

**INTERACTIVE SPECIFICATION AND ACQUISITION OF DEPTH FROM  
SINGLE IMAGES**

by

**MAX CHEN**

BACHELOR OF SCIENCE, ARCHITECTURE  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 1999

BACHELOR OF SCIENCE, COMPUTER SCIENCE  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 1999

Submitted to the Department of Architecture  
in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE IN BUILDING TECHNOLOGY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

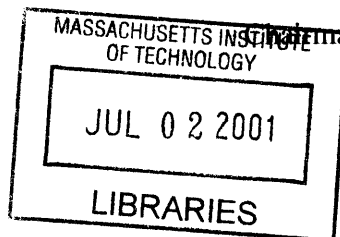
June 2001

© 2001 MASSACHUSETTS INSTITUTE OF TECHNOLOGY. All rights reserved.

Author .....  
Department of Architecture  
May 11, 2001

Certified by .....  
Julie Dorsey  
Associate Professor of Computer Science and Engineering and Architecture  
Thesis Supervisor

Accepted by .....  
Stanford O. Anderson  
Chairman, Department Committee on Graduate Students



**ROTCH**

# **INTERACTIVE SPECIFICATION AND ACQUISITION OF DEPTH FROM SINGLE IMAGES**

by

**MAX CHEN**

Submitted to the Department of Architecture  
on May 11, 2001, in partial fulfillment of the  
requirements for the degree of  
**MASTER OF SCIENCE IN BUILDING TECHNOLOGY**

## **ABSTRACT**

We describe a system for interactively acquiring depth for an image-based representation consisting of a single input image. We use layers of images with depth to represent the scene. Unlike traditional 3D modeling and rendering systems that require precise information that are usually difficult to model and manipulate, our system's emphasis is on ease of use, comprehensiveness, and use of potentially crude depth information.

Depth is extracted by the user through intuitive, interactive tools using the powerful notion of selection. Once a set of pixels is selected, the user can assign depth by painting and chiseling, using shape from shading, applying filters, aligning and extracting shape from geometry primitives, or using level set methods. The ground plane tool provides an intuitive depth reference for all other tools and serves as an initial step in depth specification. Our system is based on pixels and selections, and therefore does not impose any restriction on the scene geometry.

Our system allows the user to interactively perform high quality editing operations on SGI O2s and Octanes. We demonstrate the application of our system in the architectural design (relighting, sketching, 3D walkthroughs from images), complex photocompositing, and fine art exploration contexts.

Thesis Supervisor: Julie Dorsey

Title: Associate Professor of Computer Science and Engineering and Architecture

## ACKNOWLEDGMENTS

### Thanks...

**to those that gave me insights into this work.** Fredo for his ideas and his cooking with no mushrooms. Mok for the system, painting tools, and interface stuff. Matt for vision, math, and latex. Gernot for OpenGL help and wk. Sini and Ruji for experimenting with my tools and creating examples. Barry for keeping my imagination alive. My word...

**to the CGG.** Adel for giving me admin access. Bob for being Bobo. Bryt for helping me with the stapler. Franck for Virtua Tennis and “putain de merde”. Hans for teaching me about normals, 3D scanning, and filling holes. Henrik for keeping me busy with Dali. Jason for the movies and starting me on my list. Justin for interface ideas. Manish for Euclidean Space, printf, and voxels, eh? Mike for letting me model examples for his research and getting me started in graphics. Mrp for sparse matrices. Osama for letting me scan stuff in his old office. Sami for showing me Alt-Esc. Stephen for holding down the architecture stronghold in the lab. Wojciech for coffee. Julie for bringing us all together.

**to MIT.** Professor Senturia and Professor Umans for keeping me involved in music as well as EECS. Professor Bose for his stories. The 4.206 students for keeping me amused.

**to Music.** The MIT Music Library for their extensive CD collection. Jascha Heifetz for his Collection. The BSO, Symphony Hall, and Wallace Sabine.

Finally, thanks Diana for making me laugh; Annie for always being awake; and Mok, for hanging with me the last 5 years. San Diego, here we come.





---

# CONTENTS

---

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                             | <b>11</b> |
| <b>2</b> | <b>Previous Work</b>                            | <b>13</b> |
| 2.1      | Geometric Modeling . . . . .                    | 13        |
| 2.2      | Range Scanning . . . . .                        | 13        |
| 2.3      | Computer Vision . . . . .                       | 14        |
| 2.4      | Image-Based Modeling . . . . .                  | 14        |
| <b>3</b> | <b>System Overview</b>                          | <b>17</b> |
| 3.1      | Layers of Images with Depth Per Pixel . . . . . | 17        |
| 3.2      | System Architecture . . . . .                   | 17        |
| 3.3      | Reference Camera . . . . .                      | 18        |
| 3.3.1    | Required Camera Parameters . . . . .            | 19        |
| 3.3.2    | Projection . . . . .                            | 20        |
| 3.3.3    | Reference Camera Transformation . . . . .       | 21        |
| 3.4      | System Workflow . . . . .                       | 21        |
| <b>4</b> | <b>Image-Based Tools</b>                        | <b>25</b> |
| 4.1      | Painting Tools . . . . .                        | 25        |
| 4.1.1    | Painting and Chiseling . . . . .                | 25        |
| 4.1.2    | Push/Pull . . . . .                             | 26        |
| 4.2      | Shading Tools . . . . .                         | 26        |
| 4.2.1    | Shape from Shading . . . . .                    | 26        |
| 4.3      | Filtering and Blurring . . . . .                | 29        |
| 4.3.1    | Median Filtering . . . . .                      | 29        |
| 4.3.2    | Gaussian Filtering . . . . .                    | 29        |
| 4.4      | Stitching Tool . . . . .                        | 30        |
| <b>5</b> | <b>Geometry Tools</b>                           | <b>33</b> |
| 5.1      | Alignment . . . . .                             | 33        |
| 5.2      | Depth Reading . . . . .                         | 33        |
| 5.3      | Ground Plane . . . . .                          | 35        |
| 5.4      | Vertical and Horizontal Planes . . . . .        | 37        |
| 5.4.1    | Automatic Vertical Planes . . . . .             | 39        |
| 5.5      | Contour Mesh . . . . .                          | 40        |
| 5.6      | Shape Primitives . . . . .                      | 40        |

|          |   |           |
|----------|---|-----------|
| 5.6.1    | Sphere . . . . .  | 42        |
| 5.6.2    | Cube . . . . .  | 42        |
| 5.6.3    | Plane . . . . .   | 46        |
| 5.6.4    | Pyramid/Wedge . . . . .   | 47        |
| 5.6.5    | Cylinder . . . . .  | 48        |
| 5.7      | Heightfield . . . . .   | 49        |
| 5.8      | Intrusion and Extrusion . . . . .                               | 51        |
| <b>6</b> | <b>Freeform Tools</b>   | <b>55</b> |
| 6.1      | Organic Shapes Tool . . . . .                                   | 55        |
| 6.2      | Shape Template Tool . . . . .                                   | 56        |
| 6.2.1    | Point Correspondence Algorithm . . . . .                        | 56        |
| 6.2.2    | Depth Image Warping Algorithm . . . . .                         | 60        |
| <b>7</b> | <b>Results</b>  | <b>63</b> |
| 7.1      | Statue . . . . .  | 64        |
| 7.2      | Cathedral . . . . .   | 67        |
| 7.3      | Hearst Castle . . . . .   | 70        |
| 7.4      | Painting by Dali . . . . .                                      | 71        |
| 7.5      | Painting by Van Gogh . . . . .                                  | 73        |
| 7.6      | Painting by Turner . . . . .                                    | 74        |
| 7.7      | Outdoor Scene . . . . .   | 75        |
| 7.8      | Notre Dame . . . . .  | 77        |
| 7.9      | Hotel Lobby . . . . .   | 79        |
| <b>8</b> | <b>Conclusions</b>  | <b>81</b> |
| <b>A</b> | <b>Definitions</b>  | <b>83</b> |
| A.1      | System Architecture . . . . .                                   | 83        |
| A.2      | Reference Image . . . . .                                       | 83        |
| A.3      | Math Conventions . . . . .                                      | 84        |
| A.4      | Traditional 3D Modeling Interface/Interaction . . . . .         | 84        |
| <b>B</b> | <b>System Interface</b>   | <b>85</b> |
| B.1      | System Interface . . . . .                                      | 85        |
| B.2      | General Interface Functions and Toggles . . . . .               | 86        |
| B.3      | Image-Based Tools . . . . .                                     | 87        |
| B.4      | Geometry Tools . . . . .  | 88        |
| <b>C</b> | <b>Implementation Issues</b>                                    | <b>93</b> |
| C.1      | Catmull-Rom Interpolation . . . . .                             | 93        |
| C.2      | Bit Dependence . . . . .  | 93        |
| C.3      | Extracting Triangles from Winged-Edge Data Structures . . . . . | 93        |
| <b>D</b> | <b>Supplemental Geometry</b>                                    | <b>95</b> |
| D.1      | Line and Plane Intersection . . . . .                           | 95        |
| D.2      | Line and Line Intersection . . . . .                            | 96        |
| D.3      | Closest point on a line to a point . . . . .                    | 96        |
| D.4      | Center and Radius of a Circle . . . . .                         | 97        |

---

# LIST OF FIGURES

---

- 3-1 Depth Per Pixel . . . . . 18
- 3-2 Basic Layer Data Structures . . . . . 18
- 3-3 System Architecture . . . . . 19
- 3-4 Camera Model . . . . . 20
- 3-5 Field of View from Three Vanishing Points . . . . . 21
- 3-6 Reference Camera Transformation . . . . . 22
- 3-7 System Workflow . . . . . 23
  
- 4-1 Chiseling, Push/Pull, and Blurring . . . . . 26
- 4-2 Depth Translation . . . . . 26
- 4-3 Shape from Shading Bounds . . . . . 27
- 4-4 Shape from Shading: Traditional Method . . . . . 28
- 4-5 Shape from Shading: Color/Luminance Method . . . . . 28
- 4-6 Median Filtering . . . . . 29
- 4-7 Gaussian Filtering . . . . . 30
- 4-8 Stitching Tool . . . . . 30
- 4-9 Stitching Tool Diagram . . . . . 31
  
- 5-1 Reference Image Identification and Alignment . . . . . 34
- 5-2 Depth Alignment . . . . . 34
- 5-3 Frustum Tiling . . . . . 35
- 5-4 Depth Update Procedure . . . . . 36
- 5-5 Ground Plane Tool . . . . . 36
- 5-6 Ground Plane Interaction . . . . . 37
- 5-7 Vertical Plane Tool Example . . . . . 38
- 5-8 Vertical Plane Tool Interaction . . . . . 38
- 5-9 Automatic Vertical Plane Tool . . . . . 39
- 5-10 Contour Mesh Tool Example . . . . . 40
- 5-11 Contour Mesh Tool Interaction . . . . . 41
- 5-12 Sphere Tool Example . . . . . 43
- 5-13 Sphere Tool Interaction . . . . . 43
- 5-14 Cube Tool Example . . . . . 44
- 5-15 Cube Tool Interaction . . . . . 45
- 5-16 Plane Tool Example . . . . . 46
- 5-17 Plane Tool Interaction . . . . . 46
- 5-18 Pyramid/Wedge Tool Example . . . . . 47
- 5-19 Pyramid/Wedge Tool Interaction . . . . . 47

---

|      |  |    |
|------|--|----|
| 5-20 | Cylinder Tool Example . . . . .                          | 48 |
| 5-21 | Cylinder Tool Interaction . . . . .                      | 49 |
| 5-22 | Heightfield Tool Example . . . . .                       | 49 |
| 5-23 | Heightfield Tool Interaction . . . . .                   | 50 |
| 5-24 | Intrusion and Extrusion Tools Example . . . . .          | 51 |
| 5-25 | Intrusion and Extrusion Tools Interaction . . . . .      | 52 |
| 5-26 | Extrusion Geometry . . . . .                             | 53 |
| 5-27 | Intrusion and Extrusion Depth Update Procedure . . . . . | 53 |
|      |  |    |
| 6-1  | Organic Shapes . . . . .                                 | 56 |
| 6-2  | Organic Shapes Pixel Update Procedure . . . . .          | 57 |
| 6-3  | Shape Template . . . . .                                 | 58 |
| 6-4  | Similar Triangles . . . . .                              | 59 |
| 6-5  | Depth Image Warping Algorithm . . . . .                  | 61 |
|      |  |    |
| A-1  | System Conventions . . . . .                             | 83 |
|      |  |    |
| D-1  | Line and Plane Intersection . . . . .                    | 95 |
| D-2  | Line and Line Intersection . . . . .                     | 96 |
| D-3  | Closest point on a line to a point . . . . .             | 96 |
| D-4  | Center and Radius of a Circle . . . . .                  | 97 |

---

# LIST OF TABLES

---

|     |  |    |
|-----|--|----|
| 5.1 | Constrained Bounding Box Manipulation. . . . . | 42 |
| B.1 | General Buttons. . . . .                       | 86 |
| B.2 | Appearance Buttons. . . . .                    | 86 |
| B.3 | Image-Based Tools. . . . .                     | 87 |
| B.4 | Geometry Movement Buttons. . . . .             | 88 |
| B.5 | Primitive Tools Sketch Interaction. . . . .    | 90 |
| B.6 | Geometry Tools Interaction. . . . .            | 91 |



---

## INTRODUCTION

---

Today's 3D modeling and rendering systems, such as 3D Studio and Lightscape, are capable of allowing a user to generate geometric models of buildings and landscapes with accuracy and realism. These programs, however, are limited in several aspects. First, to the novice user, modeling and manipulating objects in 3D is not intuitive. Second, the models created by these systems will tend to limit the photo-realism of the images generated from them since they do not undergo real-world affects, such as weathering and aging. Further, acquiring and modeling an entire 3D scene can be quite time consuming and complex and many real-world scenes are simply too complex to model. Thus, while these systems serve architects well for final presentations, they cannot be used for design exploration when quick and slight modifications to the scene are required.

On the other hand, image-editing programs, such as Adobe Photoshop [2], provide a powerful 2D framework, where users can easily edit the appearance of an image with much versatility and flexibility. Typically, the system takes a photograph or image as input and then provides users with pixel-editing tools that can freely manipulate the image in any desired fashion. The 2D operators in these systems are easy and intuitive to use; the systems are interactive and hands-on, giving users fine control; and the simplicity of most operations allow immediate response and feedback. In the hands of an expert, even dramatically edited photographs remain convincingly realistic.

Recently, image-based modeling and rendering techniques have become increasingly widespread [25, 9, 31, 26, 18]. Using photographs of real scenes, these systems incorporate information about the scene, such as geometric relationships, which allow the user to view the scene from different viewpoints. These techniques, however, still require the user to acquire geometric information. Some techniques, such as laser range scanning technology [14, 13], are capable of capturing highly accurate geometric information. Unfortunately, they require special hardware and software that are not readily available to most consumers because of their cost (at least tens of thousands of dollars). Computer vision/photogrammetry techniques, such as stereo correspondence or feature matching [15] also can result in quite convincing representations of the real scene. But, these algorithms are limited since they often require several pre-processing steps, such as image rectification, and they are highly dependent upon finding corresponding features between images, which can often be absent. In addition, when there is only one image available, these methods are not applicable.

Another approach is to allow users to interactively model the scene geometry from multiple input images using photogrammetric modeling techniques [15, 7, 35]. These systems, such as Canoma [7] and Facade [15], are capable of creating incredibly convincing 3D architectural environments from photographs by interactively reconstructing the 3D polygonal model and using the photographs as textures. Yet, these systems are rigid and permit neither quick edits nor modeling of organic shapes.

We present an interactive image-based editing system that allows the user to quickly specify and acquire depth from images. The primary motivation for our system is to provide simple tools for the novice user, which allow real-time editing and navigation through any image. Once depth information is acquired from the image, several applications can ensue. Simple virtual camera movement can provide insight into design sketches and drawings through novel viewpoints. Depth specified for panoramic photographs can provide immersive environments when the user manipulates the virtual camera. Modifications, such as lighting changes and importing new geometric objects, to a photograph can also be achieved with depth information. Moreover, the models generated with our system retain

realism because our geometric representation uses the photographs as textures.

Depth assignment in our system is based on two important ideas. First, a depth value, the distance to the camera, is encoded for every pixel in the image. This allows the user to change depth easily with intuitive 2D tools. It also facilitates real-time rendering and feedback as well as defining arbitrary and complex shapes. Second, the user can apply the tools to one or more layers or selection areas. By constraining the depth tools to selected areas, the user can customize how the depth tools are applied.

Our depth tools can be classified into three categories: Image-Based Tools, Geometry Tools, and Freeform Tools. Image-Based Tools allow either the user or the system to infer and assign depth by examining the color channel of the image. There are several variants to these tools, such as painting and chiseling, shape from shading, and filtering. Geometry Tools are 3D shapes that the user aligns to objects in the image. The depth is specified by the shape of the object as well as its distance from the camera. Freeform Tools consist of two techniques that are specifically oriented towards modeling non-architectural objects.

The following chapters are organized as follows. First, we overview previous work in this area. Second, we present our image-based modeling and editing system [33], which our depth tools are a part of. Next, we describe each of the groups of tools: Image-Based Tools, Geometry Tools, and Freeform Tools. We then give step-by-step tutorials for acquiring depth and show a variety of results generated from our tools. Finally, we conclude with future work.



---

## PREVIOUS WORK

---

Our research builds on previous work in 3D scene reconstruction. This involves recovering the geometry and spatial organization of a scene from images or photographs; drawings, sketches, or paintings; or real-world environments. There have been research and development efforts that separately address each of the aforementioned inputs. The next sections describe previous work that includes geometric modeling systems, range scanning techniques, computer vision methods, and image-based modeling.

### 2.1 GEOMETRIC MODELING

Due to the advances in 3D graphics and hardware, many geometric modeling and rendering systems have found their way to desktop PCs. Currently, there are several software products [3, 4, 30] that allow users to build polygonal models accurately, incorporate real-world material properties and textures into these models, and render them from new vantage points with extreme realism.

While the above products can produce high-quality imagery, they require many hours of expert usage. They are also limited in scope because it is difficult to model organic or bulgy objects. Recently, there have been several systems that enable the user to generate 3D models with intuitive 2D strokes [50, 22, 11]. The SKETCH system [50] contains a set of 2D drawing gestures from which the system infers 3D shapes. It also includes more complex editing tools such as constructive solid geometry operations. One important note is that the system constrains the 3D shapes onto a pre-defined orthogonal ground plane. Our system borrows some 2D gestures from the SKETCH system, but it is not limited to orthogonal scenes. The Teddy fast-modeling system [22] permits the user to draw the outline of a blobby shape, such as a stuffed-animal. The system then forms the 3D shape by finding the spine within the outline and raising it a certain distance so that the cross section of the resulting inflated outline is circular. It also gives the user the option to modify this shape with extrusions, intrusions, cuts, and smoothing. Our system includes an implementation of the Teddy system using level sets, similar to [48], thereby allowing the user to easily create blobby objects. The Harold system [11] relies on a series of user-defined flat billboards that specify distinct depths for different objects in a drawing. Their system also includes gestures that allow the user to connect billboards and to form a heightfield by specifying distinct heights on a ground plane. The three systems mentioned all operate by inferring 3D object position based on user-defined 2D strokes. These systems, however, do not attempt to match the drawn object to an arbitrary image. Our research builds on these systems by giving the user tools to draw 2D strokes that match the projection of the actual 3D object in a photograph.

### 2.2 RANGE SCANNING

In addition to interactive geometric modeling techniques, there have been several advances in automatically reconstructing a real scene. Laser range scanning technology [14, 13] is capable of reproducing accurate 3D point clouds of an existing environment. It involves a time-of-flight technique that measures the time it takes a laser beam to leave

the hardware source, hit an object in the environment, and return to the source. The 3D mesh generated by this time-of-flight technique can then be textured using a photograph of the environment taken from the same vantage point as the hardware source. Furthermore, research in robust techniques for merging multiple range images into complex models [47, 12, 40] makes it possible to recover accurate geometry and depth information for complete environments. Although these meshes can contain an extremely large number of points, which make them difficult to manipulate interactively, mesh simplification techniques [39] allow for interactive viewing of these meshes. Yet, these systems are not without flaws. Objects with a high specular component often fail with this reconstruction method. These systems also fail to reconstruct large scenes. Moreover, the high cost of these systems can prevent access to many first-time users.

## 2.3 COMPUTER VISION

Computer vision techniques are popular because they attempt to reconstruct 3D geometry automatically. Some of the earlier work includes recovering shape from a stereo pair of images [36, 34, 46]. This involves starting with two images, one left and one right, of the scene or object to be reconstructed. The typical algorithm then follows several steps. First, the pair of images needs to be rectified [23]. Second, the system finds point correspondences, points that project from the images to the same 3D point in the scene. Finally, the depth map of the scene is recovered. But, this technique often fails because there can be noise in the pair of images or occlusions in the scene, both which can lead to incorrect correspondence matches. Another technique, shape from shading, attempts to recover shape by determining the position of the light source in a scene and assuming a known or uniform reflectance model for the object to be reconstructed [20, 5, 51]. These methods, however, are limited because only objects with the assumed reflectance model can be recovered. Moreover, real-world scenes are typically lit by more than one light source, which complicates the computation of the shape from shading method.

## 2.4 IMAGE-BASED MODELING

Image-Based Modeling has been an active research area in the past few years. More recent work includes photogrammetric modeling techniques, which allow the user to reconstruct 3D geometry of architectural models from a set of photographs [15, 16, 37, 27, 7, 35]. Using computer vision tools, these systems first establish correspondences between images, then exploit the constraints available from epipolar geometry and from geometric relationships that are common in architectural scenes, such as parallelism, orthogonality, repetition, and symmetry, to extract 3D geometry. The photographs are then projected onto the geometry and used as texture maps. These systems are capable of producing realistic renderings of architectural scenes under the original lighting conditions. They are also useful for creating new views of the original scene. However, they are rigid modeling systems because the reconstructed scene is limited by the geometry of the primitives included in the system.

There are other Image-Based Modeling systems that address non-architectural imagery. Blanz and Vetter [6] provide a solution to generating a 3D model of human faces from an image. Their method is based on building a morphable human face model from a database of hundreds of 3D face models. A set of correspondence points between the morphable model and the image of the face are then identified automatically through optic flow. Their system then optimizes the linear combination of parameters for the morphable model using these correspondence points and morphs the 3D model to fit its projection onto the image. Our system provides similar functionality by extending this idea to any 3D shape template geometry. Our model, however, is not based on these morphable parameters because we argue that exact depth is not necessary in order to achieve convincing results.

Our system also builds on Horry *et. al's* *Tour into the Picture* [21] and Kang's depth painting system [24]. Horry *et. al* resort to a less precise "spidery mesh" interface to crudely approximate depth, yet are able to achieve convincing results. Their system is limited to one-point perspective images and supports only crude billboard geometry, while our system is not constrained by these limitations. Kang's depth painting system offers the user several operators to adjust depth either pixel by pixel or by selected regions. Kang's system offers the user a brush-like interface to paint depth values and four orthographic side view windows to preview depth changes in real time. Our system offers a group of tools in addition to these traditional painting and chiseling operators.

The depth representation used in our system is similar to the plenoptic editing approach of Seitz *et al.* [42]. Their goal is also to build and edit an image-based representation. However, their approach differs from ours in that their system can deal with multiple views of the same part of the scene and propagate modifications among different

images, allowing a better handling of view-dependent effects. Unfortunately, the quality of their results is limited by the volumetric representation that they use. Moreover, they need multiple images of the same object viewed from outside in.

Our system builds on these works by providing tools that are easy to use and more flexible, thus giving the power and control to the user in specifying depth. There is also an emphasis on comprehensiveness. The aforementioned systems are each directly oriented towards a distinct class of images. We wish to address all types of images by presenting a suite of tools that allow the user to acquire depth from any image.



---

## SYSTEM OVERVIEW

---

Our system requires one reference image and associated camera parameters. From these inputs, the user is able to perform a number of operations. The image is segmented into layers using selection tools. Hidden regions resulting from layering can be painted by copying from similar regions in the image. Most importantly, the user can assign depth using interactive tools. Although user-in-the-loop CAD modeling has traditionally been viewed as tedious and time consuming, our system's interactive nature allows the user to extract depth quickly. Moreover, our image-based representation permits the user to define depth for objects that are difficult or nearly impossible to represent with traditional modeling techniques. The virtual camera can be manipulated at any time, thereby allowing the user total freedom while editing. This enables operations to be performed from any camera location. The following sections describe the implementation of our system, detailing the data structures, the system architecture, and the required inputs. We then conclude with an overview of the system workflow.

### 3.1 LAYERS OF IMAGES WITH DEPTH PER PIXEL

The foundation of our system is a simple image-based representation – layers of images with depth per pixel (Figure 3-1), similar to [10]. Encoding a depth value for each pixel facilitates specifying depth. For example, the user can select certain pixels for depth assignment. This permits areas in the reference image that define particular shapes to be segregated and segmented for manipulation.

The data structures are organized into layers and channels, as in many 2D image editing systems (Figure 3-2). A layer requires three basic channels: color, depth, and alpha/transparency. Layers are used to segment objects and/or areas of the image, and together with the channels, are composited into a single 3D representation. Each layer has the camera information sufficient for projecting pixels in 2D reference image space to 3D object space according to their depth values. This allows users to construct and edit 3D representations from single images as well as a panoramic image, which can lead to 360-degree panoramic environments. Depth can also be input into the system through the output of traditional modeling programs, 3D scanners [14, 13], or 3D cameras [1]. Our system, however, does not require precise depth information. We argue that our interactive tools provide quick, crude depth, which is sufficient for several applications.

### 3.2 SYSTEM ARCHITECTURE

The architecture of our system is composed of a set of tools built around a central data structure (Figure 3-3). This enables one to add new tools and operators easily. The tools, which include selection, painting, and depth acquisition tools, operate on the data structures at the pixel level and can affect any number of channels at once. Moreover, selection can be used to restrict the effect of certain tools to particular pixels or regions. The interactive view displays the reference scene information contained in our data structures using triangles [31, 32] and hardware projective

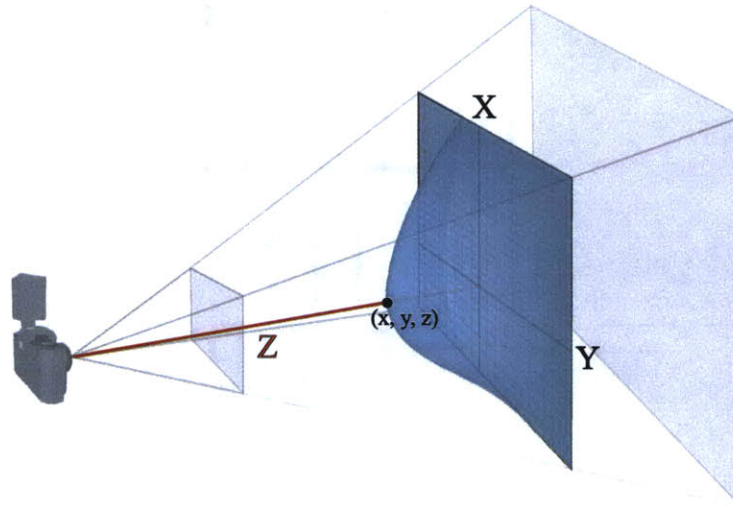


FIGURE 3-1: Depth Per Pixel. At each pixel  $(x,y)$ , there is a corresponding depth value,  $z$ .

```

layer {
  reference camera : transformation matrix
  color channels   : array of floats
  alpha channel   : array of floats
  depth channel   : array of floats
  optional channels : arrays of floats
}

```

FIGURE 3-2: Basic Layer Data Structures.

texture mapping [41]. The viewpoint can be changed at any time during the editing process, thereby allowing the user to operate on the reference scene from optimal positions.

