A System for Image-Based Modeling and Photo Editing

by

Byong Mok Oh

B.A., Computer Science
B.A., Art History and Studio Art
Oberlin College, 1992

M.S.E., Computer and Information Science
University of Pennsylvania, 1993

Submitted to the Department of Architecture
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Architecture: Design and Computation

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2002

Author .....
Department of Architecture
May 3, 2002

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

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

Committee Members

Julie Dorsey

Associate Professor of Computer Science and Engineering and Architecture

Frédo Durand

Post-Doctoral Associate of Laboratory for Computer Science

Leonard McMillan

Associate Professor of Electrical Engineering and Computer Science

Leslie Keith Norford

Associate Professor of Building Technology

A System for Image-Based Modeling and Photo Editing

by

Byong Mok Oh

Submitted to the Department of Architecture
on May 3, 2002, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Architecture: Design and Computation

Abstract

Traditionally in computer graphics, a scene is represented by geometric primitives composed of various materials and a collection of lights. Recently, techniques for modeling and rendering scenes from a set of pre-acquired images have emerged as an alternative approach, known as *image-based modeling and rendering*. Much of the research in this field has focused on reconstructing and re-rendering from a set of photographs, while little work has been done to address the problem of *editing* and *modifying* these scenes. On the other hand, photo-editing systems, such as Adobe Photoshop, provide a powerful, intuitive, and practical means to edit images. However, these systems are limited by their two-dimensional nature.

In this thesis, we present a system that extends photo editing to 3D. Starting from a single input image, the system enables the user to reconstruct a 3D representation of the captured scene, and edit it with the ease and versatility of 2D photo editing. The scene is represented as layers of images with depth, where each layer is an image that encodes both color and depth. A suite of user-assisted tools are employed, based on a painting metaphor, to extract layers and assign depths. The system enables editing from different viewpoints, extracting and grouping of image-based objects, and modifying the shape, color, and illumination of these objects.

As part of the system, we introduce three powerful new editing tools. These include two new clone brushing tools: the *non-distorted clone brush* and the *structure-preserving clone brush*. They permit copying of parts of an image to another via a brush interface, but alleviate distortions due to perspective foreshortening and object geometry. The non-distorted clone brush works on arbitrary 3D geometry, while the structure-preserving clone brush, a 2D version, assumes a planar surface, but has the added advantage of working directly in 2D photo-editing systems that lack depth information. The third tool, a *texture-illuminance decoupling filter*, discounts the effect of illumination on uniformly textured areas by decoupling large- and small-scale features via bilateral filtering. This tool is crucial for relighting and changing the materials of the scene.

There are many applications for such a system, for example architectural, lighting and landscape design, entertainment and special effects, games, and virtual TV sets. The system allows the user to superimpose scaled architectural models into real environments, or to quickly paint a desired lighting scheme of an interior, while being able to navigate within the scene for a fully immersive 3D experience. We present examples and results of complex architectural scenes, 360-degree panoramas, and even paintings, where the user can change viewpoints, edit the geometry and materials, and relight the environment.

Thesis Supervisor: Julie Dorsey
Title: Associate Professor of Architecture and Computer Science and Engineering

Acknowledgments

I would like to thank..

Julie Dorsey, for her vision, encouragement, and giving me this opportunity of a life time to work with her and incredibly motivated, intelligent, and talented group of people in MIT... And having faith that I would finish my PhD someday.

Frédo Durand, for his creative, innovative, and (more often than not) brilliant research ideas, expensive wine, superb French cooking, and for his friendship. Merci, Frédo.

Max Chen, for being such a wonderful and creative collaborator, researcher, and a partner in crime.

Committee members, Leonard McMillan and Les Norford, for their encouragement, and challenging me to do better.

Henrik Wann Jensen and Michael Monks, for being such wonderful collaborators and mentors.

My fantastic UROPs, Sini, Ruji, Aparna, Lucy, Hector, James, and Alex, for their help in using and testing our system.

Barry Webb, for his friendship, creative and motivating ideas, and his influence.

