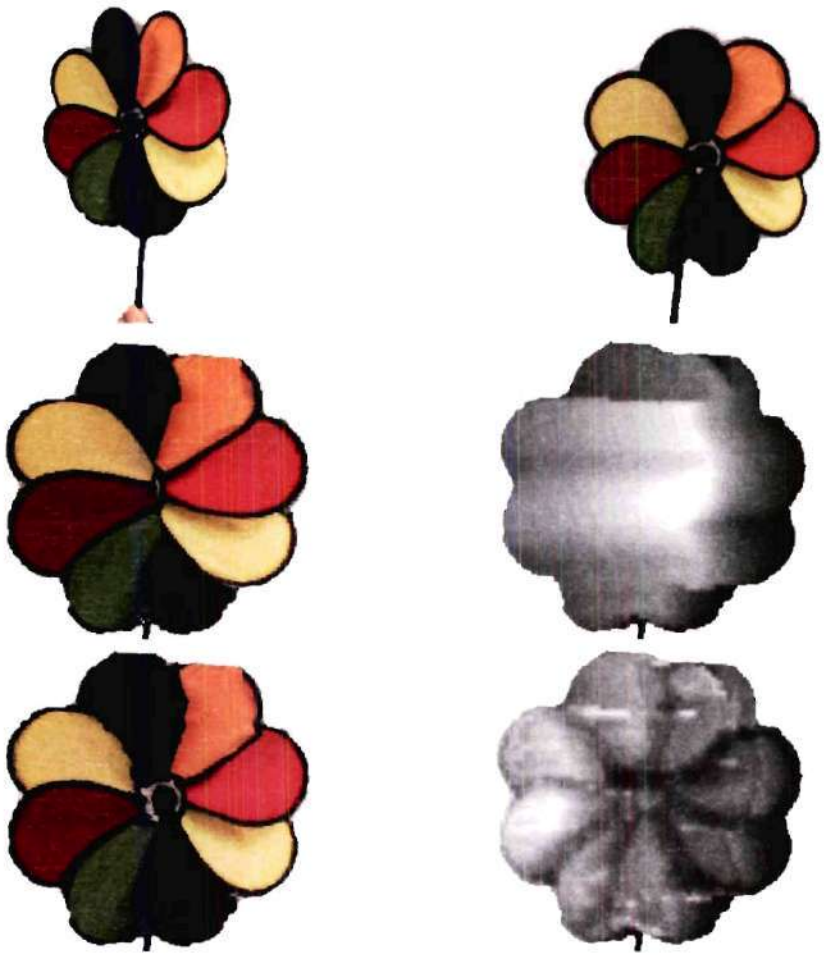


Figure 52: We show two reference views (top row) at one time instant of our Pinwheel data set, after background subtraction. The visual hull reconstruction (middle row) vs. photo hull reconstruction (bottom row) are shown for a synthetic view, along with corresponding depth maps.

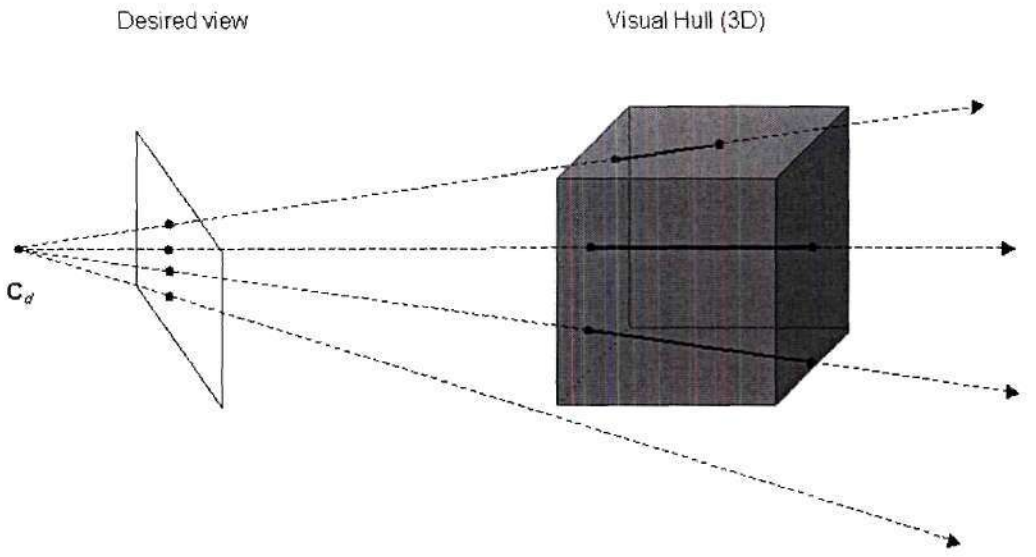


Figure 53: View-dependent geometry.

IBVH algorithm with the improved geometric accuracy of the photo hull.

7.1.2 Computing geometry

One of the unique properties of the IBVH algorithm is that it reconstructs view-dependent geometry. A user moves a virtual camera about the scene. For each virtual camera placement, the IBVH algorithm computes the extent that back-projected rays from the center of projection $\mathbf{C}_d$ intersect the visual hull in 3D space, as shown in Figure 53. These intervals are stored as a layered depth image [103]. Thus, the representation of the geometry is specified for the desired view, and changes as the user moves the virtual camera.

Consider an individual ray, as shown in Figure 54. The ray is back-projected from the desired view's center of projection, through a pixel in the image plane, and into 3D space. This ray projects to an epipolar line in each reference view. The IBVH algorithm determines the 2D intervals where the epipolar line crosses the silhouette. These 2D intervals are then "lifted" back onto the 3D ray using a simple projective transformation. The intervals along the 3D ray from all reference views are intersected. The resultant set of intervals describe where the ray pierces the visual hull. These are called *hull intervals* in this thesis.

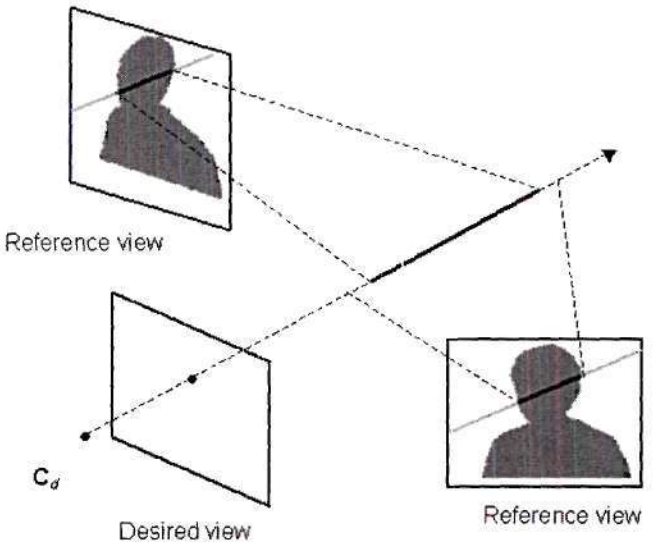


Figure 54: Determining a ray's visual hull intervals.

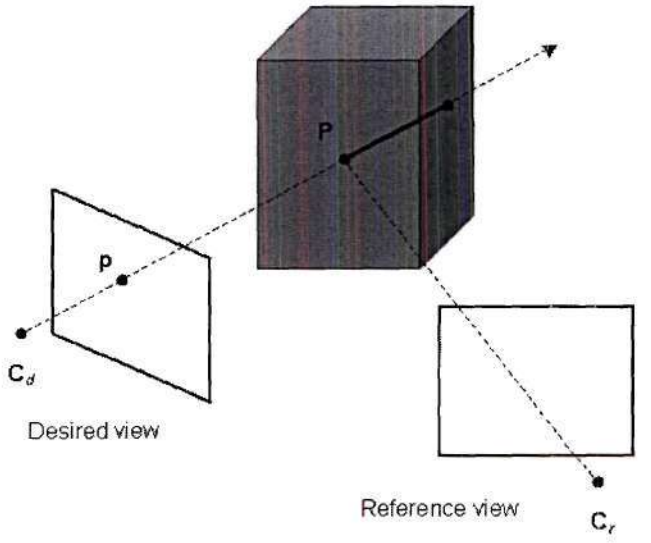
In Figure 54, one hull interval is found along the back-projected ray. Once this procedure has been performed on all rays back-projected from the desired view, the reconstruction of the view-dependent geometry of the visual hull is complete.

7.1.3 Computing visibility

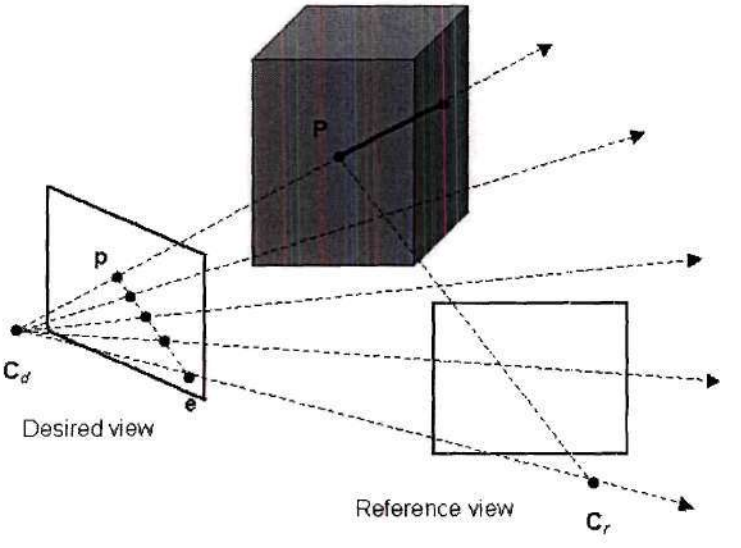
In order to color a point on the visual hull, it is necessary to determine which cameras have an unoccluded view of the point. Thus, visibility must be computed before texture-mapping the reconstructed geometry.

At a pixel **p** in the desired view, the first point (if any) along the first visual hull interval indicates a point **P** in 3D space that projects to **p** and is visible in the desired view, as shown in Figure 55 (a). To compute visibility, for each reference view we need to determine if **P** is visible. **P** must be visible in the reference view if the line segment $\overline{\mathbf{PC}_r}$ between **P** and the reference view's center of projection $\mathbf{C}_r$ does not intersect any visual hull geometry.

The layered depth image representation of the visual hull makes this easy to determine. In the desired view, $\overline{\mathbf{PC}_r}$ projects to an epipolar line segment $\overline{\mathbf{pe}}$, where **e** is the epipole, found by projecting $\mathbf{C}_r$ into the desired view, as shown in Figure 55 (b). For each pixel



(a)



(b)

Figure 55: **P** is visible in the reference view if there is no occluding geometry along $\overline{\mathbf{PC}_r}$.

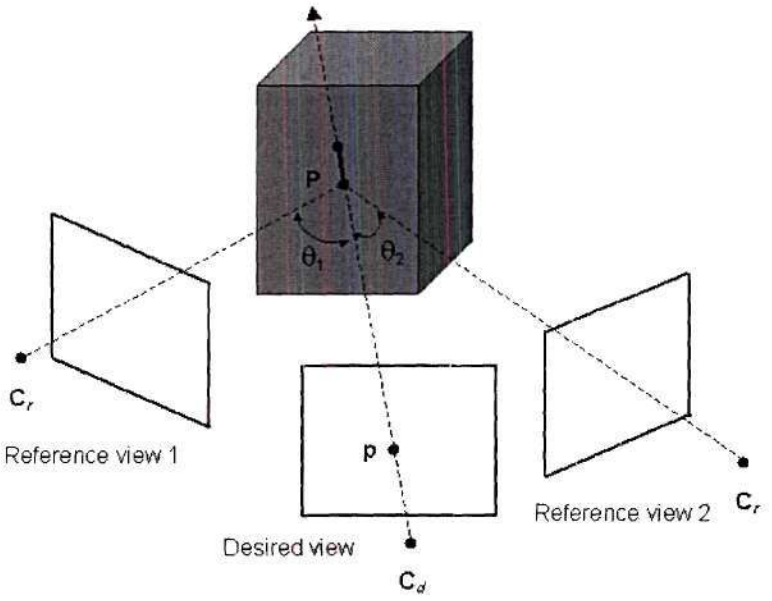


Figure 56: View-dependent texture-mapping.

along $\overline{\mathbf{pe}}$, the visual hull intervals can be checked to see if they contain geometry that intersects $\overline{\mathbf{PC}_r}$. If an intersection occurs, point **P** is not visible in the reference view, and no more pixels along $\overline{\mathbf{pe}}$ need be evaluated. Otherwise, one continues evaluating pixels along $\overline{\mathbf{pe}}$, until there are no more pixels to evaluate. If no visual hull interval has intersected $\overline{\mathbf{PC}_r}$, then the point **P** is visible in the reference view.

The IBVH paper [72] discusses discretization issues in computing visibility using this approach, as well as occlusion-compatible orderings to improve its efficiency.

7.1.4 View-dependent texture mapping

Once visibility has been computed, one can color the visual hull using the reference views. The IBVH paper employs view-dependent texture mapping, which retains view-dependent effects present in the photos, and works well with the inaccurate geometry of the visual hull. To color a point **p** in the desired view, the closest point **P** on the hull is found. Then, for each reference view that has visibility of **P**, the angle between $\overline{\mathbf{PC}_d}$ and $\overline{\mathbf{PC}_r}$ is found, as shown in Figure 56. The reference view with the smallest angle is chosen to color the

```
compute IBVH
compute visibility
pre-compute homogeneous ray steps HΔP in each reference image
do
   evaluate photo-consistency
   for each inconsistent ray in desired view
      if (number of steps along ray k<=k̄+K)
         step along inconsistent ray
      else
         set ray consistent
   if (updating visibility)
      update visibility
} while(number of inconsistent rays > M)
display hull using VDTM
```

Figure 57: Pseudo-code for the IBPH algorithm. See text for details.

visual hull. This is the reference view that has the "best" view of **P** for the virtual camera's location. For example, in Figure 56, reference view 2 would be chosen since $\theta_2 < \theta_1$.

7.2 *Image-based photo hulls*

As discussed in Section 2.5.1, the visual hull is a conservative volume that is guaranteed to enclose the surfaces being reconstructed. The image-based visual hulls algorithm therefore is able to eliminate a large part of space that does not contain any scene surfaces. And because the IBVH algorithm is very efficient, it does this quickly. Our image-based photo hulls algorithm refines geometry of the image-based visual hull using photo-consistency to yield improved results. Below we describe the approach.

7.2.1 Approach

As indicated in the pseudo-code in Figure 57, our IBPH algorithm starts with the visual hull computed by the IBVH algorithm. The IBPH algorithm then evaluates the photo-consistency of the closest point on the visual hull along each ray back-projected from the desired view. If the point is inconsistent, we take a small step along the ray, moving away from the desired view, as depicted in Figure 58. We continue stepping along an inconsistent

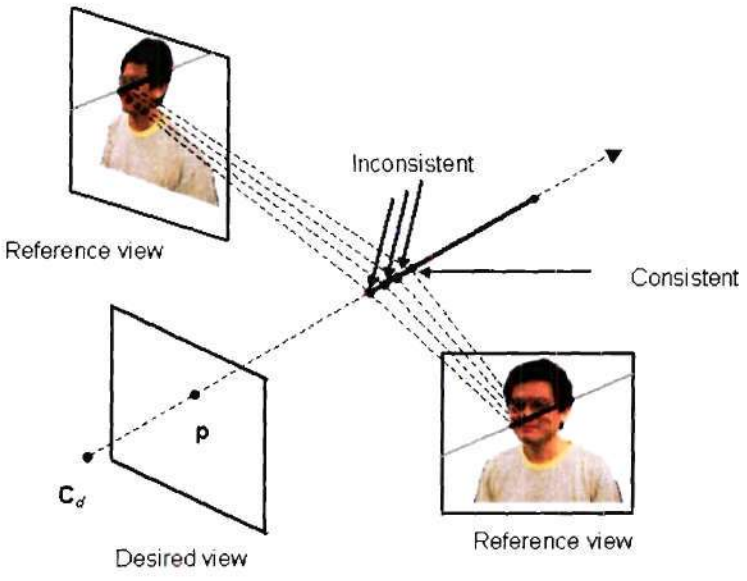


Figure 58: Computing the image-based photo hull.

ray until it either becomes consistent or we have stepped beyond all visual hull intervals along the ray. This latter case indicates that no photo-consistent geometry along the ray was found.

Note that in this approach, only the points on the hull that are visible in the desired view are processed. Initially, these points are the first points in the first visual hull interval along each back-projected ray. By stepping along the inconsistent rays until convergence, the IBPH algorithm reconstructs only the portion of the photo hull that is visible to the desired view.

7.2.2 Stepping along epipolar lines

As we step in 3D along an inconsistent ray, looking for the point at which it becomes consistent, we must simultaneously step along an epipolar line in each reference view. The brute force way of stepping along the epipolar line in a reference view is to simply project each 3D point $\mathbf{P}_i$ on the ray to the reference view point $\mathbf{p}_i$ by multiplying the reference view's projection matrix H with $\mathbf{P}_i$, i.e., $\mathbf{p}_i = H\mathbf{P}_i$. Such an approach will work, but will require a large number of matrix multiplications.

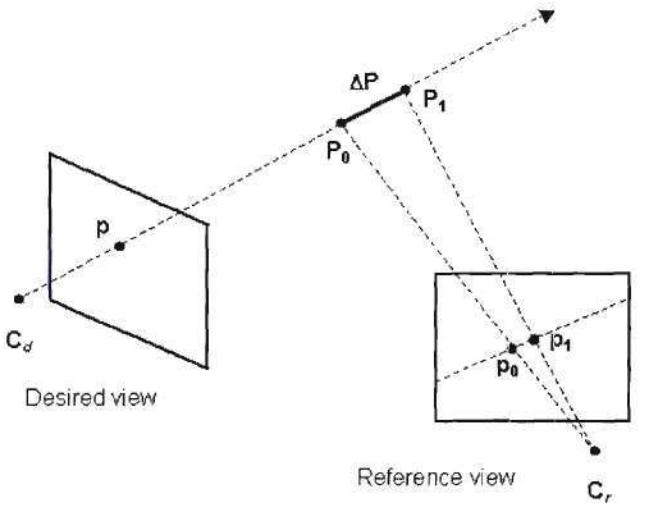


Figure 59: Stepping along an epipolar line.

While the step size $|\Delta\mathbf{P}|$ in 3D is constant, the step size between adjacent points along the epipolar line in a 2D reference view varies because of the projection. However, since the projection is a homography (linear projective transformation), the step size *is* constant in homogeneous coordinates. We use this fact to produce a more efficient procedure for stepping along the epipolar line.

Consider the 3D point $\mathbf{P}_0$ on the ray, as shown in Figure 59. It projects to a point $\mathbf{p}_0 = H\mathbf{P}_0$ in a reference image. If we take a step along the ray, we arrive at a 3D point $\mathbf{P}_1 = \mathbf{P}_0 + \Delta\mathbf{P}$. The point $\mathbf{p}_1$, the projection of $\mathbf{P}_1$ into the reference view, can be written as

$$\begin{aligned} \mathbf{p}_1 &= H\mathbf{P}_1 \\ &= H(\mathbf{P}_0 + \Delta\mathbf{P}) \\ &= \mathbf{p}_0 + H\Delta\mathbf{P} \end{aligned}$$

Thus, we can incrementally update the homogeneous position of the point along the epipolar line. That is,

$$\mathbf{p}_i = \begin{cases} H\mathbf{P}_0, & i = 0 \\ \mathbf{p}_{i-1} + H\Delta\mathbf{P}, & i > 0 \end{cases} \tag{35}$$

We pre-compute the constant $H\Delta\mathbf{P}$ for each ray and store it in a lookup table, as shown in the pseudo-code of Figure 57. As we step along the epipolar line, we use Equation 35 to compute the homogeneous position of the point $\mathbf{p}_i$. With this approach, stepping along an epipolar line is very efficient.

7.2.3 IBPH visibility

When evaluating the photo-consistency of a 3D point, only pixels from the reference views that have visibility of the 3D point should be used. As one steps along the inconsistent rays, the visibility of the scene may change. A point that was not visible in a reference view before may become visible after the step is taken. Therefore, it is necessary to update visibility after stepping. This is achieved by re-executing the visibility procedure described in Section 7.1.3.

Visibility could be updated each time a step is taken along each ray. However, such an excessive number of visibility updates results in a slow reconstruction. Instead, our algorithm takes one step along each inconsistent ray, and then updates visibility. As a result, the visibility may be out-of-date when evaluating some 3D points. However, such an approach is conservative. Pixels from only a subset of the references views that have visibility of the point will contribute to the consistency measure. For a monotonic photo-consistency function [54], this may result in some 3D points being erroneously classified as consistent, while a full visibility calculation would show that they are really inconsistent. Since visibility is updated periodically, such erroneous classifications are properly classified on a later iteration of the algorithm. Such an approach is similar to that used in the GVC-IB approach described in Section 3.5.1.