### 3.3 REFERENCE CAMERA

The camera model in our system is based on the pinhole camera model. We implement our system using the graphics API OpenGL, which offers two methods to store camera parameters, as given in [49] (Figure 3-4):

```

glFrustum (left, right, bottom, top, near, far)
gluPerspective (fovy, aspect, near, far).

```

Our tools use both frustum and perspective parameterizations, although the latter is more intuitive since its parameters are closely related to physical camera attributes. In Chapter 5, we will refer to the frustum parameterization to explain how to acquire depth from the OpenGL depth buffer. To convert from perspective to frustum parameters, we need the following relations:

$$top = near * \tan\left(\frac{fovy}{180} * \pi * \frac{1}{2}\right)$$

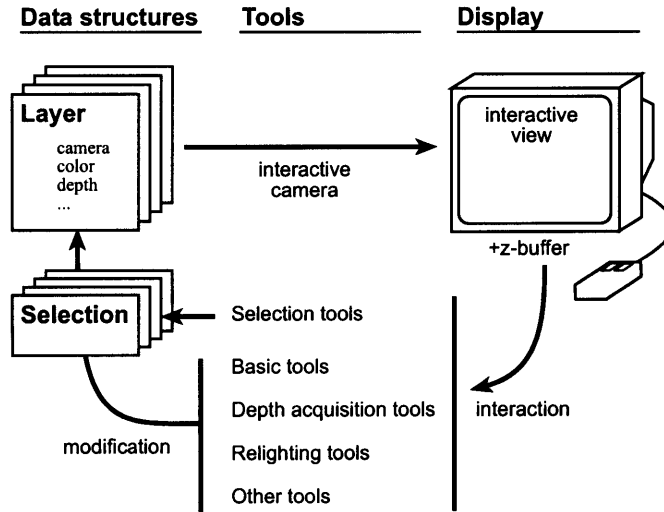


FIGURE 3-3: System Architecture

$$\begin{aligned}
 \textit{bottom} &= -\textit{top} \\
 \textit{right} &= \textit{aspect} * \textit{top} \\
 \textit{left} &= -\textit{right} .
 \end{aligned}$$

While our depth tools are specific for single images, they can also be useful when multiple images are present. This is particularly relevant for a series of images that can be stitched together to form a panorama. The next section describes the required camera parameter inputs for each reference image. We then note how we use this camera representation to project from 2D to 3D coordinates and vice versa. This discussion of the reference camera concludes by extending the use of our camera representation to layer transformations.

### 3.3.1 REQUIRED CAMERA PARAMETERS

Any image can be used as input to our system. The image dimensions, width and height, are inherent in the image. Therefore, all that is required for each image is an approximation of the field of view for each image's camera. In practice, we have found that our system works well even if the field of view is chosen arbitrarily. This is because our representation is pixels and not polygons, as opposed to most reconstruction algorithms and modeling systems. Nevertheless, our system can estimate the field of view given three vanishing points as described in [8]. If the user marks three distinct pairs of lines that are parallel in object space, the system determines the three vanishing points,  $vp_1, vp_2, vp_3$ , as the intersection of these lines. Each of these vanishing points is a certain distance  $f$ , the focal length, from the camera position and by definition is orthogonal to every other vanishing point. So we solve for the focal length as follows.

Since  $vp_1, vp_2, vp_3$  are orthogonal, as in Figure 3-5,

$$\begin{aligned}
 (vp_1 - P_c) \cdot (vp_2 - P_c) &= 0 \\
 (vp_1 - P_c) \cdot (vp_3 - P_c) &= 0 \\
 (vp_2 - P_c) \cdot (vp_3 - P_c) &= 0 ,
 \end{aligned}$$

which reduce to,

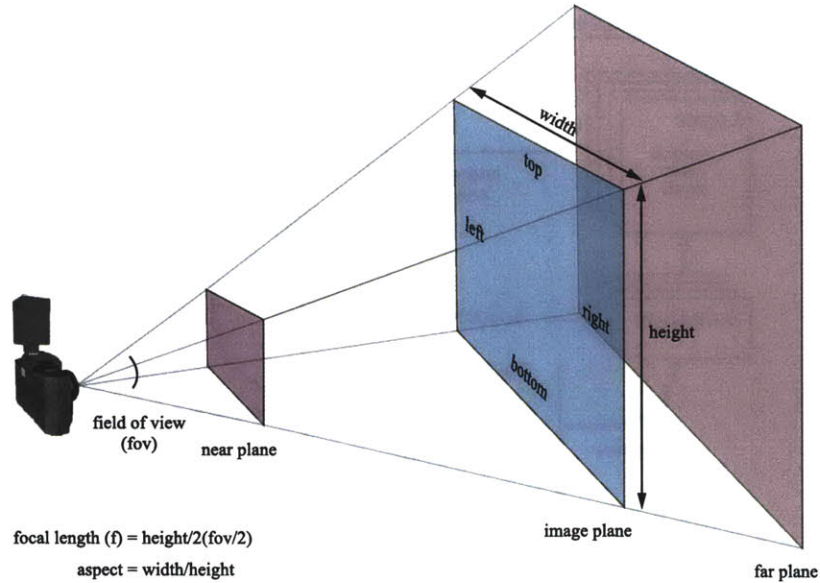


FIGURE 3-4: Camera Model.

$$vp_1 \cdot vp_2 + f^2 = 0$$

$$vp_1 \cdot vp_3 + f^2 = 0$$

$$vp_2 \cdot vp_3 + f^2 = 0$$

$$f = \sqrt{\frac{vp_1 \cdot vp_2 + vp_1 \cdot vp_3 + vp_2 \cdot vp_3}{3}}$$

We then use the focal length to derive the field of view  $fov$  as follows:

$$fov = \frac{180}{\pi} \theta,$$

where

$$\theta = 2 \arctan\left(\frac{height/2}{f}\right).$$

For single images, we set the camera position at the origin and use an arbitrary field of view or one derived from the method above. For multiple images or panoramas, the camera position for each of the images must be known. This is necessary so that the system can place each of the images in a common coordinate system. The field of view can be calculated as before.

### 3.3.2 PROJECTION

Once we have the required camera parameter inputs, we can project any 2D pixel coordinate in the reference image to its corresponding 3D coordinate in the reference scene. We perform this operation using OpenGL routines,

```
gluUnProject (winx, winy, winz, modelMatrix, projMatrix, viewport, objx, objy, objz)
gluProject (objx, objy, objz, modelMatrix, projMatrix, viewport, winx, winy, winz),
```



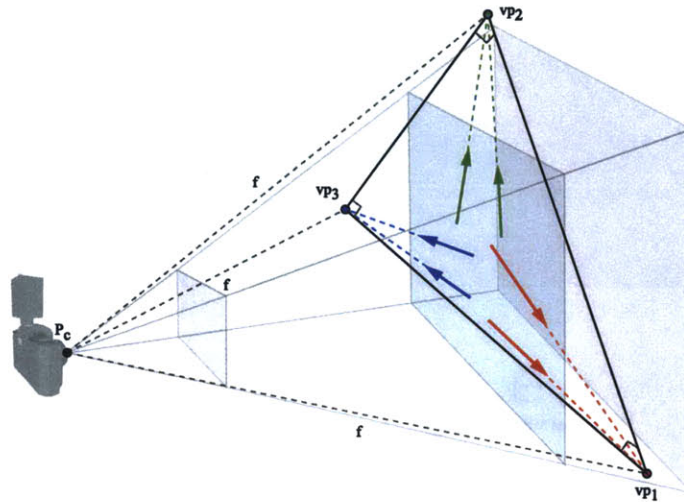


FIGURE 3-5: Field of View from Three Vanishing Points. The user draws three pairs of lines that are parallel in the reference scene. The system finds the three vanishing points that are the intersections of these pairs of lines. These vanishing points are a distance  $f$  from the camera position. We can use  $f$  to find the field of view.

where the 2D pixel coordinate is  $(winx, winy)$ , with a depth value of  $winz$ , and the corresponding 3D coordinate is  $(objx, objy, objz)$ . The depth values in OpenGL are stored ranging from 0.0 to 1.0, where the furthest depth value is 1.0.

### 3.3.3 REFERENCE CAMERA TRANSFORMATION

Typically, the parameters for the reference camera should remain constant throughout depth specification, because depth per pixel is dependent upon the camera matrix. This, however, is not mandatory. Once depth is specified for the reference image, the user can transform an entire layer through interactive translation, rotation, and scaling of the layer's reference camera. This is useful for adjusting layers as a whole, rather than modifying individual pixels. Therefore, if the user segments a chair in the reference image, he can move this chair's location in the reference image. In addition, if the chair lies between two images, OpenGL depth reading can be unreliable and produce inconsistent depth. In Figure 3-6, the resultant depth does not match. We therefore move one side of the chair so that it is flush against the other side of the chair. To some extent, these operations can easily be performed with image-editing programs such as Photoshop [2]. Translating and scaling are trivial. Rotation, however, is not. This is because there is no sense of 3D space in 2D image-editing programs. Because our system has a notion of 3D space, the user can intuitively rotate objects in 2D images using our manipulation schemes.

Transforming the camera parameters is performed as follows. For translation, the system translates the camera position the same amount as the translation of the layer. For rotation, the system rotates the camera position around the center of the layer. For scaling, the system changes the aspect ratio of the reference camera accordingly. The user can also select a point on the layer and a corresponding point in the reference scene. The layer will be translated by the distance between these points.

## 3.4 SYSTEM WORKFLOW

With these system elements at hand, the typical workflow of our system is as follows (Figure 3-7). The user begins with a single image. The next step is to segment the image into layers and fill hidden regions created by segmentation. This facilitates depth specification because it delineates regions in the image where depth modifications should occur. Our interactive tools then allow the user to specify depth at varying levels of detail. The final depth representation



FIGURE 3-6: Reference Camera Transformation. (a) Chair lies between two images, so depth reading is unreliable and results in a discontinuity between layers. (b) Depth is adjusted by moving one side of the chair closer to other side.

permits the viewer to navigate through the scene with a virtual camera, thus generating new views of the image. The next three chapters describe our depth tools in detail.

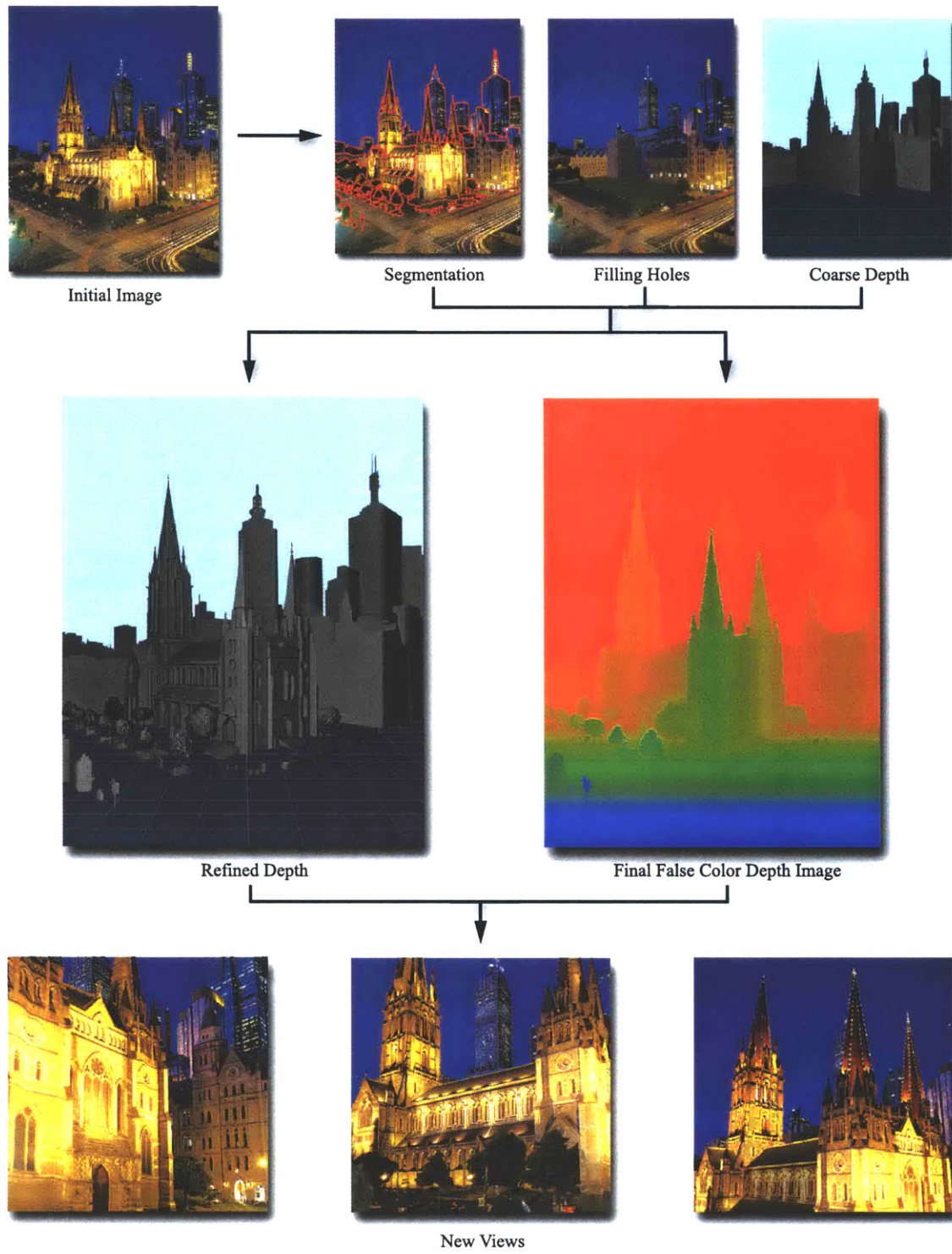


FIGURE 3-7: System Workflow.



---

## IMAGE-BASED TOOLS

---

Our discussion of depth tools begins with Image-Based Tools. These tools derive depth solely based on what the eye can discern from the image. From the perspective of the human eye, we are able to examine an image, segment the image into distinct objects, and perceive the depth of these objects by examining shading, texture, color, or placement. One group of Image-Based Tools parallels this process by allowing the user to segment the image and paint, chisel, and push/pull those segments in 3D with 2D strokes. From the perspective of the computer's eye, another group of Image-Based Tools derive much from the computer vision field. These tools acquire depth by determining shape and depth from the shading inherent in the image.

This chapter is divided into four parts. First, we discuss Painting Tools, which include painting, chiseling, and push/pull. Then, we introduce Shading Tools, which comprise shape from shading methods and its variants. Third, we describe the effects of filtering and blurring. Finally, we conclude with the stitching tool, which ensures that layer junctions are seamless by mending discontinuities between layers.

### 4.1 PAINTING TOOLS

Painting Tools offer the user direct access to the data structures by allowing modifications at the pixel level with a paintbrush interface. This is particularly useful when incremental depth modifications are necessary, such as cutting away the surface of a wall, carving the indentations of a sculpture, or adjusting the depth of a façade. These tools, however, also depend on the user's ability to make decisions about the depth variations in the scene. The effect of Painting Tools is not limited to the depth channel. Rather, they can be useful when adjusting luminance or transparency values in certain layers. For example, they can be used to change the highlights or colors in a room or surface, or remove portions of a layer by painting on the alpha channel. The following sections describe the two Painting Tools in our system, painting and chiseling and push/pull (Figure 4-1 (b)).

#### 4.1.1 PAINTING AND CHISELING

Painting and chiseling tools offer the user a brush interface to directly paint depth, as described by [24]. This is analogous to painting colors in photo-editing software. Painting assigns absolute depth values, while chiseling adds to or subtracts from current depth values. As with all depth tools, the virtual camera can be manipulated throughout the editing process. This permits the user to view and adjust depth changes simultaneously.

For painting, the user sets an absolute depth value either by directly selecting a grey value from the palette or by selecting a depth value in the reference scene with a color picker. For chiseling, the user specifies either an additive or subtractive chisel and brushes over the pixel regions to modify. The size and softness of the brush can be interactively chosen.



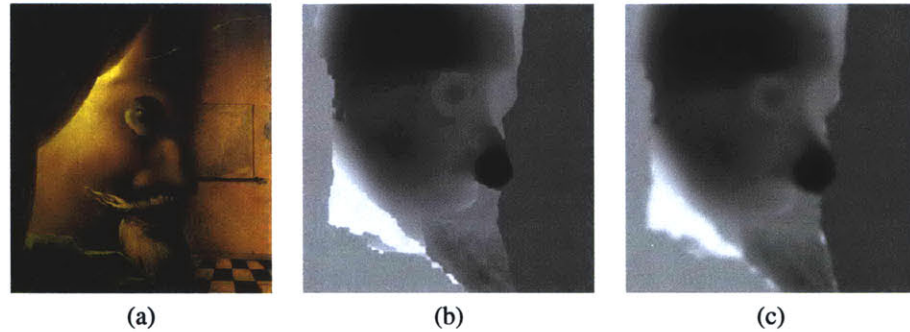


FIGURE 4-1: Chiseling, Push/Pull, and Blurring. (a) Original image with no depth. (b) Depth applied through chiseling and push/pull. (c) Depth blurred to smooth transition areas.

### 4.1.2 PUSH/PULL

Push/Pull allows the user to drag a selection area toward or away from the reference camera (Figure 4-2). This is useful for applying details such as doors, windows, and balconies, which often protrude or recede from building façades.

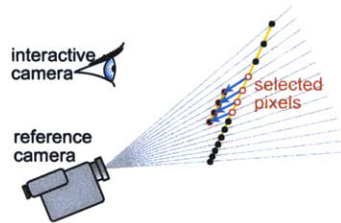


FIGURE 4-2: Depth Translation. Translation is performed along lines of sight with respect to the reference camera.

Push/Pull operates in the following manner. Since we wish that the translation remains planar, we cannot add or subtract a constant value to the current depth value, because the depth values are nonlinear. Instead, we must multiply the depth values by a constant value. This action therefore results in a parallel planar translation.

## 4.2 SHADING TOOLS

Images can be segmented into distinct objects, as previously mentioned. Each object often has specific material properties that result in predictable appearance through the physics of light transport and given information about the light sources. We can therefore operate on these classes of objects as a whole, deriving the entire shape or geometry from the luminance values in the reference image only.

Shading Tools operate on user-defined regions in the reference image by determining approximate depth from that region's luminance values. We describe two general methods of acquiring shape from shading. First, we overview traditional shape from shading methods in computer vision. Second, we introduce a variant upon this shape from shading method. There are no new techniques in the following sections. Rather, the emphasis on these sections is how these shape from shading methods can be used to acquire and refine depth from images.

### 4.2.1 SHAPE FROM SHADING

There have been many efforts [20, 5, 51] to derive shape or geometry from shading. The general principle is to assume a reflectance model for the object to be reconstructed and obtain normal vectors in the reference scene by examining the relationship between the luminance values in the reference image and a known light source position. While the

results can be somewhat promising, the method has several limitations. First, it requires the location of the light sources and works best with objects in the reference scene that are illuminated by a single light source. Since scenes are typically illuminated by a variety of light sources and determining the positions of light sources is difficult, shape from shading techniques fail for many images. Second, shape from shading depends on an assumed reflectance model. If there are many objects in the scene with differing material properties, shape from shading will likely fail. Because of these limitations, shape from shading is often unreliable and does not reconstruct the exact geometry of a scene.

Our application of shape from shading techniques, however, generates useful results because we employ user intervention, and we do not require precise depth. Moreover, while we use traditional shape from shading methods, we also use variations of shape from shading, shape from color and luminance, to obtain approximate depth from irregular scenes. Both variational methods rely upon user-defined bounds within which the tools operate. The bound specifies either the range that the depth values should map to, or a multiplicative amount that will be applied to the original depth values. The former bound would specify a more absolute shape as shown in Figure 4-3 (a), than the latter, which would simply alter existing depth as in Figure 4-3 (b). This gives the user more control over the resulting depth values generated from shape from shading.

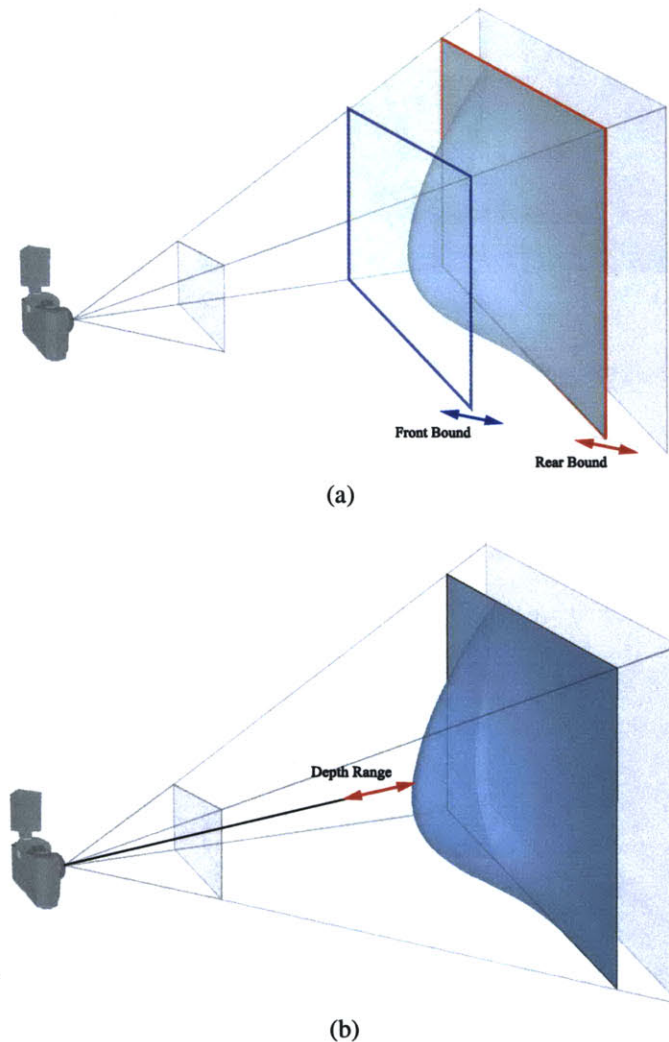


FIGURE 4-3: Shape from Shading: Bounds. (a) User interactively defines front and rear bounded regions. (b) User interactively defines a depth range that serves as a multiplier for the current depth values.

#### 4.2.1.1 TRADITIONAL METHOD

We implemented the Tsai model from [51]. This model assumes a Lambertian reflectance model, which has the reflectance function,

$$R(p_{x,y}, q_{x,y}) = \frac{-s_x p_{x,y} - s_y q_{x,y} + s_z}{\sqrt{1 + p_{x,y}^2 + q_{x,y}^2}},$$

where  $s_{x,y,z}$  is the light source position and  $(p, q)$  is the surface gradient at pixel  $(x, y)$ . Their method consists of using discrete approximations for the surface gradient and then linearizing the reflectance function to solve for the depth at each pixel. Our implementation, however, has one slight modification to this algorithm. Since our system is user driven, the user can specify a range within which the depth values from shape from shading should be normalized to. This affords the user more control over the output from shape from shading methods, which can be erratic. Figure 4-4 illustrates our implementation and the interface.

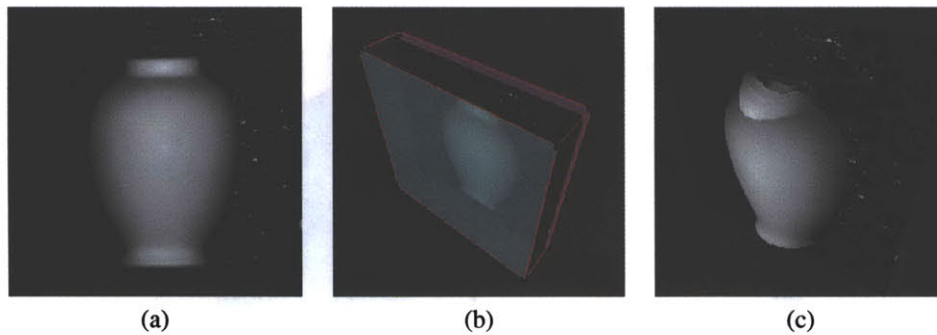


FIGURE 4-4: Shape from Shading: Traditional Method. (a) Original image. (b) Depth range. (c) Applied depth

#### 4.2.1.2 COLOR AND LUMINANCE METHODS

The color and luminance methods take the color and luminous intensity values of the reference image, respectively, as the depth values. In the color method, the user has the option of assigning a distinct weight to each color value. For example, the user can place more emphasis on extracting depth from the green channel in the forest scene shown in Figure 4-5. In the luminance method, we convert the color values,  $R$ ,  $G$ , and  $B$ , from the reference image selection to a grey-scale value  $L$ ,

$$L_{x,y} = 0.299R_{x,y} + 0.587G_{x,y} + 0.114B_{x,y}.$$

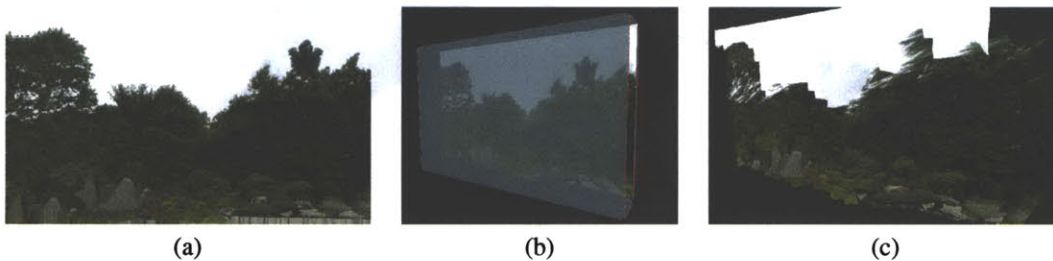


FIGURE 4-5: Shape from Shading: Color/Luminance Method. (a) Original image. (b) Depth range. (c) Applied depth



### 4.3 FILTERING AND BLURRING

Our system contains several filters that can be used to smooth values from pixel to pixel. The filters are especially important for smoothing the depth values generated from painting, chiseling, and shape from shading since those values can vary dramatically. Figure 4-1 demonstrates the use of filters on chiseled depth. The current system supports median and Gaussian filtering, although other filters can be easily added. Again, there are no new techniques introduced in the following sections.

#### 4.3.1 MEDIAN FILTERING

Median filtering (Figure 4-6) operates by taking the median value of the pixels in a user-specified region, or kernel, surrounding the current pixel being considered [28]. This type of filtering results in plateaus of differing depth values. It is most useful when the current reference image contains noise, or sharply contrasting pixels in areas of local consistency.

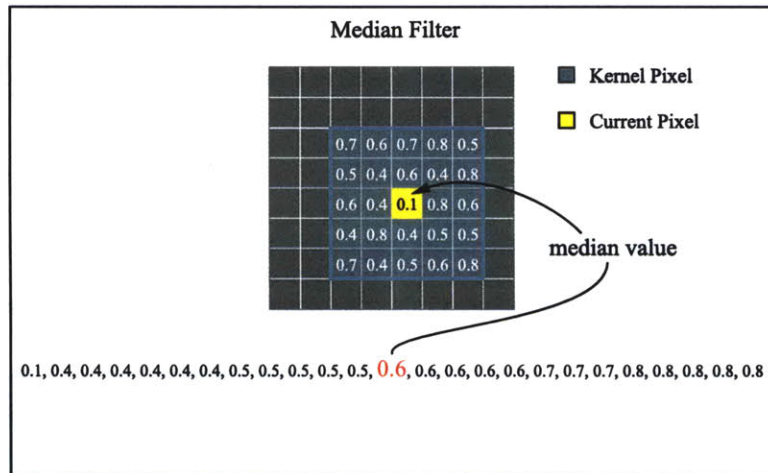


FIGURE 4-6: Median Filtering. Median filtering replaces the value of the current pixel with the median value within a user-specified kernel.

#### 4.3.2 GAUSSIAN FILTERING

Gaussian filtering (Figure 4-7) operates by taking user-defined weights for the pixels in a user-specified region surrounding the current pixel being considered [28]. Unlike the median filter, it will not eliminate pixels that are sharply contrasted from its neighboring pixels. Instead, it will tend to provide a smooth transition between all pixels.