Bryt and Adel, for all their help and support.

Everyone in the fantastic MIT Computer Graphics Group!

Most of all..

The Oh family — dad, mom, and bro — for their support, love, and being such wonderful family members.

Carol, for her patience, support in all facets of my life, *always* being there for me, and being such a wonderful partner. I could not have done any of this without you.

And thank you Lord, my God, for being with me through this wonderful journey.

To Carol

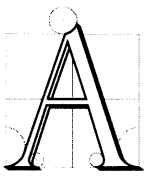
CONTENTS

LIST OF FIGURES

LIST OF TABLES

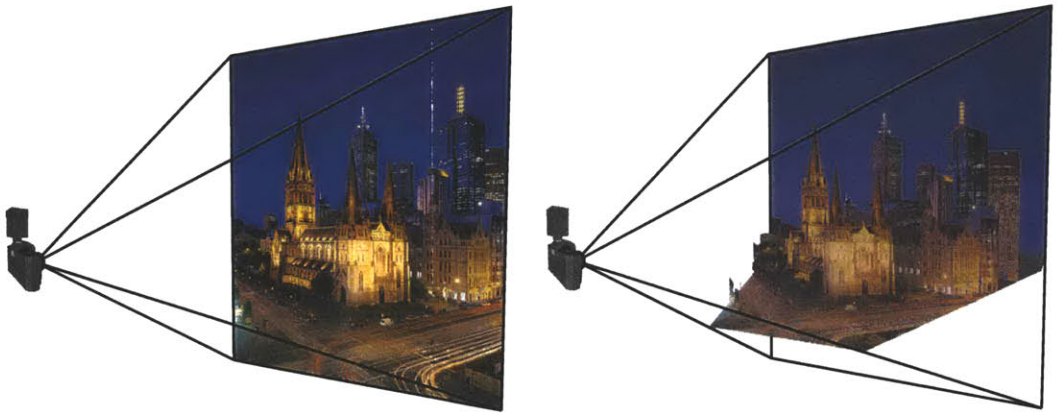
CHAPTER 1

Introduction



major goal and endeavor in the field of computer graphics is realistic image synthesis — to recreate, using the computer, realistic images of 3D environments. In pursuit of this goal, several sub-fields have emerged, including geometric modeling, reflection and material modeling, light transport simulation, and perceptual modeling. Despite the advances in these numerous fields, photorealism remains elusive. The real world is highly complex, and has shapes, material properties, and lighting effects that are not easily described through traditional means, both quantitatively and qualitatively. Tools for modeling complex geometric shapes require a great deal of effort and skill; accurate models of materials are difficult and time consuming to acquire; and the techniques used for lighting simulation remain computationally expensive and continue to undergo much research and experimentation. Today, creating convincing photorealistic images of real and complex scenes is still a difficult and time-consuming task.

An alternative approach known as *image-based modeling and rendering* (IBMR) is becoming increasingly popular and has received much interest in both the vision and computer graphics communities. IBMR techniques allow users to create 3D renderings starting from photographs captured



(a) (b)

Figure 1-1: (a) A flat photograph used as input to our system. (b) A result from our system.

from the real world. Typically, a set of photographs of a scene is used to derive an *image-based representation*, which can then be used to explore the recreated virtual scene (Figure 1-1). This approach has the advantage of producing photorealistic results since the input photographs are reused as the underlying representation. Also, unlike traditional computer graphics techniques, the cost of rendering is independent of the scene complexity.

Much attention in IBMR-related research has been devoted to modeling and rendering captured scenes from photographs, but little work has been done to address the problem of *editing* the image-based representations. A necessary and essential part of many applications is not only the ability to capture and rerender existing scenes from novel viewpoints, but also the capability to change the scene in a desired fashion. In this thesis, we address the issue of interactively manipulating and editing an image-based representation. Applications involving the editing of real images — for aesthetic, design or illustration purposes — can benefit from such an approach. In many cases, the user wants to modify existing scenes: when designing a new building in an existing context, when changing the layout and lighting of an interior space, when designing virtual TV sets, or for special effects. In these examples, modeling and editing the scenes using the traditional 3D tools is tedious and costly, and obtaining photorealistic images is challenging at best. Because these applications deal with existing scenes, they are logical candidates for image-based approaches. We believe that the ability to interactively manipulate and edit an image-based representation can prove extremely valuable in this context.