7.2.4 Sampling

One way to trade off accuracy for speed in both the IBVH and IBPH algorithms is to compute the hull in a multi-resolution fashion. The algorithm is executed not for every pixel in the desired view, but rather on a coarse raster. One first computes the hull at sampling

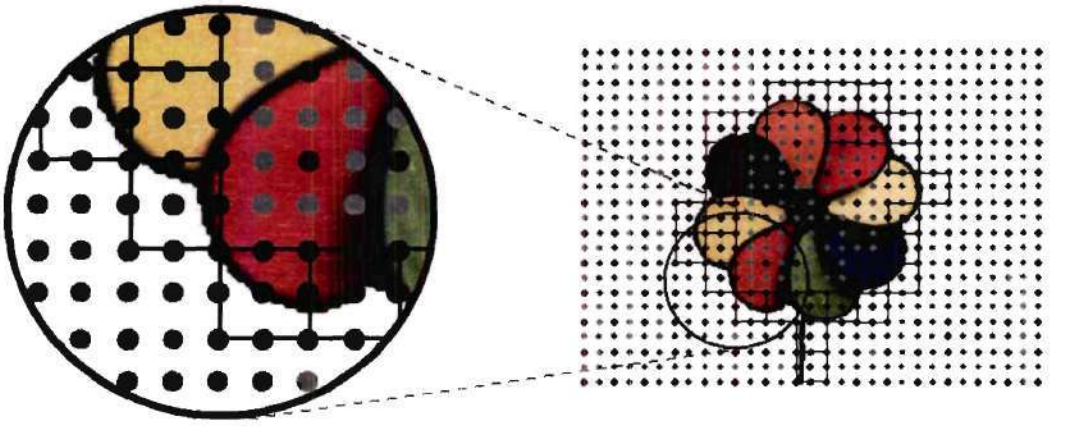
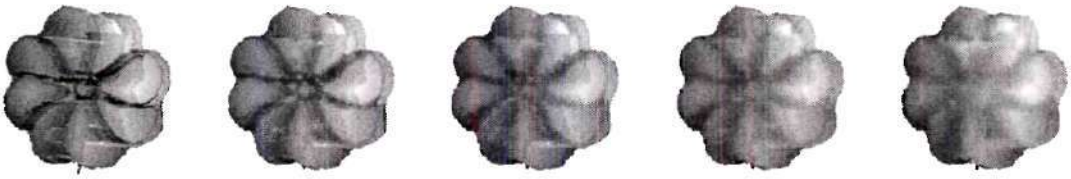


Figure 60: Multi-resolution sampling for faster performance. In this example, DX = DY = 2.

locations $(x \cdot \mathrm{DX}, y \cdot \mathrm{DY})$ in the desired image, where DX and DY are constants that specify the sampling size. The sampling locations are shown as black dots in Figure 60. For in-between pixels on the boundary, indicated using black squares in the figure, the hull is computed at every pixel so that the edges of the synthesized image are at full resolution. For pixels inside the boundary, the closest point of the hull interval (i.e., depth) is interpolated from adjacent samples. This approach significantly reduces the number of rays that must be processed, resulting in a faster reconstruction.

Figure 61 shows the effect increasing the sampling size DX and DY for a pinwheel photographed from five viewpoints. The leftmost image shows the depth map when DX and DY are 1. In this case, we are computing a depth value for every pixel in the desired image. While the depth map is crisp, the frame rate is only 0.4 frames per second (FPS). Increasing the sampling size has a significant impact on the frame rate. For the rightmost image, the DX and DY are both five, and the frame rate is 8.3 FPS. The tradeoff for this improvement in frame rate is blurring of the depth map, since the depth values at pixels inside the border are interpolated from the sampling locations. Continuing to increase the sampling size further blurs the depth map, but has little impact on the frame rate, as more pixels become boundary pixels, which are sampled at full resolution.



1x1	2x2	3x3	4x4	5x5
0.4 FPS	1.6 FPS	4.7 FPS	6.8 FPS	8.3 FPS
(a)	(b)	(c)	(d)	(e)

Figure 61: Effect of increasing sampling size DX, DY on the reconstructed depth map and frame rate.

For IBPH, there is an additional sampling parameter, $\Delta\mathbf{P}$. This is the size of the 3D step taken along an inconsistent ray. In our application, we set $|\Delta\mathbf{P}|$ to a size that results in a projected size $|\Delta\mathbf{p}|$ of about one pixel for most reference views.

7.2.5 Convergence

The IBPH algorithm steps along the inconsistent rays, stopping at the point at which each ray becomes photo-consistent. For convergence, one can require that all rays are photo-consistent. However, often during a reconstruction, a significant majority of the rays will become consistent quickly. Continuing to process a handful of inconsistent rays will yield little impact on the overall quality of the reconstruction, but can take a lot of time. In our implementation, we have introduced a mechanism to terminate the reconstruction when M or less rays are inconsistent, as shown in the pseudo-code of Figure 57. When M is a small number, good quality hulls are produced quickly.

Figure 62 justifies our use of M to terminate the reconstruction before all rays are photo-consistent. This plot shows ray classifications versus iteration for reconstruction of our Sam data set, which consists of a person's head photographed from four viewpoints. The visual hull projected to 1333 of the 80 x 60 points on the coarse raster. Rays back-projected through these points were analyzed using the IBPH algorithm. Initially, 635 were inconsistent and 698 were consistent, as shown in the figure. At each iteration of the algorithm, a step was taken along each inconsistent ray. The plot of the number of inconsistent rays is

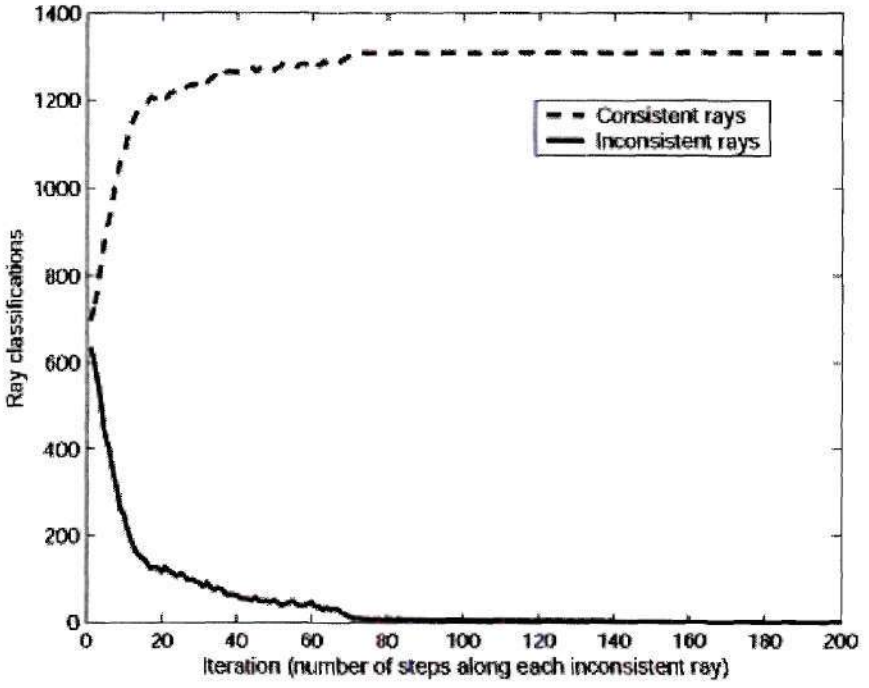


Figure 62: Ray classifications vs. iteration.

very steep at first, indicating that many rays become consistent quickly. After 60 iterations, most rays are consistent. However, it takes an additional 140 iterations for the few remaining inconsistent rays to become consistent. For real-time application, one would rather not continue processing these rays, as they will not significantly contribute to the quality of the reconstructed model.

7.2.6 Spatial coherence

Most scenes exhibit a high degree of spatial coherence, as they consist of surfaces that do not radically change their position over a small region of space. Accordingly, many stereo vision algorithms impose a regularization criterion that requires the reconstructed geometry to be smooth. In a similar vein, we have developed a very simple and efficient smoothing technique that we incorporate into our IBPH algorithm. The smoothing also helps mitigate reconstruction errors because of noise and specularities.

When stepping along an inconsistent ray, we keep track of the number of steps we

Figure 63: Varying K. From left to right, $K = 1$, 2, and 5.

have taken, k. As shown in the pseudo-code of Figure 57, before taking another step, we compare k to a local mean computed by averaging the number of steps taken along rays in a small neighborhood around the inconsistent ray. We denote this local average $\bar{k}$. If $k > \bar{k} + K$, where K is a small constant, we do not step along the ray. This ensures that the number of steps taken along a ray is not significantly different from that of its neighbors, resulting in a surface that is spatially coherent. This smoothing approach requires very little computation and works naturally with the representation of the hull geometry used in our algorithm.

Figure 63 shows the effect of changing K for a reconstruction of a person's head. Notice that there are less abrupt transitions in the depth map in the left-most reconstruction ($K = 1$) compared to the right-most reconstruction ($K = 5$).

7.3 Results

We have implemented the IBPH algorithm on a multi-camera system, shown in Appendix C. We have five calibrated Sony DFW-V500 digital cameras. The cameras are synchronized so that they take images of the scene at the same instant of time. Each camera is connected to an 800 MHz HP Kayak machine. These machines perform background subtraction on the incoming frames, segmenting them into foreground and background regions. The resolution of the reference images is 320 x 240 pixels.

The segmented reference images are sent over a 100 Mb/s switch to our server machine, which computes the 3D reconstruction. Our server machine is a dual processor 2 GHz HP x4000 workstation. Our algorithm has been multi-threaded to take advantage of our multi-processor machine. The ith thread reconstructs the scene using a set of images corresponding to time t_i. In this way, the IBPH algorithm can very naturally be parallelized.

The bottom row of Figure 52 shows the results of IBPH algorithm in reconstructing a pinwheel. We placed the pinwheel in front of the cameras and spun the wheel. Figure 52 shows the reconstruction at one time instant. For this reconstruction, the sampling rate parameters DX and DY were 4, and the resolution of the desired view was 320 x 240. The algorithm reconstructed the scene and generated new views at 6 FPS. The IBPH algorithm produces more geometrically accurate results than the IBVH algorithm. However, the IBVH algorithm ran at 25 FPS for this data.

Figure 64 shows a view from a real-time 3D telepresence application we are currently developing with HP labs. The 3D model of the person's head and neck is reconstructed online using the IBPH algorithm. The reconstructed geometry of the person is then depth composited with a 3D model of a conference room. New synthesized views of this composited scene are generated at 7.5 frames per second. The upper image in the figure shows the texture-mapped model, while the lower image shows the depth map.

Figure 65 compares the results produced using the multi-resolution generalized voxel coloring (GVC) algorithm presented in Chapter 3. The resolution of the final GVC reconstructed model is 160 x 200 x 160 voxels. At this resolution, a voxel projects to about one pixel in the synthesized view. To make a suitable comparison, we show the results of the IBPH algorithm using a 1x1 sampling size. As expected, the synthesized new views and depth maps are similar in quality. The pinwheel's center and petals can be identified in the depth maps, and both new views are similarly realistic.

This IBPH reconstruction was texture-mapped view-dependently; while the GVC reconstruction was texture-mapped view-independently by setting the color of each voxel

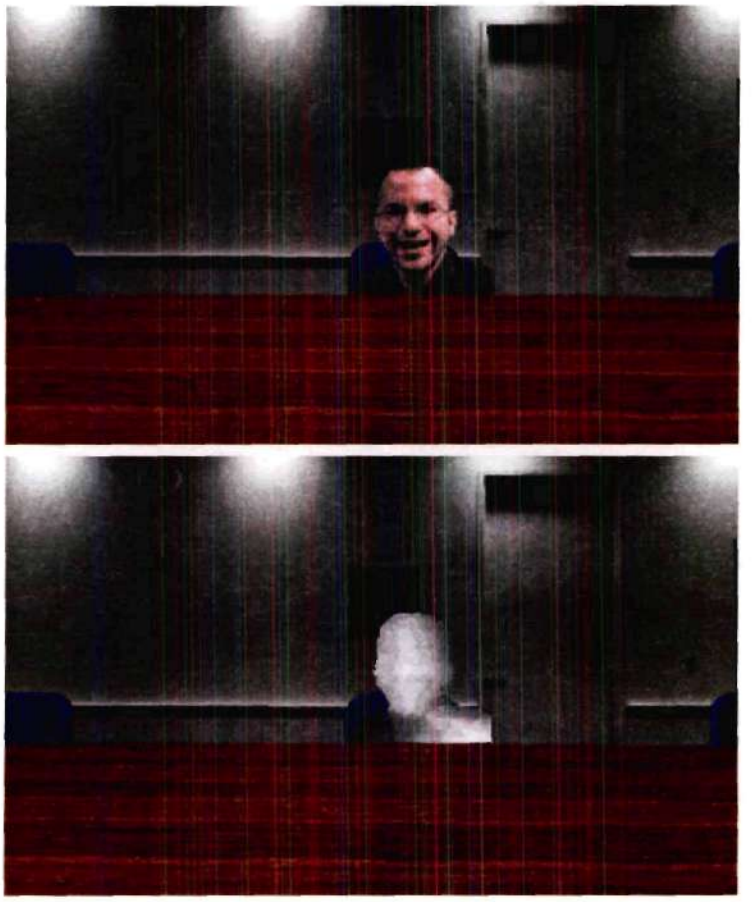


Figure 64: Using IBPH in a real-time 3D telepresence application.

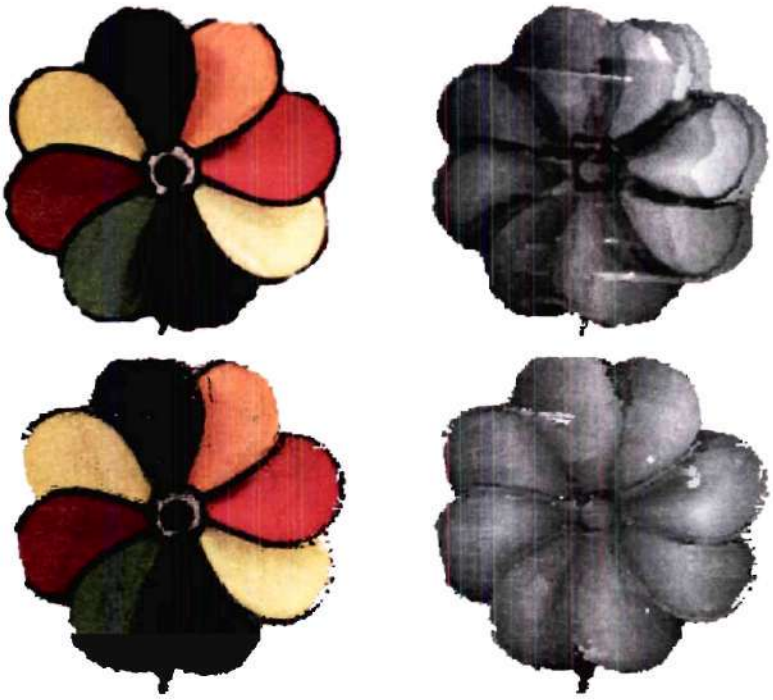


Figure 65: Comparing IBPH (top) to generalized voxel coloring (bottom). Texture-mapped view view (left) and corresponding depth map (right).

equal to its mean of the pixels in its projection in all the reference views. This explains why some speckles appear in some of the petals of the pinwheel in the GVC reconstruction. Also, the GVC result shows some breaking apart of the surface along the outer edges of the pinwheel, which had some specularity. The IBPH algorithm was implemented to preserve edges for this data set.

The GVC reconstruction took 2 minutes and 47 seconds using one processor on our dual processor 2 GHz machine. Even if we multi-threaded GVC to take advantage of the second processor in our machine, it would still run well over an order of magnitude slower than IBPH for a 1x1 sampling size. More detailed comparisons between GVC and IBPH will be presented in Chapter 8.

In Chapter 8 we will compare IBPH to other reconstruction approaches presented in this thesis.

7.4 Summary

In this chapter we have presented our image-based photo hulls algorithm that efficiently reconstructs views of the photo hull. Our algorithm extends the image-based visual hulls algorithm by utilizing the color information in the reference views to further constrain the 3D space containing scene surfaces. The more accurate geometry reconstructed by our technique often results in better new views synthesized of the scene.

Our IBPH algorithm's efficiency comes from the following sources. First, it computes the photo hull starting from the visual hull. Using the IBVH algorithm, we are able to quickly find the visual hull, efficiently removing a large part of the 3D space that does not contain the surfaces being reconstructed. Second, our IBPH algorithm computes only that portion of the photo hull that is visible to the virtual viewpoint. Since this is the viewpoint that is being synthesized, it is not necessary to reconstruct that part of the 3D scene that is not visible in the rendered view. Finally, we have demonstrated the effect of increasing the sampling size DX and DY to tradeoff accuracy for speed.

Our method inherits all the limitations of algorithms that reconstruct the photo hull. Scene surfaces must be sufficiently colorful in order to be properly reconstructed. The photo-consistency measure we use assumes a Lambertian scene. Reconstruction of significantly non-Lambertian scenes requires a more sophisticated approach for determining photo-consistency. We should note that visual hull reconstruction algorithms like IBVH do not have a problem with non-Lambertian scenes, as they do not perform color matching. The depth maps that are reconstructed with the IBPH algorithm, while geometrically more accurate, can be noisier that those of the IBVH algorithm. Finally, the IBPH algorithm is significantly slower that the IBVH algorithm.

There are several possible future directions for this research. The photo-consistency measure presented here does not take into account surface orientation or the distance of the reference view from the surface. Correlation-based matching similar to [34] might improve reconstruction quality. Additionally, we do not take into consideration temporal

coherence. Motion constraints could be imposed to improve efficiency and reconstruction quality. Finally, an adaptive approach to determining $|\Delta \mathbf{P}|$, the step size taken along each ray might improve the efficiency of our algorithm. When the photo-consistency measure is *very* inconsistent, one might take larger steps. As the consistency improves, one could take smaller steps until the ray becomes photo-consistent.

CHAPTER VIII

EVALUATION

This thesis has introduced a variety of methods that reconstruct a 3D scene using multiple photographs. The reconstructed geometry is then used to synthesize views of the scene from new viewpoints. In this chapter, we evaluate the generalized voxel coloring (GVC) algorithm of Chapter 3, the volumetric optimization approach of Chapter 4, the level set method of Chapter 6, and the image-based photo hulls (IBPH) algorithm of Chapter 7 by analyzing the quality of reconstructions produced by each approach. We also compare these methods to the visual hull and the image-based visual hull.

A common problem when working with this class of algorithms is how one should evaluate different 3D scene reconstruction methods. That is, how can one objectively quantify the quality of a 3D scene reconstruction?

An intuitive approach is to use 3D ground truth information. When there is known 3D geometry, one can compute a 3D spatial error measure that characterizes the mismatch between actual and reconstructed geometry. We perform such an analysis in Section 8.1.

Since our application is new view synthesis, such a 3D spatial error measure can potentially be an inadequate indicator for how well a reconstructed scene will produce new views. An alternative evaluation approach leaves one image out during reconstruction. For example, if there are N reference views, the reconstruction is performed using $N-1$ of the reference views. The reconstructed model is then projected to the reference view that was left out, forming an image. This projected image is then compared with the reference view that was left out. We perform such an analysis in Section 8.2.

8.1 3D error evaluation

In this section, we reconstruct a scene of known 3D geometry. We then compute a 3D spatial error that measures the difference between the reconstructed geometry and the known scene geometry.

Using computer graphics methods, it is easy to produce synthetic images of a known 3D scene. One simply specifies the geometry and color of the scene in 3D space, and then renders the scene to a camera viewpoint, forming a reference view. For such a synthetic scene, the cameras are perfectly calibrated and the scene can be made perfectly Lambertian.

For simplicity, we specified a plane located on the xy axis. We call this data set SynthPlane. The plane extends six units in the x and y dimensions each, and has a different randomly assigned color in each 1 unit x 1 unit square. The plane was photographed from 24 different viewpoints, each with a radius $r = 10.5$ units from the center of the plane. Eight viewpoints were taken at 45° intervals of azimuth angle θ at elevation angles of $\phi = 15°, 40°$, and 65°. Figure 66 shows the camera placements, as well as a bounding box containing the plane. Figure 67 shows three of the reference views for $\theta = 0°$ and $\phi = 15°, 40°$, and 65° degrees. These images were generated to have a size of 640 x 480 pixels.

While a multi-colored plane is not the most exciting surface to reconstruct, it is useful and interesting for several reasons. First, it is quite simple to compare reconstructed geometry to that of a plane; we will describe our method below. Second, while the planar geometry is simple, it is effective for comparing different reconstruction algorithms since the reconstructed geometry exhibits the characteristics present in reconstructions of more geometrically complex surfaces. Thus, understanding how well these algorithms reconstruct a plane offers important insight into more complex surfaces. Finally, a plane is easy to model and render, so generating synthetic reference views is a simple endeavor.

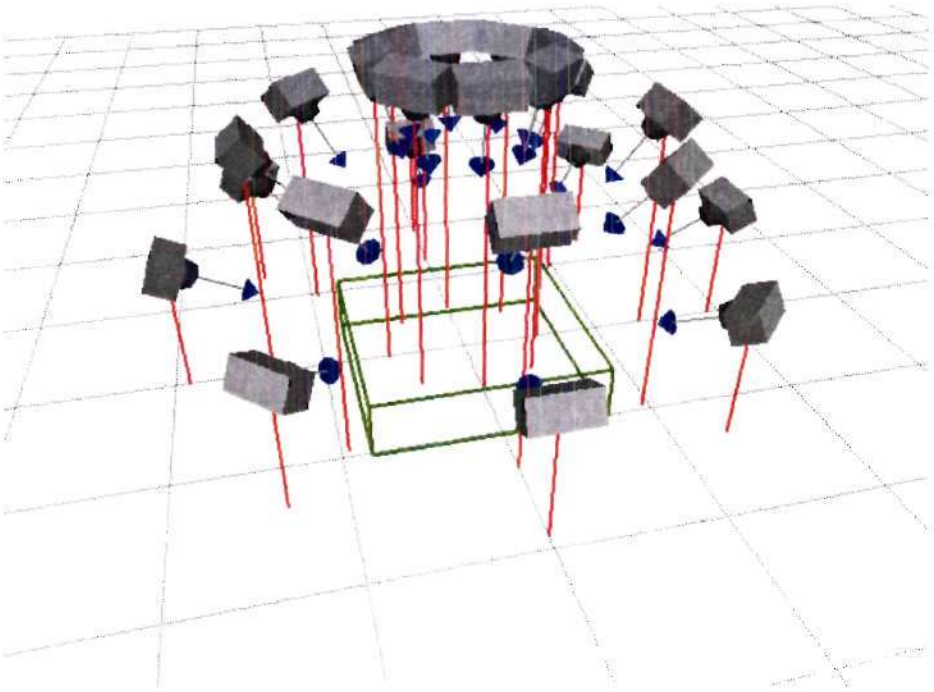


Figure 66: Camera placements and reconstruction volume for the SynthPlane scene.