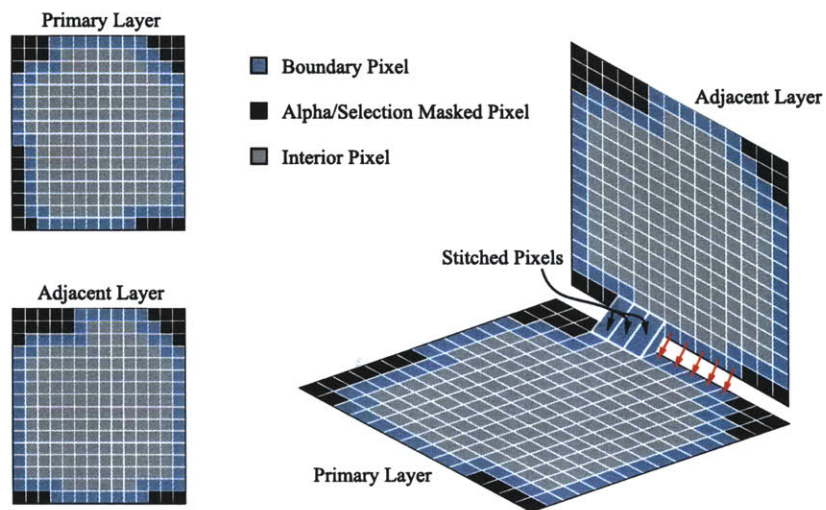


FIGURE 4-9: Stitching Tool Diagram. The stitching tool finds the closest pixels in adjacent layers to boundary pixels in the primary layer and matches the depth of the closest pixels to that of the boundary pixels.



---

## GEOMETRY TOOLS

---

**G**eometry Tools are 3D shapes that the user places into the reference scene through 2D interaction, as viewed from the reference camera. There are several advantages to using Geometry Tools over Image-Based Tools. First, if the image contains geometric objects, associating 3D shapes with them is intuitive for the user. It is much easier to imagine the depth of a box-shaped building, such as in Figure 5-1, extracted from a cube rather than derived from several paint/chisel strokes. Moreover, if the spatial layout of the scene can be easily associated with a combination of primitive geometries, the user can apply Geometry Tools since our system supports stacking and aligning several geometric objects with one another. Second, if an object is visible in multiple images, depth acquisition is propagated across each reference image since depth is acquired from the same 3D shape, and each image's camera is aligned to a common coordinate system. This is not necessarily the case with Image-Based Tools. Either each image would have to be edited individually, or our system could be modified to function as in [42]. Third, using polygonal shapes to give layers crude depth is often faster and easier than painting pixel by pixel. Geometry Tools are therefore essential components of our system because they quickly afford depth information through 2D gestures that define 3D shapes.

There are two steps that occur when using Geometry Tools. First, the user identifies an object in the image and aligns the projection of a corresponding 3D object using 2D interaction. Second, depth is then obtained by extracting the distance from each pixel to the reference camera of the reference image according to the selected area or layer. The following two sections describe alignment and depth reading.

### 5.1 ALIGNMENT

Alignment for Geometry Tools requires the user to perform two steps. First, the user must perform Reference Image Alignment (Figure 5-1). In this step, the Geometry Tool must be aligned with its 2D projection onto the reference image. Our system offers many manipulation schemes that aid the user in aligning Geometry Tools to a wide variety of scenes. Once the tool is aligned with its projection in the image, the only remaining degree of freedom is its distance from the camera. Specifying this distance is the second alignment step, Depth Alignment (Figure 5-2). The user specifies the depth, or the distance from the tool to the camera, by interactively dragging the tool or by clicking on a depth in the reference scene. The tool is initially situated at the depth of the reference image. As the user drags the tool nearer to or farther from the camera, it is scaled proportionally smaller or larger according to similar triangles.

### 5.2 DEPTH READING

We acquire the depth at each pixel by reading the depth buffer in OpenGL. The depth buffer in OpenGL contains the 3D distance of each pixel in the display window relative to the camera's position. The depth values are stored ranging from 0.0 to 1.0, where the furthest depth value is 1.0. The depth buffer, however, only contains the number of pixels

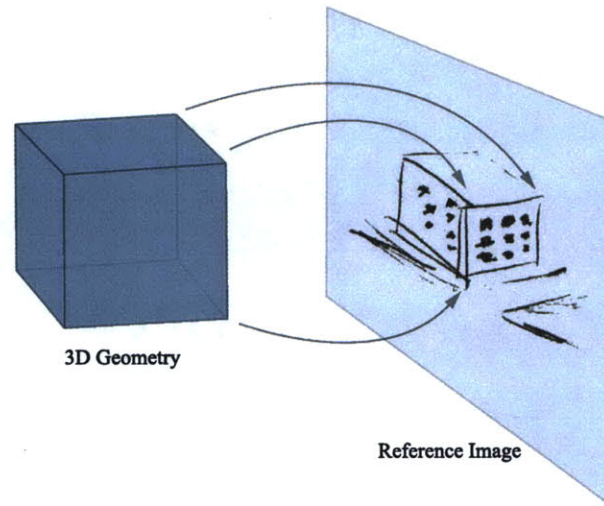


FIGURE 5-1: Reference Image Identification and Alignment. The user identifies a Geometry Tool that corresponds to an object in the image, in this case a cube and a building, respectively. The user then aligns the Geometry Tool to its projection as the object in the image through 2D gestures.

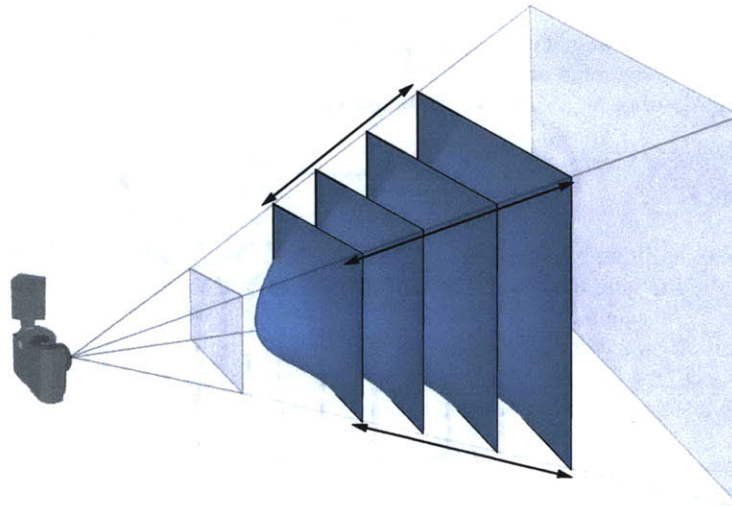


FIGURE 5-2: Depth Alignment. The user aligns the depth of the Geometry Tool in the reference scene. The Geometry Tool is initially at the depth of the reference image. It is scaled proportionally according to its distance from the camera.



specified by the resolution of the current window. Therefore, if our reference image contains more pixels than the pixel resolution of the current window, simply reading the current depth buffer will not give us correct depth values for the entire scene. Our solution to this problem is through frustum tiling. First, we set the camera frustum to a fixed size, the size of the current window's pixel resolution. Next, we determine the number of times to tile across and down the reference image as shown in Figure 5-3.

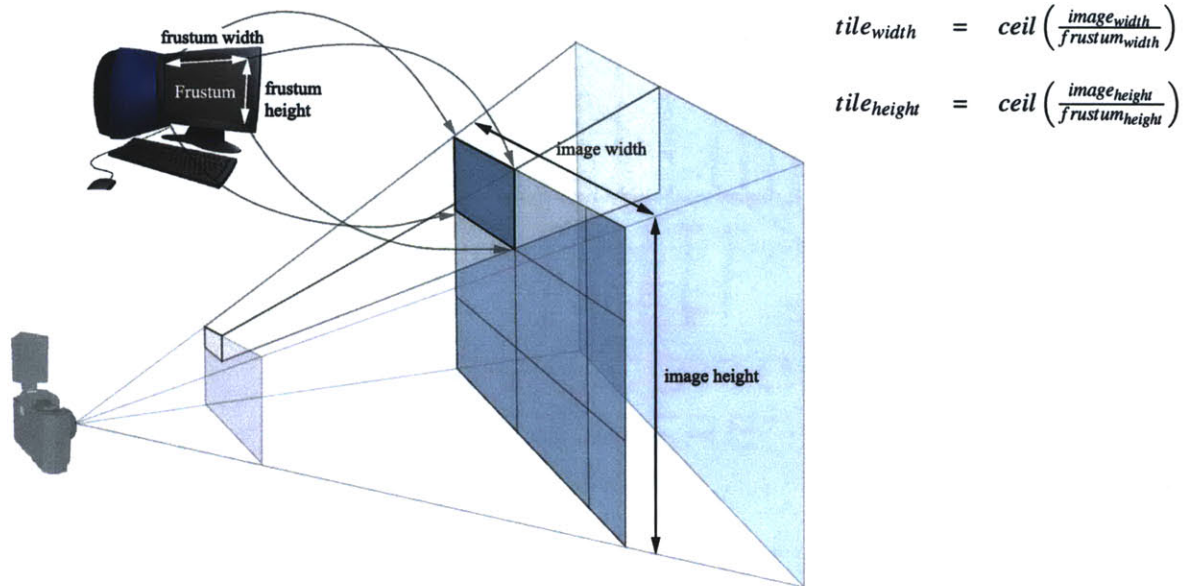


FIGURE 5-3: Frustum Tiling. The frustum size is set to the pixel resolution of the current window. The system iterates through the tiles, reading the OpenGL depth buffer and updating the data structures at each iteration.

We iterate through these tiles, and at each iteration, we display only the Geometry Tool object, read the depth buffer, and update our data structures. We update the depth values in our data structures as follows (Figure 5-4). If the depth value is 1.0, the Geometry Tool does not affect this pixel since a value of 1.0 corresponds to depth at the far plane. In other words, there is no new depth from the Geometry Tool. If the selection value is 1.0, this indicates that this pixel is in the user-selected area. Therefore, at each pixel, if the depth value is not equivalent to 1.0 and the selection value is 1.0, we replace that pixel's depth in our data structure with the new value. Otherwise, we do not modify this pixel's depth value and move onto the next pixel.

Table B.6 in Appendix B list the Geometry Tools with a brief diagram describing their manipulation schemes. The following sections describe each of the Geometry Tools, detailing typical uses and their corresponding alignment schemes.

## 5.3 GROUND PLANE

The ground plane corresponds to the horizon line of the reference image. This tool is useful for specifying the ground, floor, or terrain of the reference image (Figure 5-5). It should be the first step in extracting depth from most images because it can provide a basis for placing all other Geometry Tools in the reference image. This is similar to [21] in that all of their depth values are dependent upon their "spidery mesh". Our ground plane, however, does not limit the reference image to one-point perspective images, as in [21]. Moreover, it is important to note that an image will not have a clear horizon line if the line is beyond the image's view frustum. Under these circumstances, it will be difficult to align the ground plane to features in the image. Other tools in our system, however, can be used.

Reference Image Alignment and Depth Alignment is a one step process for the ground plane since it extends indefinitely in all directions. There are four methods of alignment for the ground plane (Figure 5-6). First, the user

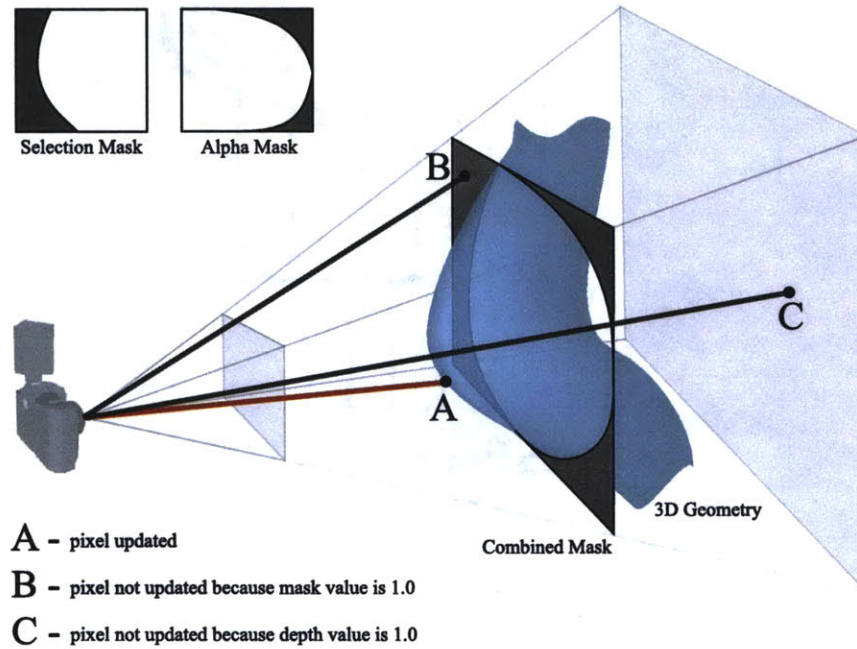


FIGURE 5-4: Depth Update Procedure. At each pixel, depth is updated according to the OpenGL depth buffer and the Selection and Alpha Masks. At pixel A, depth is updated. At pixel B, depth is not updated because the combined mask value is 1.0. At pixel C, depth is not updated because the depth value is 1.0 (depth at the far plane).

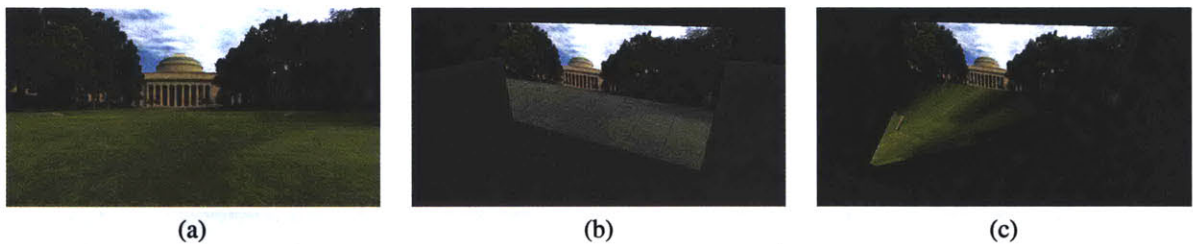


FIGURE 5-5: Ground Plane Tool. Here, we align the ground plane tool by picking the horizon line. (a) Original image. (b) Ground plane geometry. (c) Applied depth.



can pick any point on the horizon line of the reference image. This will translate the ground plane to the picked point, assuming that the horizon line can be seen or inferred in the reference image, and that it is parallel to the width of the reference image. For images that do not fulfill this criteria, we offer alternative methods of manipulation. The user can adjust the rotation and translation of the ground plane with traditional 3D modeling interaction as defined in Appendix A.4. Third, the user can draw two or three line segments that are perpendicular in the reference scene. Finally, the user can pick three points in the reference scene that already have assigned depth values. The last method is often useful when it is easier to first apply depth using other tools, such as shape primitives, and then derive the ground plane from points in the reference scene.

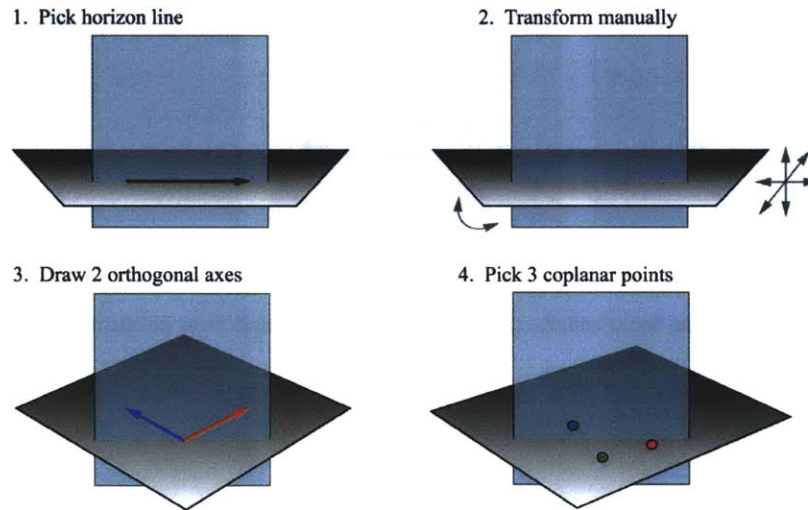


FIGURE 5-6: Ground Plane Interaction. The ground plane tool has 4 alignment schemes.

## 5.4 VERTICAL AND HORIZONTAL PLANES

Vertical and horizontal planes are relative to user-defined planes and are useful for providing quick, crude “billboard” depth as seen in [21]. They are also useful for providing more exact depth for objects in the reference image that have unique horizontal and vertical elements, such as walls and ceilings (Figure 5-7).

Reference Image Alignment and Depth Alignment is a one step process for the vertical and horizontal planes because they are relative to a pre-defined plane (Figure 5-8). Vertical plane alignment is as follows. The user first picks points on an arbitrary plane, by default, the ground plane. These points represent the point of contact between an object in the reference image and the ground plane. The user then specifies an extrusion height that corresponds to the object’s height. If the object is on its own layer or a selection area is chosen, the extrusion height can be infinite since the user has already restricted the region of influence. If the object is not on its own layer, we do not want the height to extend indefinitely and therefore leave it to the user to specify the correct height. The system duplicates and raises the set of lines connecting the points in the direction of the plane’s normal to the user-specified height. Quadrilaterals are then connected between the extruded lines and are used to extract depth. Our system also includes features such as Catmull-Rom Interpolation, as described in Appendix C.1, and manipulating existing points. These features allow the user to generate smooth curves and to make alterations to the vertical planes without redrawing all the previously defined points.

Interaction for the horizontal plane tool is similar to the vertical plane tool, aside from two differences. First, there is no extrusion. Second, the user picks points on the plane defined by the extruded vertical planes. The user has the option of snapping to points at the top of the vertical plane to create an enclosure or picking arbitrary points on the plane. In this manner, vertical and horizontal planes can be used in conjunction to generate depth for unique shapes that contain vertical and horizontal elements.

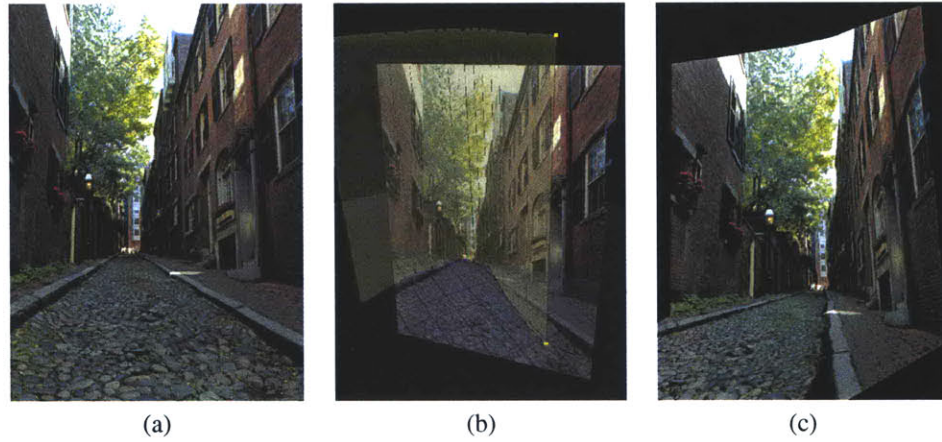


FIGURE 5-7: Vertical Plane Tool Example. (a) Original image. (b) Vertical plane geometry. (c) Applied depth.

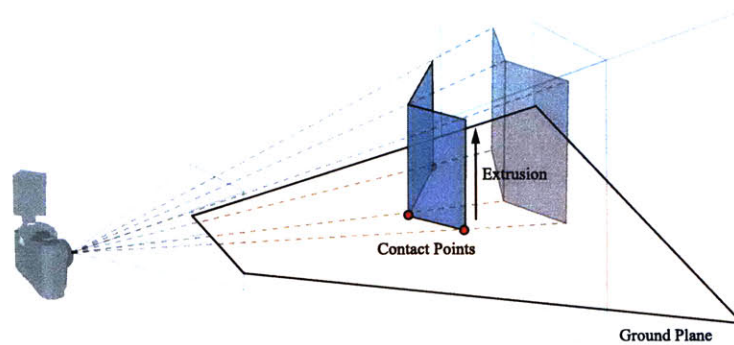


FIGURE 5-8: Vertical Plane Tool Interaction. The user picks points of contact in the reference image and specifies an extrusion height.

### 5.4.1 AUTOMATIC VERTICAL PLANES

We have also implemented an automatic version of the vertical planes that processes the whole selection or layer at once. There are two variants to this version (Figure 5-9). First, if the layer or selection is concave, then we fit the best straight line from the bottom of the area. Otherwise, the algorithm assumes that in each column of pixels in the layer or selection, there is a pixel that is in contact with the ground plane. The algorithm determines this pixel as the lowest visible pixel by marching upwards from the bottom of the layer or selection area until it finds a pixel with an alpha value larger than 0.0. A vertical plane is then placed at each lowest visible pixel, and depth is updated by reading the OpenGL depth buffer.

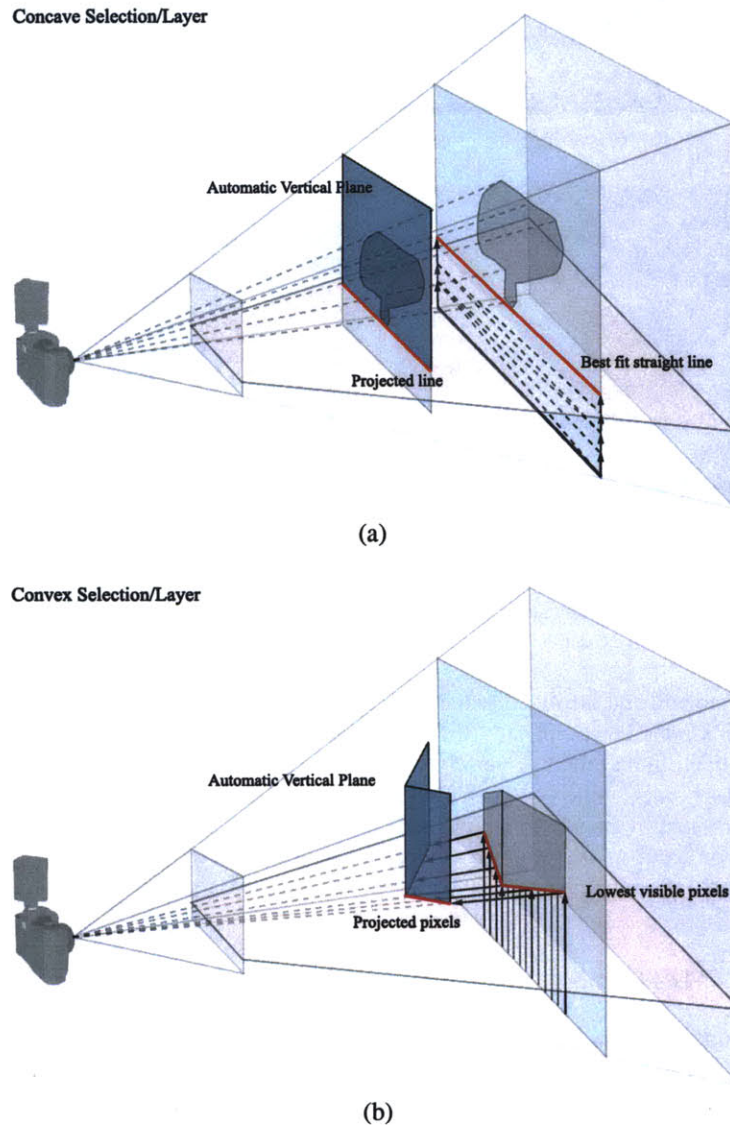


FIGURE 5-9: Automatic Vertical Plane Tool. (a) If the layer or selection area is concave, the system fits a straight line from the bottom of the layer. (b) Otherwise, the system finds the lowest visible pixel and generates an automatic vertical plane.



## 5.5 CONTOUR MESH

The contour mesh is a Bilinear Coons Patch [17] generated from four user-defined curves. It can provide crude depth for elements in the reference image that have no regular shape. This includes hills, trees, and bushes as in Figure 5-10.

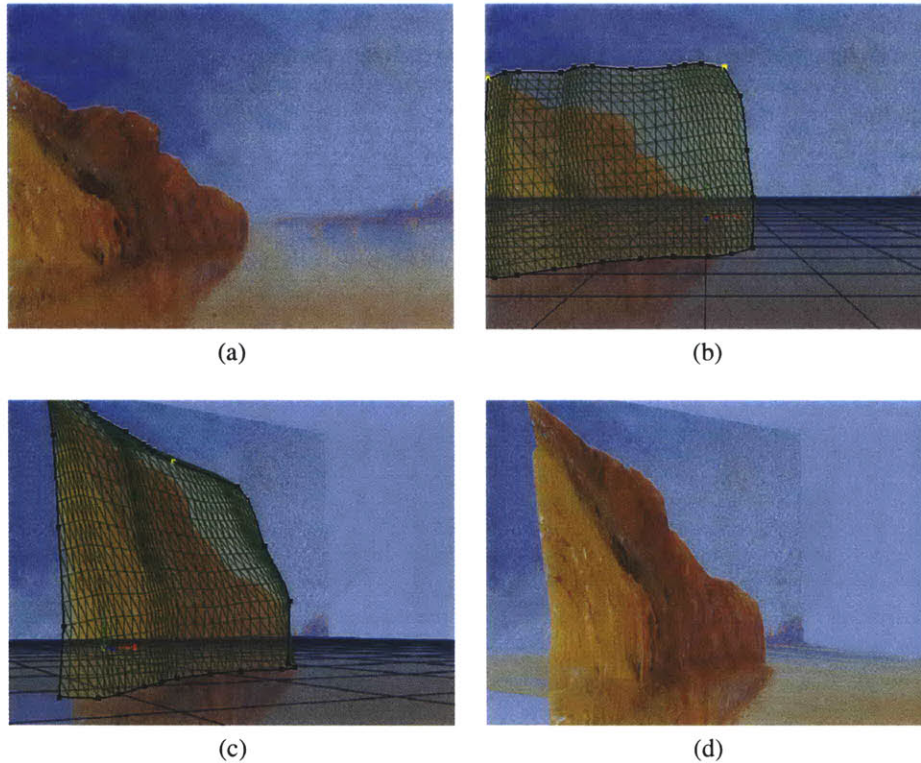


FIGURE 5-10: Contour Mesh Tool Example. (a) Original image. (b) Contour mesh geometry with ground plane. (c) Contour Mesh geometry from another view. (d) Applied depth.

Reference Image Alignment and Depth Alignment occur simultaneously as the user picks four boundary curves, which define the Bilinear Coons Patch, on four distinct planes (Figure 5-11). The first curve consists of points on an arbitrary plane, by default, the ground plane. The second and third curves are specified by points on the plane defined by the first and last points on the first curve, respectively, and another user-defined point on the ground plane. The fourth curve is determined by points on the plane defined by the last points on the second and third curves. A Bilinear Coons Patch is formed from these four curves using the algorithm described in [19]. The contour mesh tool also supports Catmull-Rom Interpolation between the points of each curve to produce a smoother mesh.

## 5.6 SHAPE PRIMITIVES

Our system includes the following shape primitives: sphere, cube, cylinder, and pyramid/wedge. Our shape primitives and the interface for their manipulation build upon previous work in two ways. First, our shape primitives include surfaces of revolution. This is not possible in [7, 35] because interaction is by selecting and transforming vertices individually. Second, manipulating our shape primitives always leads to predictable shapes. Because [15, 7, 35] allow the user to arbitrarily place vertices in 3D space that subsequently alter reference camera parameters, the resulting shape parameterized by these vertices can be unpredictable and is often undesirable.