Our work was inspired, in part, by the simplicity and versatility of popular photo-editing pack-

ages, in particular *Adobe Photoshop* [Photoshop]. Such systems provide a powerful means to alter the appearance of an image via simple and intuitive editing operations. A photo-montage, where the color of objects has been changed, and people have been removed, added or duplicated, still remains convincing and "photorealistic." The process involves almost no automation and is entirely driven by the user, which provides freedom to directly access and edit the image with a suite of tools and operators. Unfortunately, the lack of 3D information often imposes restrictions or makes editing more tedious. While a photograph can provide an amazing amount of information about the appearance of the scene, it is, nonetheless, a flat image. One cannot move about the scene, freely transform and modify the shape of objects, or convincingly change the lighting or the materials of the scene. Our work overcomes these limitations by extending the image representation to 3D, while maintaining the user interaction associated with a more controllable 2D context. In our system, the user can interactively navigate into the photograph and can not only change the viewpoint but also edit the scene with photo-editing tools and operators. The familiar class of tools and filters provided in photo-editing systems is extended to our system. We also introduce new tools that take advantage of the 3D information.

In photo editing, *clone brushing* (also known as "clone stamping") is one of the most powerful and widely-used tools for the seamless alteration of pictures. Unlike typical painting tools that use a simple color to brush over an image, the advantage of the clone brush is that it reuses existing parts of the image. It is often used to remove undesirable portions of an image, such as blemishes or distracting objects in the background, or to replicate parts of the photograph. The clone brush works as follows: The user chooses a source region of the image and then paints over the destination region using a brush interface that interactively copies and pastes pixels from the source to the destination region. However, clone brushing has its limitations — copying and pasting within a flat 2D image is restrictive when the photograph is of a 3D environment. The shape of the objects in the scene as well as the perspective projection of the camera lens cause foreshortening: only parts of the image with similar orientation and distance can be clone brushed effectively. Artifacts also appear when the intensity of the target and source regions do not match.

The existing illumination also limits image editing. Lighting changes can be achieved by painting the effects of new light sources using semi-transparent layers. However, discounting the existing illumination is difficult in photo editing. Manually brightening or darkening regions of the image to remove existing lighting is laborious and usually results in artifacts, especially at shadow boundaries. This affects copying and pasting between images with different illumination conditions,

relighting applications and, as mentioned above, clone brushing.

In this thesis, we present an approach that combines the strengths of IBMR techniques and photo editing. The goal is to bring the two approaches together: to build a flexible image-based representation that places no constraints on the geometry and to provide powerful, yet intuitive, editing tools. Our system takes a single image as input and allows the user to create, manipulate, and edit an image-based representation. More precisely, the scene is represented as a collection of layered images with depth, where a pixel encodes both color and depth information. The system provides the means to change scene structure, appearance, and illumination via a simple set of editing operations, which overcome a number of limitations of photo editing. By incorporating depth, a whole new dimension of editing is possible, from changing of the camera viewpoint to completely changing the lighting of the environment.

1.1 Thesis Contributions

The contributions of this thesis are the following:

An image-based editing system

We introduce an image-based editing system that combines the strengths and advantages of photo editing and IBMR techniques. The philosophy of the system is to retain, as much as possible, the intuitive and familiar class of tools and user interactions of 2D photo-editing systems. In addition, our image-based representation enables the user to change view points, navigate into the scene, and apply the editing operators at any given viewing position. We demonstrate the system on high resolution images.

The representation of our scenes is based on *layers of images with depth*, inspired by both photo editing and IBMR techniques. Layers are used to superimpose and composite multiple images together. Each layer has a depth channel, which enables the user to change viewpoints and navigate into the scene. The representation also enables the creation of a new class of editing tools that incorporate depth information.