Figure 67: Three of 24 reference views for the SynthPlane scene, for $\phi = 15°, 40°$, and 65°, left to right.

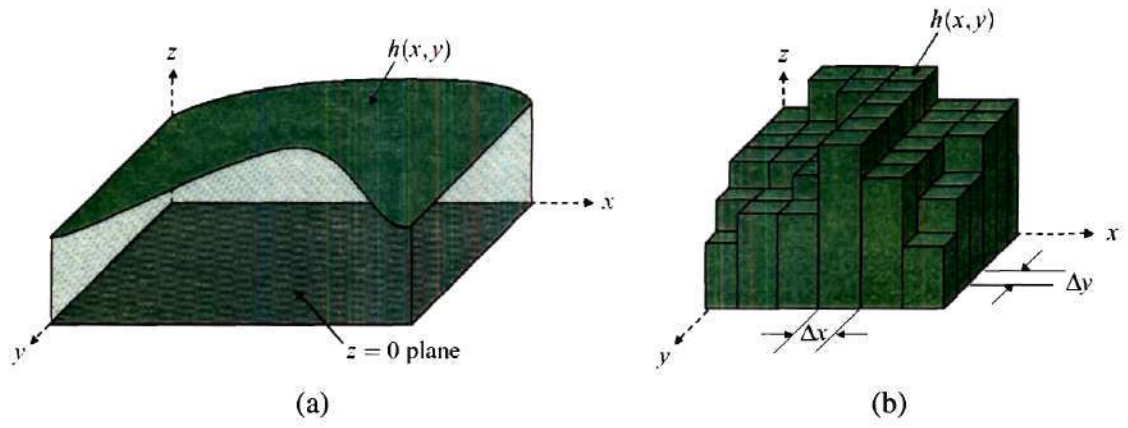


Figure 68: Computing the volume under the reconstructed surface for (a) continuous $h(x,y)$ and (b) discrete $h(x,y)$. In (b), the volume of a box-shaped region under a sample in of $h(x,y)$ is simply $h(x,y)\Delta x\Delta y$.

8.1.1 3D error metric

Our 3D error metric E_{3D} measures the volume of space that exists between the reconstructed surface and true surface located on the $z = 0$ plane, as illustrated in (a) of Figure 68. This volume can be determined by solving the double integral,

$$E_{3D} = \int\int |h(x,y)|dxdy, \tag{36}$$

where $h(x,y)$ is the height of reconstructed surface. We take the absolute value of $h(x,y)$ to ensure that all contributions to the 3D error are positive.

We generate $h(x,y)$ from the reconstructed surface by storing the largest z value for all surface voxels that are indexed by x and y. We call $h(x,y)$ the *height field* of the reconstruction. Since the height field is generated by a set of voxels, it is piece-wise planar, as shown in (b) of Figure 68. Therefore, without approximation, we can replace the integrals of Equation 36 with summations, yielding

$$E_{3D} = \sum_{x=1}^{N}\sum_{y=1}^{M} |h(x,y)|\Delta x\Delta y, \tag{37}$$

where $\Delta x = \Delta y$ is the size of a voxel in one dimension, N is the number of voxels along the x dimension, and M is the number of voxels along the y dimension of the voxel space.

If the surface does not exist at a point (x,y), the height field is undefined. This situation occurs at places, such as holes, where the reconstructed surface has been incorrectly carved away. For the 3D error, we set the height field to zero at such a point. Thus, E_{3D} will not penalize holes that exist in the model. However, the new view synthesis error measure we use in Section 8.2 will penalize holes in the surface.

In the experiments that follow, all reconstructions were performed on a dual processor 2 GHz machine with 1 GB of shared memory. All algorithms, with the exception of image-based visual hulls and image-based photo hulls ran on one processor.

For each approach, we reconstruct the scene three times. The first reconstruction is performed using 8 ($\phi = 15°$) reference views. These correspond to the highest 8 camera placements in Figure 66. The second reconstruction is performed using 16 ($\phi = 15°, 40°$) reference views. These are the upper 16 camera placements in Figure 66. The third reconstruction is performed using all 24 ($\phi = 15°, 40°$, and $65°$) reference views. Performing these three reconstructions shows the effect of reconstructing the scene using more oblique views.

8.1.2 Visual hull reconstruction

The SynthPlane scene was reconstructed in a 160 x 160 x 140 voxel space using a voxel-based visual hull algorithm. At this resolution, a voxel projects to just a few pixels in each reference view.

The top row of Figure 70 displays the height field for the three visual hull reconstructions. As demonstrated in the left-most image, the visual hull reconstructed using just top view photographs results in a very poor surface. The reconstructed model has a pyramidal shape, with a maximum value of 5.3 units at the apex. Clearly, this reconstruction deviates significantly from a plane. We compute E_{3D} using Equation 37, which yields a value of 67.2 units^3. The middle image of the top row of Figure 70 shows the visual hull reconstruction achieved using 16 reference views. The reconstructed model still has a pyramidal shape,

however, the apex of the pyramid is much lower, at a height of 2.45 units. The 3D error for this reconstruction is 30.4 units^3. Finally, the right image of the top row of Figure 70 shows the reconstruction attained using all 24 reference views. This reconstructed model still has a pyramidal shape, but more closely represents a plane. The apex of the pyramid is located at a height of 1.05 units, and the 3D error for this reconstruction is 13.3 units^3. Note that in the figure, the color map ranges from 0 to 6 for the visual hull reconstructions.

We report the time required to perform a reconstruction using a min:sec format. The 8, 16, and 24 reference view reconstructions required a time of 8:28, 9:12, and 11:26, respectively. When one evaluates a voxel to determine if it is part of the visual hull, one must project it into all reference views. Thus, the execution time of the algorithm is proportional to the number of reference views, as observed in these timing results. The visual hull was computed using unoptimized code; therefore these timing results should not be taken too seriously. We tabulate the error and timing results in Table 7.

As more oblique views are used, the visual hull reconstruction more closely approximates the true scene. However, the visual hull geometry significantly deviates from that of the true scene in this experiment.

We will show new views generated from these reconstructions in Section 8.2.

8.1.3 GVC reconstruction

Next, we reconstructed the SynthPlane data set using the generalized voxel coloring algorithm. Again, we performed the reconstruction using 8, 16, and 24 reference views as done for the visual hull reconstruction. However, in this experiment we reconstructed the scene in a single resolution 160 x 160 x 45 voxel space. We lowered the number of voxels in the z dimension to avoid processing large regions of empty space that will not represent the photo hull.

The second row of Figure 70 displays the height field for the three GVC reconstructions. Note that color map the figure ranges from $z = 0$ to $z = 2.5$, which is less than half the range

of the visual hull plots in the top row of the figure. We did this so that more surface detail of the reconstruction is apparent. The three images show the cusping effect discussed in Section 4.1.1. These are wedge-shaped protrusions on the reconstructed surface that result from homogeneously colored regions on the true surface. Any point on the cusp projects to a photo-consistent color in the reference views. As reference views that are more oblique to the plane are included, the height of the cusps diminishes. The reconstruction attained using all 24 reference views is quite close to planar.

We computed E_{3D} for the three GVC reconstructions, which yielded 21.4 units^3, 6.13 units^3, and 2.17 units^3, for reconstructions using 8, 16, and 24 reference views, respectively. The maximum value of the height map was 1.7 units, 0.55 units, and 0.25 units, respectively, for the three reconstructions. At this resolution, the voxel size was 0.05 units. Note that both the 3D error E_{3D} and the maximum height value are significantly decreased in the GVC reconstructions compared to the visual hull reconstructions for the same number of reference views.

In Figure 69 we show a close-up of the height map, and one of the reference views from a similar viewpoint. Notice how the colors that are similar to each other, like the two adjacent cyan colors, form a plateau in the height map. This is because the photo-consistency measure could not disambiguate the two colors. Colors that the photo-consistency measure finds to be different have valleys along the edges where they touch.

The 8, 16, and 24 reference view reconstructions required a time of 5:24, 12:03, and 16:27, respectively. Like the visual hull reconstruction algorithm, the time required for a reconstruction to complete is proportional to the number of reference views. We tabulate the error and timing results in Table 7.

(a)

(b)

Figure 69: Zooming in on the GVC height map. Similar colors form plateaus in the depth map, while different colors have valleys along the edges where they touch.

8.1.4 Volumetric optimization

We executed the volumetric optimization algorithm on the GVC reconstructions of the previous subsection. Recall that the volumetric optimization algorithm refines the reconstructed surface by minimizing reprojection error. For each experiment, we ran the greedy algorithm, first doing a pass where we carved voxels, and then a pass where we added voxels. During both passes, only modifications to the surface that decreased the reprojection error were accepted.

Table 6 tabulates the experimental results from the volumetric optimization of the three reconstructions. Volumetric optimization decreased the reprojection error by about 34% on average for the three reconstructions. Reprojection error is more closely related to the new view synthesis error measure we use in Section 8.2 than the 3D error we compute in this section.

The 3D error for the GVC reconstruction was 21.4, 6.4, and 2.2 units^3, for the reconstructions attained using 8, 16, and 24 reference views, respectively. After volumetric optimization, the 3D error of the optimized model was 20.3, 6.1, and 2.1 units^3, as shown in Table 6. Thus, the improvement in 3D error, about 5% on average for the three reconstructions, was not as dramatic as the change in reprojection error. However, the optimized surface more closely resembled the true planar surface, since the 3D error decreased for all three reconstructions. Also, the optimized surface has few floaters for the reconstruction using 8 reference views, unlike the GVC reconstruction. The third row of Figure 70 displays the height fields for the three optimized reconstructions.

The 8, 16, and 24 reference view volumetric optimizations required times of 35:28, 32:07, and 51:19, respectively. The time required for the volumetric optimization to complete depends primarily on how close the unoptimized surface is to a local minimum in the reprojection error. Therefore, the execution time may increase or decrease with the number of reference views. We tabulate the error and timing results in Table 7.

Views	8	16	24
Time (min:sec)	35:28	32:07	51:19
Number of voxels carved	10324	9011	8038
Number of voxels added	670	613	212
RE_{start}	4204	2681	3027
RE_{end}	2648	1788	2083
Improvement	37.0 %	33.3 %	31.1 %
E_{3D} (units^3)	20.29	6.13	2.07
Max. height (units)	1.7	0.55	0.25

Table 6: Volumetric optimization of the SynthPlane data set.

8.1.5 Level set reconstruction

Next, we reconstructed the SynthPlane data set using the level set algorithm. Our multi-resolution approach started with a 40 x 40 x 12 voxel space, and allowed the algorithm pass through two resolution increases, resulting in a 160 x 160 x 48 voxel space.

We reconstructed the scene using 8, 16, and 24 reference views as done for previous algorithms. For each reconstruction we set the constants α_0 and α_1 both to 0.5 so that the normal flow and the curvature flow contributed equally to the evolution. The incorporation of the curvature flow term resulted in smoothing of the model, as seen in height fields shown in the fourth row of Figure 70. The cusps located at each homogeneously colored square have a more rounded shape compared with the other reconstruction methods. This smoothed surface more closely resembles a plane. Computation of the 3D error resulted in values of 17.4, 5.46, and 1.24 units^3, for the 8, 16, and 24 reference view reconstructions, respectively. These 3D error values are the lowest of all algorithms evaluated in this thesis. However, the reconstructed model had some carved away edges that were not penalized in our 3D error metric. However, these carved edges will be penalized in our error measure described in Section 8.2.

While this approach produced the best results in terms of 3D error, it generally required the most to execute. The 8, 16, and 24 reference view reconstructions required times of 41:51, 46:06, and 51:17, respectively.

8.1.6 Image-based photo hulls reconstruction

Next, we ran the image-based visual hulls and image-based photo hulls algorithms on the SynthPlane data set. Recall that these algorithms view-dependently reconstruct the scene, producing the visible portion of the hull as seen by a virtual camera placed in the scene. We placed the virtual camera directly above the plane, for an azimuth angle $\phi = 0°$, and a radius 10.5 units up the z axis, shown as the orange camera in Figure 72. Using a sampling lattice of 4 x 4, we generated a synthetic view with resolution 640 x 480 pixels. The algorithm produced a layered-depth image at the virtual camera position. We then converted the closest point on the hull into a 3D height map $h(x,y)$ for analysis.

The IBVH algorithm produced results very similar in quality to those of the visual hull reported in Section 8.1.2, but ran much more quickly. For the 8, 16, and 24 reference view reconstructions, 3D errors of 64.7, 30.0, and 17.7 units^3 were attained in 0.217, 0.377, and 1.16 seconds, respectively. Note that the images of the SynthPlane data set have a black background, and are therefore pre-segmented into foreground and background. For photographic data sets, the background segmentation would require additional computation. We do not show results attained using the IBVH algorithm in Figure 70 because they are nearly identical to the visual hull shown in the top row of the figure. However, we do tabulate the results in Table 7.

Finally, we performed reconstructions using the image-based photo hulls algorithm. For the 8, 16, and 24 reference view reconstructions, 3D errors of 17.7, 7.65, and 5.43 units^3 were attained. The 3D surface using 8 reference views compared quite well to other methods like GVC. However, the surfaces generated using 16 and 32 reference views were significantly noisier than those produced using the other methods, as seen in bottom row of Figure 70. In particular, notice that there is a significant floater in the center of the of the height field for the middle image. This noise in the height fields results in a higher 3D error.

The inaccuracy of the IBPH height fields is hardly surprising, since the IBPH algorithm sacrifices quality for speed. By reconstructing the scene on a coarse 4 x 4 raster, the algorithm produces a depth map that is lower resolution and less accurate. Also, when computing photo-consistency, IBPH uses a small, fixed size footprint in each reference view that is independent of the location of the reference view relative to the surface being reconstructed. This approximation to the projection of a voxel is imprecise. However, it is simple to implement and quite fast.

While the IBPH algorithm produces somewhat imprecise reconstructions, it does so orders of magnitude faster than competing space carving approaches. The 8, 16, and 24 reference view reconstructions required times of 6.21, 5.55, and 12.3 seconds, respectively.

Recall that the IBPH algorithm computes the photo hull starting from the visual hull. As seen in the top row of Figure 70, the visual hull reconstructed using 8 reference views has a pyramidal shape that is significantly non-planar. Therefore, the IBPH algorithm must do a lot of work stepping along inconsistent rays to reach the photo hull. The visual hull computed using 16 reference views is much closer to a planar surface, which saves the IBPH algorithm a lot of work. For this reason, the algorithm execution time actually *decreases* when going from 8 to 16 reference views. The time increases when going from 16 to 24 reference views for two reasons. First, in general the runtime of the algorithm is proportional to the number of reference views, since a point on a ray must be projected to all reference views to determine photo-consistency. Second, the multi-threaded version of the IBPH algorithm did not fit into main memory when processing 24 reference views. Therefore, we ran the algorithm only using one processor, which required approximately half the memory. As a result, the reconstruction was roughly two times slower that it would have been had it ran on two processors.

8.1.7 Summary of 3D error analysis

In this section we have performed a 3D error analysis of most of the reconstruction approaches developed in this thesis. The results are summarized in Figure 70 and Table 7. These experiments reconstructed a simple scene of known 3D geometry.

The results show that the level set reconstruction method performed the best in terms of most accurately reconstructing the geometry. During reconstruction, this algorithm smoothes the surface. This helps mitigate the cusps in the reconstructed model. As a result, the algorithm produces results that have the lowest 3D error. Despite these good results, the level set approach does have some detriments. It is sensitive to its parameters - the thresholds used in the photo-consistency measure, the value of σ_0, and X, the number of iterations that serve in the stopping criterion for the algorithm. Furthermore, we will show in Section 8.2 that this method carved away some photo-consistent edges of the plane.

By taking advantage of the color information in the reference views, GVC produces better results than the visual hull. In terms of 3D error, the volumetrically optimized surface is even more accurate than the GVC reconstruction, but only by a slim margin. The benefit one gets by executing this algorithm might not be worth the amount of time required for it to run. Finally, the IBPH algorithm produces a lower quality surface, but orders of magnitude faster than GVC, volumetric optimization, and the level set approach. Thus, IBPH is best suited to time-critical applications.

Figure 71 plots the 3D error vs. time for all reconstructions of the SynthPlane data set. Some general observations can be made from the figure. First, as more reference views are used, the 3D error decreases. Second, the IBVH and IBPH algorithms are significantly faster than the other approaches. For a given number of images, the level set method achieves a reconstruction with the lowest 3D error for this data set. Finally, the visual hull geometry, reconstructed from silhouettes, is much less accurate than the geometry produced by methods that utilize the color information in the photographs.

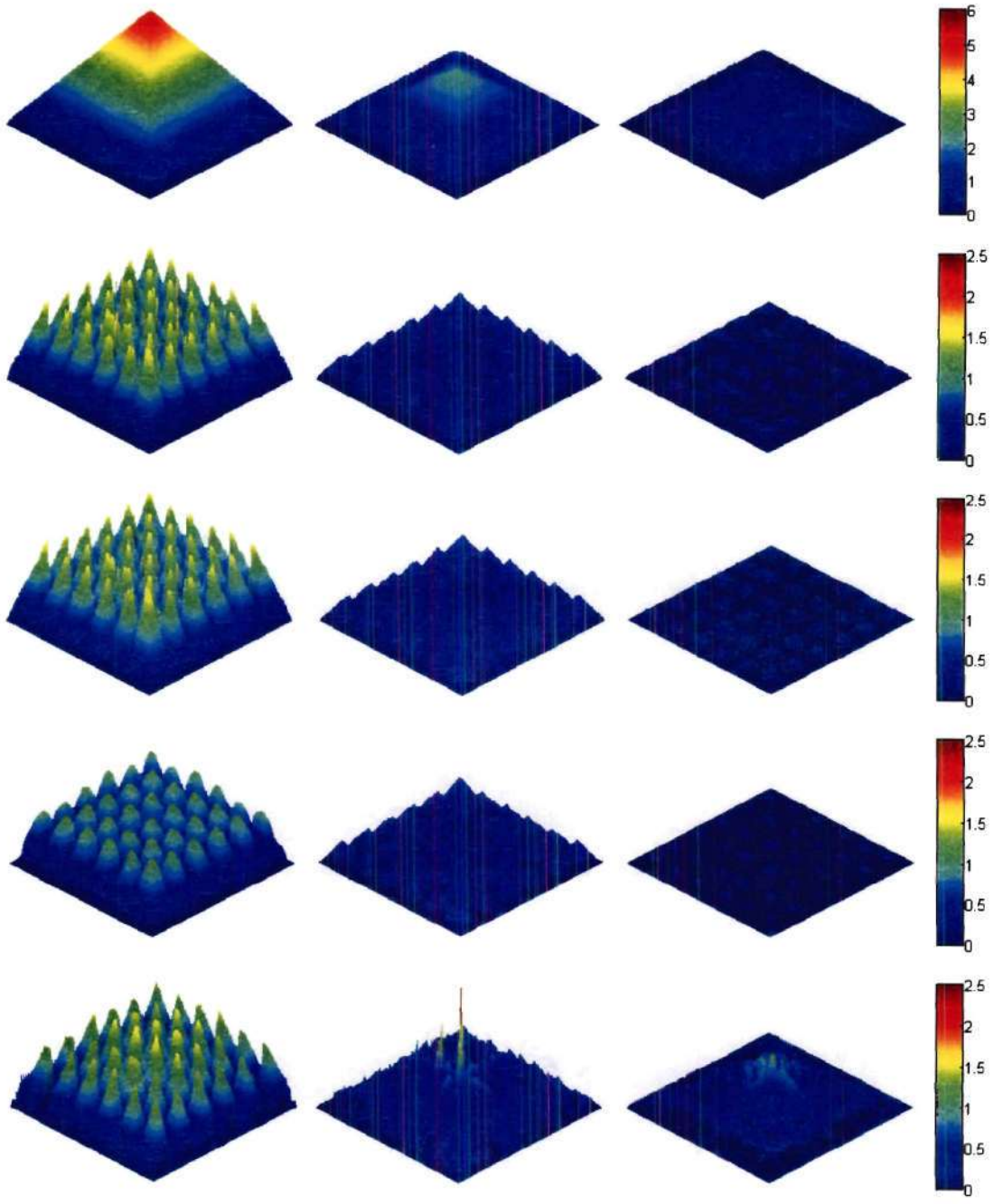


Figure 70: Reconstructions of the SynthPlane data set. From left to right: using 8, 16, and 24 reference views. From top to bottom: visual hull, GVC, volumetric optimization, level set method, and image-based photo hulls. See text for analysis.

Approach	No. ref. views	E_{3D} (units^3)	Max. height (units)	Time (m:s)
Visual Hull	8	67.2	5.30	08:28
GVC	8	21.4	1.70	05:24
Volumetric optimization	8	20.3	1.70	35:28
Level sets	8	17.4	1.13	41:51
IBVH	8	64.7	5.32	00:0.217
IBPH	8	17.7	1.63	00:6.21
Visual Hull	16	30.4	2.45	09:12
GVC	16	6.41	0.55	12:03
Volumetric optimization	16	6.13	0.55	32:07
Level sets	16	5.46	0.47	46:06
IBVH	16	30.0	2.47	00:0.377
IBPH	16	7.65	2.42	00:5.55
Visual Hull	24	13.3	1.05	11:26
GVC	24	2.17	0.25	16:27
Volumetric optimization	24	2.08	0.25	51:19
Level sets	24	1.24	0.125	51:17
IBVH	24	12.5	1.05	00:1.16
IBPH	24	5.43	1.05	00:12.3

Table 7: 3D error analysis results.