Shape primitive alignment is a two-step process. The user performs Reference Image Alignment by transforming the shape primitive either through each shape primitive's individual sketch operation, which will be described in the

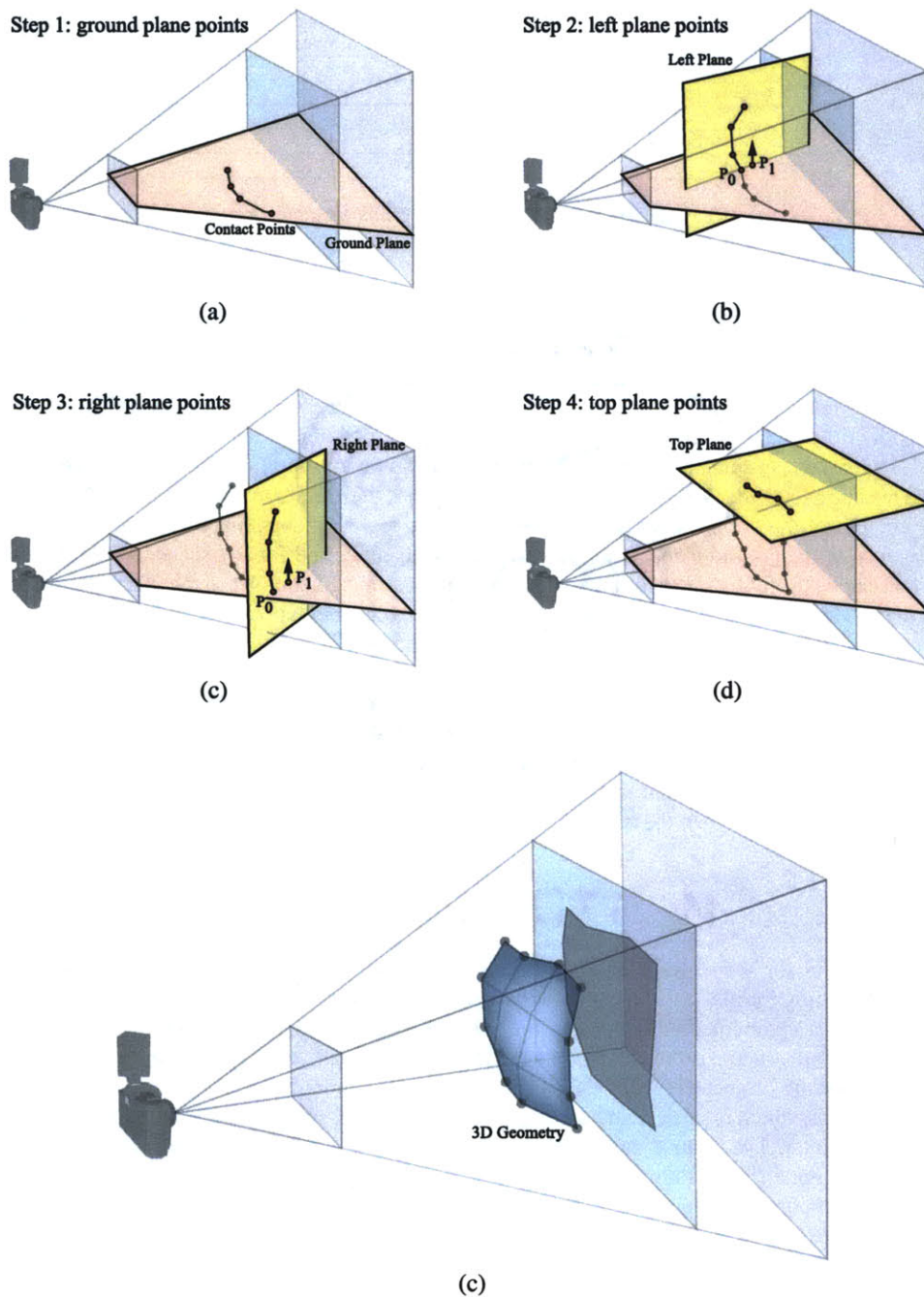


FIGURE 5-11: Contour Mesh Tool Interaction. The contour mesh is created by drawing four boundary curves. (a) The first curve is on the ground plane. (b) and (c) The second and third curves consist of points on the plane defined by  $P_0$ ,  $P_1$ , and the normal to the ground plane, as shown in Steps 2 and 3. (d) The fourth curve is on the top plane. (e) The resulting 3D geometry as a Bilinear Coons Patch.

| Number of Vertices Pinned | Degrees of Freedom | Constrained Manipulation       |
|---------------------------|--------------------|--------------------------------|
| 0                         | 9                  | 3 translate, 3 scale, 3 rotate |
| 1                         | 6                  | 3 scale, 3 rotate              |
| 2                         | 3                  | 2 scale, 1 rotate              |
| 2 (non planar face)       | 1                  | 1 rotate                       |
| 3                         | 2                  | 2 scale                        |
| 3 (non planar face)       | 0                  | none                           |
| 4                         | 1                  | 1 scale                        |
| 4 (non planar face)       | 0                  | none                           |
| greater than 5            | 0                  | none                           |

TABLE 5.1: Constrained Bounding Box Manipulation.

following sections, or through traditional 3D modeling techniques. The latter method operates through the shape's bounding box. The bounding box can be rotated, translated, and scaled, along its primary axes. Vertices on the bounding box can also be pinned or restrained, thereby reducing the degrees of freedom and constraining bounding box manipulation. Table 5.1 details these relationships. Once the user performs Reference Image Alignment, he specifies the depth of the object by moving it along the ray from the position of the bounding box to the center of the camera. As this occurs, the system scales the shape primitive an amount proportional to the distance to the camera as in Figure 5-2. This completes the Depth Alignment stage as well as the alignment process.

It is important to note that shape primitives, unlike the vertical and horizontal planes and the contour mesh, do not rely on the existence of the ground plane. Therefore, these tools can be used for any scene, not only those with clear horizon lines, and can be useful for determining the position of the ground plane. When the user has defined the ground plane, however, the system offers the option of aligning the primitive's orientation to be coincident with the normal of the ground plane. This is similar to stacking and aligning objects in [15, 7] and helps to preserve orthogonality among primitives.

The following sections describe each of the shape primitives in detail with a description of an alternative to the bounding box alignment scheme – a sketching interface. These methods are similar to those described in [50]. The user's strokes define the orientation and scale of each shape primitive. In our case, however, the user's strokes will match features not only in an orthogonal image, but also in a perspective image. Also, our system must solve for the full nine degrees of freedom since these shape primitives do not necessarily lie on a pre-defined ground plane, as in [50].

### 5.6.1 SPHERE

The sphere tool is useful for scenes such as architectural environments with domes. One typical use is shown in Figure 5-12. The sketching interface for the sphere tool consists of three user-defined points in the reference scene. Each of the three points is a point on the silhouette of the projected spherical object in the reference image. These points define the plane that the sphere tool projects onto as a circle. If  $P_1$ ,  $P_2$ , and  $P_3$  are the user-defined points (Figure 5-13), the center and radius of the projected sphere can be determined (Appendix D.4). We use these calculated values to transform the sphere tool, which is initialized as a unit sphere. The center of the sphere is the translation factor, while the radius is the scale factor. The rotation values are arbitrary because a sphere is symmetric about its center.

Since a sphere is symmetric, it is always oriented in the direction of the ground plane's normal. So there are no additional constraints imposed on the sphere if the user wishes to align the sphere tool to the normal of the ground plane.

### 5.6.2 CUBE

The cube tool can be used for architectural scenes such as building exteriors. Although the vertical plane is as useful for similar scenes, the cube is more beneficial for scenes in which the contact point with the ground is not clearly defined (Figure 5-14). Moreover, if there is no horizon line in the scene, the vertical plane tool will not function because the ground plane cannot be specified.

The sketching interface relies on four user-defined points. One point is the origin for the other three points that define three orthogonal edges in object space. If  $P_0$  is the origin and  $P_1$ ,  $P_2$ ,  $P_3$  are the other three user-defined points

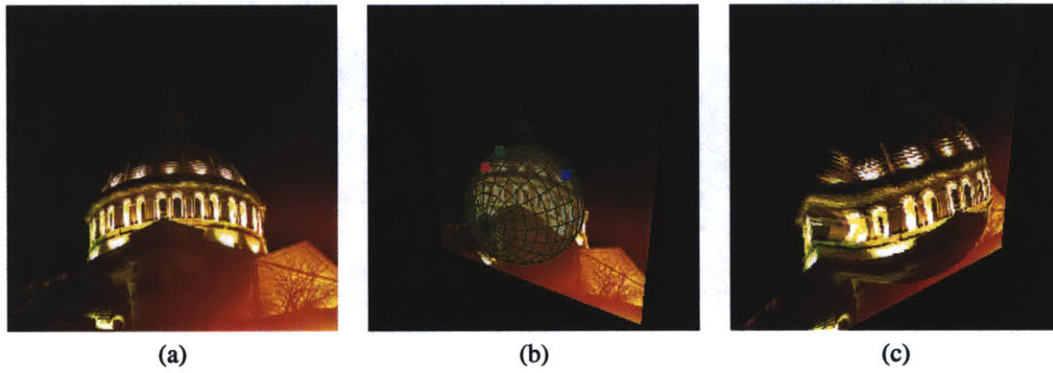


FIGURE 5-12: Sphere Tool Example. Here, we specify three points on the dome and apply depth only to the selected dome area. (a) Original image. (b) Sphere geometry. (c) Applied depth.

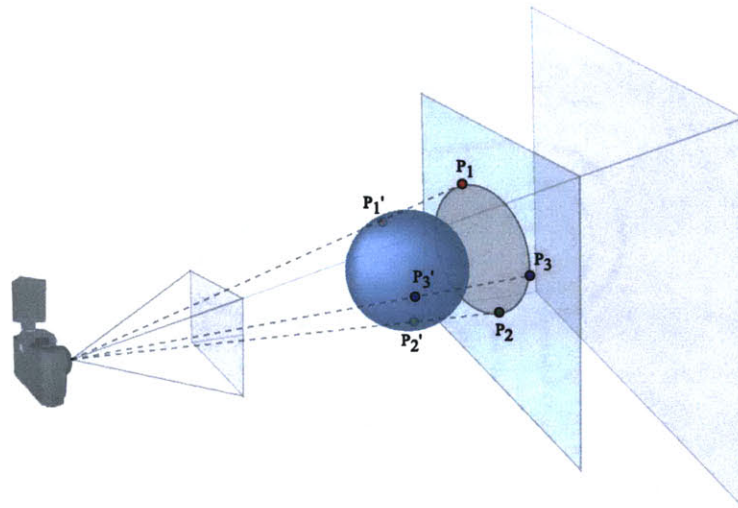


FIGURE 5-13: Sphere Tool Interaction. The user picks 3 points on the reference image. These are points on a circle, which is a projection of the sphere onto the reference image.



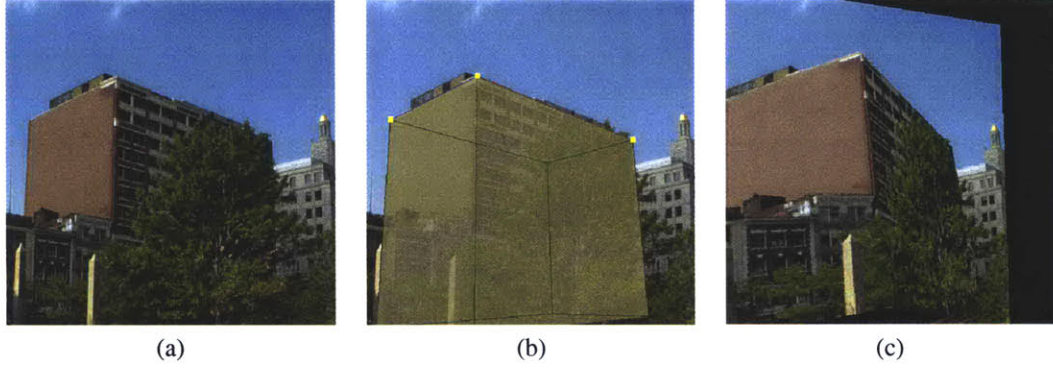


FIGURE 5-14: Cube Tool Example. Here, we specify three orthogonal edges on the top corner of the building. Since the ground is not visible, using the vertical planes would be difficult. (a) Original image. (b) Cube geometry. (c) Applied geometry.

in image space, with  $P'_0, P'_1, P'_2, P'_3$  as their projections into object space, and  $P_c$  is the position of the reference image's camera as shown in Figure 5-15, we assume that the following conditions hold:

$$P_0 = P'_0 \quad (5.1)$$

$$(P'_1 - P_0) \cdot (P'_2 - P_0) = 0 \quad (5.2)$$

$$(P'_1 - P_0) \cdot (P'_3 - P_0) = 0 \quad (5.3)$$

$$(P'_2 - P_0) \cdot (P'_3 - P_0) = 0, \quad (5.4)$$

where

$$P'_1 = p_1 N_1$$

$$P'_2 = p_2 N_2$$

$$P'_3 = p_3 N_3,$$

and

$$N_1 = (\widehat{P_1 - P_c})$$

$$N_2 = (\widehat{P_2 - P_c})$$

$$N_3 = (\widehat{P_3 - P_c}).$$

Expanding the dot products results in,

$$(N_1 N_2) p_1 p_2 - (N_1 P_0) p_1 - (N_2 P_0) p_2 + P_0^2 = 0$$

$$(N_1 N_3) p_1 p_3 - (N_1 P_0) p_1 - (N_3 P_0) p_3 + P_0^2 = 0$$

$$(N_2 N_3) p_2 p_3 - (N_2 P_0) p_2 - (N_3 P_0) p_3 + P_0^2 = 0,$$

which is of the form,

$$A_1 p_1 p_2 + B_1 p_1 + C_1 p_2 = D_1$$

$$A_2 p_1 p_3 + B_2 p_1 + C_2 p_3 = D_2$$



$$A_3 p_2 p_3 + B_3 p_2 + C_3 p_3 = D_3.$$

We solve for  $p_1$ ,  $p_2$ , and  $p_3$  using the Least Squares Method [45]. The solution will result in two sets of values. One will place  $P_1$ ,  $P_2$ , and  $P_3$  behind the camera, and the other will position those points in front of the camera. We take the set of values in front of the camera and use them to transform a unit cube. The three vectors  $(P_1 - P_0)$ ,  $(P_2 - P_0)$ , and  $(P_3 - P_0)$  are the rotation vectors of the cube and their magnitudes (before normalization) constitute the scale factors for the cube. The origin  $P_0$  is the translation factor.

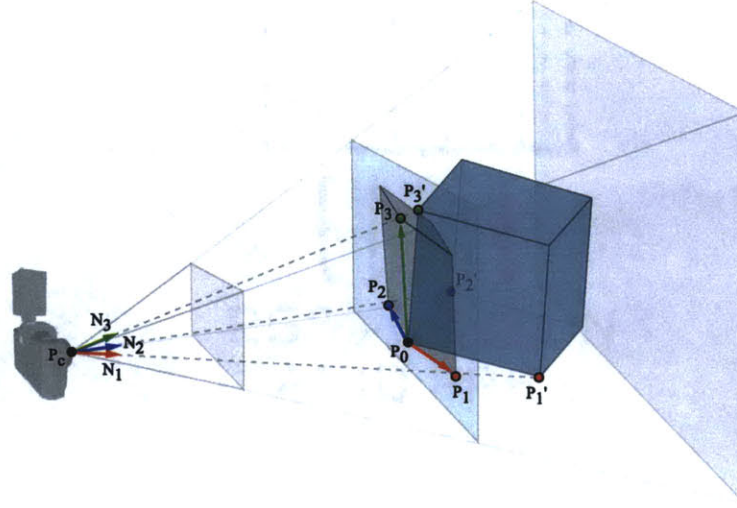


FIGURE 5-15: Cube Tool Interaction. The user draws three axes, which are orthogonal in the reference scene, on the reference image.

When the ground plane is known, we can orient the cube so that its upward direction coincides with the ground plane's normal. Since we know the upward direction, we can solve for  $P_3'$  as the intersection of  $\overline{P_c P_3}$  and  $\overline{P_0 P_{0n}}$ , where,  $P_{0n} = P_0 + N_{groundplane}$ .

Then, all that remains is to solve Equations 5.3 and 5.4 for  $p_1$  and  $p_2$ :

$$p_1 = \frac{P_0 \cdot P_3' - P_0 \cdot P_0}{N_1 \cdot P_3' - N_1 \cdot P_0} \quad (5.5)$$

$$p_2 = \frac{P_0 \cdot P_3' - P_0 \cdot P_0}{N_2 \cdot P_3' - N_2 \cdot P_0} \quad (5.6)$$

This solution loses the orthogonal relationship between  $\overline{P_1 P_0}$  and  $\overline{P_2 P_0}$  because it ignores Equation 5.2, which, if considered, would lead to an over-constrained system of equations. This is unacceptable because we would like to maintain orthogonality since the 3D object in the image most likely contains these relationships. To remedy this, we take only the solution to  $p_1$  and find a point  $P_2''$  closest to the user picked point  $P_2$  that still maintains the three orthogonal vectors. In solving for  $P_2''$ , we know that it must lie on the line orthogonal to  $P_1$  and  $P_3$ ,

$$P_2''' = (P_1' - P_0) \times (P_3' - P_0).$$

We project this line  $\overline{P_0 P_2'''}$  to the reference image plane and find the closest point on this line to  $P_2$  using Appendix D.3. This point is  $P_2''$ . We then solve for  $p_2$  in Equation 5.5, by recalculating  $N_2$  as,

$$N_2 = (P_2'' - P_c).$$

### 5.6.3 PLANE

The plane tool is useful for any scene with flat faces. It is particularly useful when the vertical plane tool cannot be used because of a missing horizon line (Figure 5-16). The plane tool can also provide a basis for pushing/pulling façade features. The sketching interface for the plane is identical to that of the cube, except that the fourth user-defined point is not necessary (Figure 5-17). With three points, we can derive the fourth point since the following is true,

$$P'_3 = (P'_1 - P_0) \times (P'_2 - P_0).$$

Therefore, three points is sufficient to align a plane with its projection on the reference image. Our system, however, allows the user to specify this fourth point to adjust the orientation of the plane. The orientation, scale, and translation factors are derived in the same manner as from the cube tool.

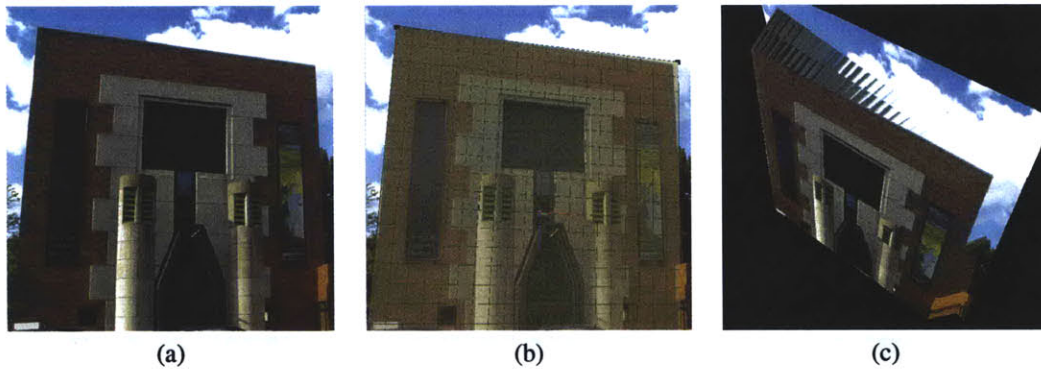


FIGURE 5-16: Plane Tool Example. This scene requires the plane tool because the horizon line is missing, eliminating the use of the ground plane tool, and only 2 orthogonal edges of the building are present, eliminating the use of the cube tool. (a) Original image. (b) Plane geometry. (c) Applied depth.

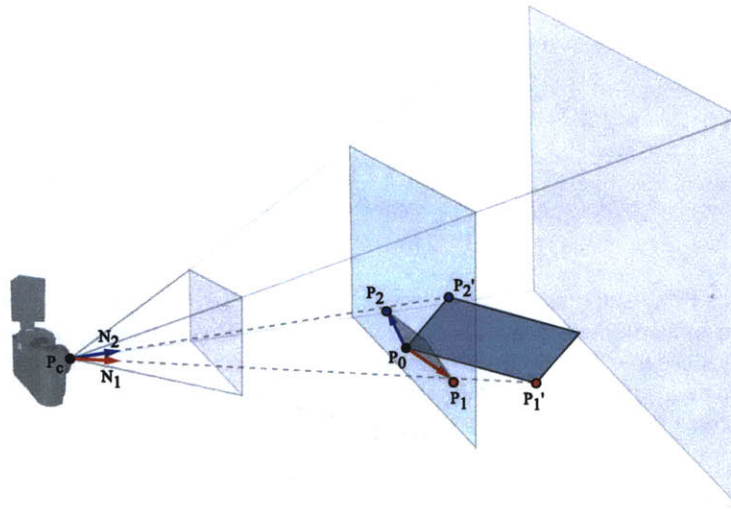


FIGURE 5-17: Plane Tool Interaction. The user draws two axes, which are orthogonal in the reference scene, on the reference image.

### 5.6.4 PYRAMID/WEDGE

The pyramid/wedge tools are useful for rooftops and slanted faces (Figure 5-18). The sketching interface for the pyramid/wedge is similar to that of the cube. It is alike in that the user specifies four points that comprise three orthogonal axes. At first, this may seem unorthodox for the user because a pyramid only contains two orthogonal axes at its base. Moreover, specifying only two orthogonal axes is not enough for the system to solve for the transformation factors of the pyramid. If we examine Figure 5-18, however, the pyramidal rooftop is connected to a shaft that does contain three orthogonal axes. This is typical of rooftops. Therefore, the pyramid tool requires that the user specify three orthogonal axes. The translation, scale, and rotation factors are derived in the same fashion as from the cube tool, and the system fits a pyramid within the bounds of the cube (Figure 5-19). For further control, the user can adjust the height of the pyramid's apex. Adjusting the apex will simply alter the scale factor.

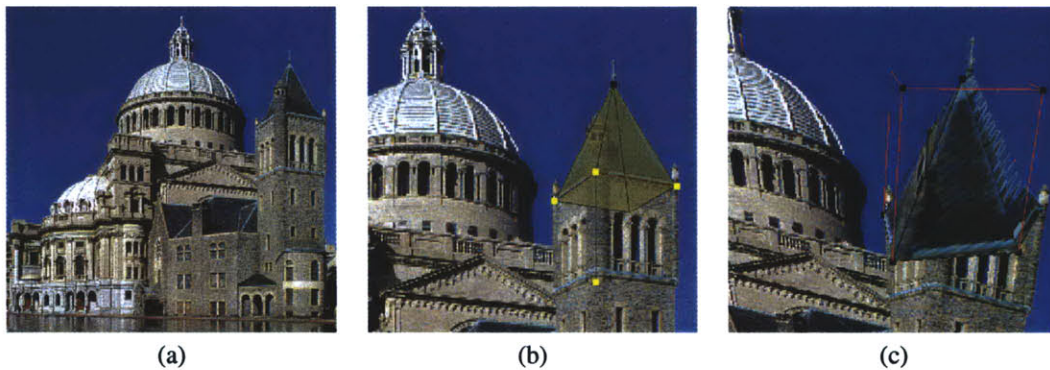


FIGURE 5-18: Pyramid/Wedge Tool Example. We specify three orthogonal axes at the corner of the tower. This sets the translation and orientation for the pyramid. We set the apex of the pyramid by scaling the pyramid. (a) Original image. (b) Cylinder geometry. (c) Applied depth.

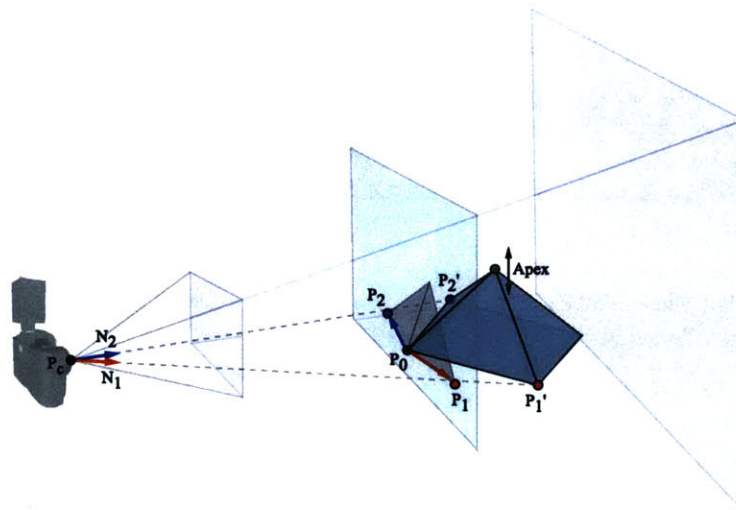


FIGURE 5-19: Pyramid/Wedge Tool Interaction. The user draws three axes, which are orthogonal in the reference scene, on the reference image. The apex can be adjusted manually.

Orienting the pyramid/wedge tool so that its upward direction is the same as the direction of the ground plane's normal can also be done. The process is identical to that of the cube tool.



### 5.6.5 CYLINDER

The cylinder tool is useful for columns, poles, and circular drums, the vertical walls that support a dome (Figure 5-20). The sketching interface is based on two user-defined lines. The user draws the left and right edges of the cylinder as shown in Figure 5-21. If  $\overline{P_0P_1}$  and  $\overline{P_2P_3}$  are the line segments,  $P'_0$ ,  $P'_1$ ,  $P'_2$ , and  $P'_3$  are their projections into world space, and  $P_c$  is the position of the reference image's camera as shown in Figure 5-21, we assume that the following conditions hold:

$$\begin{aligned} P_0 &= P'_0 \\ P_2 &= P'_2 \\ (P_2 - P_0) \cdot (P'_1 - P_0) &= 0 \\ (P_0 - P_2) \cdot (P'_3 - P_2) &= 0. \end{aligned}$$

Expanding the dot products, we solve for  $p_1$  and  $p_3$ ,

$$\begin{aligned} p_1 &= \frac{P_2P_0 + P_0^2}{(P_2N_1) - (P_0N_1)} \\ p_3 &= \frac{P_0P_2 + P_2^2}{(P_0N_3) - (P_2N_3)}. \end{aligned}$$

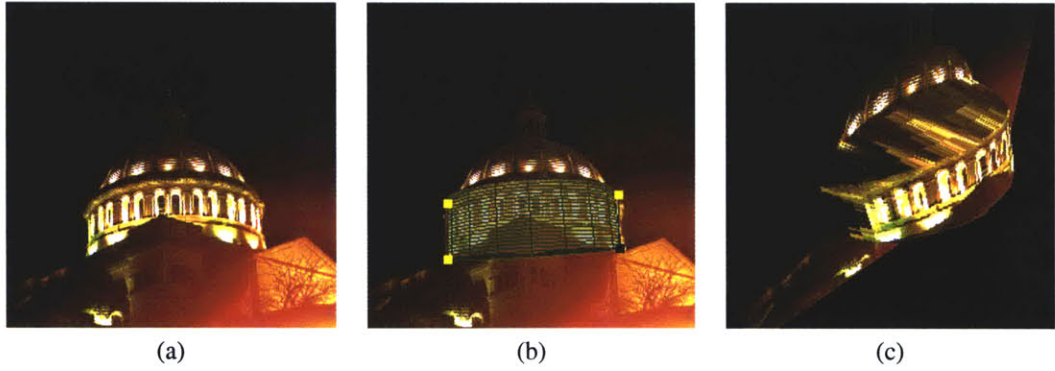


FIGURE 5-20: Cylinder Tool Example. (a) Original image. (b) Cylinder geometry. (c) Applied depth.