Our image-based representation is simple, easy to display, permits direct user control, and offers a more meaningful organization of the captured scene for editing. Furthermore, due to the simplicity of the system architecture, it is easy to plug in new tools and techniques, either for modeling or editing.

Structure-preserving clone brush

The clone brush is one of the most powerful and widely-used tools in photo editing. It copies from a selected region of the image and pastes to another via a brush interface. The brush interface provides fine user control, and the reuse of existing pixels makes it easier to perform seamless alterations and maintain photorealism. Yet, like all photo-editing tools, there are limitations due to its 2D nature. The input photograph is a 2D depiction of a 3D scene, and the traditional clone brush is a 2D tool that cannot correctly handle perspective foreshortening and object shapes in 3D space.

We present a new structure-preserving clone brush that alleviates the foreshortening artifacts of the traditional clone brush on flat surfaces. We exploit the structural cues present in the image to determine the vanishing line, which in turn is used to handle perspective foreshortening. One of the main advantages of this tool is that it works on purely 2D images, i.e. images without the depth information, and can be directly plugged into existing photo-editing systems. We further improve the tool by introducing two useful features: color correction and snapping. The color-correction feature adjusts for the varying colors between source and destination regions, and snapping provides a more precise means to align the clone-brushed regions.

Non-distorted clone brush

We introduce a non-distorted clone brush that exploits the depth information from our image-based representation to alleviate the distortions due to foreshortening or surface orientation. Unlike the structure-preserving clone brush discussed above, it works on arbitrary 3D geometric shapes. We parameterize the source and destination regions using an optimization technique that conforms to the underlying geometry. We then use this parameterization to map source points to the destination points. This minimizes distortion artifacts from perspective and surface shape. This tool is crucial for filling in gaps that appear due to occlusions in the input image when the viewpoint changes.

Texture-illuminance decoupling filter

In order to discount the existing lighting, we present a filter to separate texture and illuminance components from images of uniformly textured objects. It factors the image into two channels, one containing the large-scale features (assumed to be from illumination) and one containing only the small-scale features (assumed to be from texture). This filter works even in the presence of sharp illumination variations, since we use an edge-preserving filter that decouples the lighting discon-

tinuities as illuminance. The filter produces uniform textures and is useful for clone brushing and relighting applications. After the image has been decoupled into illuminance and texture channels, it is possible to change the scene illumination by painting over the illuminance channel or change the materials of the image, while maintaining photorealism.

1.2 Thesis Overview

The rest of the thesis is organized as follows. In the next chapter, we discuss related work in photo editing, IBMR, and image-based editing. In Chapter 3, we discuss the architecture of our system. We describe the three main components: the image-based the representation, the interactive display, and a suite of editing tools. The interactive display enables the user to interactively visualize the current state of the image-based representation, and editing tools are used to modify the representation. We also introduce a class of depth assignment tools based on a 2D painting metaphor. Most of these depth tools are the work of Max Chen for his Master's Thesis [Chen 2001]. Along with the versatility of our representation, the new class of depth tools provides a means to convert a flat image to a 3D scene. In the following three chapters, we describe two new clone brush tools in Chapters 4 and 5, and the texture-illuminance decoupling filter in Chapter 6. Finally, we show results in Chapter 7 and discuss some possible future directions in Chapter 8.