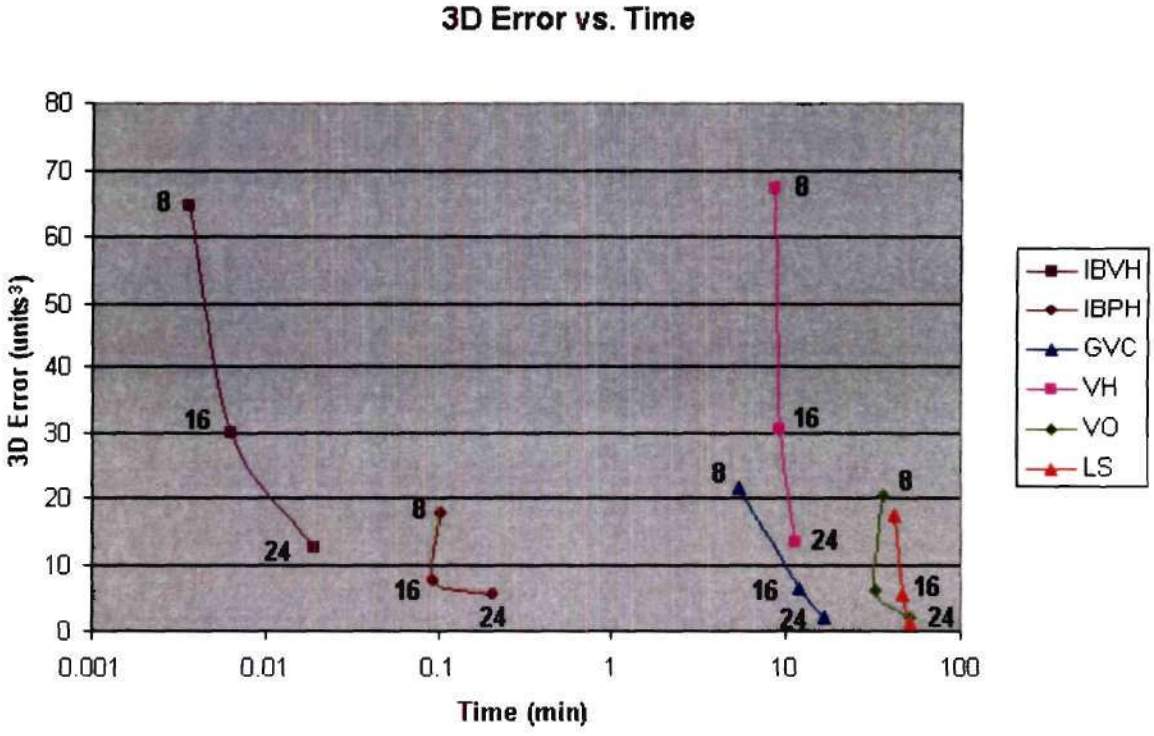


Figure 71: 3D error vs. time, SynthPlane scene. Note that time is plotted on a logarithmic scale. The figure shows results using 8, 16, and 24 reference views for the image-based visual hulls (IBVH), image-based photo hulls (IBPH), generalized voxel coloring (GVC), visual hull (VH), volumetric optimization (VO), and level set (LS) reconstruction algorithms.

8.2 New view synthesis evaluation

The previous section computed the 3D error between the true and reconstructed surface geometry for approaches described in this thesis. The 3D error is a useful characterization of the reconstruction algorithms, and is especially relevant if the application of the algorithm requires a geometrically accurate 3D model.

However, the 3D error is not necessarily indicative of how well a reconstruction will synthesize new views. For some synthesized viewpoints, errors in the 3D geometry will not generate any objectionable artifacts in the synthesized view. Therefore, an alternate measure for evaluating a reconstructed model's ability to synthesize new views is needed. We describe such a method in this section.

8.2.1 2D mean square error

After performing a 3D reconstruction, we can render the reconstructed model to generate a synthetic image R at a new viewpoint V. We can then compare the pixels in R to a photograph P, taken at location V, that is not a viewpoint for one of the reference views used to reconstruct the scene.

This 2D error measures the difference between R and P. Since R and P are color images, they have pixels with r, g, and b components. Let the color components in the ith pixel of R be referenced as $R(i).r$, $R(i).g$, and $R(i).b$, respectively. Similarly, we denote the r, g, and b components of the ith pixel of P as $P(i).r$, $P(i).g$, and $P(i).b$. We then define the new view synthesis error to be the mean square error between R and P,

$$E_{2D} = \frac{\begin{array}{r}(P(i).r - R(i).r)^2 + \\ \Sigma_{i=1}^{M}\ (P(i).g - R(i).g)^2 + \\ (P(i).b - R(i).b)^2\end{array}}{M}, \tag{38}$$

where M is the number of pixels used in the comparison.

This error metric is similar to the reprojection error of Chapter 4 but differs in several

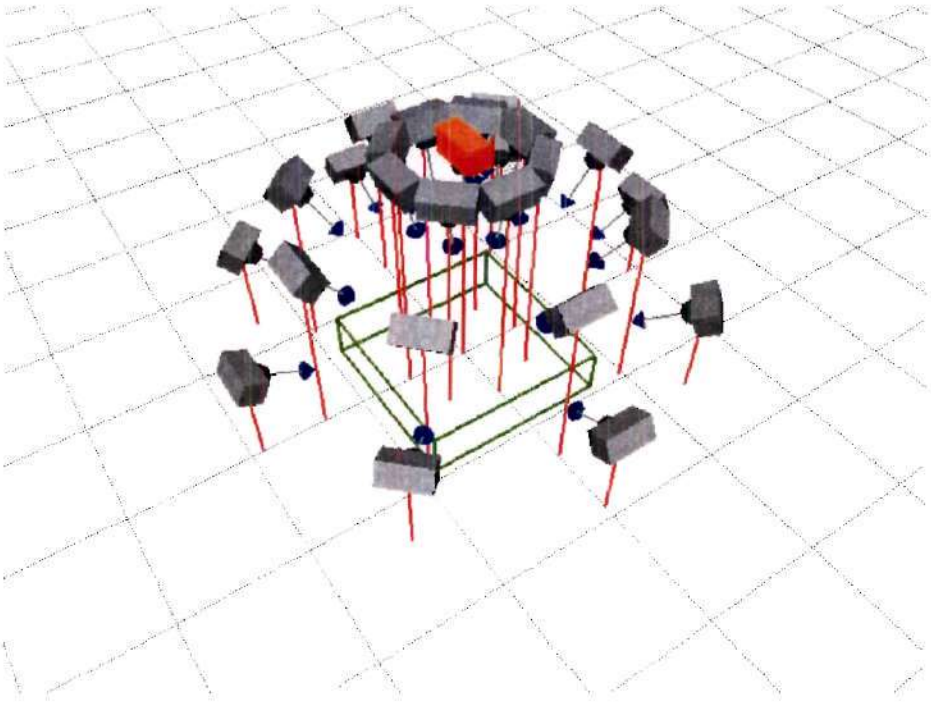


Figure 72: The camera placement used for new view synthesis evaluation is shown highlighted in orange. The camera placements of the reference views are shown in gray.

important ways. First, the reprojection error is computed over all reference views. In contrast, E_{2D} is computed at one view - the view being synthesized. Second, the photograph P used in E_{2D} is not used to produce the reconstruction.

We generated an image P of the SynthPlane scene by rendering the multi-colored plane to a camera location V directly above the plane, for an elevation angle $\phi = 0°$, and a radius 10.5 units up the z axis. The camera placement is shown in orange in Figure 72. This image P, shown in Figure 73, is the image we will attempt to synthesize by rendering the reconstructions previously generated.

8.2.2 Visual hull

We rendered the voxel-based visual hull to synthesize a new view at V. The results are shown in Figure 74. The left image of the first row of the figure shows the synthesized view achieved using the 8 reference view reconstruction. The synthesized view is quite

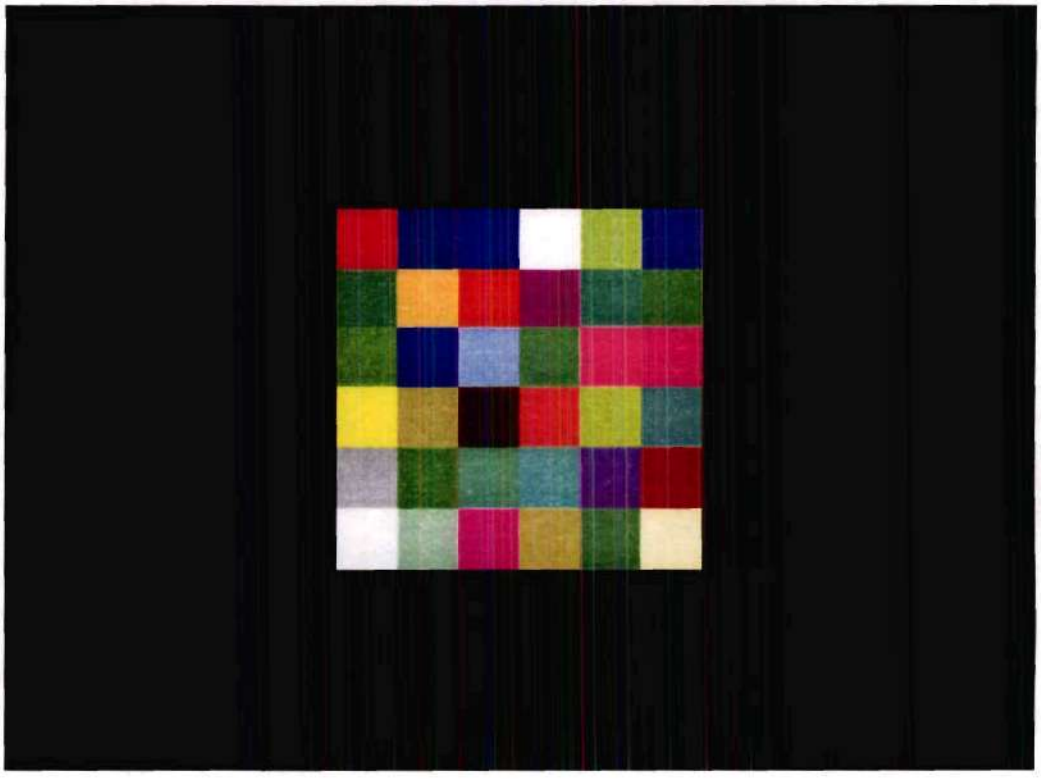


Figure 73: An image formed by rendering the true scene to viewpoint V. In the experiments that follow, we will synthesize this viewpoint by rendering 3D models produced by the different reconstruction algorithms.

poor, since the pyramidal 3D geometry of the visual hull significantly deviates from true planar geometry of the scene. Each voxel is assigned a color equal to the mean of the pixels in its projection in all reference views that have visibility of the voxel. Due to the inaccurate geometry, voxels project to incorrect pixels in the images. These pixels are then averaged together, causing many of the colors bleed together.

The right images of the figure show square error as a grayscale image. A darker color indicates a larger error. This image was produced by comparing the image on the left with the image in Figure 73 using Equation 38.

The middle and bottom rows show the view synthesized using the reconstruction achieved with 16 and 24 reference views, respectively. Adding more reference views produces a more accurate new synthesized view. However, even for the reconstruction achieved using 24 reference views, the inaccurate geometry of the visual hull still results in significant mean squared error. From top to bottom, the mean square error was computed to be 4889, 2484, and 1299. Table 8 summarizes these results.

8.2.3 Generalized voxel coloring

Next, we rendered the GVC reconstructions to synthesize new views at V. The results are shown in Figure 75. From top to bottom, the rows indicate the new views synthesized using the reconstructions achieved using 8, 16, and 24 reference views. Note that these synthesized images exhibit fewer artifacts than those produced using the visual hull. A number of floaters are present in the 8 reference view reconstruction. They appear a speckles in the new synthesized view, as seen in the first row of the figure. The reconstructions achieved using more reference views better synthesize the new view, as shown in the figure. From top to bottom, the mean square error for these three synthesized views was computed to be 596.3, 405.5, and 360.2, for the 8, 16, and 24 reference view reconstructions, respectively. Table 8 summarizes these results.

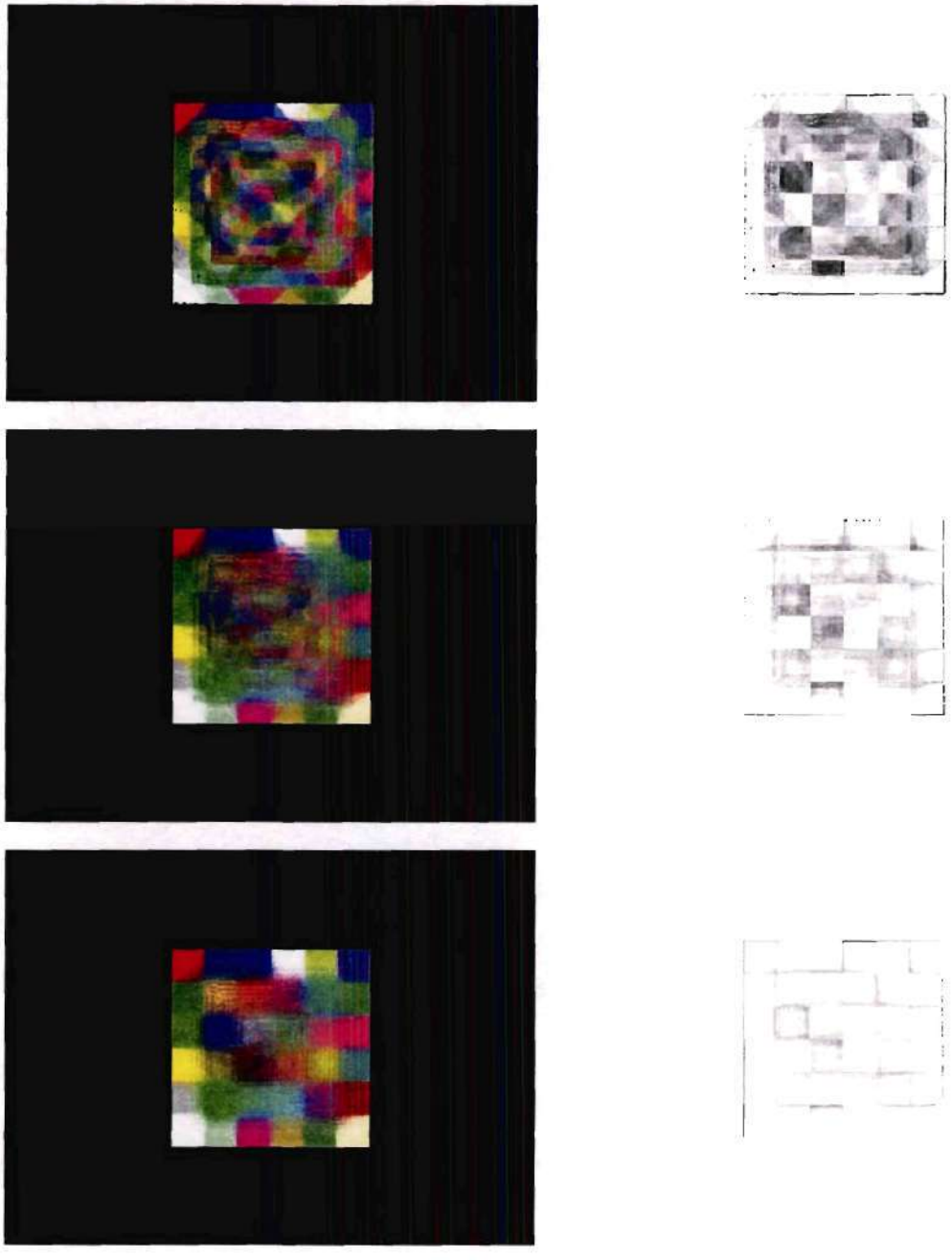


Figure 74: New view synthesis results for the visual hull algorithm. From top to bottom: using 8, 16, and 24 reference views. The synthesized views are on the left, and the corresponding squared error images are on the right.

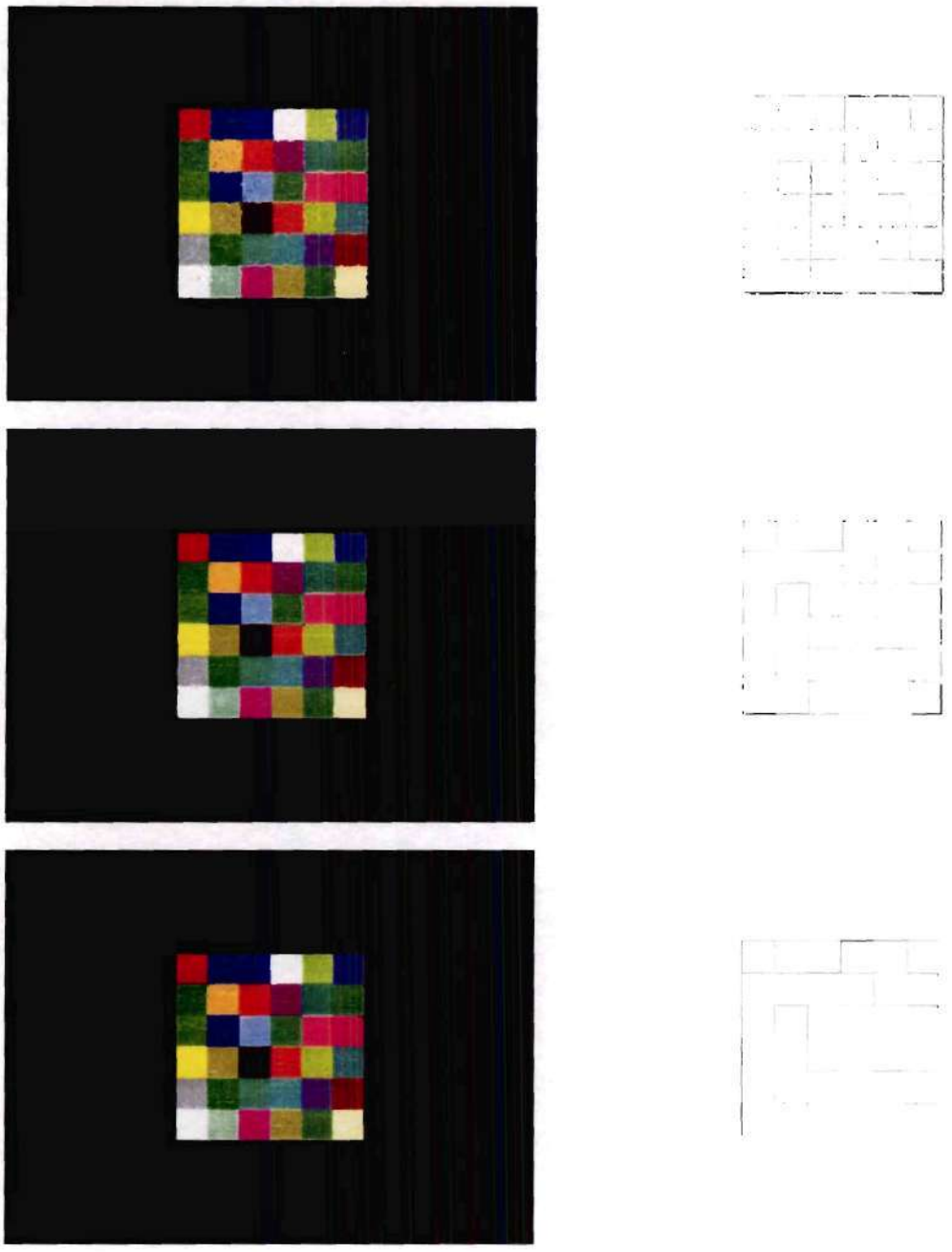


Figure 75: New view synthesis results for the GVC algorithm. From top to bottom: using 8, 16, and 24 reference views. The synthesized views are on the left, and the corresponding squared error images are on the right.

8.2.4 Volumetric optimization

We synthesized views using the results of the volumetric optimization. The new views and the squared error images are shown in Figure 76. The volumetric optimization approach produced synthetic views that had the lowest mean squared error of all the reconstruction algorithms analyzed. For the three reconstructions using 8, 16, and 24 reference views, the mean square error E_{2D} was computed to be 428.8, 329.1, and 300.5. This volumetric optimization improves upon the results of the GVC algorithm by removing many floaters and thinning the surface without forming holes in the model. The improvement in E_{2D} that resulted from optimizing the surface was 28.1%, 18.8%, and 16.6%, respectively. Table 8 summarizes these results.

8.2.5 Level set method

Figure 76 shows the new view synthesis results attained by rendering the polygonal surface reconstructed by the level set approach. While in Section 8.1 we showed that the smoothing property of this approach helps mitigate the effect of floaters and cusps, it has the detriment that it can penalize high curvature parts of the object being reconstructed. As a consequence, the evolving surface can pass through photo-consistent edges and corners that project to different colors in the reference views. This artifact is observable in the reconstructions in the figure, as the edges of the plane are not properly reconstructed. Since the mean square error E_{2D} penalizes holes in the model, the level set method did not fare well in this analysis. For the 8, 16, and 24 view reconstructions, E_{2D} was computed to be 1568, 722, and 1854, respectively. The reprojection error increased when using all 24 reference views, since when using more images, some of the edges looked less photo-consistent. This allowed the evolving surface to propagate more readily through photo-consistent regions. Table 8 summarizes these results.

Figure 76: New view synthesis results for the volumetric optimization algorithm. From top to bottom: using 8, 16, and 24 reference views. The synthesized views are on the left, and the corresponding squared error images are on the right.