The x and y rotation vectors of the cylinder are  $(P_2 - P_0)$  and  $(P_1 - P_0)$  respectively. We assume that the cross-section of the cylinder is a circle, that is, the cylinder is symmetric about its vertical axis. Therefore, the z rotation vector is the cross product of the x and y rotation vectors. The scale factors are the magnitudes of the rotation vectors (before normalization). The center, or translation factor  $C$ , of the cylinder is,

$$C = P_0 + \frac{P_2 - P_0}{2} + \frac{P_1 - P_0}{2}.$$

To orient the cylinder in the direction of the ground plane normal, we can solve for  $P'_1$  as the intersection of  $\overline{P_cP_1}$  and  $\overline{P_0P_{0n}}$ , where  $P_{0n}$  is defined as in Equation 5.6.2, using Appendix D.2. We then solve for  $P'_3$ , similarly.

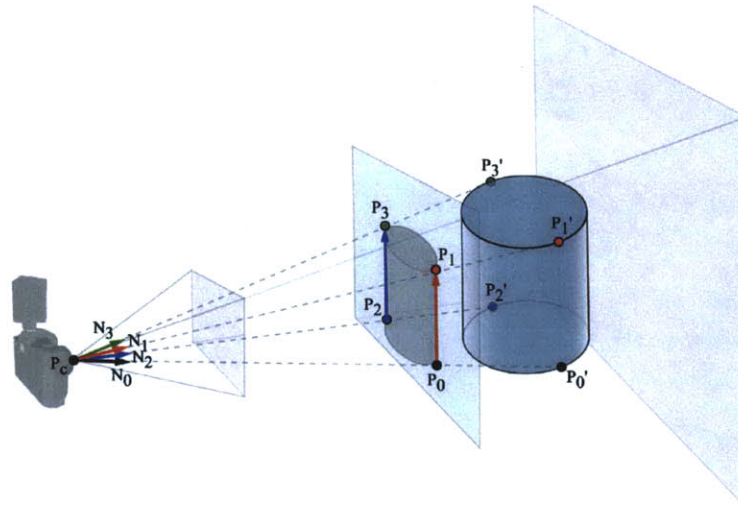


FIGURE 5-21: Cylinder Tool Interaction. The user draws the two edges of a cylinder.

## 5.7 HEIGHTFIELD

The heightfield tool is a mesh constructed from a set of points whose heights can be interactively adjusted by the user. Heightfields are useful for creating landscapes that contain hills and mountains (Figure 5-22).

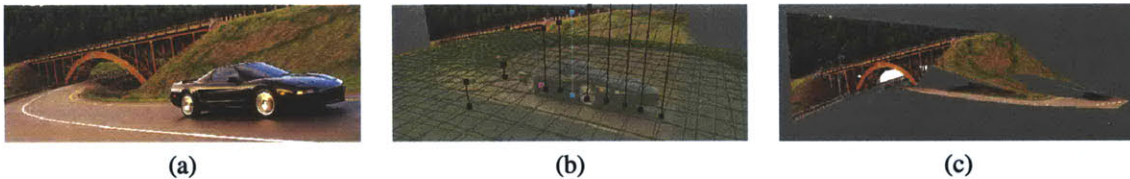


FIGURE 5-22: Heightfield Tool Example. (a) Original image. (b) Heightfield geometry. (c) Applied depth.

The heightfield begins as a planar mesh, by default, the ground plane. The user modifies the heightfield by first picking points in the reference image. This is the Reference Image Alignment step (Figure 5-23). The picked points correspond to boundaries, or silhouette edges, such as those between a hill and the sky. Once these points are defined, they can be adjusted by translating them in the plane parallel to the reference image. The user must then perform Depth Alignment for each of the points by dragging each point along the ray from its current position to the camera position. The height will scale proportionally based on the distance from the camera to the point. The ground plane can also be toggled on to assist the user in Depth Alignment. As each point is aligned, the mesh will be updated based on the bounds of the ground plane and the user-defined height points. This mesh is constructed by a variant of a 2D scattered data interpolation method, Shepard's Method, since the user-defined points are not uniformly spaced and our data points are in 3D. The system first defines a gridded surface based on the bounds of the ground plane. Each grid point is then raised by an interpolating function  $f(x)$  based on a weight determined by the inverse distance  $d$  from each user-defined point and the height  $h$  of each user-defined point:

$$f(x) = \sum_{i=1}^N w_i(x) f_i,$$

with weight function  $w_i$ ,

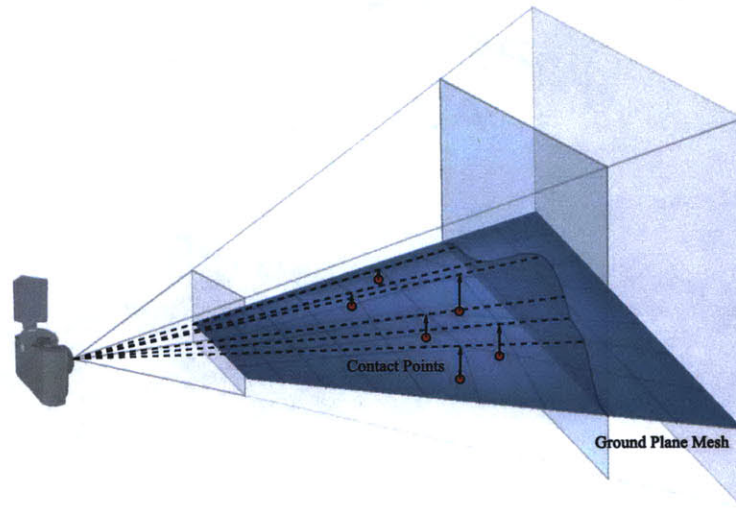


FIGURE 5-23: Heightfield Tool Interaction. The user picks points of contact in the reference image that correspond to boundaries, for example, between the sky and a hill. These points can then be modified, either by translating them within the plane parallel to the reference image, or by adjusting their heights and depths.

$$w_i(x) = \frac{\sigma_i(x)}{\sum_{j=1}^N \sigma_j(x)} h_i(x)$$

$$\sigma_i(x) = \frac{1}{d_i(x)}.$$

The system also applies two smoothing factors. First, the system applies a damping function  $\lambda$  in order to accentuate local regions around the user-defined points. This ensures that there will be smoother interpolation between user-defined points that are close together. The damping function is based on a given neighborhood distance  $R$  from each user-defined point:

$$w_i(x) = w_i(x)\lambda_i(x)$$

$$\lambda_i(x) = \left(1 - \frac{d_i}{R_i}\right)^n.$$

Large values of  $n$  will increase the damping around each point. This will force each point to have a lesser effect on areas further away. We arbitrarily choose  $n = 3$ . Second, we eliminate discontinuities and flat spots in the surface by modifying our weight function with:

$$\sigma_i(x) = \frac{1}{d_i(x) + c_i},$$

where  $c_i \neq 0$ . From these grid points, the system creates quadrilateral patches. The sampling of the grid points is user-defined, since in some cases, the user may want the heightfield tool to be coarser and more rugged for applications like steep mountain gorges or smoother for small hills and valleys.



## 5.8 INTRUSION AND EXTRUSION

This tool is a special case of the horizontal and vertical plane tools. Intrusion and extrusion tools allow the user to drag a selection area in the direction specified by the normal to the selection area. This type of operation is useful for defining detailed façades by specifying the depth of windows, overhangs, balconies, and doors (Figure 5-24).

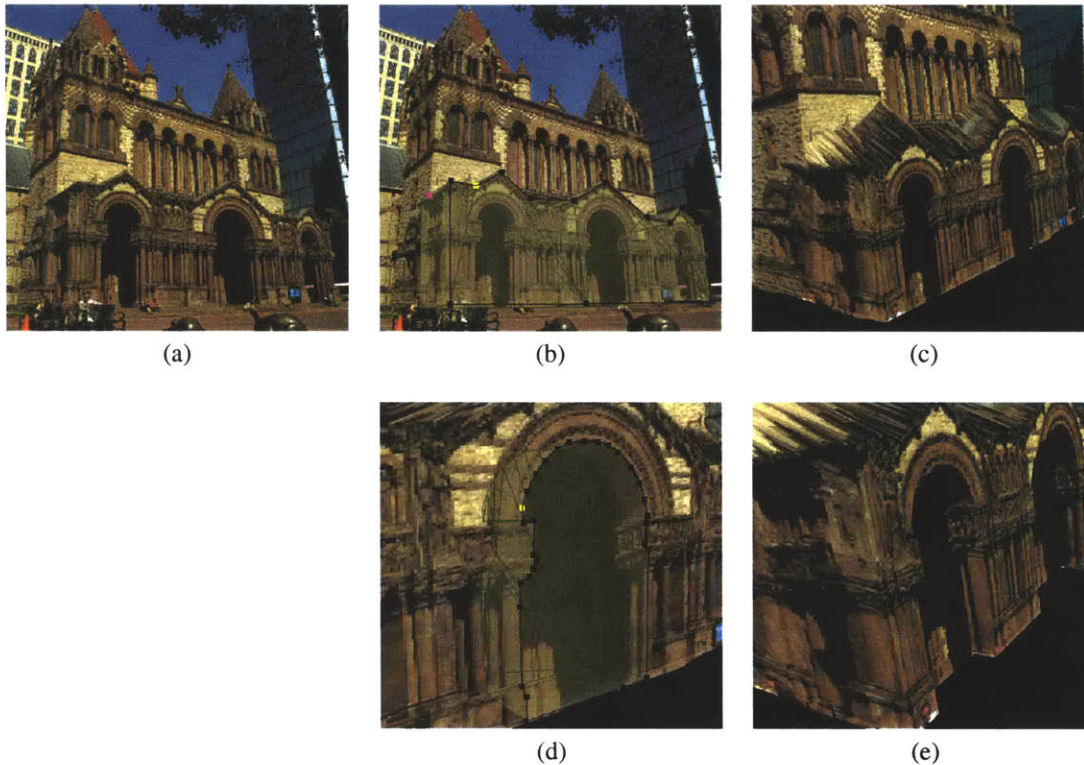


FIGURE 5-24: Intrusion and Extrusion Tools Example. (a) Original image. (b) Extrusion geometry. (c) Applied extrusion depth. (d) Intrusion geometry. (e) Applied intrusion depth.

Alignment for intrusion and extrusion tools rely on what the user can see in the reference image. For the intrusion tool, the user performs Reference Image Alignment by selecting the area with points in the reference image that will be intruded. The first three user-defined points will define a plane on which the user picks points. The user can also use Catmull-Rom Interpolation between user-defined points to obtain smooth curves. For Depth Alignment, the user then interactively drags the selected area inwards, until the correct intruded depth is achieved. As the user drags the selected area inwards, the system displays triangles extruded along the direction of the intrusion and also tessellates the planar area bounded by the selection with triangles (Figure 5-25). The system then uses these triangles to update depth by reading the OpenGL depth buffer.

For Reference Image Alignment in the extrusion tool, the user also makes a selection area. This selection area, however, is not the area in the reference image to be extruded. This area is not visible. Instead, the user selects the part of the reference image that appears extruded from some surface. Prior to this step, the user must first pick a point  $P_0$  that defines the reference depth from which the selection area should be extruded. This is Depth Alignment. We intersect the line  $\overline{P_0N}$  with a plane  $P$  defined by the first point of the selection area  $P_1$ , the camera position  $P_c$ , and another user-defined point  $P_2$ , where  $N$  is the normal to the selection area. The three points needed to define  $P$  cannot be colinear so we define  $P_2$  as:

$$P_2 = P_1 + (-N \times (P_1 - P_0)).$$

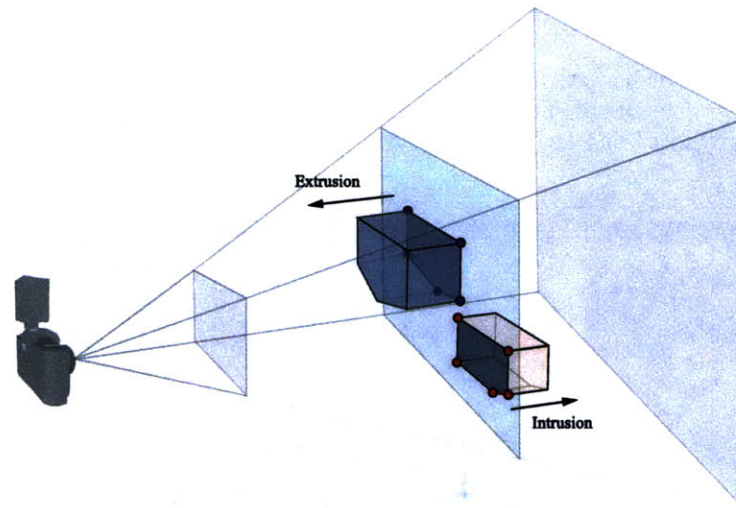


FIGURE 5-25: Intrusion and Extrusion Tools Interaction. The user picks points on the reference image and specifies an intrusion or extrusion depth by dragging.

The distance of this intersection indicates the perpendicular extrusion depth  $d$  (Figure 5-26) for each of the points in the selection area.

Depth buffer reading requires we read the depth buffer twice because of overlapping triangles (Figure 5-27). We read the depth buffer once with backface culling enabled, and a second time with it disabled. The former can result in an incorrect pixel depth value, as does  $P_1$  in Figure 5-27 (a), if there is an overlapping triangle with its surface normal facing away from the camera. The latter will always take the closest intersection as the depth value. Therefore, at each pixel, if the depth from the two depth buffers is identical, we update our data structures with the identical value. Otherwise, we take the depth value resulting from backface culling disabled.



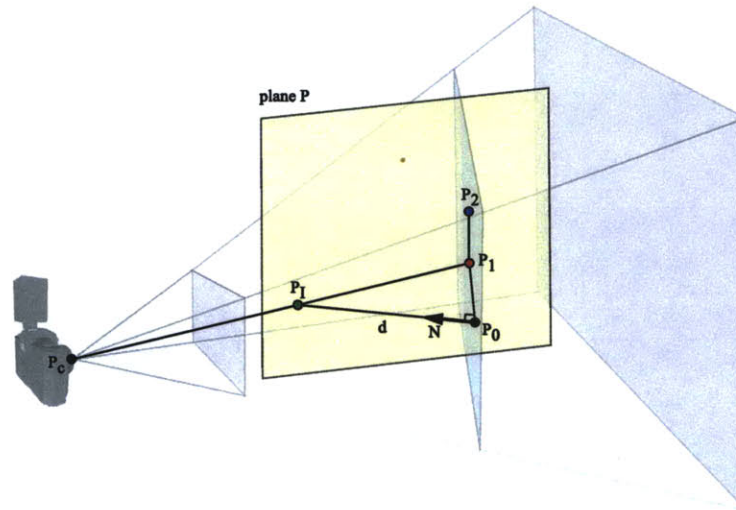


FIGURE 5-26: Extrusion Geometry. For the extrusion tool, the user picks points in the reference image that lie on extruded geometry. These points are a perpendicular distance  $d$  from the reference image. We determine  $d$  by finding the intersection  $P_1$  between  $\overline{P_0N}$  and plane  $P$ . Three points  $P_1$ ,  $P_2$ , and  $P_c$  define  $P$ .

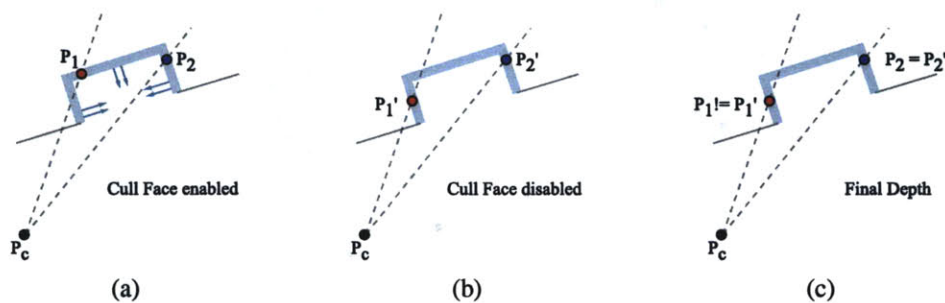


FIGURE 5-27: Intrusion and Extrusion Depth Update Procedure. (a) If we read the OpenGL depth buffer as usual with backface culling enabled, the depth at pixel  $P_1$  is incorrect. (b) We must read the depth a second time with backface culling disabled and (c) compare the results to obtain the correct depth. The depth at pixel  $P_2$  is correct because there are no overlapping triangles at this pixel.



---

## FREEFORM TOOLS

---

While images often contain geometric and architectural objects, we do not wish to limit the range of our depth acquisition tools to this class of objects. In this section, we describe two Freeform Tools that address organic shapes and objects that can be generalized. One tool operates through an image-based method, while the other functions through geometric means. The first, the organic shapes tool, assumes that a layer or selection area is a complete entity whose shape is bulgy. This idea is similar to Williams [48] and is related to the Teddy fast-modeling system [22]. The second tool is the shape template tool. This tool is similar to the shape primitives described in Chapter 5 in that it associates a 3D model to its projection onto the image. In this case, however, we generalize the shape primitive to be any arbitrary shape. By determining the correct transformation for this shape primitive to approximate the pose of its projection, we can achieve convincing depth for any generic shape.

### 6.1 ORGANIC SHAPES TOOL

We propose a tool that assigns depth using level set methods [43]. It is particularly useful for organic shapes since it gives them a bulgy appearance by specifying more distant depth at the boundary of the object and closer depth in the center of the object (Figure 6-1).

In our level set method, we use erosion techniques [44]. First, every pixel has a distance counter  $d$  that is initialized to 0. This distance counter will be used to determine the depth of the pixel. We then categorize each pixel as either a boundary pixel or not, using the following criteria (Figure 6-2):

- If the pixel borders a pixel whose selection value is 0.0, it is a boundary pixel.
- If the pixel borders a pixel whose alpha value is 0.0, it is a boundary pixel.

Next, we process each pixel by incrementing its distance counter if it is not adjacent to a boundary pixel. If it is adjacent to a boundary pixel, the current pixel also becomes a boundary pixel. We continue this operation until all pixels become boundary pixels. Finally, we update the depth of each pixel using their respective distance counters. The user also specifies a depth range  $r$  between which the depth values can vary. We then use the normalized distance to the centroid:

$$d' = 1 - d_{bound} / d_{boundmax} ,$$

and update depth according to,

$$z = z - r\sqrt{1 - d'^2} .$$

This formula was chosen because it assigns a spherical shape to a disk under orthographic projection.

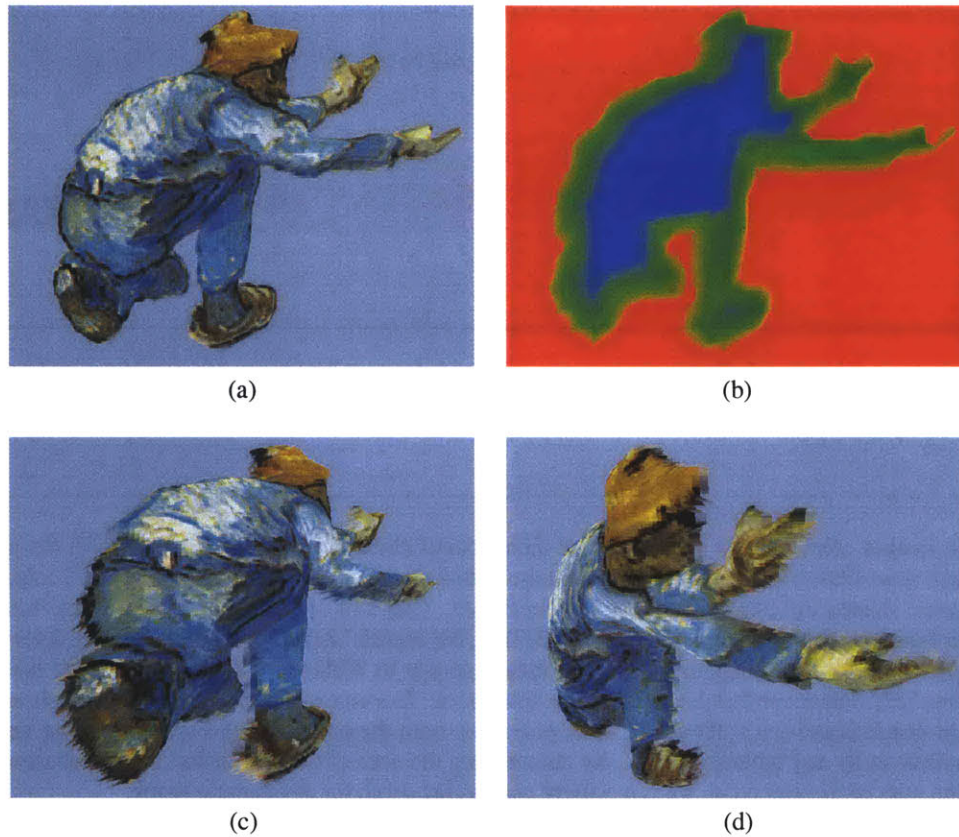


FIGURE 6-1: Organic Shapes. (a) Single layer (b) Depth map through level set methods. (c) and (d) Applied depth.

## 6.2 SHAPE TEMPLATE TOOL

The shape template tool takes as input an arbitrary generic 3D model that the user can align to the reference image. This tool is particularly useful for generating depth for elements in the reference image that can be generalized. Shape templates are therefore suitable for faces and cars (Figure 6-3). Our implementation currently accepts 3D geometry in the form of Open Inventor and VRML 2.0 files, but this can be easily extended to other 3D file formats.

Shape templates require alignment as described in Chapter 5. The user can perform Reference Image Alignment in two ways. The first method is through traditional 3D modeling interaction. The second method is through point correspondence and image warping as follows. First, the user specifies feature points on the 3D model. Next, the user picks points on the 2D reference image that correspond to these 3D feature points. The system then approximately aligns the 3D model with a transformation that minimizes the error between the projection of the 3D points onto the reference image and their corresponding 2D points. Finally, we warp the 2D depth image acquired from the point correspondence step in order to match the point correspondences exactly.

### 6.2.1 POINT CORRESPONDENCE ALGORITHM

The user specifies correspondences between a set of points  $Q$  in the image and a set of points  $P$  on the 3D model. If a point  $q_i$  in the image corresponds to a point  $p_i$  on the 3D model, they are related in the following manner by similar triangles (Figure 6-4):

$$\frac{p_i}{p_{iz}} = \frac{q_i}{f}$$

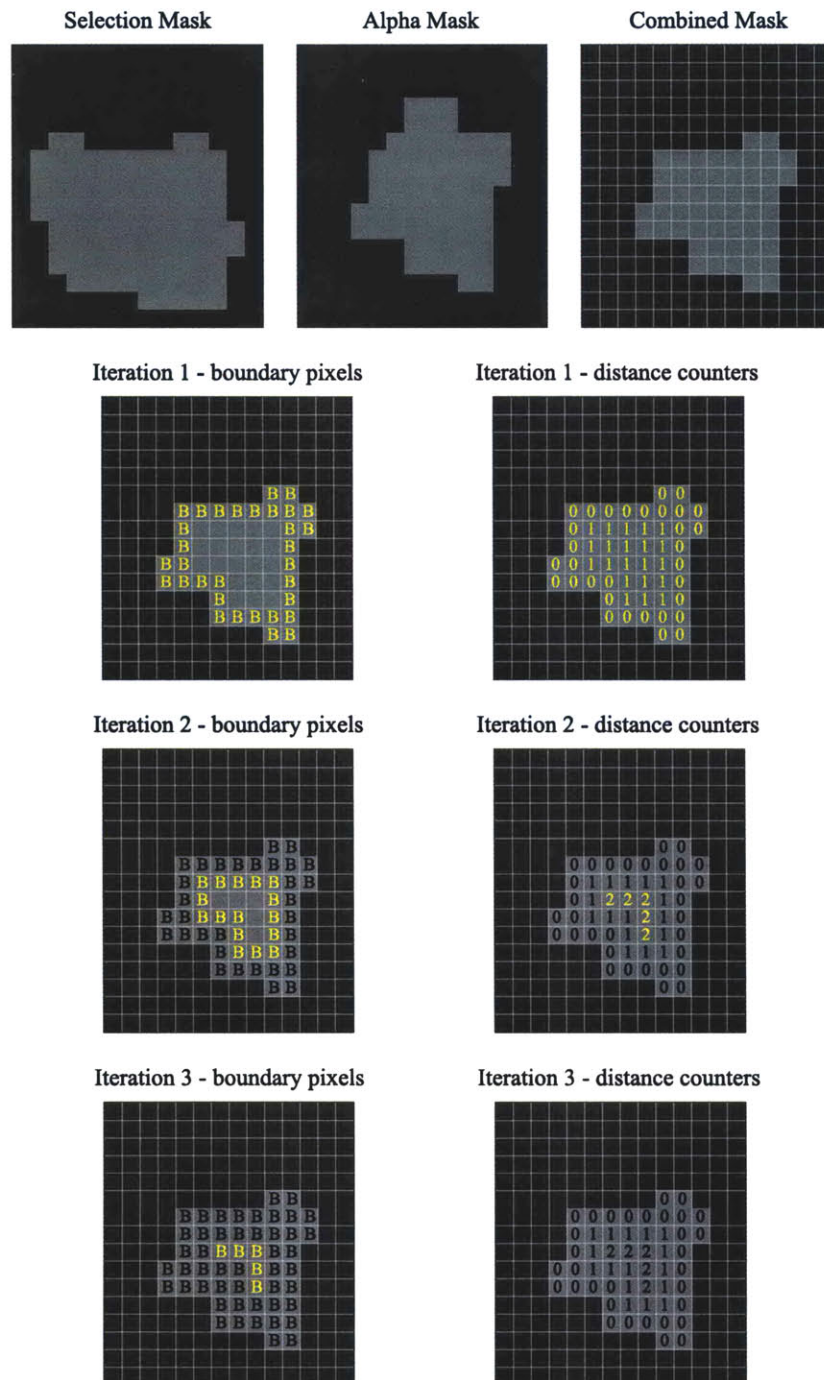


FIGURE 6-2: Organic Shapes Pixel Update Procedure. First, we initialize all pixels' distance counters to 0. At each iteration, we find the boundary pixels and increment the distance counter of each pixel that borders a boundary pixel. Each of these border pixels becomes a boundary pixel for the next iteration. This process continues until all pixels within the mask are boundary pixels.



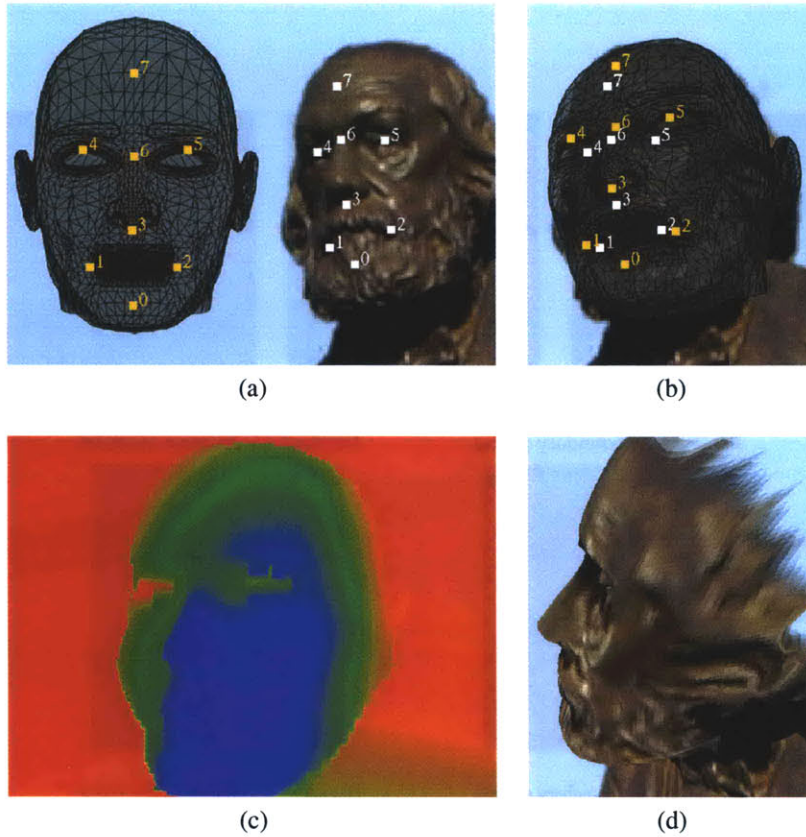


FIGURE 6-3: Shape Template. (a) User specified correspondence points. (b) Transformation after LM optimization. (c) Depth map. (d) Face after depth is applied.