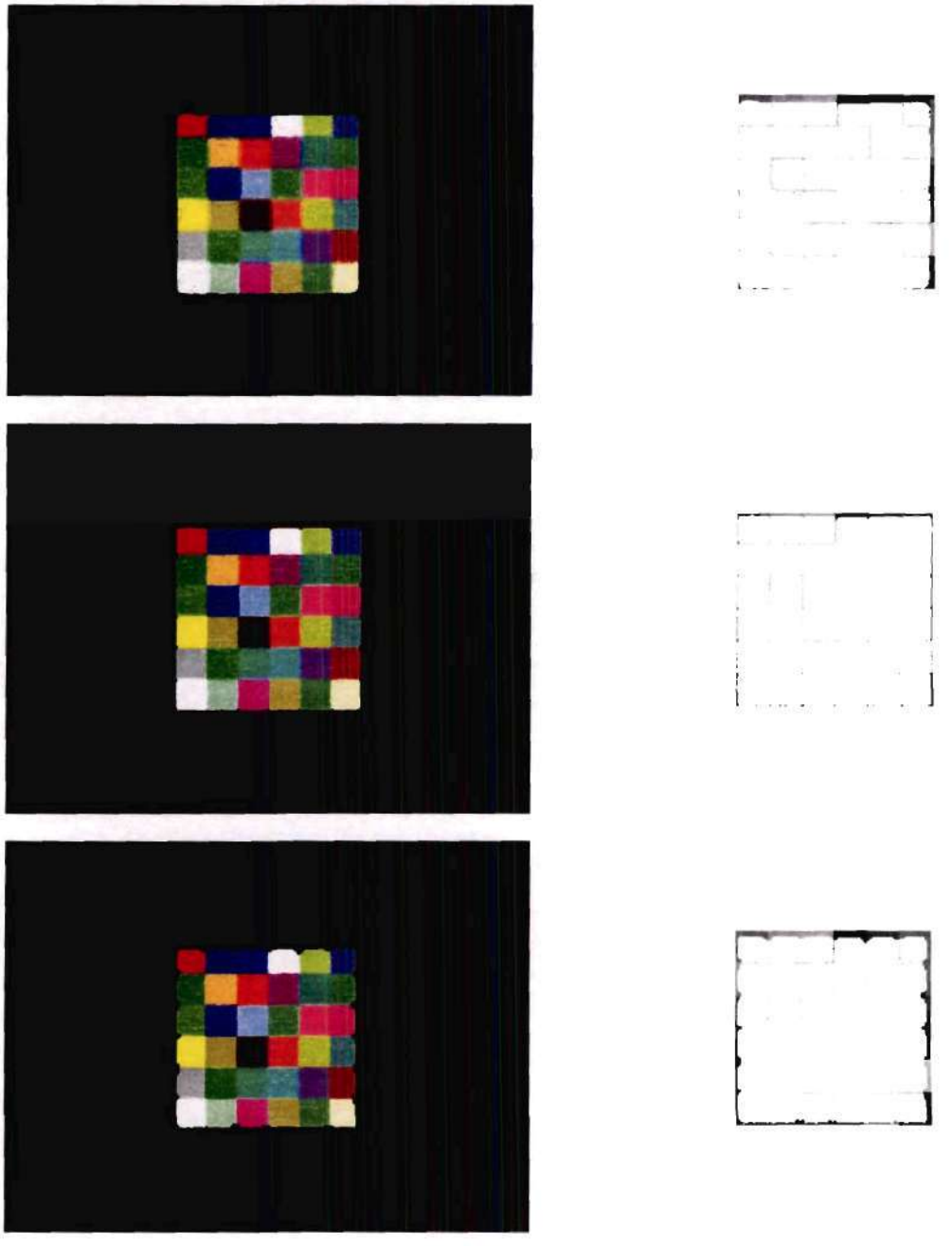


Figure 77: New view synthesis results for the level set algorithm. From top to bottom: using 8, 16, and 24 reference views. The synthesized views are on the left, and the corresponding squared error images are on the right.

8.2.6 Image-based visual hulls

Next, we performed new view synthesis using the image-based visual hulls algorithm. While the IBVH algorithm reconstructs the visual hull, just like the voxel-based visual hull reconstruction algorithm, it texture maps the reconstructed geometry using view-dependent texture mapping. Thus, the new synthesized views produced by the IBVH algorithm are quite different from the view-independent texturing of the voxel-based visual hull in Section 8.2.2, even though both have very similar geometry.

Figure 78 shows the results of synthesizing the viewpoint V using the IBVH algorithm with 8, 16, and 24 reference views. The poor results are a consequence of the inaccurate geometry of the visual hull. Compared with the results from the voxel-based visual hull of Figure 74, the view-dependent texture mapping (VDTM) by the IBVH algorithm produces less blurry images.

Recall that the VTDM approach assigns a color to a pixel based on the reference view that has the best viewpoint of the reconstructed geometry being colored. By not averaging colors across the reference views, blurring is avoided, but when the geometry is inaccurate, new synthesized views lack photo-realism. As more reference views are used, the image-based visual hull geometry improves, but artifacts still remain, as seen in Figure 78. For the 8, 16, and 24 view reconstructions, the 2D mean squared error E_{2D} was computed to be 6770, 3776, and 1778, respectively. Table 8 summarizes these results.

8.2.7 Image-based photo hulls

Finally, we synthesized new views at viewpoint V using the image-based photo hulls algorithm. The results appear in Figure 79. For the 8, 16, and 24 view reconstructions, the 2D mean squared error E_{2D} was computed to be 522.3, 773.3, and 807.1, respectively. Contrary to intuition, this error *increased* as more reference views were used. We demonstrated in Section 8.1.6 that using more reference views with the IBPH algorithm results in a more geometrically accurate reconstruction. However, the view-dependent texture

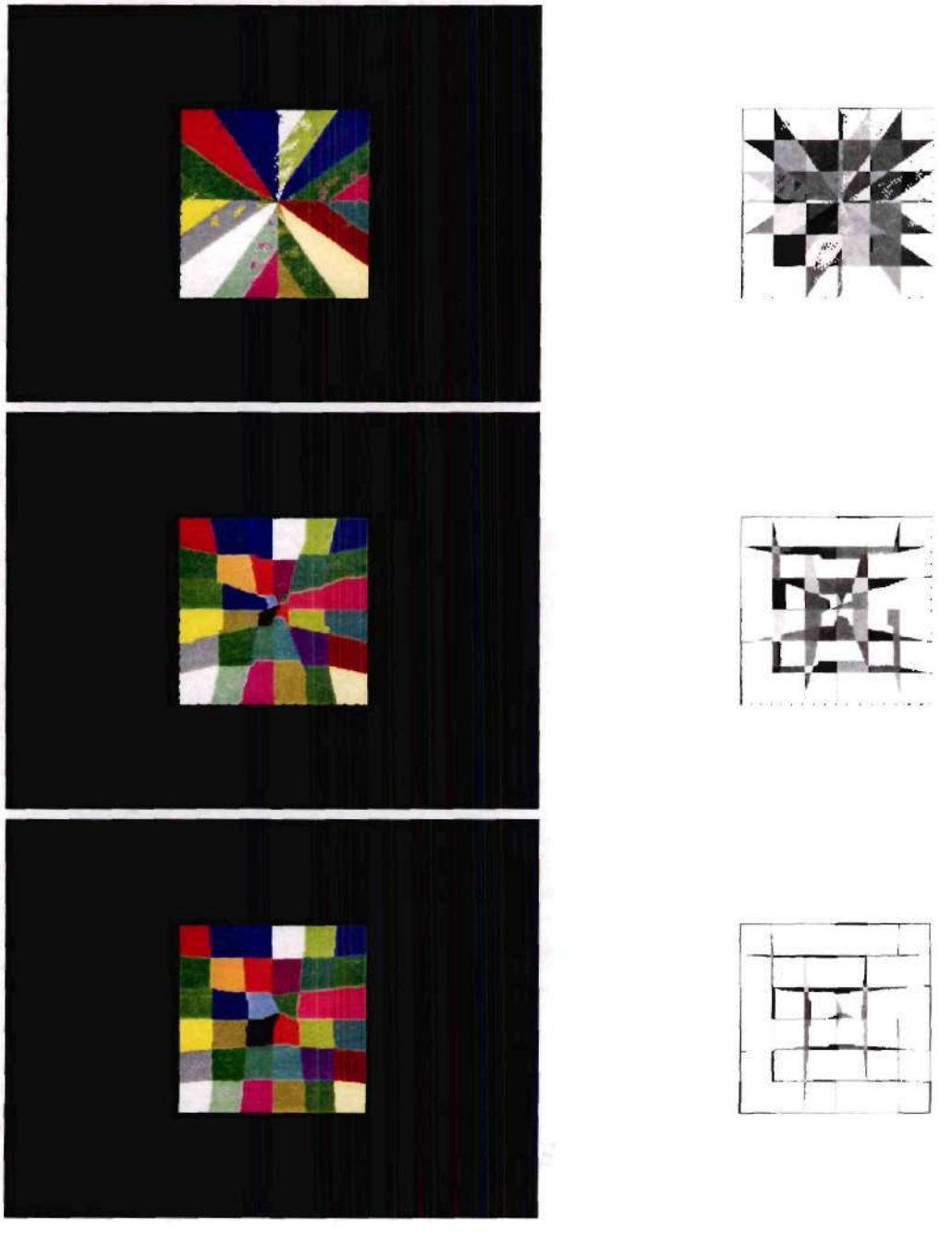


Figure 78: New view synthesis results for the IBVH algorithm. From top to bottom: using 8, 16, and 24 reference views. The synthesized views are on the left, and the corresponding squared error images are on the right.

mapping strategy becomes problematic when many reference views are used, since the reference view being used to texture-map a local point on the surface changes quickly along the surface. In the presence of inaccurate geometry, this results in small miscolorings, especially on the edges between two different colors as seen in Figure 79. Table 8 summarizes these results.

8.2.8 Summary of 2D mean square error analysis

In this section we performed a 2D mean square error analysis that evaluates a method's ability to synthesize new views. The mean square error computes the difference between a synthesized image R and a photograph P generated at the same viewpoint as R but was not used to reconstruct the scene. The results are tabulated in Table 8, and Figure 80 plots the mean square error (MSE) as a function of time for the various reconstructions.

For the SynthPlane data set, the volumetric optimization technique had the lowest mean square error for all the methods evaluated. It produced synthetic images with the lowest mean square error for the three reconstructions using 8, 16, and 24 reference views. While this approach produced the best results, it did require more time to run than any of the other methods except for the level set algorithm. If the desired application is new view synthesis, the cost of volumetrically optimizing the surface may well be worth the amount of time it takes for the algorithm to execute.

As with the 3D error, the methods like GVC that utilize the color information of the reference views produce better results than those that reconstruct the visual hull. The IBVH and IBPH methods again are orders of magnitude faster than competing methods, as seen in Figure 80, and are best suited to realtime applications. The level set method does encounter problems with surfaces that have sharp corners and edges, especially those that project to photo-consistent, but different colors in the reference views. Finally, we note that the view-dependent texture mapping approach does not perform well with a large (greater than 10) number of reference views.

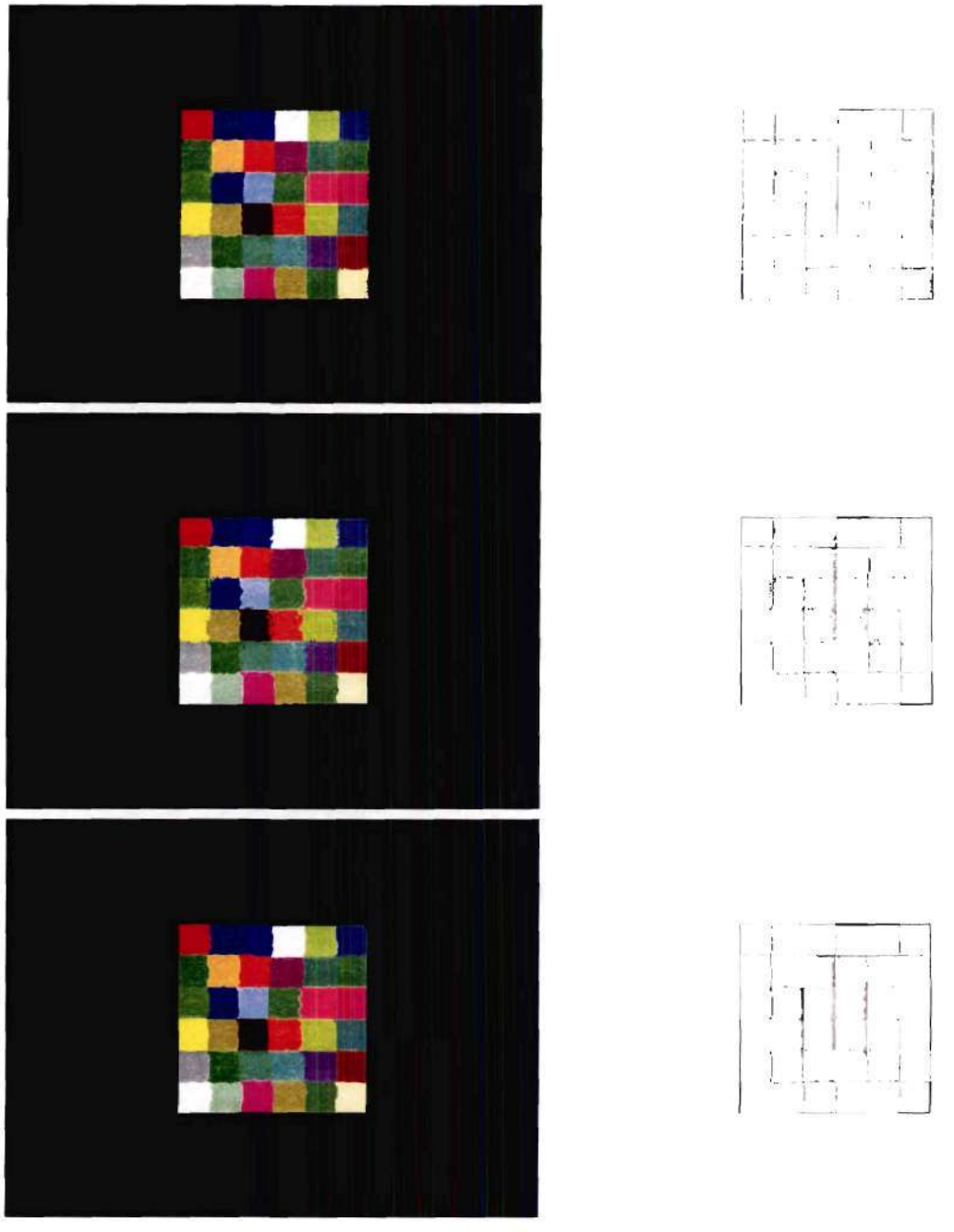


Figure 79: New view synthesis results for the IBPH algorithm. From top to bottom: using 8, 16, and 24 reference views. The synthesized views are on the left, and the corresponding squared error images are on the right.

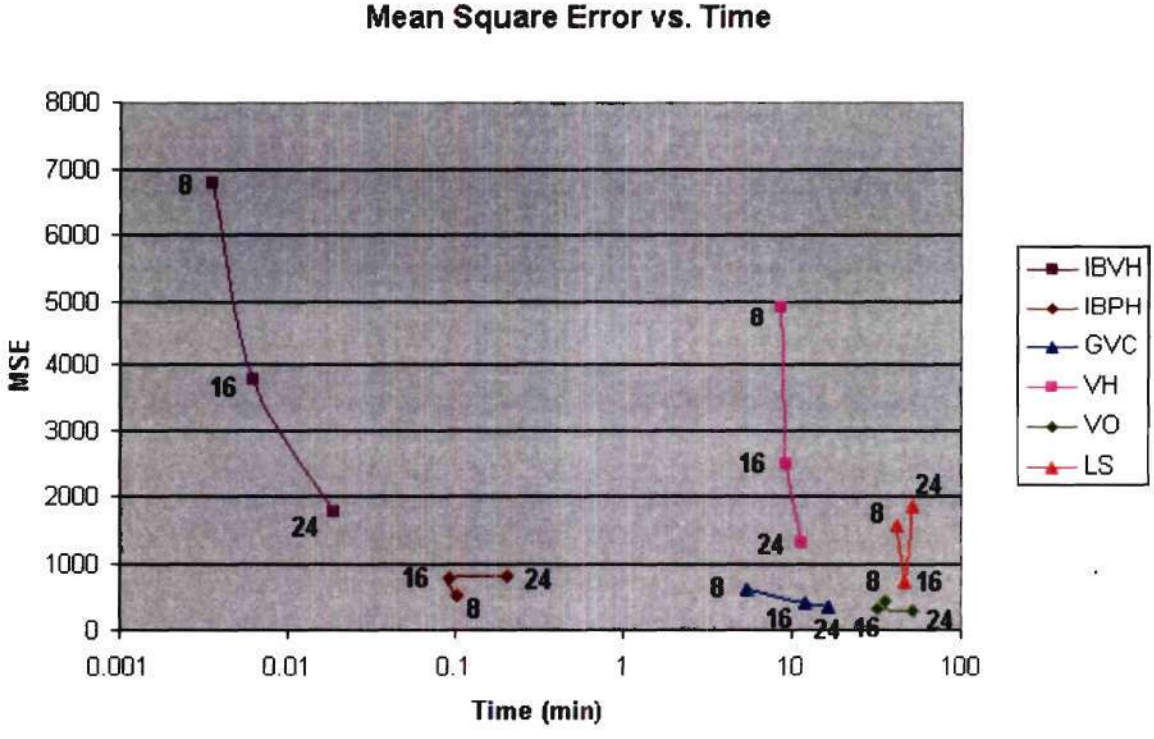


Figure 80: Mean square error vs. time, SynthPlane scene. Note that time is plotted on a logarithmic scale. The figure shows results using 8, 16, and 24 reference views for the image-based visual hulls (IBVH), image-based photo hulls (IBPH), generalized voxel coloring (GVC), visual hull (VH), volumetric optimization (VO), and level set (LS) reconstruction algorithms.

Approach	No. ref. views	E_{2D}	Time (m:s)
Visual Hull	8	4889	08:28
GVC	8	596.3	05:24
Volumetric optimization	8	428.8	35:28
Level sets	8	1568	41:51
IBVH	8	6770	00:0.217
IBPH	8	522.3	00:6.21
Visual Hull	16	2484	09:12
GVC	16	405.5	12:03
Volumetric optimization	16	329.1	32:07
Level sets	16	722	46:06
IBVH	16	3776	00:0.377
IBPH	16	773.3	00:5.55
Visual Hull	24	1299	11:26
GVC	24	360.2	16:27
Volumetric optimization	24	300.5	51:19
Level sets	24	1854	51:17
IBVH	24	1778	00:1.16
IBPH	24	807.1	00:12.3

Table 8: Mean square error analysis results.

CHAPTER IX

CONCLUSION

In this thesis we have presented novel 3D scene reconstruction methods for new view synthesis. The goal of these methods is to compute the photo hull, which is the tightest possible bound on the shape of the true scene that can be inferred from N photographs. Rendering the photo hull (or the visible part of it in IBPH) often produces photo-realistic new views of the scene.

9.1 Review of contributions

The generalized voxel coloring algorithm presented in Chapter 3 introduces two new methods for computing visibility during a voxel-based reconstruction. The GVC-IB algorithm uses less memory, while the GVC-LDI algorithm always uses up-to-date visibility and minimizes photo-consistency checks. GVC was the first space carving approach to solve the visibility problem exactly, which has been shown to produce superior results [25, 114] compared to algorithms that approximate scene visibility. The GVC-LDI algorithm was also the first 3D scene reconstruction approach to use layered depth images.

In Chapter 4 we presented various post-processing methods to refine space carving reconstructions. Simple morphological filtering was demonstrated to improve model fidelity. We then introduced a volumetric optimization approach that minimizes reprojection error. This approach was the first space carving method to carve the model below the global threshold, and to evaluate modifications to the surface on a per scene, rather than per voxel basis.

We introduced a volumetric warping algorithm in Chapter 5 that warps the voxel space

so that infinitely large scene can be modeled with a finite number of voxels. We demonstrated that such a technique is useful for reconstructing surfaces very far away from the cameras, in addition to a foreground scene.

The level set approach to space carving discussed in Chapter 6 represented the surface implicitly in the voxel space. In this framework, surface smoothing can naturally be incorporated into the 3D reconstruction instead of applied as a post-processing step.

The image-based photo hulls algorithm of Chapter 7 adopted a view-dependent approach to reconstructing only the portion of the computed photo hull that is visible to a virtual camera being moved about the scene. This was the first algorithm to demonstrate interactive reconstruction and view synthesis of the computed photo hull for nontrivial resolutions.

Additionally, the multi-view triangulation approach and analysis in Appendix B is a novel contribution in this thesis.

9.2 Future work

Significant progress on the problem of 3D reconstruction from multiple views has been made. Despite these developments, many techniques fall somewhat short of their ultimate goal of generating photo-realistic new views of arbitrary scenes.

One area of future work is the reconstruction of non-Lambertian scenes. The Lambertian assumption commonly made in reconstruction algorithms simplifies the problem at the expense of limiting the class of scenes that can be reconstructed. Clearly though, real surfaces interact with light in complex ways, producing view-dependent effects such as specularities and reflections. Thus, more sophisticated modeling of the bidirectional reflectance distribution function will be required to improve the flexibility of existing reconstruction algorithms. Some work on this problem has started to emerge in the literature [15, 23, 64].

Future work is needed to fully automate scene reconstruction. Perhaps the most significant obstacle in achieving this goal is camera calibration. For the space carving algorithms

described in this thesis, one calibrates the cameras in an often tedious pre-process before reconstruction. Self-calibration [85, 133] methods have appeared in the literature, but have been implemented in short baseline approaches that use a single camera moving about the scene. Future research is needed for accurate, wide baseline multi-view self-calibration.

Reconstruction from multi-view video poses new research opportunities and applications. Incorporating motion constraints into the reconstruction algorithm can improve reconstruction quality in addition to compute nonrigid motion [15, 126]. Viewpoint control of dynamic scenes could find application in a variety of settings such as interactive television and virtual reality. Future research in this area includes 4D video compression (3D model + time) and model-based coding. The ability to reason about moving objects in three dimensions, rather than their projection into two, could benefit many video-based applications, such as motion estimation, tracking, and automatic target recognition.

APPENDIX A

CAMERA GEOMETRY

This appendix provides a brief description of the geometry of single and multiple views. The amount of material on the subject is vast; our intent here is to present only the basic concepts that are relevant to the techniques described in this thesis. For an in-depth treatise, please refer to Hartley and Zisserman's [48] or Faugeras and Luong's [37] textbook.

We begin with a discussion of a single view's projection matrix, which is used by all of the techniques in this thesis. We then present multi-view concepts of bundle adjustment and epipolar geometry.

A.1 A camera's projection matrix

A camera's projection matrix H is a 3 x 4 matrix that describes how a point $\mathbf{P} = (X, Y, Z)$ in 3D space projects to a point $\mathbf{p} = (x, y)$ in an image. Using a homogeneous representation for $\mathbf{P}$ and $\mathbf{p}$, the relationship is

$$\mathbf{p} = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = H\mathbf{P} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{39}$$

where s is a scale factor. Since H is known only up to an arbitrary scale factor, it has 11 degrees of freedom, and can be computed from six known point correspondences $\mathbf{P} \leftrightarrow \mathbf{p}$ using one of a number of different techniques [123, 124, 132].

The projection matrix can be composed using the camera calibration. Correspondingly, if known, the projection matrix can be decomposed [48] to reveal the camera calibration

parameters. The relationship is

$$H = K[R \quad | -R\mathbf{T}]. \tag{40}$$

The matrix K in Equation 40 is a 3 x 3 matrix

$$K = \begin{bmatrix} \frac{f}{d_x} & 0 & C_x \\ 0 & \frac{f}{d_y} & C_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{41}$$

composed of the intrinsic camera parameters, which are physically measurable quantities that do not vary as the camera is moved in space. They are:

- d_x: Horizontal size of a pixel
- d_y: Vertical size of a pixel
- f: Focal length
- C_x: Horizontal center of image
- C_y: Vertical center of image

Here we assume that the camera has no skew, i.e., pixels have a square or rectangular shape, which is typically the case for modern cameras.

The matrix R in Equation 40 is a 3 x 3 rotation matrix that describes the orientation of the camera in 3D space. R has three degrees of freedom, and can be described by three rotations (ψ, θ, ϕ) about the x, y, and z axes, respectively, as

$$R = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix}. \tag{42}$$

Finally, $\mathbf{T} = [T_x, T_y, T_z]^T$ in Equation 40 is the position of the camera's center of projection in world coordinates. The six extrinsic camera parameters, (ψ, θ, ϕ) and $\mathbf{T}$ specify the camera pose, and vary as the camera is moved in space.

A.2 Bundle adjustment

Calibrating each camera individually using point correspondences $\mathbf{P} \leftrightarrow \mathbf{p}$ can achieve acceptable calibration results. However, *bundle adjustment* [122] is often used to refine the calibration to achieve a jointly optimal solution over all reference views. Bundle adjustment can also used to refine the 3D points if desired.

Bundle adjustment assumes that a set of 3D points $\mathbf{P}_j$ are viewed by cameras with projection matrices H_i, and that these quantities are approximately known. Given the set of image coordinates $\mathbf{p}_{ij}$ to which $\mathbf{P}_j$ projects in the ith image, bundle adjustment refines the projection matrices H_i and/or the 3D points $\mathbf{P}_j$ to minimize the image-space error

$$\sum_{ij} d(H_i \mathbf{P}_j, \mathbf{p}_{ij})^2, \tag{43}$$

where $d(\mathbf{x}, \mathbf{y})$ is a geometric image distance measure. This procedure is called bundle adjustment since it adjusts the bundle of rays between each camera center and the set of 3D points.

Bundle adjustment relies on gradient descent subject to constraints on the camera parameters, and is typically implemented using the Levenberg-Marquardt algorithm [89]. The optimization is subject to local minima, so the technique requires a reasonably accurate initialization. For a large number of 3D points and cameras, the procedure can be computationally intensive; however, Triggs et al. [122] describe methods that reduce computational cost by taking advantage of the mathematical structure of the problem.

A.3 Epipolar geometry

Epipolar geometry is the intrinsic projective geometry between two views. It is used extensively in the IBPH algorithm of Chapter 7. Epipolar geometry is independent of the scene structure, and depends only on the cameras' intrinsic parameters and relative pose.

Consider a 3D point $\mathbf{P}$ that is imaged by two different viewpoints, and projects to points

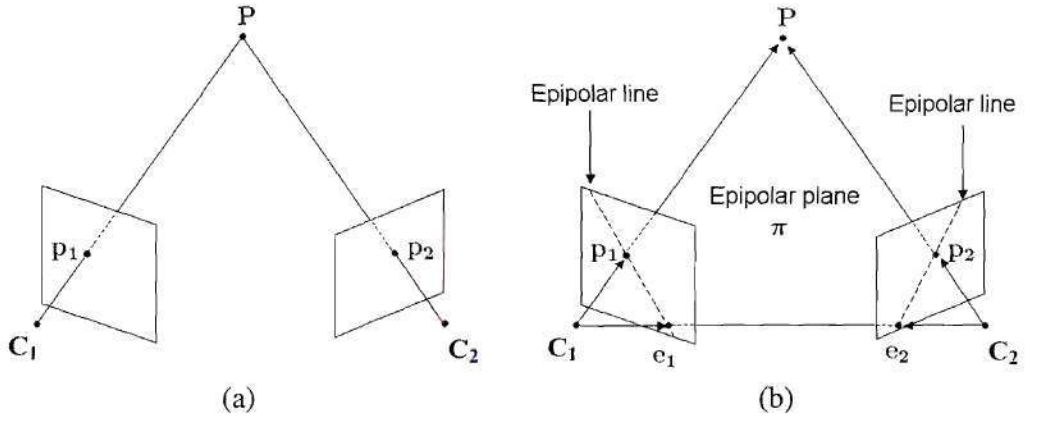


Figure 81: Epipolar geometry. In (a), a point **P** is imaged by two cameras. In (b), the epipolar plane going through the point **P** and the camera centers $\mathbf{C_1}$ and $\mathbf{C_2}$ is formed.

$\mathbf{p_1}$ and $\mathbf{p_2}$ as shown in Figure 81 (a). What are the geometric constraints on the two corresponding points $\mathbf{p_1}$ and $\mathbf{p_2}$? Consider the plane π defined by **P** and the center of projection for each camera, $\mathbf{C_1}$ and $\mathbf{C_2}$, shown in Figure 81 (b). It is clear from the figure that rays back-projected from $\mathbf{p_1}$ and $\mathbf{p_2}$ intersect at **P**, and that the rays lie in the *epipolar plane* π.

Now consider the left reference view. The epipolar plane cuts through the image plane, forming an *epipolar line* in Figure 81 (b). The epipolar line goes through the image point $\mathbf{p_1}$ and the *epipole* $\mathbf{e_1}$, which is the projection of camera 2's center of projection into the image plane of camera 1. Regardless of the location of the 3D point P in space, the epipole remains fixed. Symmetric results hold for the right image.

This epipolar geometry is very useful when searching for correspondences between views, as shown in Figure 82. Given the image point $\mathbf{p_1}$, one can back-project a ray from the center of projection $\mathbf{C_1}$ and into 3D space. In image 2, this ray projects to the epipolar line, along which the correspondence must lie. Thus, the epipolar geometry restricts the search for the correspondence to a one-dimensional region along the epipolar line.

The epipolar geometry is typically encoded into a 3 x 3 matrix F called the *fundamental matrix*. The fundamental matrix can be estimated from image correspondences, or computed from the camera calibration of the two views if known. We do the latter, since our

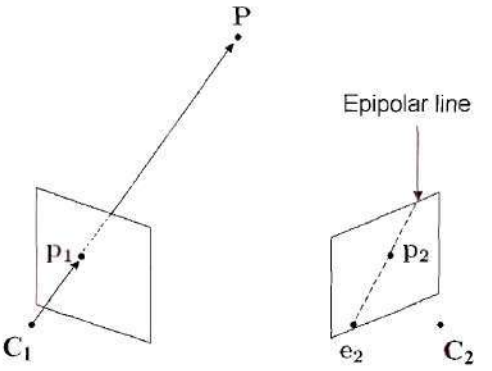


Figure 82: Correspondence search using epipolar geometry.

cameras are calibrated, and determine F as

$$F = K_2^{-T}[\mathbf{T}_\times]RK_1^{-1}, \tag{44}$$

where K_1 and K_2 are the intrinsic camera matrices, R is a rotation matrix specifying the relative change in orientation between the two views, and $[\mathbf{T}_\times]$ is a 3 x 3 skew-symmetric matrix formed from the relative translation (T_x, T_y, T_z) between the two views as

$$[\mathbf{T}_\times] = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix}. \tag{45}$$

Recall that the correspondence for a point $\mathbf{p}_1$ from image 1 must lie along the epipolar line in image 2. This epipolar line is very easily computed using the fundamental matrix as $F\mathbf{p}_1$. For points $\mathbf{x}$ on the epipolar line, including the correspondence $\mathbf{p}_2$,

$$\mathbf{x}^T F\mathbf{p}_1 = 0. \tag{46}$$

The *trifocal tensor* and *quadrifocal tensor* are similar geometrical entities that relate sets of three and four views, respectively. In this thesis we make use only of epipolar geometry, so we conclude our discussion of multi-view geometry here.

APPENDIX B

MULTI-VIEW TRIANGULATION

B.1 Introduction

Often in computer vision, one must triangulate corresponding points between two images in order to compute depth. Because of errors in quantization, camera calibration, and correspondence, rays back-projected from the images into three-dimensional space rarely intersect. As a result, one must find a point that is optimally close to the two rays. This appendix generalizes this problem for the case of multiple images; i.e., how to compute the optimal position of three-dimensional point given rays emanating from corresponding pixels in N images, where $N \geq 2$. We develop a simple, closed-form technique with a linear solution and provide an example to demonstrate the method. To our knowledge, this technique is novel.

B.2 Computing depth using two views

We begin by considering a common method [123] for stereo triangulation. Here, we assume that the camera parameters are known, and that the projections, $\mathbf{P}_1$ and $\mathbf{P}_2$ of a 3D point into two images, I_1 and I_2 are known, as shown in Figure 83. Points $\mathbf{P}_1$ and $\mathbf{P}_2$ are called a *correspondence*, as they both correspond to the same point $\mathbf{P}$ in 3D space. Our task is to compute the location of point $\mathbf{P}$, given $\mathbf{P}_1$, $\mathbf{P}_2$, and the camera parameters.

Figure 83 shows that for each camera, we can back-project a ray from the camera center $\mathbf{C}_i$, through the image pixel of the correspondence, forming a ray $\hat{\mathbf{d}}_i$ into 3D space. Ideally, these rays would intersect exactly at the same 3D point. However, since the camera parameters and correspondence locations in image space are known only approximately, the rays will not actually intersect in 3D space. So instead, we seek to find a 3D point that

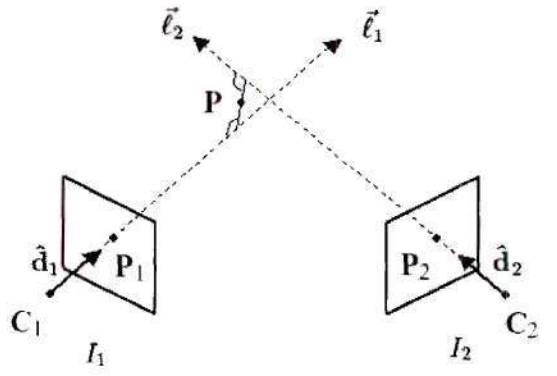


Figure 83: A common method for stereo triangulation.

has minimal distance from both rays. This point will be located on a line segment that is orthogonal to the rays, as shown in Figure 83. A standard approach then, first computes the endpoints of this line segment. From these, one computes the midpoint **P** of the line segment. Point **P** is the point in 3D space that is optimally close to the two non-intersecting rays.

This approach works well for two-view triangulation. However, what if one has a correspondence visible in N views, for $N > 2$? One could triangulate the correspondence between M different pairs of views, possibly all $\begin{pmatrix} N \\ 2 \end{pmatrix}$ pairs, resulting in M estimates of the point **P**. These estimates could then be combined to produce a single value for the point P. While this would result in a reasonable solution, there are a few drawbacks to this approach. First, the triangulation algorithm would be executed M times, once for each pair of views chosen. If all pairs of views are chosen, this requires executing the triangulation algorithm $O(N^2)$ times. For large N this can be computationally intensive. Perhaps even more significant is that the combination of these pairwise results does not guarantee a solution that is optimal in the sense of having minimum distance to all rays.

In the next section, we describe a simple technique that executes in $O(N)$ time and additionally finds a point **P** that is optimally close to all N rays.

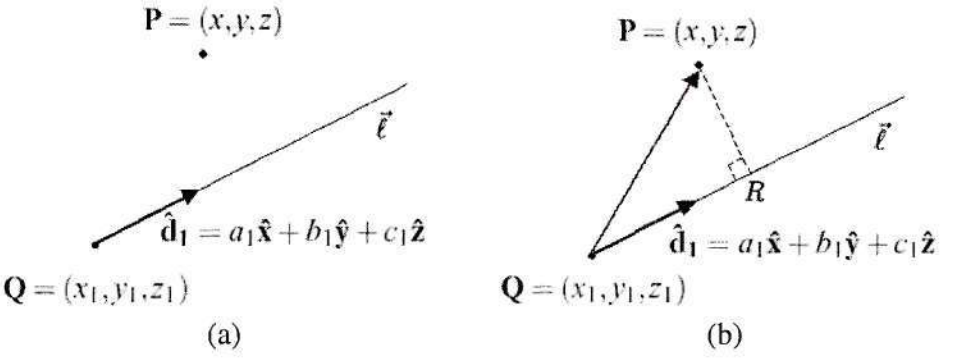


Figure 84: Finding the shortest distance between a point and a ray.

B.3 Computing depth using N views

B.3.1 The distance between a point and a ray

Consider Figure 84(a), which shows a point $\mathbf{P} = (x,y,z)$ and a ray $\vec{\ell}$ that starts at point $\mathbf{Q} = (x_1,y_1,z_1)$ and has a normalized direction $\hat{\mathbf{d}}_1 = a_1\hat{\mathbf{x}} + b_1\hat{\mathbf{y}} + c_1\hat{\mathbf{z}}$. Our task in this subsection is to derive an equation for the distance between the point $\mathbf{P}$ and the ray $\vec{\ell}$. Intuitively, this distance is along a line that goes through $\mathbf{P}$ and is orthogonal to $\vec{\ell}$. In Figure 84(b), this distance is represented by the length of vector **RP**.

The vector **QR** is the projection of **QP** onto the ray $\vec{\ell}$. From the figure, we note that

$$\mathbf{QP} = (x-x_1)\hat{\mathbf{x}} + (y-y_1)\hat{\mathbf{y}} + (z-z_1)\hat{\mathbf{z}}, \tag{47}$$

and this vector has a length

$$||\mathbf{QP}|| = \sqrt{(x-x_1)^2 + (y-y_1)^2 + (z-z_1)^2}. \tag{48}$$

QR, the projection of **QP** onto $\hat{\mathbf{d}}_1$ is

$$\begin{aligned} \mathbf{QR} &= (\mathbf{QP}\cdot\hat{\mathbf{d}}_1)\,\hat{\mathbf{d}}_1 && (49)\\ &= [a_1(x-x_1) + b_1(y-y_1) + c_1(z-z_1)]\,\hat{\mathbf{d}}_1 && (50) \end{aligned}$$

Since $\hat{\mathbf{d}}_1$ has unit magnitude, the length of **QR** is then

$$||\mathbf{QR}|| = a_1(x-x_1) + b_1(y-y_1) + c_1(z-z_1). \tag{51}$$

Our goal is find the length of the vector **RP**. Since the points **PQR** form a right triangle, we can invoke the Pythagorean thereom,

$$||\mathbf{QP}||^2 = ||\mathbf{QR}||^2 + ||\mathbf{RP}||^2, \tag{52}$$

or

$$||\mathbf{RP}||^2 = ||\mathbf{QP}||^2 - ||\mathbf{QR}||^2 \tag{53}$$

Substituting in values gives

$$||\mathbf{RP}||^2 = (x-x_1)^2 + (y-y_1)^2 + (z-z_1)^2 - [a_1(x-x_1) + b_1(y-y_1) + c_1(z-z_1)]^2 \tag{54}$$

Thus,

$$||\mathbf{RP}|| = \sqrt{(x-x_1)^2 + (y-y_1)^2 + (z-z_1)^2 - [a_1(x-x_1) + b_1(y-y_1) + c_1(z-z_1)]^2} \tag{55}$$

Equation 55 is the distance between a point $\mathbf{P} = (x,y,z)$ and a ray $\vec{\ell}$ that starts at point $\mathbf{Q} = (x_1,y_1,z_1)$ and has a normalized direction $\hat{\mathbf{d}}_1 = a_1\hat{\mathbf{x}} + b_1\hat{\mathbf{y}} + c_1\hat{\mathbf{z}}$.

B.3.2 Minimizing the sum of squared distance

To compute the total distance, $D(x,y,z)$ between a point and N rays, Equation 55 is evaluated for each ray i and the results are summed. This yields an equation of the form

$$D(x,y,z) = \sum_{i=1}^{N} \sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2 - [a_i(x-x_i) + b_i(y-y_i) + c_i(z-z_i)]^2}. \tag{56}$$

One might try to analytically optimize $D(x,y,z)$. However, because of the square root, the solution is nonlinear. So instead, we optimize the sum of squared distances,

$$E(x,y,z) = \sum_{i=1}^{N} (x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2 - [a_i(x-x_i) + b_i(y-y_i) + c_i(z-z_i)]^2, \tag{57}$$

which is a sum of terms similar to Equation 54, one for each ray. Thus, our goal is to find a point that globally minimizes $E(x,y,z)$.

B.3.3 Derivation of the optimal point

To find the optimal point, we differentiate $E(x,y,z)$, set the partial derivatives to zero, and evaluate the critical points.

$$\begin{aligned}
\frac{\partial E(x,y,z)}{\partial x} &= \sum_{i=1}^{N} \{2(x-x_i) - 2a_i[a_i(x-x_i)+b_i(y-y_i)+c_i(z-z_i)]\} = 0 \\
\frac{\partial E(x,y,z)}{\partial y} &= \sum_{i=1}^{N} \{2(y-y_i) - 2b_i[a_i(x-x_i)+b_i(y-y_i)+c_i(z-z_i)]\} = 0 \\
\frac{\partial E(x,y,z)}{\partial z} &= \sum_{i=1}^{N} \{2(z-z_i) - 2c_i[a_i(x-x_i)+b_i(y-y_i)+c_i(z-z_i)]\} = 0
\end{aligned}$$

We seek the optimal point (x,y,z) that satisfies the above equations. Expanding the terms and dividing by 2 yields

$$\sum_{i=1}^{N} \left[x - x_i - a_i^2 x + a_i^2 x_i - a_i b_i y + a_i b_i y_i - a_i c_i z + a_i c_i z_i\right] = 0 \tag{58}$$

$$\sum_{i=1}^{N} \left[y - y_i - a_i b_i x + a_i b_i x_i - b_i^2 y + b_i^2 y_i - b_i c_i z + b_i c_i z_i\right] = 0 \tag{59}$$

$$\sum_{i=1}^{N} \left[z - z_i - a_i c_i x + a_i c_i x_i - b_i c_i y + b_i c_i y_i - c_i^2 z + c_i^2 z_i\right] = 0 \tag{60}$$

Placing the terms involving (x_i, y_i, z_i) on the right side of the equation gives

$$\sum_{i=1}^{N} \left[(1-a_i^2)x - a_i b_i y - a_i c_i z\right] = \sum_{i=1}^{N} \left[(1-a_i^2)x_i - a_i b_i y_i - a_i c_i z_i\right] \tag{61}$$

$$\sum_{i=1}^{N} \left[-a_i b_i x + (1-b_i^2)y - b_i c_i z\right] = \sum_{i=1}^{N} \left[-a_i b_i x_i + (1-b_i^2)y_i - b_i c_i z_i\right] \tag{62}$$

$$\sum_{i=1}^{N} \left[-a_i c_i x - b_i c_i y + (1-c_i^2)z\right] = \sum_{i=1}^{N} \left[-a_i c_i x_i - b_i c_i y_i + (1-c_i^2)z_i\right] \tag{63}$$

Next, we write this in matrix form

$$\begin{bmatrix} \sum_i (1-a_i^2) & -\sum_i a_i b_i & -\sum_i a_i c_i \\ -\sum_i a_i b_i & \sum_i (1-b_i^2) & -\sum_i b_i c_i \\ -\sum_i a_i c_i & -\sum_i b_i c_i & \sum_i (1-c_i^2) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \sum_i \left[(1-a_i^2)x_i - a_i b_i y_i - a_i c_i z_i\right] \\ \sum_i \left[-a_i b_i x_i + (1-b_i^2)y_i - b_i c_i z_i\right] \\ \sum_i \left[-a_i c_i x_i - b_i c_i y_i + (1-c_i^2)z_i\right] \end{bmatrix} \tag{64}$$

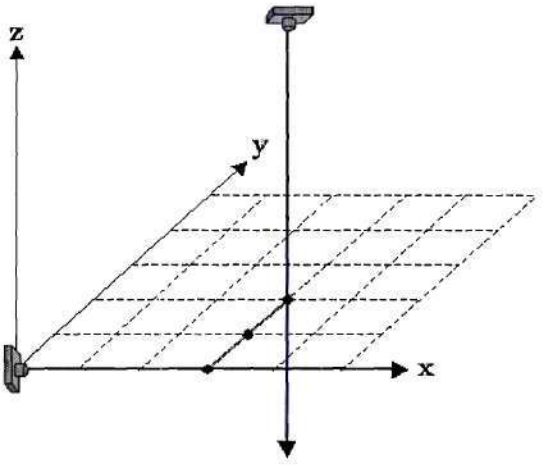


Figure 85: An example showing our multi-view triangulation algorithm.