We form the general projection equation, whose unknowns are the parameters of the rotation and scale matrix,  $R$  and  $S$ , that transforms  $p_i$  to a position at which its corresponding point  $q_i$  projects through from the reference camera,

$$\frac{C_i}{C_{iz}} = \frac{q_i}{f}, \quad (6.1)$$

where,

$$C_i = RS(p_i - p_0) + p_0.$$

There are two important points to note. First, we avoid solving for translation by assuming that  $p_0 = q_0$ . Second, we solve for  $R$  and  $S$  separately because we do not want shear in the transformation. A shearing transformation can lead to very undesirable results since the 3D geometry will be warped. If we define  $R$  and  $S$  as a quaternion and a scale matrix, respectively,

$$R = \begin{bmatrix} 1 - 2b^2 - 2c^2 & 2ab - 2sc & 2ac + 2sb \\ 2ab + 2sc & 1 - sa^2 - 2c^2 & 2bc - 2sa \\ 2ac - 2sb & 2bc + 2sa & 1 - 2a^2 - 2b^2 \end{bmatrix}$$

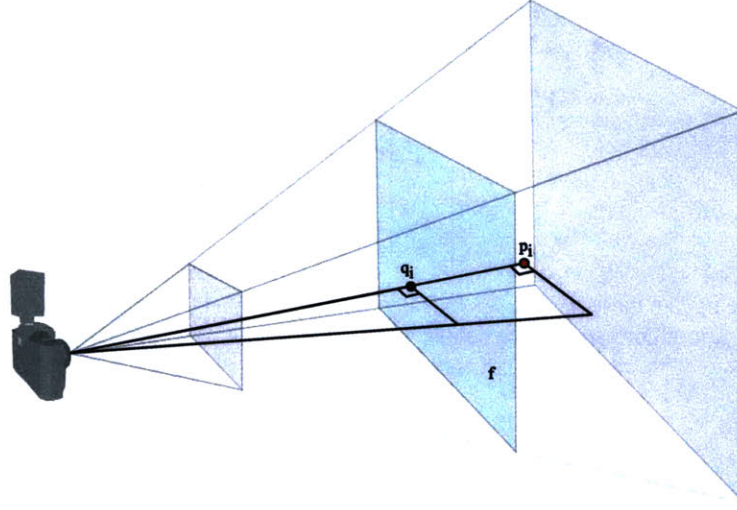


FIGURE 6-4: Similar Triangles.

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix},$$

we therefore have a total of seven unknown parameters, four rotation and three scale factors, to solve for. We use Levenberg-Marquadt (LM) optimization [38] to minimize the error between the points and extract the unknown parameters. LM optimization operates as follows. If we expand Equation 6.1,

$$RS(p_{ix} - p_{0x}) + p_{0x} = \frac{q_{ix}}{f} RS(p_{iz} - p_{0z}) + p_{0z} \quad (6.2)$$

$$RS(p_{iy} - p_{0y}) + p_{0y} = \frac{q_{iy}}{f} RS(p_{iz} - p_{0z}) + p_{0z} \quad (6.3)$$

$$RS(p_{iz} - p_{0z}) + p_{0z} = \frac{q_{iz}}{f} RS(p_{iz} - p_{0z}) + p_{0z}, \quad (6.4)$$

substitute matrices  $R$  and  $S$ , and simplify, Equations 6.2, 6.3, and 6.4 reduce to:

$$A_i = \frac{q_{0x} - q_x((2ac - 2sb)s_x(p_x - p_{0x}) + (2bc + 2sa)s_y(p_y - p_{0y}) + (1 - 2a^2 - 2b^2)s_z(p_z - p_{0z}) + q_{0z})/f + (1 - 2b^2 - 2c^2)s_x(p_x - p_{0x}) + (2ab - 2sc)s_y(p_y - p_{0y}) + (2ac + 2sb)s_z(p_z - p_{0z})}{(2ab + 2sc)s_x(p_x - p_{0x}) + (1 - 2a^2 - 2c^2)s_y(p_y - p_{0y}) + (2bc - 2sa)s_z(p_z - p_{0z})} \quad (6.5)$$

$$B_i = \frac{q_{0y} - q_y((2ac - 2sb)s_x(p_x - p_{0x}) + (2bc + 2sa)s_y(p_y - p_{0y}) + (1 - 2a^2 - 2b^2)s_z(p_z - p_{0z}) + q_{0z})/f + (2ab + 2sc)s_x(p_x - p_{0x}) + (1 - 2a^2 - 2c^2)s_y(p_y - p_{0y}) + (2bc - 2sa)s_z(p_z - p_{0z})}{(2ab + 2sc)s_x(p_x - p_{0x}) + (1 - 2a^2 - 2c^2)s_y(p_y - p_{0y}) + (2bc - 2sa)s_z(p_z - p_{0z})} \quad (6.6)$$

$$1 = 1. \quad (6.7)$$

We use  $A_i$  and  $B_i$  to define the error  $E$  between  $q_i$  and the projected point  $p_i$  as:

$$E = \sum A_i^2 + B_i^2.$$



In order to minimize this error, we iterate until this error drops below a certain threshold. First, we initialize a vector  $V$  with the unknown parameters of  $R$  and  $S$  as identity transformations:

$$V = \begin{bmatrix} a \\ b \\ c \\ s \\ s_x \\ s_y \\ s_z \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} .$$

At each iteration, we update these parameters by calculating the gradient  $G$  and its corresponding hessian matrix  $H$  as follows. For every pair of correspondence points, we calculate  $G_a$  and  $G_b$ , and increment  $G$  and  $H$ :

$$\begin{aligned} G & += G_a + G_b \\ H & += G_a G_a^T + G_b G_b^T . \end{aligned}$$

$G_a$  and  $G_b$  are defined as follows:

$$G_a = \frac{\partial A_i}{\partial V_k} \quad G_b = \frac{\partial B_i}{\partial V_k} .$$

After obtaining  $G$  and  $H$  for the current iteration, we update  $V$  as follows:

$$V' = V_0 + \Delta V ,$$

where

$$\Delta V = H^{-1} G .$$

When  $E$  falls below some pre-defined threshold, the optimization terminates, and we assign the values in  $V$  as the parameters for  $R$  and  $S$ . This completes the point correspondence algorithm for the shape template tool.

### 6.2.2 DEPTH IMAGE WARPING ALGORITHM

Since LM optimization gives us an approximation of  $R$  and  $S$ , the set of points  $Q$  will not project directly through the set of points  $P$  on the model. We therefore apply 2D warping techniques as described by [29]. The process is as follows (Figure 6-5). First, we update the depth channel for the current layer according to the transformed 3D object. This gives us depth according to the LM optimization. Then, we project the set of points  $P$  to the reference image to form a second set of points  $T$ . We perform 2D Delaunay Triangulation using [19] with this set of points  $T$  and apply the reference depth channel to these triangles as texture. See Appendix C.3 for details on extracting triangles from the winged-edge data structure. Next, we morph these triangles by translating each point  $t_i$  to its corresponding point  $q_i$ . A critical note is that we do not alter texture coordinates for these triangles. Finally, we update the depth channel by setting the interactive camera to the reference camera, reading the pixel values from the texture values of the resulting triangulation, and assigning these values to the depth channel.

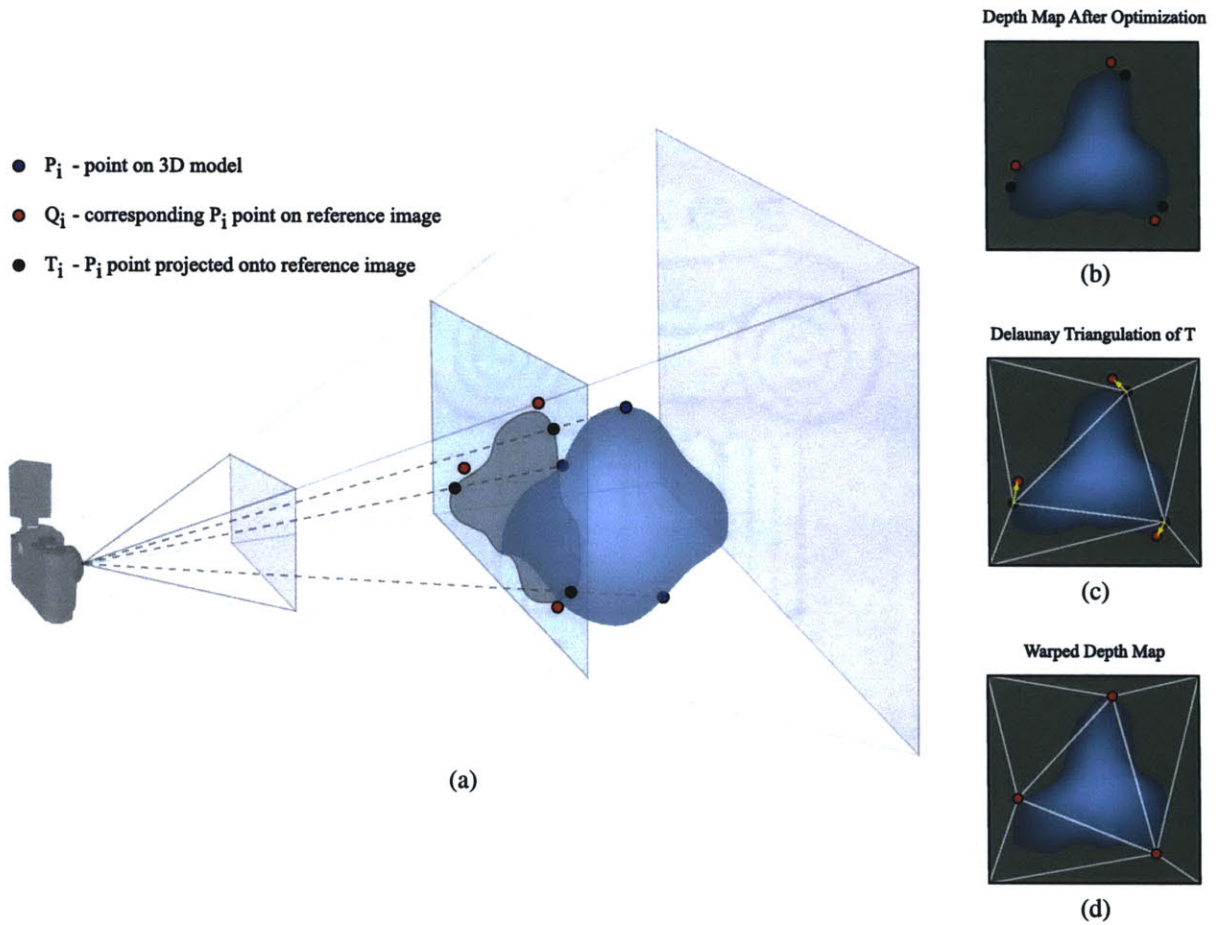


FIGURE 6-5: Depth Image Warping Algorithm. (a) We would like  $P_i$  to project onto the reference image through the user-defined  $Q_i$ . The Point Correspondence Algorithm gives us an approximation of this transformation. To achieve exact correspondence, we (b) generate the depth map from the optimization, (c) triangulate the projected points  $T_i$  using Delaunay Triangulation, and (d) warp these points to their desired projections and update the depth values.



---

# RESULTS

---




We show a series of examples that exhibit the versatility of our system. We will demonstrate that our system works well not only for real photographs but for images that do not adhere to the rules of perspective. As a result, our system can be useful for quickly generating walkthroughs or new views of an image for design purposes.

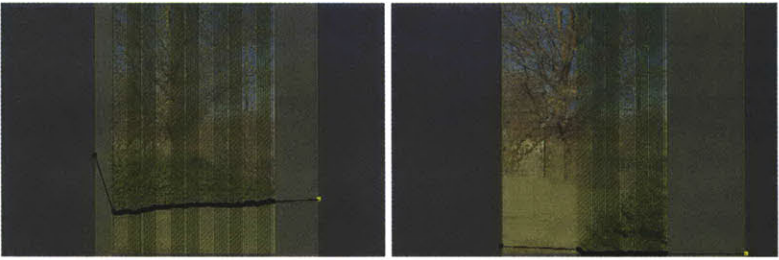



First, we begin with a step-by-step tutorial for specifying depth from a single photograph. We then continue with two photographs of architectural scenes that exhibit extensive use of the Geometry Tools. The third set of examples deals with paintings. These images rely heavily on the use of Image-Based and Freeform Tools since artists are not forced to adhere to strict perspective rules. Nevertheless, depth can be acquired from this class of images because our representation does not rely on reconstructing the true geometry of a scene. Finally, we conclude with cubic panoramas. These panoramas have been generated from a series of images by aligning the images to one another, sewing and stitching them to each other, and then finally re-projecting the stitched image onto the interior faces of a cube. This results in six images whose camera center is the center of the cube. We then apply depth to each of the images and composite them to complete the scene geometry.

We hope to show from these examples the great versatility in the types of images that can be used in our system. This allows even the novice user to achieve convincing results from any image by specifying crude depth.

## 7.1 STATUE

In this example, we present a step-by-step tutorial for the novice user. This photograph was taken by a mega-pixel digital camera, and the image has been down-sized to a pixel resolution of 1000 x 1500. We will use the system's default field of view of 30 degrees. Specifying an accurate field of view is not crucial in this example because none of the tools used in this scene will require an accurate field of view measurement. Segmentation and filling holes was the most time consuming step, as will be evidenced by each of the subsequent examples.

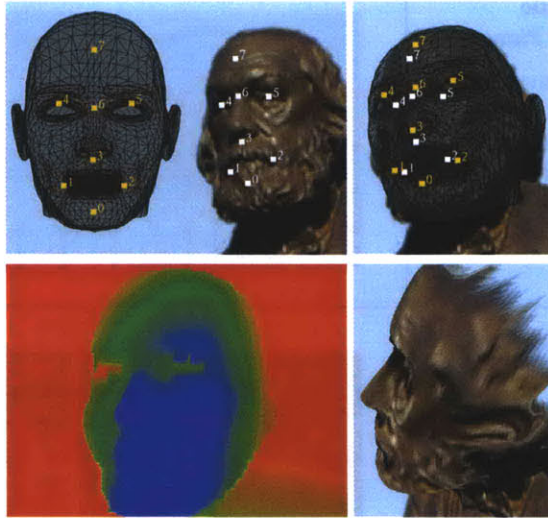
|  |  |
|--|--|
| <p><b>1. Initial Image</b></p> <p>Begin with a single image of a statue and its surroundings.</p>  |  <p>Pixel Resolution : 1000 x 1500<br/>         Number of Layers : 6<br/>         Segmentation Time : 2 hours<br/>         Depth Specification Time : 5 minutes</p> |
| <p><b>2. Layering and Filling Holes</b></p> <p>Segment the image into layers as outlined in red (left) and fill holes by copying from similar regions (right).</p> |   |
| <p><b>3. Ground Plane</b></p> <p>Pick the horizon line to specify the ground plane (left) and apply the depth of this geometry to the ground layers (right).</p>   |    |

|  |  |
|--|--|
| <p><b>4. Automatic Vertical Planes</b></p> <p>Use the automatic vertical plane tool to acquire depth for the layers of the foreground and background bushes.</p>   |    |
| <p><b>5. Cylinder Primitive</b></p> <p>Extract depth for the statue base by drawing the left and right edges of a cylinder (left) and applying the depth of this geometry to the statue base layer (center). The right image is the resulting depth map.</p> |    |
| <p><b>6. Vertical Plane</b></p> <p>Use the vertical plane tool to place the statue on top of its base by drawing a vertical plane that extends from the base of the statue to infinity.</p>  |  |
| <p><b>7. Organic Shape</b></p> <p>Generate a bulgy shape for the statue by using the organic shapes tool on that layer. The depth map is shown on the right.</p>   |  |



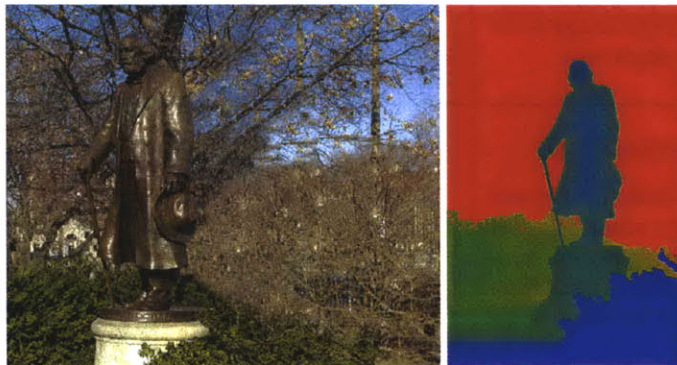
### 8. Shape Template

Align a generic 3D face model to the statue's face by picking correspondence points (top left). Acquire depth from the optimized position of the 3D model (top right). The resulting depth map is shown in the bottom left, with a new view in the bottom right.



### 9. Final Result


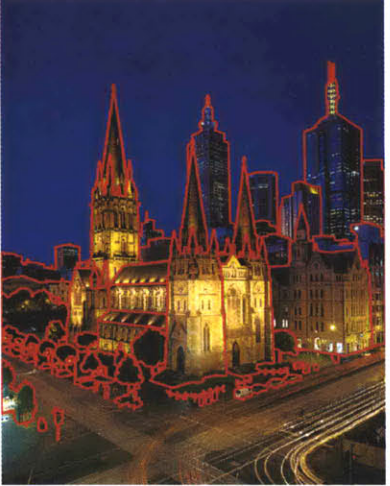
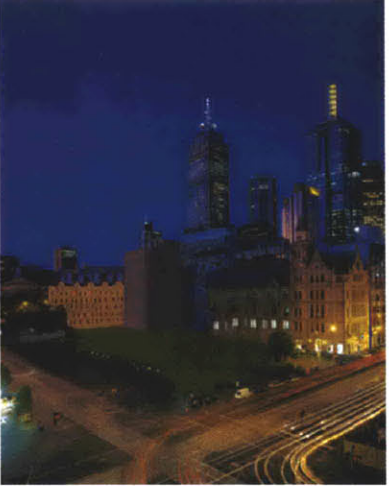
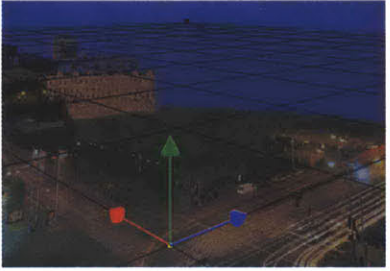

An alternative view (left) and the final depth map (right).


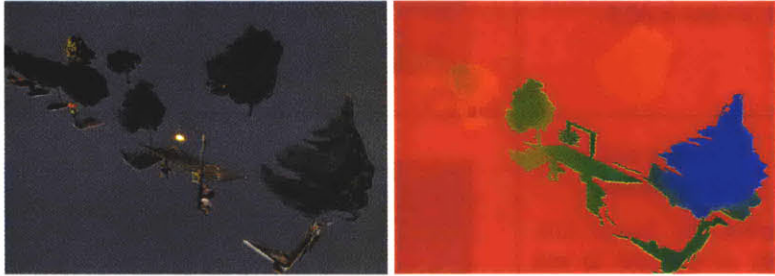
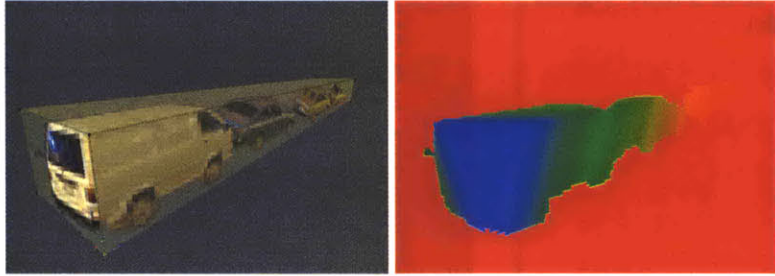


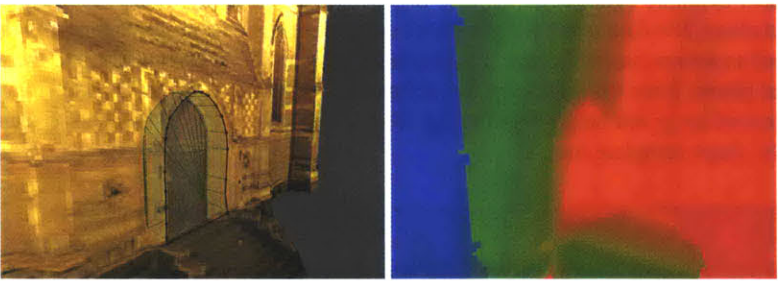
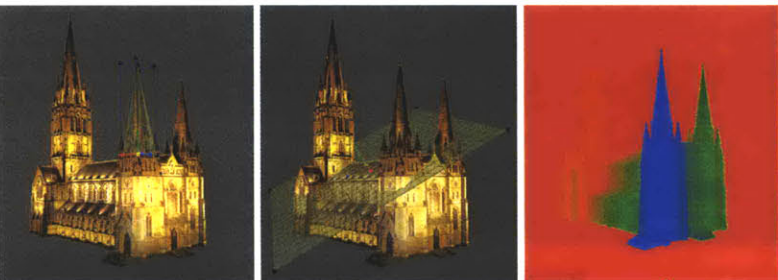
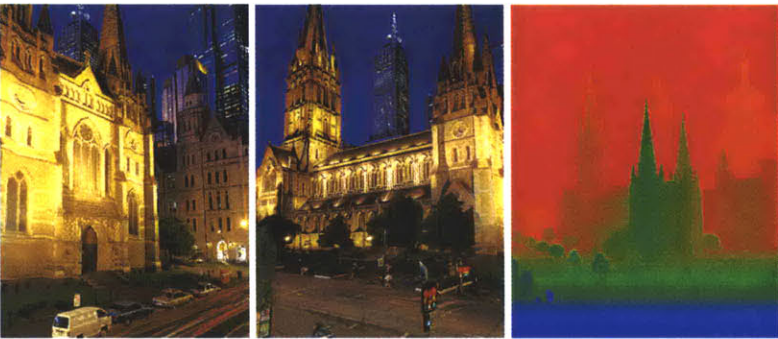


## 7.2 CATHEDRAL

The following example illustrates extensive use of Geometry Tools, which are applied to a photograph that contains many orthogonal and architectural elements. As a result, we require a more precise field of view estimation, as opposed to the previous example. With a precise field of view, the shape primitives will be aligned properly with features in the image. This, however, is not necessary for layers in the scene whose depth is not derived from shape primitives. For example, the trees are simply vertical planes that have been given a bulgy appearance using the organic shapes tool. Similar to the previous example, we show many of the steps taken to acquire depth from this image.

|   |   |   |
|---|---|---|
| <p><b>1. Initial Image and Field of View Estimation</b></p> <p>Begin with a single image of a cathedral. Estimate the field of view by drawing three orthogonal pairs of lines. The calculated field of view is 60.5.</p> |    | <p>Pixel Resolution : 1280 x 960<br/>         Number of Layers : 61<br/>         Segmentation Time : several days<br/>         Depth Specification Time : 6 hours</p> |
| <p><b>2. Layering and Filling Holes</b></p> <p>Segment the image into layers as outlined in red (left) and fill the corresponding holes by copying from similar regions (right).</p>                                      |  |   |
| <p><b>3. Ground Plane</b></p> <p>Define the ground plane by drawing two lines that follow the perpendicular road intersection in the image (left). The depth map is shown on the right.</p>                               |  |   |

|  |  |
|--|--|
| <p><b>4. Automatic Vertical Planes</b></p> <p>Give each layer a rough billboard depth by utilizing the automatic vertical plane tool. For those layers without a clear point of contact with the ground, estimate this contact point and define a vertical plane. The image on the left shows the automatic vertical plane tool applied to the cathedral layer. On the right is the resulting depth map.</p> |    |
| <p><b>5. Organic Shapes</b></p> <p>Add bulgy depth to the trees and bushes using the organic shapes tool (left). The depth map is shown on the right.</p>  |   |
| <p><b>6. Cube Primitive</b></p> <p>Improve the depth for the cars and street furniture with the cube tool (left). The resulting depth map (right).</p>   |  |

|   |  |
|---|--|
| <p><b>7. Intrusion and Extrusion</b></p> <p>Use the intrusion and extrusion tool to add relief to the façade windows and buttresses of the church (left). Again, the depth map is shown on the right.</p>   |    |
| <p><b>8. Pyramid and Plane Primitive</b></p> <p>Align the pyramid and the plane tool to the turrets (left) and the roofs (center), respectively, to define the depth for those slanted elements. The depth map for the entire church is on the right.</p> |    |
| <p><b>9. Final Result</b></p> <p>New views (left and middle) and the final depth map (right).</p>   |  |



### 7.3 HEARST CASTLE

This next example again demonstrates heavy use of Geometry Tools. The depth of the pool is taken by setting the ground plane. We then make the plane tool coincident with the ground plane and raise it slightly so that it matches with the pool's surrounding walkway. The depth of the temple, its colonnade, and the peristyle is specified by establishing vertical planes from the walkway. We refine the curvature of the columns using the cylinder tool. The bushes in the background are given bulgy depth using the organic shapes tool. The face of the statue in the foreground is matched with the shape template tool.



Initial Image

|                          |   |            |
|--------------------------|---|------------|
| Pixel Resolution         | : | 1280 x 960 |
| Number of Layers         | : | 19         |
| Segmentation Time        | : | 6 hours    |
| Depth Specification Time | : | 1 hour     |



Layering



Filling Holes



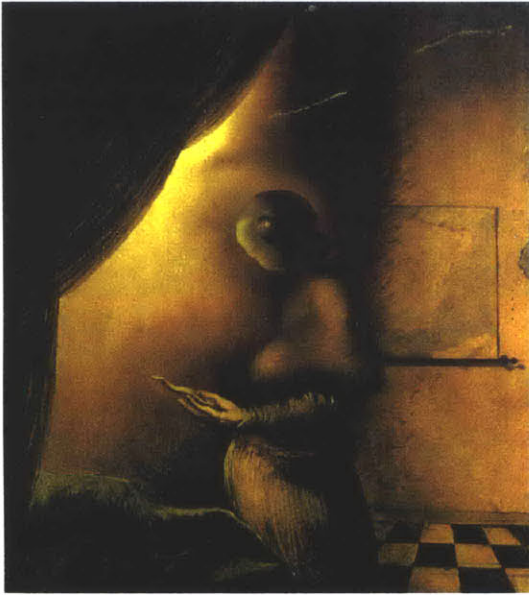
New View



Final Depth Map

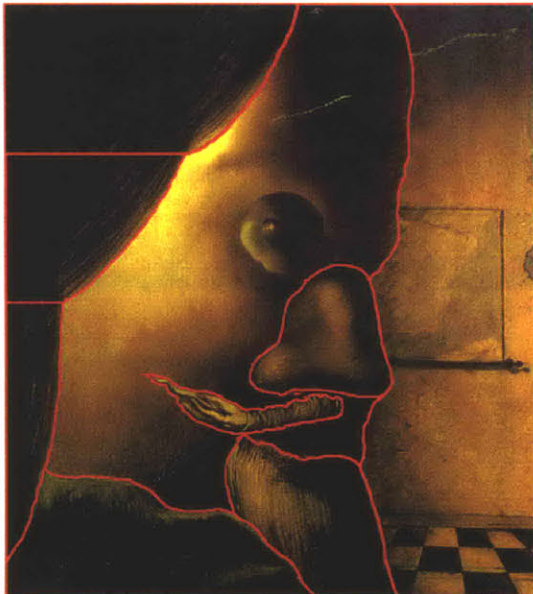
## 7.4 PAINTING BY DALI

The next set of examples are paintings. This example illustrates the use of combining Geometry Tools with Painting Tools. The depth for the face was first initialized with the shape template tool and then refined with the chisel/paint tool. The depth for the woman was generated using the organic shapes tool. As evidenced by this example, the user does not have to reconstruct the actual geometry, especially in situations in which the image is ambiguous.

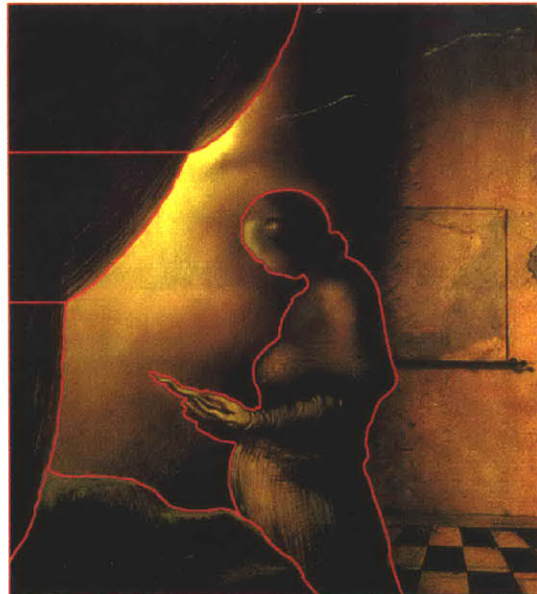


Initial Image

|                          |   |             |
|--------------------------|---|-------------|
| Pixel Resolution         | : | 1000 x 1124 |
| Number of Head Layers    | : | 3           |
| Number of Woman Layers   | : | 6           |
| Segmentation Time        | : | 5 hours     |
| Depth Specification Time | : | 2 hours     |

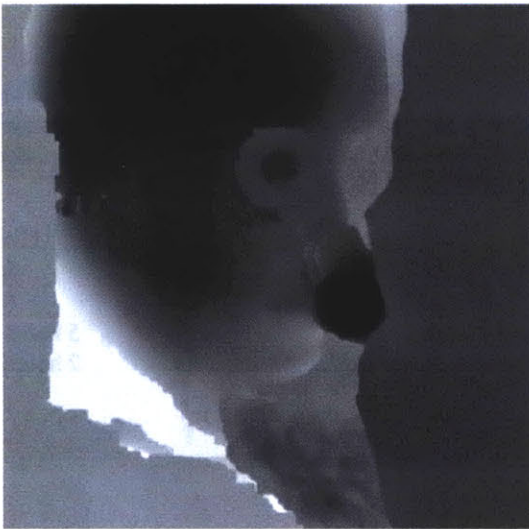


Face Layering



Woman Layering





Chiseling



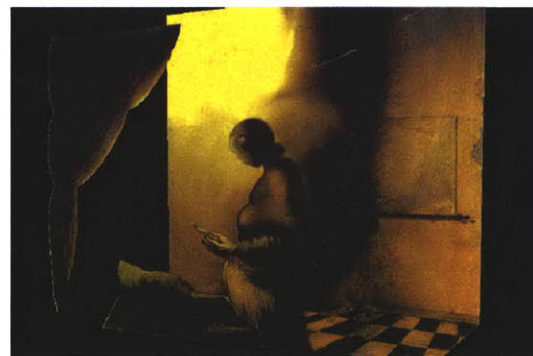
Smoothing



Head Depth



Woman Depth



New Views

### 7.5 PAINTING BY VAN GOGH

This painting shows the versatility of Geometry Tools by demonstrating that they can be applied to images that do not exactly follow the rules of perspective. We began by specifying the ground plane as the point at which the house touches the ground. We can assume that the vertical direction is directly upwards, since the subsequent vertical planes for each of the layers will be relative to this direction. The shape of the man, woman, child, and bushes was bulged using the organic shapes tool.



Initial Image

|                          |   |           |
|--------------------------|---|-----------|
| Pixel Resolution         | : | 640 x 504 |
| Number of Layers         | : | 14        |
| Segmentation Time        | : | 6 hours   |
| Depth Specification Time | : | 3 hours   |



Layering



Filling Holes



New View



Final Depth Map



## 7.6 PAINTING BY TURNER

This final painting again applies Geometry Tools to a scene that is not perspective correct. As usual, we began with the ground plane at the level of the water. Each of the layers was given basic depth using the vertical plane tool. Because the reflections of the boats and the hills in the water were segmented into their same corresponding layers, the depths of the reflections are vertical as well. This simulates movement of reflections as the virtual camera moves. We also used the contour mesh tool to give a rounded shape to the hills on the left.



Initial Image

|                          |   |           |
|--------------------------|---|-----------|
| Pixel Resolution         | : | 640 x 472 |
| Number of Layers         | : | 11        |
| Segmentation Time        | : | 3 hours   |
| Depth Specification Time | : | 1 hour    |



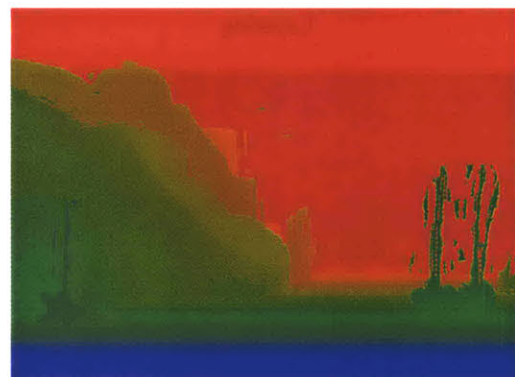
Layering



Filling Holes



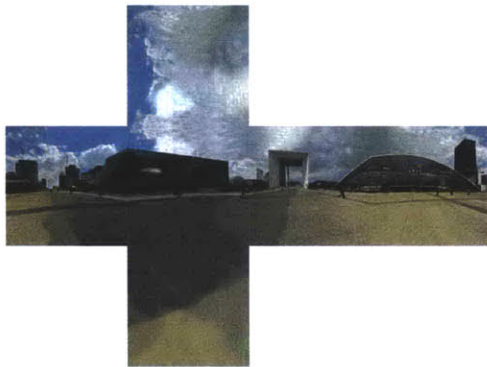
New View



Final Depth Map

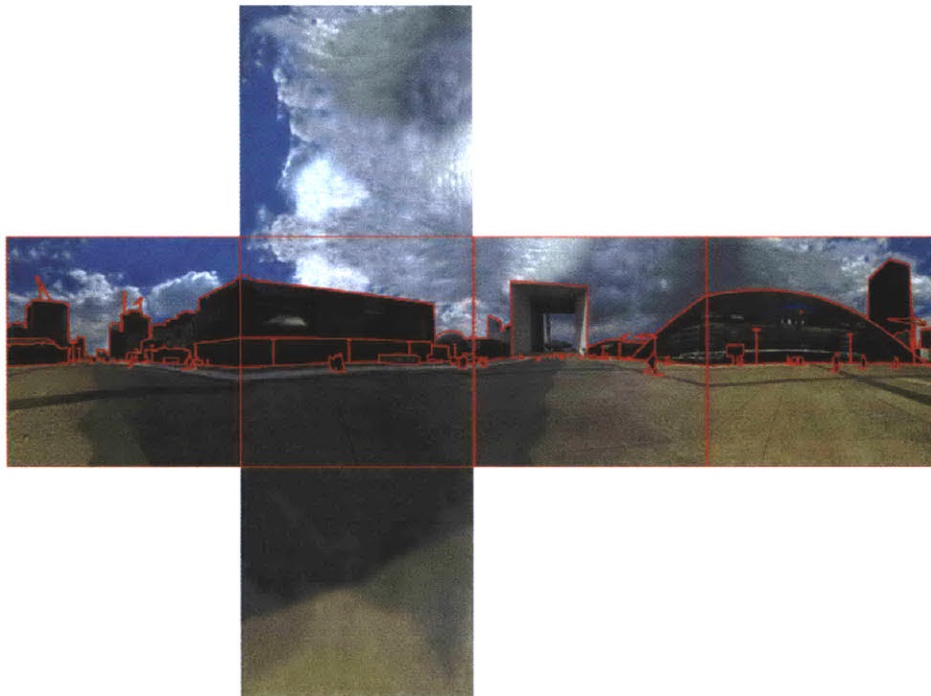
## 7.7 OUTDOOR SCENE

Our system is suited not only for single images, but also for panoramic images. In the following examples, we applied depth to each of the sides of a cubic panorama. The first example is a large outdoor scene. The depth has been applied mostly with the ground plane tool and the vertical plane tool. By using these tools, we can already achieve convincing results because the extent of this scene is very large. Therefore, crude depth is sufficient to achieve parallax effects. At boundaries between images of the cubic panorama, discontinuities can occur. We use the stitching tool to resolve these issues.

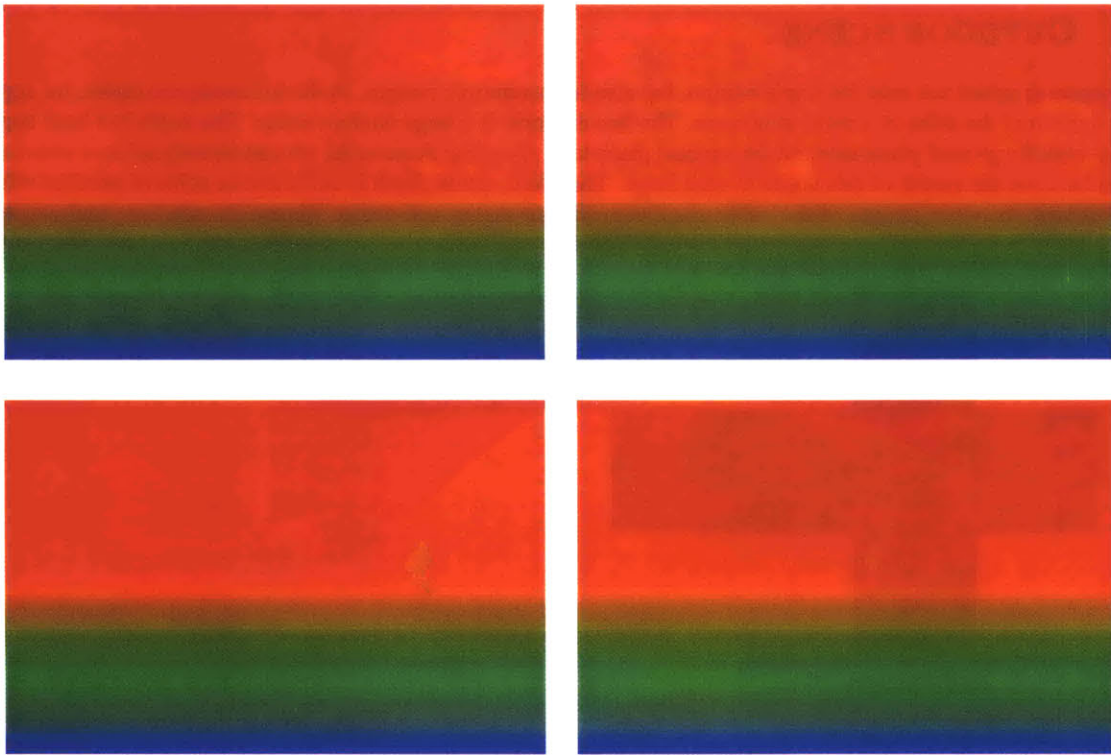


Initial Images

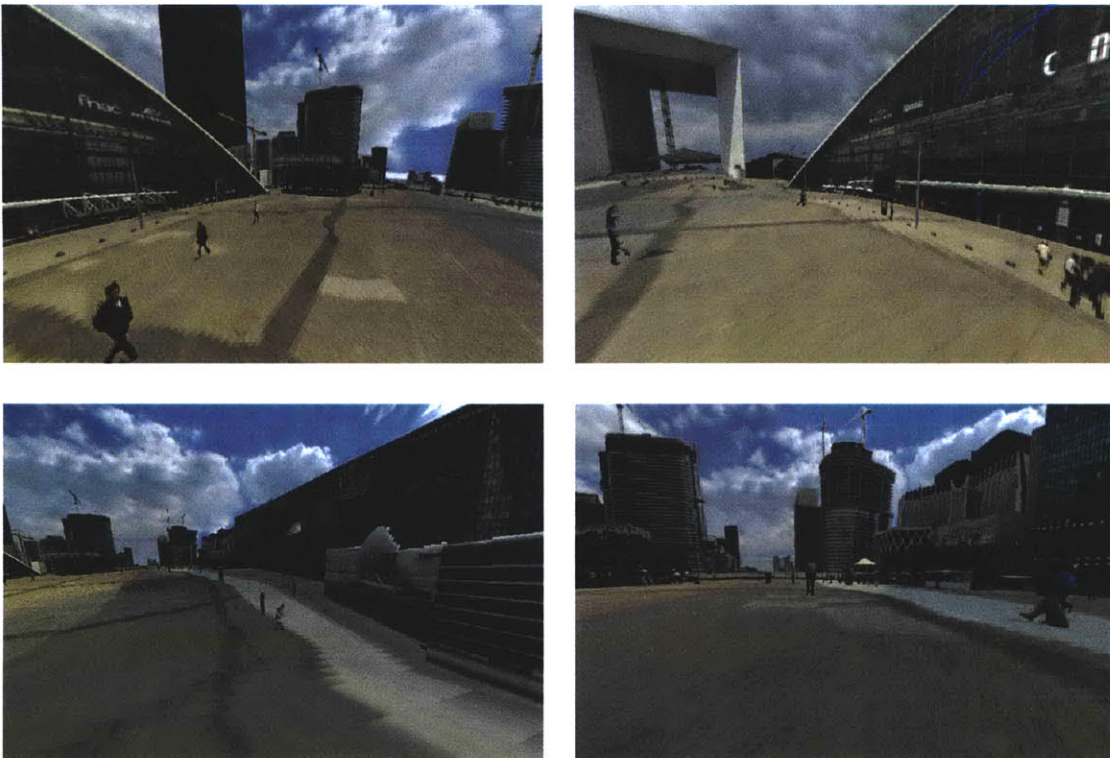
|                          |   |                       |
|--------------------------|---|-----------------------|
| Pixel Resolution         | : | 640 x 640 (each side) |
| Number of Layers         | : | 47                    |
| Segmentation Time        | : | several days          |
| Depth Specification Time | : | 4 hours               |



Layering



Final Depth Maps

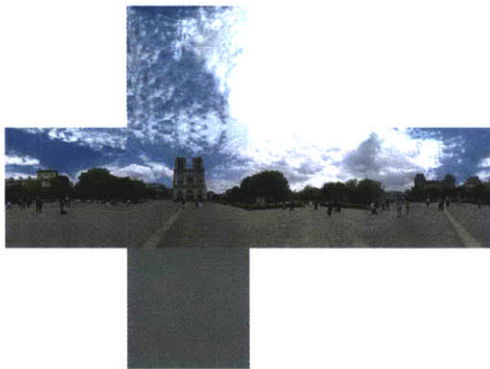


New Views



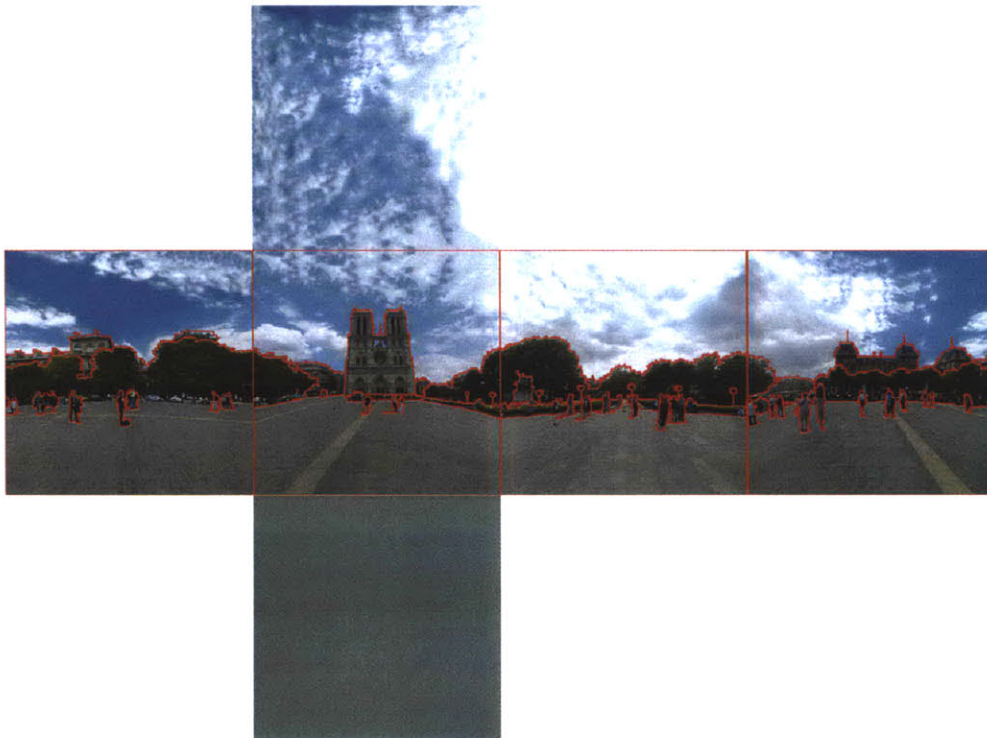
## 7.8 NOTRE DAME

The second panorama is the courtyard in front of the Notre Dame cathedral. This photographs of this large expansive space contain several tourists, which have been given depth using the automatic vertical plane tool. We used the organic shapes tool to give rounded depth to the surrounding bushes and trees. The depths for the far buildings are simply vertical planes. This simple depth took a short amount of time to complete, but the results are effective because the separation of layers and the corresponding applied depth allow for dramatic parallax effects.

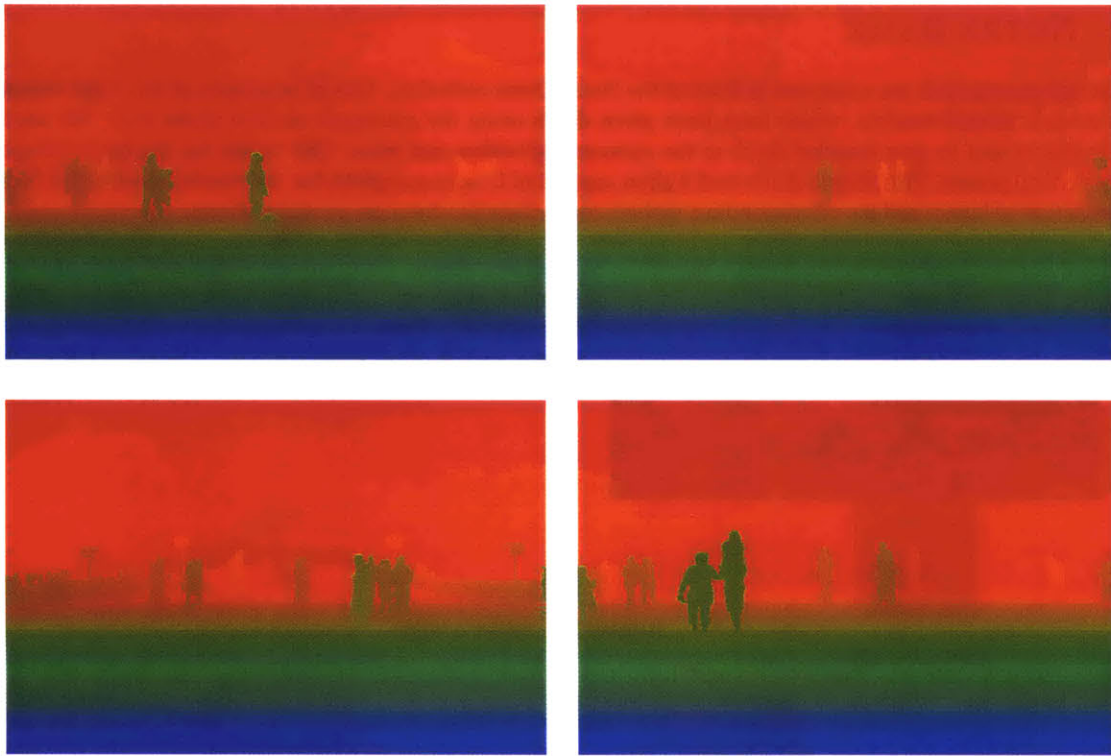


Initial Images

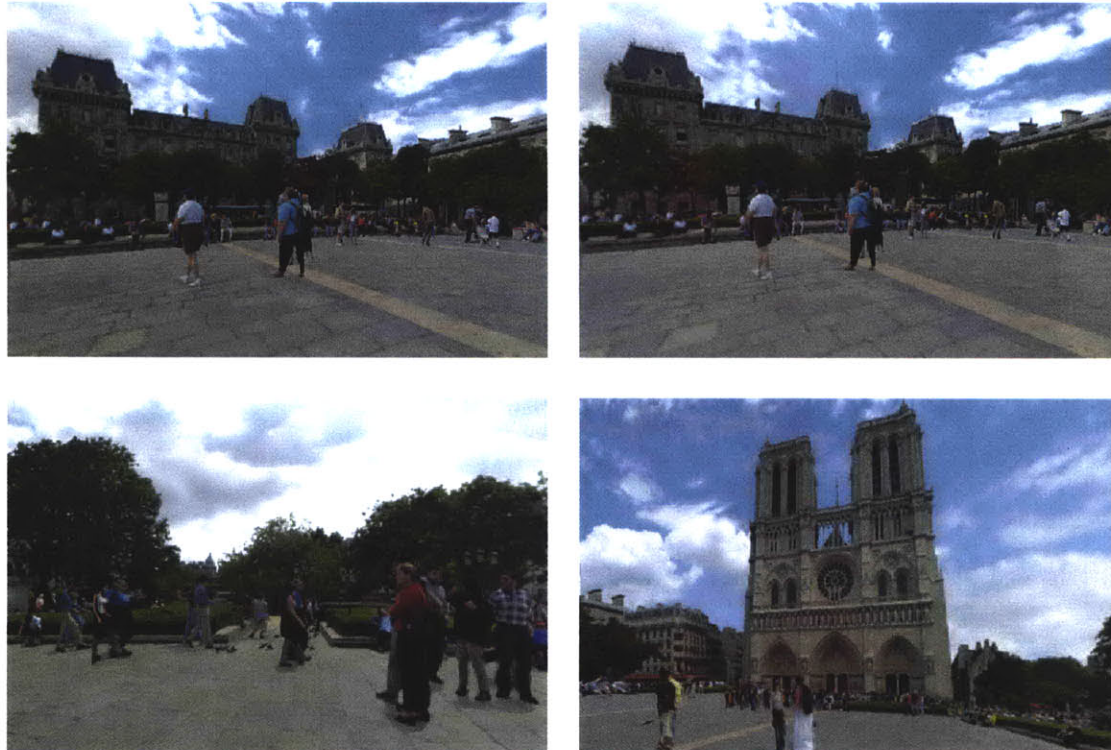
|                          |   |                       |
|--------------------------|---|-----------------------|
| Pixel Resolution         | : | 640 x 640 (each side) |
| Number of Layers         | : | 43                    |
| Segmentation Time        | : | several days          |
| Depth Specification Time | : | 2 hours               |



Layering



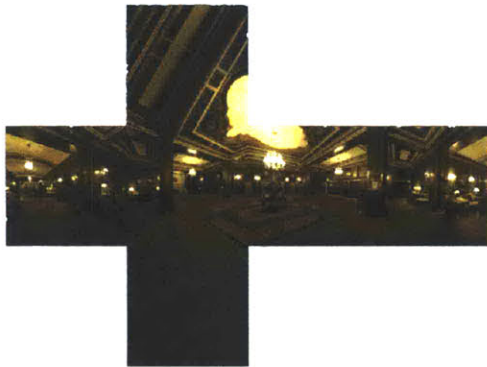
Final Depth Maps



New Views

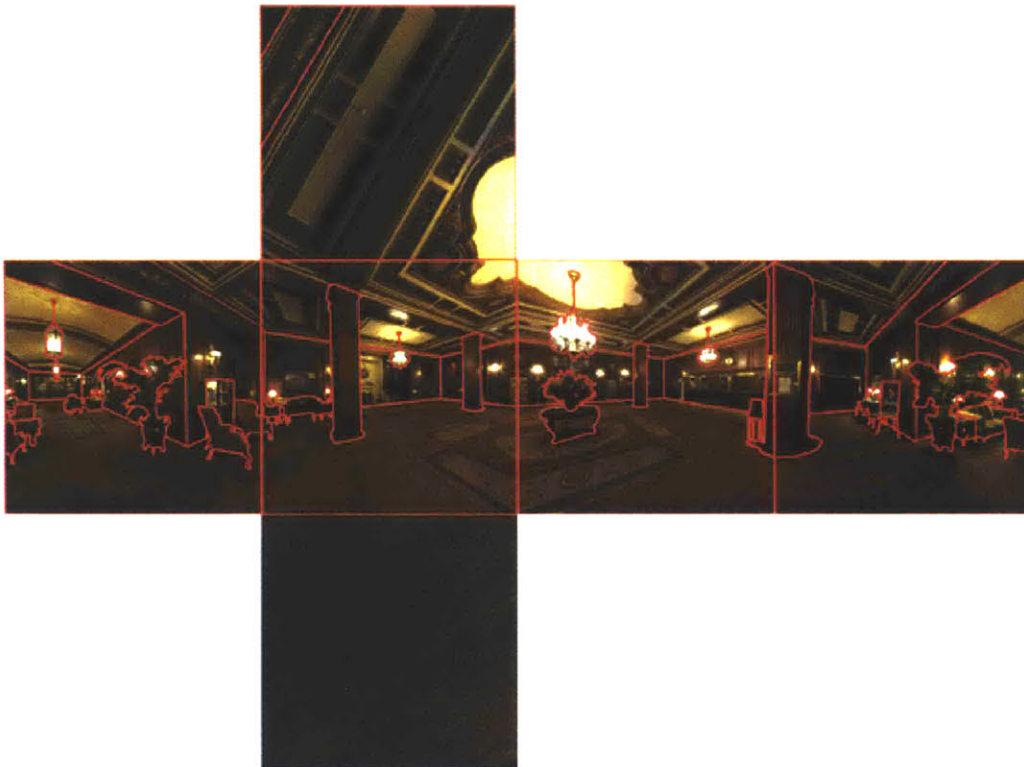
## 7.9 HOTEL LOBBY

Our final example is an interior scene featuring a hotel lobby. To specify depth, we first set the ground plane for the four side images. The depth of the walls was specified using the vertical plane tool, and the ceiling was capped using the horizontal plane tool. We used the cube, cylinder, and contour mesh tools to define depths for the chairs, tables, and couches. The organic shapes tool was used to give depth to the plants and flowers. Again, the stitching tool was used to resolve discontinuities between the images.