This expression is of the form $A\mathbf{x} = \mathbf{b}$. For each ray i, the starting point of the ray (x_i, y_i, z_i) and the ray direction $\hat{\mathbf{d}}_i = a_i\hat{\mathbf{x}} + b_i\hat{\mathbf{y}} + c_i\hat{\mathbf{z}}$ are known. Thus, all the terms in the matrix A and the vector $\mathbf{b}$ are known. We compute these matrices, and solve for $\mathbf{x}$,

$$\mathbf{x} = A^{-1}\mathbf{b}. \tag{65}$$

The point $\mathbf{x}$ then, is the point that is is closest to all of the rays in the sense of minimizing the sum of squared distance.

B.4 Example

In this section, we consider a simple example that shows how this approach works for triangulation. While this example computes the optimal point using two images, the approach works for an arbitrary number of images.

Suppose we have two cameras as shown in Figure 85. Camera 1 is centered at the origin, so $(x_1, y_1, z_1) = (0,0,0)$. Camera 2 is centered at the point $(x_2, y_2, z_2) = (3,2,5)$. A correspondence is found in the two images, and in each image a ray is back-projected from the camera center through the pixel in the image plane, as shown in the figure. The ray from camera 1 has a direction $(a_1, b_1, c_1) = (1,0,0)$, parallel to the x-axis, and the ray from

camera 2 has a direction $(a_2, b_2, c_2) = (0, 0, -1)$, parallel to the z-axis. Our goal is to find the point in 3D space that is closest to both rays. By inspection, we expect the solution to this problem to be $(3, 1, 0)$.

Using the matrix equation in the previous section, we get

$$\begin{bmatrix} 1-a_1^2+1-a_2^2 & -a_1b_1-a_2b_2 & -a_1c_1-a_2c_2 \\ -a_1b_1-a_2b_2 & 1-b_1^2+1-b_2^2 & -b_1c_1-b_2c_2 \\ -a_1c_1-a_2c_2 & -b_1c_1-b_2c_2 & 1-c_1^2+1-c_2^2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} =$$

$$\begin{bmatrix} (1-a_1^2)x_1 - a_1b_1y_1 - a_1c_1z_1 + (1-a_2^2)x_2 - a_2b_2y_2 - a_2c_2z_2 \\ -a_1b_1x_1 + (1-b_1^2)y_1 - b_1c_1z_1 - a_2b_2x_2 + (1-b_2^2)y_2 - b_2c_2z_2 \\ -a_1c_1x_1 - b_1c_1y_1 + (1-c_1^2)z_1 - a_1c_1x_2 - b_1c_1y_2 + (1-c_1^2)z_2 \end{bmatrix} \tag{66}$$

Plugging in known values gives

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 0 \end{bmatrix} \tag{67}$$

which yields the correct solution of

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 0 \end{bmatrix} \tag{68}$$

B.5 Analysis

B.5.1 When Is a unique solution not possible?

Equation 65 shows that if the matrix A is invertible, then a unique solution exists. In this subsection we determine under what circumstances A becomes singular, resulting in no unique solution. To do this, we first compute the determinant of A.

$$|A| = \begin{vmatrix} \sum_i (1-a_i^2) & -\sum_i a_i b_i & -\sum_i a_i c_i \\ -\sum_i a_i b_i & \sum_i (1-b_i^2) & -\sum_i b_i c_i \\ -\sum_i a_i c_i & -\sum_i b_i c_i & \sum_i (1-c_i^2) \end{vmatrix} \tag{69}$$

Each element of A is a sum from $i = 1 \cdots N$. Evaluating and simplifying the terms in this determinant for general N is rather challenging. Using the constraint $a_i^2 + b_i^2 + c_i^2 = 1$, with some work it is possible to show that this determinant can be rewritten as

$$\begin{aligned} |A| \;=\; & \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1\\ j\neq i}}^{N}\sum_{\substack{k=1\\ k\neq i,j}}^{N}(b_ic_j - b_jc_i)^2(1-a_k^2) + \\ & \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1\\ j\neq i}}^{N}\sum_{\substack{k=1\\ k\neq i,j}}^{N}(a_ic_j - a_jc_i)^2(1-b_k^2) + \\ & \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1\\ j\neq i}}^{N}\sum_{\substack{k=1\\ k\neq i,j}}^{N}(a_ib_j - a_jb_i)^2(1-c_k^2) + \\ & \sum_{i=1}^{N}\sum_{\substack{j=1\\ j\neq i}}^{N}\sum_{\substack{k=1\\ k\neq i,j}}^{N}(a_ib_jc_k - a_kb_ic_j)^2 \end{aligned} \tag{70}$$

A unique solution to this problem cannot be found when Equation 70 equals zero. $|A|$ is the sum of nonnegative expressions, as each ray direction is normalized. Thus, the only way Equation 70 can equal zero is if each expression evaulates to zero. This fact leads to the following theorem:

Theorem 1 *A unique solution exists except when all the rays are either parallel or anti-parallel. That is,*

$$\hat{\mathbf{d}}_i = \begin{cases} a\hat{\mathbf{x}} + b\hat{\mathbf{y}} + c\hat{\mathbf{z}} \\ \quad or \\ -a\hat{\mathbf{x}} - b\hat{\mathbf{y}} - c\hat{\mathbf{z}} \end{cases} \tag{71}$$

Proof: To prove Theorem 1, we must show that each term in Equation 70 equals zero only when all the rays are parallel or anti-parallel. The only way the expression $(1-a_k^2)$ can be zero is if each normalized ray direction is either $(1,0,0)$ or $(-1,0,0)$, i.e., all the rays are parallel or anti-parallel. A similar result holds for the $(1-b_k^2)$ and $(1-c_k^2)$ expressions in Equation 70. For the expressions $(b_ic_j - b_jc_i)$, $(a_ic_j - a_jc_i)$, and $(a_ib_j - a_jb_i)$ to be zero,

we must have

$$b_i c_j = b_j c_i \tag{72}$$

$$a_i c_j = a_j c_i \tag{73}$$

$$a_i b_j = a_j b_i \tag{74}$$

Using the fact that $a_i^2 + b_i^2 + c_i^2 = 1$, we can rewrite Equation 74 as

$$a_i b_j = a_j b_i \tag{75}$$

$$a_i \sqrt{1 - a_j^2 - c_j^2} = a_j \sqrt{1 - a_i^2 - c_i^2} \tag{76}$$

$$a_i^2 (1 - a_j^2 - c_j^2) = a_j^2 (1 - a_i^2 - c_i^2) \tag{77}$$

$$a_i^2 - a_i^2 c_j^2 = a_j^2 - a_j^2 c_i^2 \tag{78}$$

Since $(a_i c_j - a_j c_i)^2 = 0$, we know that $a_i^2 c_j^2 = 2 a_i a_j c_i c_j - a_j^2 c_i^2$. We substitute this into Equation 78, yielding

$$a_i^2 - 2 a_i a_j c_i c_j + a_j^2 c_i^2 = a_j^2 - a_j^2 c_i^2 \tag{79}$$

$$a_i^2 + 2 a_j^2 c_i^2 = a_j^2 + 2 a_i a_j c_i c_j \tag{80}$$

Using Equation 73, we get

$$a_i^2 = a_j^2 \tag{81}$$

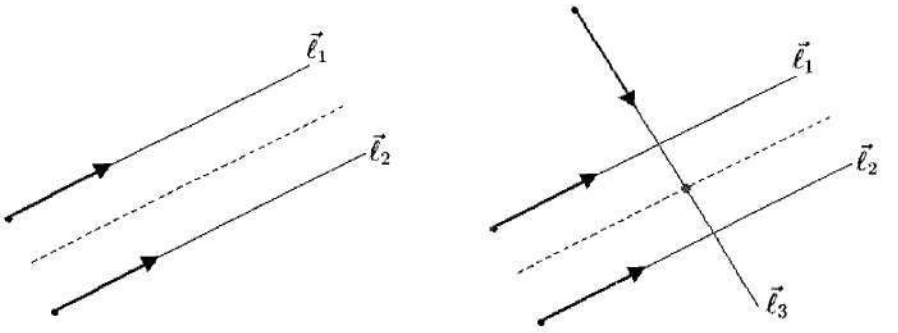
Similar results hold for b and c, resulting in

$$a_i = \pm a_j \tag{82}$$

$$b_i = \pm b_j \tag{83}$$

$$c_i = \pm c_j \tag{84}$$

In order for Equations 72 to 74 to be satisfied, the signs on the above three equations must either all be positive or negative. Thus, the first three terms of Equation 70 are zero if and only if all the rays are parallel or anti-parallel.



(a) (b)

Figure 86: No unique solution exists when all rays are parallel or anti-parallel. In (a), any point that is along the line that is equidistant from the parallel rays will be optimally close to the two rays. However, if just one ray is not parallel, as in (b), a unique solution exists, depicted with a gray dot.

The fourth term of Equation 70 can be re-expressed to look one of the Equations 72 to 74. For example, using the fact that $a_i b_j = a_j b_i$,

$$(a_i b_j)c_k = a_k b_i c_j \tag{85}$$

$$(a_j b_i)c_k = a_k b_i c_j \tag{86}$$

$$a_j c_k = a_k c_j \tag{87}$$

Thus, the fourth term of Equation 70 is zero when the other three terms of Equation 70 are zero, namely when all the rays are parallel or anti-parallel. This concludes the proof. $\triangle$

Theorem 1 is consistent with one's intuition. When all the rays are parallel (or anti-parallel), one would not expect that a unique solution exists, as shown in Figure 86 (a). In this case, any point along a line that is equidistant from the parallel rays will be a valid solution. This equidistant line is depicted with a dotted line. However, if at least one of the rays is not parallel to the others, then a unique solution exists, as shown in in Figure 86 (b).

B.5.2 Discussion of extremum

In the derivation above, we found an extremal point of the sum of squared distance $E(x, y, z)$. By nature of the problem, one can assume that this extremal point is a minimum and not

a maximum. For the finicky reader, we prove in this subsection that the extremal point is indeed a minimum.

Theorem 2 *The extremal point found by this method minimizes the sum of squared distance.*

Proof: To show that the extremal point is indeed a minimum, we must analyze the matrix of second partial derivatives [5],

$$D = \begin{bmatrix} \frac{\partial^2 E}{\partial x^2} & \frac{\partial^2 E}{\partial x \partial y} & \frac{\partial^2 E}{\partial x \partial z} \\ \frac{\partial^2 E}{\partial x \partial y} & \frac{\partial^2 E}{\partial y^2} & \frac{\partial^2 E}{\partial y \partial z} \\ \frac{\partial^2 E}{\partial x \partial z} & \frac{\partial^2 E}{\partial y \partial z} & \frac{\partial^2 E}{\partial z^2} \end{bmatrix} = \begin{bmatrix} 2\sum_i (1-a_i^2) & -2\sum_i a_i b_i & -2\sum_i a_i c_i \\ -2\sum_i a_i b_i & 2\sum_i (1-b_i^2) & -2\sum_i b_i c_i \\ -2\sum_i a_i c_i & -2\sum_i b_i c_i & 2\sum_i (1-c_i^2) \end{bmatrix} \tag{88}$$

and show that the determinants of the upper-left 1x1 submatrix, upper-left 2x2 submatrix, and D are all strictly positive.

First, we compute the determinant of D_1, the upper-left 1x1 submatrix,

$$|D_1| = 2\sum_i (1 - a_i^2) \tag{89}$$

Since a_i is a component from a normalized vector, the value of each $(1-a_i^2)$ term must be nonnegative. Thus, D_1 must be nonnegative, since it is a sum of nonnegative terms. The only situation for which D_1 could be zero is if each $a_i = \pm 1$. In that case, all rays would be parallel or anti-parallel, with directions $(a_i, b_i, c_i) = (1,0,0)$ or $(a_i, b_i, c_i) = (-1,0,0)$. As shown earlier, we cannot expect a unique solution for this case, so we do not care about the extremal point. Thus, D_1 is strictly positive except when the all rays are parallel or anti-parallel.

Next, we compute the determinant of D_2, the upper-left 2x2 submatrix,

$$|D_2| = \begin{vmatrix} 2\sum_i (1-a_i^2) & -2\sum_i a_i b_i \\ -2\sum_i a_i b_i & 2\sum_i (1-b_i^2) \end{vmatrix} \tag{90}$$

Using the fact that $a_i^2 + b_i^2 + c_i^2 = 1$, this determinant can be expressed as

$$|D_2| = 4N \sum_{i=1}^{N} (c_i^2) + 2 \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} (a_i b_j - a_j b_i)^2 \tag{91}$$

Since D_2 is a sum of nonnegative terms, it must also be nonnegative. The only way D_2 can be zero is if all $c_i = 0$, and each $a_i b_j = a_j b_i$ for $i \neq j$. This can only happen when all the rays are parallel or anti-parallel, since

$$
\begin{aligned}
a_i b_j &= a_j b_i && (92)\\
\sqrt{1 - b_i^2 - c_i^2} b_j &= \sqrt{1 - b_j^2 - c_j^2} b_i && (93)\\
\sqrt{1 - b_i^2} b_j &= \sqrt{1 - b_j^2} b_i && (94)\\
(1 - b_i^2) b_j^2 &= (1 - b_j^2) b_i^2 && (95)\\
b_j^2 - b_i^2 b_j^2 &= b_i^2 - b_i^2 b_j^2 && (96)\\
b_i^2 &= b_j^2 && (97)
\end{aligned}
$$

Likewise, D_2 can only be zero for $a_i^2 = a_j^2$. This equation, along with Equations 92 and 97 are true only when the rays are parallel or anti-parallel. Thus, D_2 is strictly positive except when all rays are parallel or anti-parallel.

Finally, we must show that the determinant of D is nonnegative. Comparing equations 69 and 88, we note that $|D| = 8|A|$. In the previous section, we showed that the determinant of A was strictly positive except when all rays were parallel or anti-parallel. Therefore, this result also applies to the determinant of D. This concludes the proof. $\triangle$

Thus, the extremal point $\mathbf{P}$ found by our optimal ray intersection algorithm minimizes the sum of squared distance to each ray.

APPENDIX C

EXPERIMENTAL SETUP

In this appendix we describe our experimental setup used for interactive new view synthesis.

As Figure 87 shows, our lab is equipped with five Sony DFW-V500 digital video cameras positioned on an arc. These cameras are synchronized for simultaneous image acquisition. Synchronization is achieved by sending a short pulse out the parallel port of a computer to the external trigger input on each camera.

For applications that require maximum performance, we connect each camera via FireWire to a personal computer. These computers perform background subtractions on the incoming video. The background is attained by first collecting a short video clip of the scene before the object to be reconstructed is placed in front of the cameras. The segmented multi-view video is sent over a 100 Mbps switch to our server machine, which is a dual-processor Pentium 4 2.0 GHz machine. For each time instant, the server computes the image-based photo hull.

The image-based photo hull can be displayed on the server machine. Since the reconstruction is a 3D representation, it can be depth composited with synthetic objects using standard computer graphics methods. Our server machine is connected to Internet2. With this connection, we are able to perform 3D video-conferencing with remote sites. In particular, we are developing a 3D video-conferencing application with Hewlett-Packard Labs.

Figure 87: Our experimental setup for interactive new view synthesis.

REFERENCES

[1] ADELSON, E. H. and BERGEN, J. R., "The Plenoptic Function and the Elements of Early Vision," in *Computational Models of Visual Processing* (LANDY, M. and MOVSHON, J. A., eds.), Cambridge, MA: MIT Press, 1991.

[2] ALIAGA, D. G. and CARLBOM, I., "Plenoptic Stitching: A Scalable Method for Reconstructing 3D Interactive Walkthroughs," in *SIGGRAPH 2001, Computer Graphics Proceedings*, 2001.

[3] AVIDAN, S. and SHASHUA, A., "Novel View Synthesis by Cascading Trilinear Tensors," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 4, pp. 293–306, 1998.

[4] BAKER, H. H. and BINFORD, T. O., "Depth from Edge and Intensity Based Stereo," in *International Joint Conference on Artificial Intelligence*, pp. 631–636, 1981.

[5] BARTLE, R., *The Elements of Real Analysis*. New York: John Wiley and Sons, second ed., 1976.

[6] BECKER, S. and BOVE, V. M., "Semiautomatic 3-D Model Extraction from Uncalibrated 2-D Camera Views," *Proc. SPIE Visual Data Exploration and Analysis II*, vol. 2410, pp. 447–461, Feb. 1995.

[7] BEVERIDGE, J. R. and STEVENS, M., "Precise Matching of 3-D Target Models to Multisensor Data," *IEEE Transactions on Image Processing*, vol. 6, no. 1, pp. 126–142, 1997.

[8] BLACK, M. J. and RANGARAJAN, A., "On the Unification of Line Processes, Outlier Rejection, and Robust Statistics with Applications in Early Vision," *International Journal of Computer Vision*, vol. 19, no. 1, pp. 57–91, 1996.

[9] BLINN, J. and NEWELL, M., "Texture and Reflection on Computer Generated Images," *Communications of ACM*, vol. 19, no. 10, pp. 542–547, 1976.

[10] BLOOMENTHAL, J., *Introduction to Implicit Surfaces*. San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1997.

[11] BONET, J. D. and VIOLA, P., "Roxels: Responsibility Weighted 3D Volume Reconstruction," in *International Conference on Computer Vision*, pp. 415–425, 1999.

[12] BROADHURST, A., DRUMMOND, T. W., and CIPOLLA, R., "A Probabilistic Framework for Space Carving," in *International Conference on Computer Vision*, vol. 1, pp. 388–393, 2001.

[13] BROADHURST, A., *A Probabilistic Framework for Space Carving*. Ph.d. thesis, University of Cambridge, 2001.

[14] BUEHLER, C., BOSSE, M., MCMILLAN, L., GORTLER, S. J., and COHEN, M. F., "Unstructured Lumigraph Rendering," in *SIGGRAPH 2001, Computer Graphics Proceedings*, pp. 425–432, 2001.

[15] CARCERONI, R. and KUTULAKOS, K., "Multi-View Scene Capture by Surfel Sampling: From Video Streams to Non-Rigid Motion, Shape, and Reflectance," in *International Conference on Computer Vision*, vol. 2, pp. 60–67, 2001.

[16] CASTLEMAN, K. R., *Digital Image Processing*. Englewood Cliffs, NJ: Prentice Hall, 1996.

[17] CHANG, N. and ZAKHOR, A., "A Multivalued Representation for View Synthesis," in *International Conference on Image Processing*, pp. 505–509, 1999.

[18] CHEN, Q. and MEDIONI, G., "Image synthesis from a sparse set of views," in *Proceedings of the Conference on Visualization '97*, pp. 269–275, ACM Press, 1997.

[19] CHEN, Q. and MEDIONI, G., "A Volumetric Stereo Matching Method: Application to Image-Based Modeling," *Proc. Computer Vision and Pattern Recognition*, pp. 29–34, June 1999.

[20] CHEN, S. E. and WILLIAMS, L., "View Interpolation for Image Synthesis," in *SIGGRAPH 1993, Computer Graphics Proceedings*, pp. 279–288, 1993.

[21] CHEN, S. E., "QuickTime VR — An Image-Based Approach to Virtual Environment Navigation," in *SIGGRAPH 1995, Computer Graphics Proceedings*, pp. 29–38, 1995.

[22] CHEN, W., BOUGET, J., CHU, M., and GRZESZCZUK, R., "Light Field Mapping: Efficient Representation and Hardward Rendering of Surface Light Fields," in *SIGGRAPH 2002, Computer Graphics Proceedings*, 2002.

[23] CHHABRA, V., "Reconstructing Specular Objects with Image-based Rendering Using Color Caching," Master's thesis, Worcester Polytechnic Institute, 2001.

[24] COLOSIMO, A., SARTI, A., and TUBARO, S., "Image-Based Object Modeling: A Multiresolution Level-Set Approach," in *International Conference on Image Processing*, pp. 181–184, 2001.

[25] CULBERTSON, W. B., MALZBENDER, T., and SLABAUGH, G., "Generalized Voxel Coloring," in *Proceedings of the International Workshop on Vision Algorithms*, pp. 100–115, Springer-Verlag LNCS 1883, 1999.

[26] CURLESS, B. and LEVOY, M., "A Volumetric Method for Building Complex Models from Range Images," in *SIGGRAPH 1996, Computer Graphics Proceedings*, pp. 303–312, 1996.

[27] DEBEVEC, P., TAYLOR, C., and MALIK, J., "Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-based Approach," in *SIGGRAPH 1996, Computer Graphics Proceedings*, pp. 11–20, 1996.

[28] DEDIEU, S., GUITTON, P., SCHLICK, C., and REUTER, P., "Reality: an Interactive Reconstruction Tool of 3D Objects from Photographs," in *Workshop on Vision, Modeling, and Visualization*, 2001.

[29] DINH, H. Q., TURK, G., and SLABAUGH, G., "Reconstructing Surfaces Using Anisotropic Basis Functions," in *International Conference on Computer Vision*, vol. 2, pp. 606–613, 2001.

[30] DINH, H. Q., TURK, G., and SLABAUGH, G., "Reconstructing Surfaces by Volumetric Regularization Using Radial Basis Functions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[31] DYER, C. R., "Volumetric Scene Reconstruction from Multiple Views," in *Foundations of Image Understanding* (DAVIS, L. S., ed.), pp. 469–489, Kluwer, 2001.

[32] EISERT, P., STEINBACH, E., and GIROD, B., "Multi-Hypothesis, Volumetric Reconstruction of 3-D Objects From Multiple Calibrated Camera Views," in *ICASSP*, vol. 6, pp. 3509–3512, 1999.

[33] EISERT, P., STEINBACH, E., and GIROD, B., "Automatic Reconstruction of 3-D Stationary Objects from Multiple Uncalibrated Camera Views," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 2, pp. 261–277, 2000.

[34] FAUGERAS, O. and KERIVEN, R., "Variational Principles, Surface Evolution, PDE's, Level Set Methods and the Stereo Problem," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 336–344, 1998.

[35] FAUGERAS, O., LAVEAU, S., ROBERT, L., CSURKA, G., and ZELLER, C., "3-D Reconstruction of Urban Scenes from Sequences of Images," *INRIA Tech. Report 2572*, pp. 1–24, June 1995.

[36] FAUGERAS, O., *Three-Dimensional Computer Vision: A Geometrical Viewpoint*. Cambridge, MA: The MIT Press, second printing ed., 1993.

[37] FAUGERAS, O. and LUONG, Q.-T., *The Geometry of Multiple Images*. Cambridge: MIT Press, 2001.

[38] FITZGIBBON, A. and ZISSERMAN, A., "Automatic 3D Model Acquisition and Generation of New Images From Video Sequences," *Proc. European Signal Processing Conference*, pp. 1261–1269, 1998.

[39] FOLEY, J. D., VAN DAM, A., FEINER, S. K., and HUGHES, J. F., *Computer Graphics: Principles and Practice*. Reading, MA: Addison-Wesley, second edition ed., 1996.

[40] FROMHERZ, T. and BICHSEL, M., "Shape from Contours as Initial Step in Shape from Multiple Cues," in *ISPRS Commission III Symposium on Spatial Information from Digital Photogrammetry and Computer Vision*, pp. 240–256, 1994.

[41] FUA, P. and LECLERC, Y., "Object-Centered Surface Reconstruction: Combining Multi-Image Stereo and Shading," *International Journal of Computer Vision*, vol. 16, pp. 35–56, Sep 1995.

[42] FUA, P. and SANDER, P., "Reconstructing Surfaces from Unstructured 3D Points," *Proc. European Conference on Computer Vision*, 1992.

[43] GARCIA, B. and BRUNET, P., "3D Reconstruction With Projective Octrees and Epipolar Geometry," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1067–1072, 1998.

[44] GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., and COHEN, M. F., "The Lumigraph," in *SIGGRAPH 1996, Computer Graphics Proceedings*, pp. 43–54, 1996.

[45] GREENE, N., "Environment Mapping and Other Applications of World Projections," *International Journal of Computer Vision*, pp. 21–29, 1986.

[46] HAN, M. and KANADE, T., "Multiple Motion Scene Reconstruction from Uncalibrated Views," in *International Conference on Computer Vision*, vol. 1, pp. 163–170, 2001.

[47] HARDING, C. M. and LANE, R. G., "Passive Navigation from Image Sequences by Use of a Volumetric Approach," *Journal of the Optical Society of America*, vol. 19, no. 2, pp. 295–305, 2002.

[48] HARTLEY, R. and ZISSERMAN, A., *Multiple View Geometry*. Cambridge: Cambridge Unversity Press, 2000.

[49] IRANI, M., HASSNER, T., and ANANDAN, P., "What Does the Scene Look Like from a Scene Point?," in *European Conference on Computer Vision*, vol. 2, pp. 883–897, 2002.

[50] ISODORO, J. and SCLAROFF, S., "Stochastic Mesh-Based Multiview Reconstruction," in *1st International Symposium on 3D Processing, Visualization, and Transmission*, pp. 568–577, 2002.

[51] JEBARA, T., AZARBAYEJANI, A., and PENTLAND, A., "3D Structure from 2D Motion," *IEEE Signal Processing Magazine*, vol. 16, no. 3, pp. 66–84, 1999.

[52] JIN, H., YEZZI, A., and SOATTO, S., "Variational Multiframe Stereo in the Presence of Specular Reflections," in *1st International Symposium on 3D Processing, Visualization, and Transmission*, pp. 626–630, 2002.

[53] KIMBER, D., FOOTE, J., and LERTSITHICHAI, S., "FlyAbout: Spatially Indexed Panoramic Video," in *ACM Multimedia*, pp. 339–347, 2001.

[54] KUTULAKOS, K. and SEITZ, S., "A Theory of Shape by Space Carving," *International Journal of Computer Vision*, vol. 38, no. 3, pp. 199–218, 2000.

[55] KUTULAKOS, K. N., "Approximate N-View Stereo," in *Proceedings of the the Sixth European Conference on Computer Vision*, vol. 1, pp. 67–83, 2000.

[56] KUTULAKOS, K. N. and SEITZ, S. M., "A Theory of Shape by Space Carving," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 1, pp. 307–314, 1999.

[57] LAURENTINI, A., "The Visual Hull Concept for Silhouette-based Image Understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, pp. 150–162, 1994.

[58] LAVEU, S. and FAUGERAS, O., "3-D Scene Representation as a Collection of Images," in *International Conference on Pattern Recognition*, pp. 689–691, 1994.

[59] LEON-GARCIA, A., *Probability and Random Processes for Electrical Engineering*. Addison-Wesley, 1993.

[60] LEVOY, M. and HANRAHAN, P., "Light Field Rendering," *Computer Graphics*, vol. 30, no. Annual Conference Series, pp. 31–42, 1996.

[61] LIPPMAN, A., "Movie-Maps: An Application of the Optical Video-disk to Computer Graphics," *Computer Graphics*, vol. 14, no. 3, 1980.

[62] LORENSON, W. and CLINE, H., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," in *SIGGRAPH 1987, Computer Graphics Proceedings*, pp. 163–170, 1987.

[63] M. KIMURA, SAITO, H. and KANADE, T., "3D Voxel Construction Based on Epipolar Geometry," in *International Conference on Image Processing*, pp. 135–139, 1999.

[64] MAGDA, S., KREIGMAN, D., ZICKLER, T., and BELHUMEUR, P., "Beyond Lambert: Reconstructing Surfaces with Arbitrary BRDFs," in *International Conference on Computer Vision*, vol. 2, pp. 391–398, 2001.

[65] MAGNOR, M. and GIROD, B., "Adaptive Block-based Light Field Coding," *Proc. International Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging*, pp. 140–143, 1999.

[66] MANNING, R. A. and DYER, C. R., "Interpolating View and Scene Motion by Dynamic View Morphing," in *Proc. Computer Vision and Pattern Recognition Conf.*, vol. 1, pp. 388–394, 1999.

[67] MARAGOS, P. and SCHAFER, R. W., "Morphological Systems for Multidimensional Signal Processing," *Proceedings of the IEEE*, vol. 78, pp. 690–710, April 1990.

[68] MARK, W. R., *Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping*. Ph.d. thesis, University of North Carolina, 1999.

[69] MARR, D. and POGGIO, T., "Cooperative Computation of Stereo Disparity," *Science*, vol. 194, pp. 283–287, 1976.

[70] MARTIN, W. and AGGARWAL, J. K., "Volumetric Descriptions of Objects from Multiple Views," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, no. 2, pp. 150–158, 1983.

[71] MATSUYAMA, T. and TAKAI, T., "Generation, Visualization, and Editing of 3D Video," in *The First International Symposium on 3D Data Processing, Visualization, and Transmission*, pp. 234–245, 2002.

[72] MATUSIK, W., BUEHLER, C., RASKAR, R., GORTLER, S. J., and MCMILLAN, L., "Image-Based Visual Hulls," in *SIGGRAPH 2000, Computer Graphics Proceedings*, pp. 369–374, 2000.

[73] MAX, N., PUEYO, S., and SCHRODER, P., "Hierarchical Rendering of Trees from Precomputed Multi-Layer Z-Buffers," in *Eurographics Workshop on Rendering*, pp. 165–174, 1996.

[74] MCMILLAN, L., "A List-Priority Rendering Algorithm for Redisplaying Projected Surfaces," Tech. Rep. Technical Report TR95-005, University of North Carolina, 1995.

[75] MILLER, G., RUBIN, S., and PONCELEON, D., "Lazy Decompression of Surface Light Fields for Precomputed Global Illumination," in *Eurographics Workshop on Rendering*, pp. 281–292, 1998.

[76] MOEZZI, S., TAI, L. C., and GERARD, P., "Virtual View Generation for 3D Digital Video," *IEEE Multimedia*, vol. 4, no. 1, pp. 18–26, 1997.

[77] MOSTAFA, M. G., HEMAYED, E., and FARAG, A., "Target Recognition via 3D Object Reconstruction from Image Sequence and Contour Matching," *Pattern Recognition Letters*, vol. 20, no. 11-13, pp. 1381–1387, 1999.

[78] NARAYANAN, P., RANDER, P., and KANADE, T., "Constructing Virtual Worlds Using Dense Stereo," in *International Conference on Computer Vision*, pp. 3–10, 1998.

[79] NEUMANN, J. and ALOIMONOS, Y., "Spatio-temporal Stereo Using Multi-resolution Subdivision Surfaces," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 181–193, 2002.

[80] NISTER, D., "Reconstruction from Uncalibrated Sequences with a Hierarchy of Trifocal Tensors," in *European Conference on Computer Vision*, pp. 649–663, 2000.

[81] NISTER, D., "Frame Decimation for Structure and Motion," in *European Workshop on 3D Structure from Multiple Images for Large-scale Environments* (POLLEFEYS, M., VAN GOOL, L., ZISSERMAN, A., and FITZGIBBON, A., eds.), pp. 17–34, Springer LNCS 2018, 2001.

[82] OKUTOMI, M. and KANADE, T., "A Multiple-Baseline Stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, Apr. 1993.

[83] OSHER, S. and SETHIAN, J., "Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations," *Journal of Computational Physics*, vol. 79, pp. 12–49, 1988.

[84] OZUN, O., "Comparison of Photo-consistency Measures Used in the Voxel Coloring Algorithm," Master's thesis, Middle East Technical University, 2002.

[85] POLLEFEYS, M., KOCH, R., and GOOL, L. V., "Self-Calibration and Metric Reconstruction Inspite of Varying and Unkonwn Intrinsic Camera Parameters," *International Journal of Computer Vision*, vol. 32, pp. 7–25, Jan. 1999.

[86] POLLEFEYS, M., KOCH, R., VERGAUWEN, M., and GOOL, L. V., "Hand-Held Acquisition of 3D Models With a Video Camera," *Proc. 2nd International Conference on 3-D Digital Imaging and Modeling*, pp. 14–23, 1999.

[87] POTMESIL, M., "Generating Octree Models of 3D Objects from Their Silhouettes in a Sequence of Images," *Computer Vision, Graphics and Image Processing*, vol. 40, no. 1, pp. 1–29, 1987.

[88] POULIN, P., OUIMET, M., and FRASSON, M. C., "Interactively Modeling with Photogrammetry," in *Proceedings of the Eurographics Workshop on Rendering*, pp. 93–104, 1998.

[89] PRESS, W., TEUKOLSKY, S., VETTERLING, W., and FLANNERY, B., *Numerical Recipes in C.* Cambridge University Press, second ed., 1992.

[90] PRITCHETT, P. and ZISSERMAN, A., "Wide Baseline Stereo Matching," *Proc. International Conference on Computer Vision*, pp. 754–760, Jan. 1998.

[91] PROCK, A. and DYER, C., "Towards Real-Time Voxel Coloring," in *Image Understanding Workshop*, pp. 315–321, 1998.

[92] REGAN, M. J. P., MILLER, G. S. P., RUBIN, S. M., and KOGELNIK, C., "A Real Time Low-Latency Hardware Light-Field Renderer," in *Siggraph 1999, Computer Graphics Proceedings*, pp. 287–290, Addison Wesley Longman, 1999.

[93] ROCKWOOD, A. and WINGET, J., "Three-Dimensional Object Reconstruction from Two-Dimensional Images," *Computer-Aided Design*, vol. 29, pp. 279–285, Mar. 1997.

[94] ROY, S. and COX, I. J., "A Maximum-Flow Formulation of the N-camera Stereo Correspondence Problem," in *Proceedings of the Interntational Conference on Computer Vision*, pp. 492–499, 1999.

[95] SAITO, H. and KANADE, T., "Shape Reconstruction in Projective Grid Space from Large Number of Images," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 49–54, 1999.

[96] SAWHNEY, H. S. and KUMAR, R., "True Multi-Image Alignment and Its Application to Mosaicing and Lens Distortion Correction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 3, pp. 235–243, 1999.

[97] SCHAFFALITZKY, F. and ZISSERMAN, A., "Viewpoint Invariant Texture Matching and Wide Baseline Stereo," in *International Conference on Computer Vision*, 2001.

[98] SCHARSTEIN, D., "Stereo Vision for View Synthesis," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 852–858, IEEE Computer Society Press, 1996.

[99] SEITZ, S. and DYER, C., "Physically-Valid View Synthesis by Image Interpolation," in *Proceedings, Workshop on the Representation of Visual Scenes*, IEEE Computer Society Press, June 1995.

[100] SEITZ, S. and DYER, C., "Photorealistic Scene Reconstruction by Voxel Coloring," *International Journal of Computer Vision*, vol. 35, no. 2, pp. 151–173, 1999.

[101] SEITZ, S. M. and DYER, C. R., "View Morphing," in *SIGGRAPH 1996, Computer Graphics Proceedings*, pp. 21–30, 1996.

[102] SETHIAN, J., *Level Set Methods and Fast Marching Methods*. Cambridge University Press, second edition ed., 1999.

[103] SHADE, J., GORTLER, S., HE, L., and SZELISKI, R., "Layered Depth Images," in *SIGGRAPH 1998, Computer Graphics Proceedings*, pp. 231–242, 1998.

[104] SHARMAN, K., NIXON, M., and CARTER, J., "Extraction and Description of 3D (Articulated) Moving Objects," in *1st International Symposium on 3D Processing, Visualization, and Transmission*, pp. 664–667, 2002.

[105] SHUM, H. Y., HEI, M., and SZELISKI, R., "Interactive Construction of 3D Models from Panoramic Mosaics," *Proc. Computer Vision and Pattern Recognition*, pp. 427–433, June 1998.

[106] SHUM, H.-Y. and HE, L.-W., "Rendering with Concentric Mosaics," in *SIGGRAPH 1999, Computer Graphics Proceedings*, pp. 299–306, 1999.

[107] SLABAUGH, G., CULBERTSON, W. B., MALZBENDER, T., LIVINGSTON, M., SOBEL, I., STEVENS, M., and SCHAFER, R., "A Collection of Methods for Volumetric Reconstruction of Visual Scenes," *Submitted to the International Journal of Computer Vision*.

[108] SLABAUGH, G., CULBERTSON, W. B., MALZBENDER, T., and SCHAFER, R., "Improved Voxel Coloring Via Volumetric Optimization," Tech. Rep. 3, Center for Signal and Image Processing, Georgia Tech., 2000.

[109] SLABAUGH, G., CULBERTSON, W. B., MALZBENDER, T., and SCHAFER, R., "A Survey of Volumetric Scene Reconstruction Methods from Photographs," in *Volume Graphics 2001, Proc. of Joint IEEE TCVG and Eurographics Workshop* (MUELLER, K. and KAUFMAN, A., eds.), pp. 81–100, Springer Computer Science, 2001.

[110] SLABAUGH, G., MALZBENDER, T., and CULBERTSON, W. B., "Volumetric Warping for Voxel Coloring on an Infinite Domain," in *European Workshop on 3D Structure from Multiple Images for Large-scale Environments* (POLLEFEYS, M., VAN GOOL, L., ZISSERMAN, A., and FITZGIBBON, A., eds.), pp. 109–123, Springer LNCS 2018, 2000.

[111] SLABAUGH, G., SCHAFER, R., and HANS, M., "Image-Based Photo Hulls for Fast and Photo-Realistic New View Synthesis," *Submitted to Real-Time Imaging*.

[112] SLABAUGH, G., SCHAFER, R. W., and HANS, M., "Image-Based Photo Hulls," in *1st International Symposium on 3D Processing, Visualization, and Transmission*, pp. 704–708, 2002.

[113] SLABAUGH, G., SCHAFER, R. W., and HANS, M., "Multi-Resolution Space Carving Using Level Set Methods," in *International Conference on Image Processing*, vol. 2, pp. 545–548, 2002.

[114] STEINBACH, E., GIROD, B., EISERT, P., and BETZ, A., "3-D Reconstruction of Real-World Objects Using Extended Voxels," in *International Conference on Image Processing*, 2000.

[115] STEVENS, M. R., *Reasoning About Object Appearance in the Context of a Scene*. Ph.d. thesis, Colorado State University, 1999.

[116] SZELISKI, R., "Rapid Octree Construction from Image Sequences," *CVGIP: Image Understanding*, vol. 58, no. 1, pp. 23–32, 1993.

[117] SZELISKI, R., "A Multi-View Approach to Motion and Stereo," *International Conference on Computer Vision and Pattern Recognition*, pp. 157–153, 1999.

[118] SZELISKI, R. and SHUM, H.-Y., "Creating Full View Panoramic Image Mosaics and Environment Maps," in *SIGGRAPH 1997, Computer Graphics Proceedings*, pp. 251–258, 1997.

[119] TAO, H., SAWHNEY, H. S., and KUMAR, R., "Dynamic Depth Recovery from Multiple Synchronized Video Streams," in *Conference on Computer Vision and Pattern Recognition*, pp. 118–124, 2001.

[120] TAO, H., SAWHNEY, H. S., and KUMAR, R., "A global matching framework for stereo computation," in *International Conference on Computer Vision*, pp. 532–539, 2001.

[121] TORR, P., FITZGIBBON, A., and ZISSERMAN, A., "Maintaining Multiple Motion Model Hypothesis over Many Views to Recover Matching and Structure," in *International Conference on Computer Vision*, pp. 485–491, 1998.

[122] TRIGGS, B., MCLAUCHLAN, P., HARTLEY, R., and FITZGIBBON, A., "Bundle Adjustment – A Modern Synthesis," in *Proceedings of the International Workshop on Vision Algorithms*, pp. 298–372, Springer-Verlag LNCS 1883, 1999.

[123] TRUCCO, E. and VERRI, A., *Introductory Techniques for 3-D Computer Vision*. New Jersey: Prentice-Hall, 1998.

[124] TSAI, R., "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses," *IEEE Transactions on Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987.

[125] TUYTELAARS, T. and GOOL, L. V., "Wide Baseline Stereo based on Local, Affinely invariant Regions," in *British Machine Vision Conference*, pp. 412–422, 2000.

[126] VEDULA, S., BAKER, S., SEITZ, S., and KANADE, T., "Shape and Motion Carving in 6D," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 592–598, 2000.

[127] VEDULA, S., RANDER, P., SAITO, H., and KANADE, T., "Modeling, Combining, and Rendering Dynamic Real-World Events from Image Sequences," *Proc. Virtual Systems and Multimedia*, pp. 323–344, 1998.

[128] WEGHORST, H., HOOPER, G., and GREENBERG, D. P., "Improving Computational Methods for Ray Tracing," *ACM Transactions on Graphics*, vol. 3, no. 1, pp. 52–69, 1984.

[129] WOOD, D. N., AZUMA, D. I., ALDINGER, K., CURLESS, B., DUCHAMP, T., SALESIN, D. H., and STUETZLE, W., "Surface Light Fields for 3D Photography," in *SIGGRAPH 2000, Computer Graphics Proceedings* (AKELEY, K., ed.), pp. 287–296, 2000.

[130] YEZZI, A. and SOATTO, S., "Stereoscopic Segementation," in *International Conference on Computer Vision*, vol. 1, pp. 59–66, 2001.

[131] ZHANG, Y. and KAMBHAMETTU, C., "Integrated 3D Scene Flow and Structure Recovery from Multiview Image Sequences," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 674–681, 2000.

[132] ZHANG, Z., "Flexible Camera Calibration By Viewing a Plane From Unknown Orientations," in *International Conference on Computer Vision*, pp. 666–673, 1999.

[133] ZISSERMAN, A., FITZGIBBON, A., and CROSS, G., "VHS to VRML: 3D Graphical Models from Video Sequences," *Proc. International Conference on Multimedia Systems*, pp. 51–57, 1999.

VITA

Greg Slabaugh was born in Garden City, MI on May 31, 1971. In May of 1989, he graduated from Brother Rice High School in Bloomfield Hills, MI. He then attended the University of Michigan in Ann Arbor, MI, graduating with honors in 1994 with a Bachelor of Science degree in Engineering Physics. That year, Greg received the Outstanding Student of the Year Award in Engineering Physics.

From summer 1994 to fall 1996, Greg worked at Friendly Software, a start-up software company located in Toledo, OH. There, he developed a 3D graphics engine for computer video games.

In the fall of 1996 Greg enrolled at the Georgia Institute of Technology for graduate school. In August of 1998 he graduated with honors with a Master of Science degree in Electrical Engineering. He then began work towards a doctorate degree. During his Ph.D. studies, Greg interned twice at Hewlett-Packard Laboratories in Palo Alto, CA; once in the fall of 1998 and again in the summer and fall of 1999.

While a graduate student, Greg received a Presidential Fellowship from the Georgia Institute of Technology, as well as a Foundation Fellowship from Schlumberger. In spring of 2002, Greg received an award in the Georgia Tech Student Paper Competition sponsored by Science Applications International Corporation. Greg was awarded the Center for Signal and Image Processing Outstanding Service Award in fall of 2000, and the Outstanding Research Award in fall of 2002. In the early years of his graduate studies, Greg worked as a teaching assistant, developing technology-based educational tools. The latter years of his graduate studies were funded by Hewlett-Packard.

Greg is member of IEEE, Tau Beta Pi, and Phi Kappa Phi.