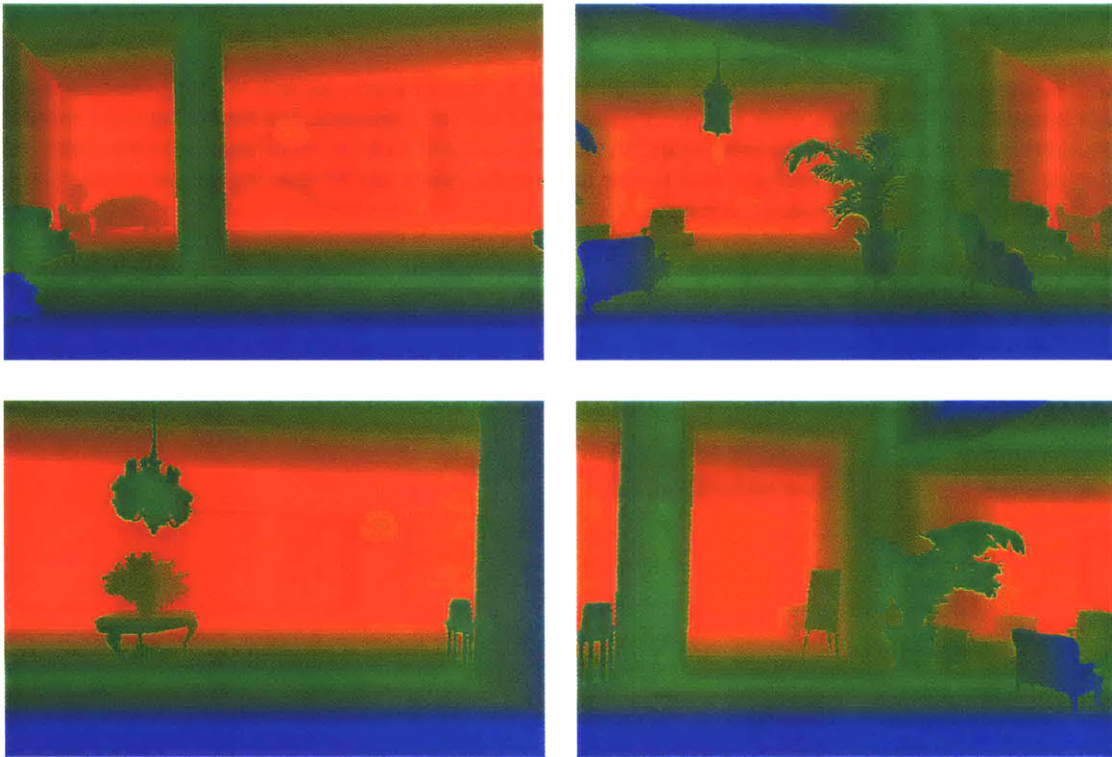
Initial Images

|                          |   |                       |
|--------------------------|---|-----------------------|
| Pixel Resolution         | : | 512 x 512 (each side) |
| Number of Layers         | : | 62                    |
| Segmentation Time        | : | several days          |
| Depth Specification Time | : | 6 hours               |



Layering





Final Depth Maps



New Views

---

## CONCLUSIONS

---

We have presented an image-based modeling system that takes a single photo as input and allows the user to extract and specify depth using interactive tools. The system allows the user to build a representation consisting of layers of images with depth. We have demonstrated the system in a variety of contexts, including architectural design and fine art exploration.

Our system includes geometric primitives and the use of a ground plane as a reference point. At the same time, we do not neglect non-architectural shapes. We have introduced two new tools that specifically address shapes such as trees, cars, and people. We also provide Image-Based Tools, which include painting and chiseling, shape from shading techniques, and filtering.

There are several avenues of future work in our system. Selection and segmentation of the image can be automated using computer vision techniques. Texture synthesis methods could be applied to automatically fill in holes created by segmentation. Our system can also be extended to multiple images. While the existing depth tools can operate on multiple images, there are several other techniques that can be applied to extract depth from multiple images. With multiple images, epipolar geometry and point correspondences would enable more automated image segmentation. In addition, stereo correspondence or computer vision techniques could be integrated to automatically extract depth. Depth with multiple images would also permit view-dependent effects.





---

## DEFINITIONS

---

In this section we delineate the conventions used through this document and the implementation. Figure A-1 illustrates these definitions.

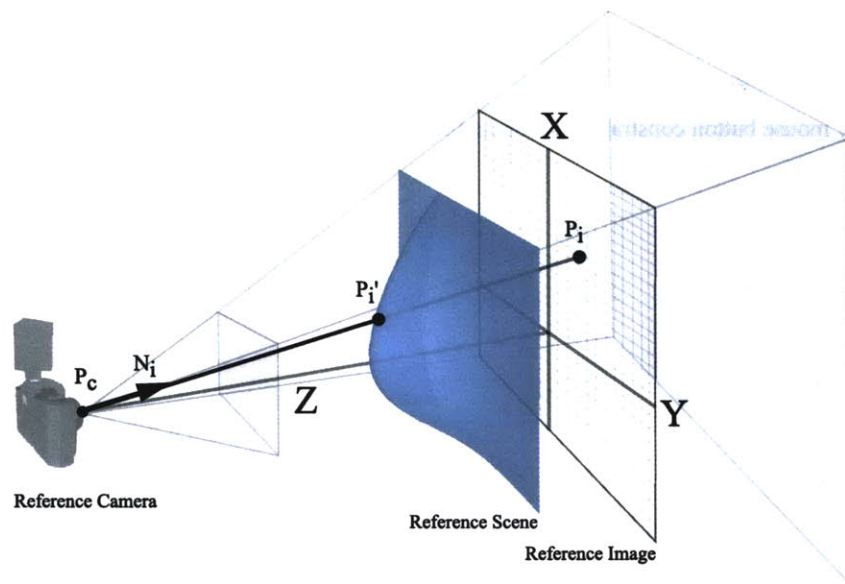


FIGURE A-1: System Conventions.

### A.1 SYSTEM ARCHITECTURE

- Coordinate System - image dimensions are in  $x$  and  $y$ ,  $z$  is the depth coordinate

### A.2 REFERENCE IMAGE

- Reference Image - initial image without depth modifications, picking points on the reference image results in coplanar points.

- Reference Scene - initial image with depth modifications, picking more than 3 points in the reference scene does not necessarily result in coplanar points.
- Reference Camera - initial image's camera parameters.

### A.3 MATH CONVENTIONS

- $P_i$  - 2D point  $i$  in reference image.
- $P'_i$  - 3D projection of  $P_i$ .
- $N_i$  - normal of  $i$ .
- $P_c$  - 3D camera position.

### A.4 TRADITIONAL 3D MODELING INTERFACE/INTERACTION

Throughout this document, we refer to traditional 3D modeling interface and interaction as transforming 3D objects in the OpenGL viewer with the mouse. The interaction is as follows.

- Mouse Button 1: Rotation.
- Mouse Button 2: Translation.
- Mouse Button 3: Scale.
- Shift and any mouse button constrains motion in direction parallel to interactive camera.

# SYSTEM INTERFACE

## B.1 SYSTEM INTERFACE



## B.2 GENERAL INTERFACE FUNCTIONS AND TOGGLES




| General Button   | Description   |
|--|---|
| <br><b>Home</b>         | <ul style="list-style-type: none"> <li>Resets active tool to home position.</li> </ul>  |
| <br><b>Update Depth</b> | <ul style="list-style-type: none"> <li>Performs depth update using active tool.</li> </ul>  |
| <br><b>Lock</b>         | <ul style="list-style-type: none"> <li>Locks active tool.</li> <li>When lock is active, camera tool becomes active tool.</li> </ul> |

Table B.1: General Buttons.





| Appearance Button  | Description  |
|--|--|
| <br><b>Wireframe</b>    | <ul style="list-style-type: none"> <li>Toggles between wireframe and filled display of active tool.</li> </ul>   |
| <br><b>Texture</b>      | <ul style="list-style-type: none"> <li>Toggles between projected texture and non textured display of active tool.</li> </ul>                                     |
| <br><b>Guides</b>       | <ul style="list-style-type: none"> <li>Toggles guideline display of active tool. Guidelines include the pivot point and the axes.</li> </ul>                     |
| <br><b>Ground Plane</b> | <ul style="list-style-type: none"> <li>Toggles display of ground plane. If ground plane is visible, depth reading will include ground plane geometry.</li> </ul> |

Table B.2: Appearance Buttons.



**B.3 IMAGE-BASED TOOLS**


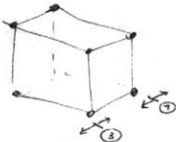


| Image-Based Tool   | Interaction   | Description  |
|--|---|--|
| <br><b>Shape from Shading</b> |  | <ul style="list-style-type: none"><li>• Move front bound (Mouse 3)</li><li>• Move rear bound (Mouse 1)</li></ul> |
| <br><b>Shape from Color</b>   |  | <ul style="list-style-type: none"><li>• Move with slider</li></ul>   |

Table B.3: Image-Based Tools.

## B.4 GEOMETRY TOOLS






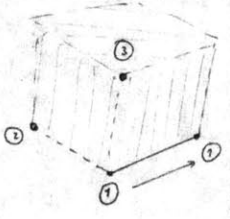



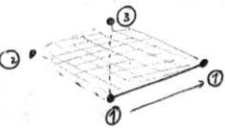

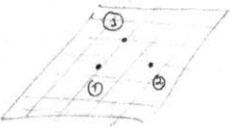

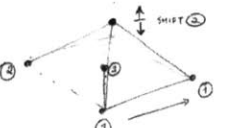
| Movement Button  | Description   |
|--|---|
|  <p><b>Transform</b></p>  | <ul style="list-style-type: none"> <li>• Transforms geometry.</li> <li>• Mouse 1: rotate, Mouse 2: translate, Mouse 3: scale.</li> </ul>              |
|  <p><b>Sketch</b></p>     | <ul style="list-style-type: none"> <li>• Sketch interface for primitive tools (See Table B.5).</li> </ul>   |
|  <p><b>Depth Move</b></p> | <ul style="list-style-type: none"> <li>• Translates and scales in depth.</li> </ul>   |
|  <p><b>Pivot</b></p>     | <ul style="list-style-type: none"> <li>• Move pivot.</li> <li>• Mouse 1: select pivot point, Mouse 2: snap pivot point to reference depth.</li> </ul> |

Table B.4: Geometry Movement Buttons.

| Primitive Tool  | Interaction   | Description   |
|---|---|---|
|  <p><b>Cube</b></p>            |    | <ul style="list-style-type: none"> <li>• Draw three orthogonal axes with one line (Mouse 1) and two points (Mouse 2, 3)</li> </ul>  |
|  <p><b>Cylinder</b></p>        |    | <ul style="list-style-type: none"> <li>• Draw two edges of cylinder (Mouse 1,3)</li> <li>• Shift constrains to vertical</li> </ul>  |
|  <p><b>Ground Plane</b></p>   |   | <ul style="list-style-type: none"> <li>• Draw three orthogonal axes with one line (Mouse 1) and two points (Mouse 2, 3)</li> </ul>  |
|  <p><b>Plane</b></p>         |  | <ul style="list-style-type: none"> <li>• Draw three points on the plane (Mouse 1, 2, 3)</li> </ul>  |
|  <p><b>Pyramid/Wedge</b></p> |  | <ul style="list-style-type: none"> <li>• Draw three orthogonal axes with one line (Mouse 1) and two points (Mouse 2, 3)</li> <li>• Scale height of peak with Shift-Mouse 2</li> </ul> |


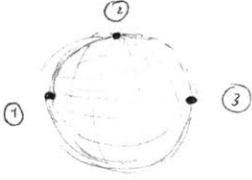

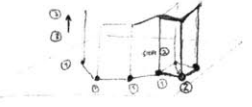

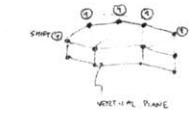

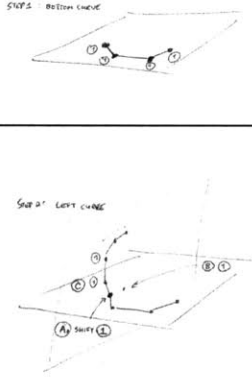
|  |   |   |
|--|---|---|
|  <p><b>Sphere</b></p> |  | <ul style="list-style-type: none"> <li>• Draw three points on the sphere (Mouse 1, 2, 3)</li> </ul> |
|--|---|---|

Table B.5: Primitive Tools Sketch Interaction.

| Geometry Tool  | Interaction   | Description  |
|--|---|--|
|  <p><b>Vertical Plane</b></p>     |    | <ul style="list-style-type: none"> <li>• Draw points on ground plane (Mouse 1)</li> <li>• Extrude points (Mouse 3)</li> <li>• Select existing points (Shift-Mouse 2)</li> <li>• Move selected points (Mouse 2)</li> </ul>  |
|  <p><b>Horizontal Plane</b></p> |  | <ul style="list-style-type: none"> <li>• Draw points in counter-clockwise order on plane defined by vertical plane (Mouse 1)</li> <li>• Snap to top of vertical plane (Shift-Mouse 1)</li> <li>• Select existing points (Shift-Mouse 2)</li> <li>• Move selected points (Mouse 2)</li> </ul>   |
|  <p><b>Contour Mesh</b></p>     |  | <ul style="list-style-type: none"> <li>• Draw Bottom Curve points on ground plane (Mouse 1)</li> <li>• Shift-Mouse 1 near first point on Bottom Curve</li> <li>• Draw another point on the ground plane (Mouse 1) to set plane</li> <li>• Draw Left Curve points on plane (Mouse 1)</li> </ul> |

|                                       |                            |  |
|---------------------------------------|----------------------------|--|
|                                       | <p>STEP 3: RIGHT CURVE</p> | <ul style="list-style-type: none"> <li>• Shift-Mouse 1 near last point on Bottom Curve</li> <li>• Draw another point on the ground plane (Mouse 1) to set plane</li> <li>• Draw Right Curve points on plane (Mouse 1)</li> </ul>   |
|                                       | <p>STEP 4: TOP CURVE</p>   | <ul style="list-style-type: none"> <li>• Shift-Mouse 1 near last point on Left Curve</li> <li>• Draw Top Curve points on plane (Mouse 1)</li> </ul>  |
| <p><b>Heightfield</b></p>             |                            | <ul style="list-style-type: none"> <li>• Draw points/peaks in image (Mouse 1)</li> <li>• Select existing points (Shift-Mouse 2)</li> <li>• Move selected points in plane parallel to image (Mouse 2)</li> <li>• Change depth of selected point (Mouse 3)</li> </ul>      |
| <p><b>Intrusion and Extrusion</b></p> |                            | <ul style="list-style-type: none"> <li>• Intrusion Mode: Draw points in counter-clockwise order in image (Mouse 1), push with Mouse 3</li> <li>• Extrusion Mode: First point defines extrude point, draw points in counter-clockwise order in image (Mouse 1)</li> </ul> |

Table B.6: Geometry Tools Interaction.





---

## IMPLEMENTATION ISSUES

---

### C.1 CATMULL-ROM INTERPOLATION

Given a set of points and the number of interpolation divisions between the points, Catmull-Rom Interpolation defines a curve that includes the original set of points and interpolated points in between them [17]. If  $P_1, P_2, P_3, P_4$  are the set of initial points, we can directly calculate a point  $Q_i$  between  $P_2$  and  $P_3$ , given the desired number of points  $n$  between  $P_2$  and  $P_3$  with,

$$Q_i(t) = 0.5 * ((-P_1 + 3P_2 - 3P_3 + P_4)t^3 + (2P_1 - 5P_2 + 4P_3 - P_4)t^2 + (-P_1 + P_3)t + 2P_2),$$

where,

$$t = \frac{1}{1.0 + n}(i + 1).$$

### C.2 BIT DEPENDENCE

For image warping, we need to read the color buffer [49]. Because our system's data structure is based on 16-bit reference images, our system relies on hardware that supports 16-bit buffers. SGI O2s support only 8-bit buffers while SGI Octanes support 12-bit buffers. We can use 12-bit buffers by storing our textures as bytes, rather than bits.

### C.3 EXTRACTING TRIANGLES FROM WINGED-EDGE DATA STRUCTURES

The image warping tool requires triangulation of points. We use Delaunay Triangulation [19]. The data structure for this implementation of Delaunay Triangulation is a winged-edge data structure. This data structure stores a list of vertices and their corresponding edges. Therefore, there is no continuous list of triangles in this data structure. To extract the triangles from this data structure, we iterate through the list of vertices. Each vertex contains edges to two other vertices, therefore two triangles. We mark these two triangles and this vertex as visited and continue finding triangles with the two connected vertices. This algorithm runs in  $O(n)$  time.



---

## SUPPLEMENTAL GEOMETRY

---

### D.1 LINE AND PLANE INTERSECTION

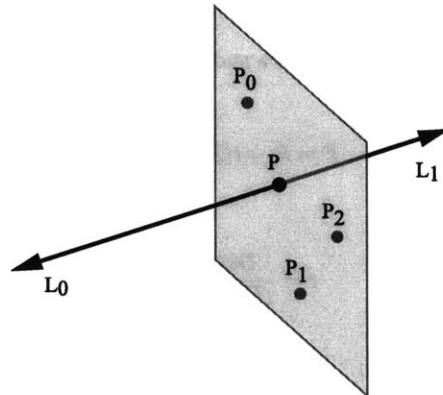


FIGURE D-1: Line and Plane Intersection.

The intersection of a line  $\overline{L_0L_1}$  and a plane defined by three points,  $P_0, P_1, P_2$ , is a point  $P$  (Figure D-1),

$$P = L_0 + (L_1 - L_0)t,$$

where,

$$t = \frac{\text{Det}(A)}{\text{Det}(B)},$$

and,

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ P_{0x} & P_{1x} & P_{2x} & L_{0x} \\ P_{0y} & P_{1y} & P_{2y} & L_{0y} \\ P_{0z} & P_{1z} & P_{2z} & L_{0z} \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 & 1 & 0 \\ P_{0x} & P_{1x} & P_{2x} & L_{1x} - L_{0x} \\ P_{0y} & P_{1y} & P_{2y} & L_{1y} - L_{0y} \\ P_{0z} & P_{1z} & P_{2z} & L_{1z} - L_{0z} \end{bmatrix}.$$

## D.2 LINE AND LINE INTERSECTION

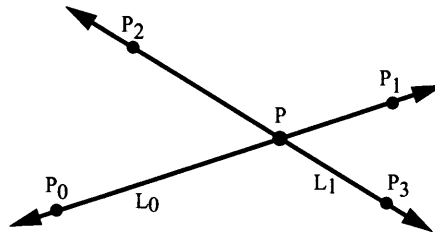


FIGURE D-2: Line and Line Intersection.

The intersection of a line  $L_1$ , whose endpoints are  $P_0$  and  $P_1$ , and a second line  $L_2$ , whose endpoints are  $P_2$  and  $P_3$ , is a point  $P$  (Figure D-2),

$$P = P_0 + t(\widehat{P_1 - P_0}),$$

where,

$$t = \frac{\text{Det}(A)}{(L_1 \times L_2)^2},$$

and,

$$A = \begin{bmatrix} P_{2x} - P_{1x} & L_{2x} & (L_1 \times L_2)_x \\ P_{2y} - P_{1y} & L_{2y} & (L_1 \times L_2)_y \\ P_{2z} - P_{1z} & L_{2z} & (L_1 \times L_2)_z \end{bmatrix}.$$

## D.3 CLOSEST POINT ON A LINE TO A POINT

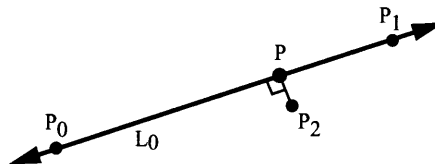


FIGURE D-3: Closest point on a line to a point.

The closest point  $P$  on  $\overline{P_0P_1}$  to a point  $P_2$  is (Figure D-3),



$$P = P_0 + t(\widehat{P_1 - P_0}) .$$

The closest point lies on the line that passes through  $P_2$  and is perpendicular to  $\overline{P_0P_1}$ ,

$$(P_2 - P) \cdot (P_1 - P_0) = 0 .$$

Substituting for  $P$  and solving for  $t$ ,

$$\begin{aligned} 0 &= (P_2 - P_0 + t(\widehat{P_1 - P_0})) \cdot (P_1 - P_0) \\ t &= \frac{(P_2 - P_0) \cdot (P_1 - P_0)}{(P_1 - P_0) \cdot (P_1 - P_0)} . \end{aligned}$$

## D.4 CENTER AND RADIUS OF A CIRCLE

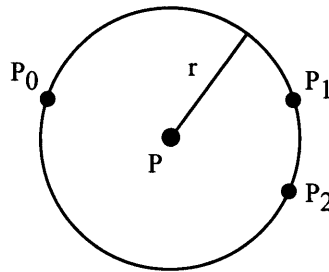


FIGURE D-4: Center and Radius of a Circle.

The center and radius of a circle can be determined if three points on the circle are known. If  $P_0, P_1, P_2$  are the user-defined points, the center  $P$  and the radius  $r$  are (Figure D-4),

$$\begin{aligned} P_x &= \frac{-d}{s} \frac{1}{a} \\ P_y &= \frac{-e}{s} \frac{1}{a} \\ r &= \sqrt{\frac{d^2 + e^2}{4a^2} - \frac{f}{a}} , \end{aligned}$$

where,

$$a = \begin{bmatrix} P_{0x} & P_{0y} & 1 \\ P_{1x} & P_{1y} & 1 \\ P_{2x} & P_{2y} & 1 \end{bmatrix}$$

$$d = - \begin{bmatrix} P_{0x}^2 + P_{0y}^2 & P_{0y} & 1 \\ P_{1x}^2 + P_{1y}^2 & P_{1y} & 1 \\ P_{2x}^2 + P_{2y}^2 & P_{2y} & 1 \end{bmatrix}$$

$$e = \begin{bmatrix} P_{0x}^2 + P_{0y}^2 & P_{0x} & 1 \\ P_{1x}^2 + P_{1y}^2 & P_{1x} & 1 \\ P_{2x}^2 + P_{2y}^2 & P_{2x} & 1 \end{bmatrix}$$

$$f = - \begin{bmatrix} P_{0x}^2 + P_{0y}^2 & P_{0x} & P_{0y} \\ P_{1x}^2 + P_{1y}^2 & P_{1x} & P_{1y} \\ P_{2x}^2 + P_{2y}^2 & P_{2x} & P_{2y} \end{bmatrix} .$$

---

## BIBLIOGRAPHY

---

- [1] Minolta 3D 1500. <http://www.minolta.com/dp/3d1500/>.
- [2] Adobe. <http://www.adobe.com>.
- [3] Alias—Wavefront. <http://www.aliaswavefront.com>.
- [4] AutoCAD. <http://www.autodesk.com>.
- [5] M. Bichsel and A. P. Pentland. A simple algorithm for shape from shading. In *CVPR*, pages 459–465, 1992.
- [6] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. *Proceedings of SIGGRAPH 99*, pages 187–194, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.
- [7] Canoma. <http://www.canoma.com>.
- [8] B. Caprile and V. Torre. Using vanishing points for camera calibration. *IJCV*, 4(2):127–140, March 1990.
- [9] Shenchang Eric Chen. Quicktime vr - an image-based approach to virtual environment navigation. *Proceedings of SIGGRAPH 95*, pages 29–38, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California.
- [10] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288, aug 1993.
- [11] J. Cohen, J. Hughes, and R. Zeleznik. Harold: A world made of drawings. In *NPAR*, 2000.
- [12] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. *Proceedings of SIGGRAPH 96*, pages 303–312, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.
- [13] Cyberware. <http://www.cyberware.com>.
- [14] Cyra. <http://www.cyra.com>.
- [15] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.
- [16] Olivier Faugeras, Stéphane Laveau, Luc Robert, Gabriella Csurka, and Cyril Zeller. 3-d reconstruction of urban scenes from sequences of images. Technical Report RR-2572, Inria, Institut National de Recherche en Informatique et en Automatique, 1995.
- [17] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer graphics —Principles and Practice—*. Addison-Wesley, 2nd edition, 1990.

- [18] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996, <http://hillbilly.deas.harvard.edu/~sjg/>.
- [19] Paul S. Heckbert, editor. *Graphics gems IV*, volume 4 of *Graphics Gems*. AP Professional, Boston, MA, USA, 1994.
- [20] Berthold K. P. Horn and Michael J. Brooks. *Shape from Shading*. The MIT Press, Cambridge, MA, 1989. Prepared with Daniel Brotsky's  $\text{\LaTeX}$ .
- [21] Youichi Horry, Ken ichi Anjyo, and Kiyoshi Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 225–232. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.
- [22] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3D freeform design. In *Siggraph*, Los Angeles, 1999.
- [23] F. Isgro and E. Trucco. Projective rectification without epipolar geometry. In *Proceedings of the IEEE Computer Science Conference on Computer Vision and Pattern Recognition (CVPR-99)*, pages 94–99, Los Alamitos, June 23–25 1999. IEEE.
- [24] S. Kang. Depth painting for image-based rendering applications. Tech. report, CRL, Compaq Cambridge Research Lab, 1998. <http://www.research.microsoft.com/Users/sbkang/publications/>.
- [25] Stephane Laveau and Olivier Faugeras. 3-D scene representation as a collection of images and fundamental matrices. In *Proceedings of 12th International Conference on Pattern Recognition*, volume 1, pages 689–691, 1994.
- [26] Marc Levoy and Pat Hanrahan. Light field rendering. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 31–42. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996, <http://www-graphics.stanford.edu/papers/light/>.
- [27] David Liebowitz, Antonio Criminisi, and Andrew Zisserman. Creating architectural models from images. In Hans-Peter Seidel and Sabine Coquillart, editors, *Eurographics '99*, volume 18, pages C39–C50, Computer Graphics Forum, c/o Mercury Airfreight International Ltd Inc, 365 Blair Road, Avenel, MI 48106, USA, 1999. Eurographics Association, Eurographics Association and Blackwell Publishers Ltd 1999.
- [28] Jae S. Lim. *Two-dimensional Signal and Image Processing*. Prentice Hall Signal Processing Series. Prentice-Hall, Englewood Cliffs, NJ, USA, 1990.
- [29] Peter Litwinowicz and Gavin Miller. Efficient techniques for interactive texture placement. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 119–122. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.
- [30] 3D Studio Max. <http://www.discreet.com>.
- [31] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics*, 29(Annual Conference Series):39–46, 1995.
- [32] Leonard McMillan. An image-based approach to three-dimensional computer graphics. Technical Report TR97-013, Department of Computer Science, University of North Carolina - Chapel Hill, May 19 1997. Mon, 9 Jun 1997 12:41:59 GMT.
- [33] B. Oh, M. Chen, F. Durand, and J. Dorsey. Image-based modeling and photo editing. In *Siggraph*, Los Angeles, 2001.
- [34] M. Okutomi and T. Kanade. A multiple baseline stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 63–69, Lahaina, Maui, Hawaii, 1991. IEEE Computer Society.

- [35] Photomodeler. <http://www.photomodeler.com>.
- [36] T. Poggio and Michael Drumheller. Parallel stereo. In *IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 1986. V86-1.
- [37] Pierre Poulin, Mathieu Ouimet, and Marie-Claude Frasson. Interactively modeling with photogrammetry. In *Proc. Eurographics Workshop on Rendering*, June 1998.
- [38] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, 2nd edition*. Cambridge University Press, 1992.
- [39] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings, Annual Conference Series*, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
- [40] Yoichi Sato, Mark D. Wheeler, and Katsushi Ikeuchi. Object shape and reflectance modeling from observation. *Proceedings of SIGGRAPH 97*, pages 379–388, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.
- [41] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):249–252, July 1992.
- [42] S. M. Seitz and K. N. Kutulakos. Plenoptic image editing. In *Proc. Fifth International Conference on Computer Vision*, 1998.
- [43] J. A. Sethian. *Level Set Methods*. Cambridge University Press, 1996.
- [44] F. Sillion and G. Drettakis. Feature-based control of visibility error: A multi-resolution clustering algorithm for global illumination. In *Proc. Siggraph*, Annual Conference Series, 1995.
- [45] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley, MA, USA, 1993.
- [46] Changming Sun. A fast stereo matching method. In *Digital Image Computing: Techniques and Applications*, pages 95–100, Massey University, Auckland, New Zealand, December 1997.
- [47] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. *Proceedings of SIGGRAPH 94*, pages 311–318, July 1994. ISBN 0-89791-667-0. Held in Orlando, Florida.
- [48] L. Williams. Image jets, level sets and silhouettes. Workshop on Image-Based Modeling and Rendering, <http://www-graphics.stanford.edu/workshops/ibr98/>, March 1998.
- [49] Mason Woo, Jackie Neider, Tom Davis, and OpenGL Architecture Review Board. *OpenGL programming guide: the official guide to learning OpenGL, version 1.1*. Addison-Wesley, Reading, MA, USA, second edition, 1997.
- [50] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: An interface for sketching 3d scenes. *Proceedings of SIGGRAPH 96*, pages 163–170, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.
- [51] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. Analysis of shape from shading techniques. In *Proceedings, Conference on Computer Vision and Pattern Recognition*, pages 377–384, Seattle, Washington, June 21–23 1994. IEEE-CS, IEEE Computer Society Press.