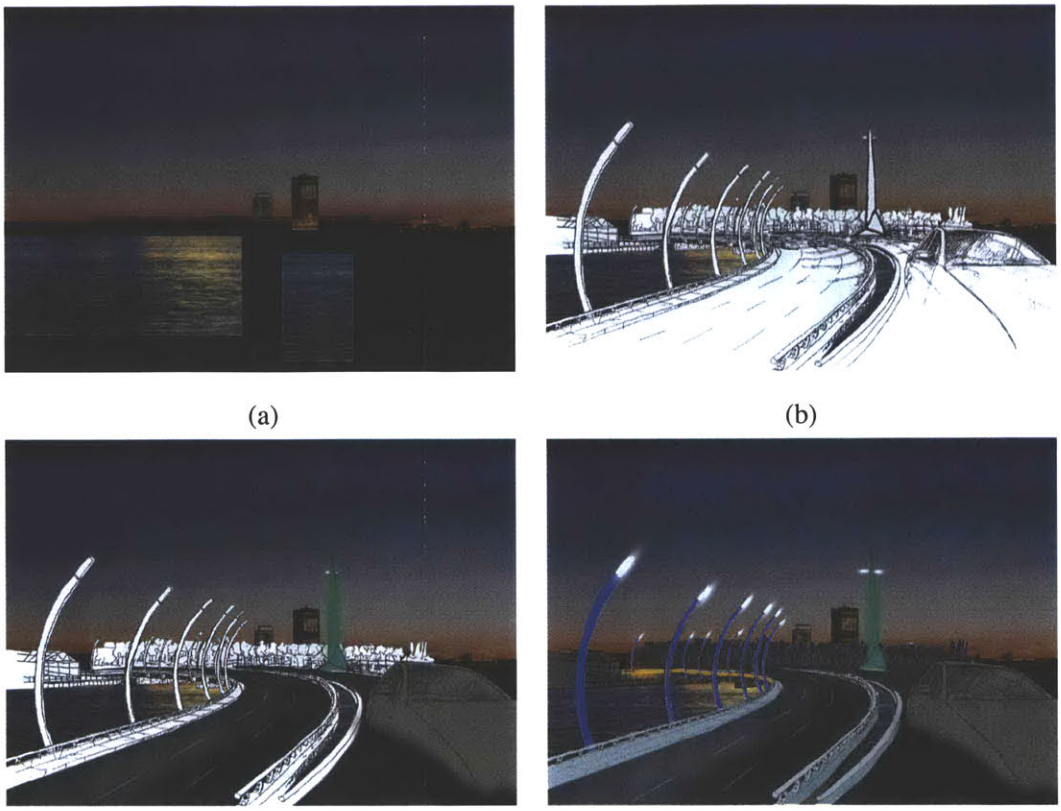
CHAPTER 2

Related Work

The system introduced in thesis builds on and extends several areas of research in computer graphics and imaging. In this chapter, we discuss these areas, noting their relevance to our system. We first provide an overview of photo-editing systems, where we discuss various classes of such systems, types of editing tools, and their application. Next, we discuss related work in image-based modeling and rendering. More specifically, we discuss IBMR approaches that model traditional polygonal representations from a set of images, or use sample-based geometric information for rendering. Finally, we discuss the related work in editing systems. Much of the focus in IBMR has been on modeling and rendering the scene from a set of images, but little attention has been given to addressing the problem of *editing* the image-based representation. We discuss some relevant image-based editing works as well as other related editing systems.

2.1 Photo-Editing Systems

With the proliferation of digital cameras, scanners, and printers, photo- and image-editing software packages, such as [Photoshop; Gimp], are becoming not only popular but an essential part of many applications and industries. Initially, 2D paint programs were introduced by Dick Shoup



(a) (b)

(c) (d)

Figure 2-1: Urban lighting design sketch by an artist using Adobe Photoshop. Layers are shown in progression from (a) to (d).

(a) (b)

Figure 2-2: Relighting applications. Transparent layers are used to add lighting effects. (a) Before. (b) After. Courtesy of Barry Webb and Associates.

[1972] and Alvy Ray Smith [1975], where a user could interactively paint on a blank canvas of limited resolution and color. These simple programs evolved into a powerful and versatile photo-editing framework that enables the user to modify and edit digital images and photographs. Such a framework provides the user with an almost unlimited choice of brushes and colors, and there are hundreds of editing tools and filters spanning a broad array of effects.

Today, a wide spectrum of photo-editing systems are available — from simple home-user systems to complex professional versions. Many applications depend heavily on photo editing: graphics and web design, photography, magazine and newspaper layout, book publishing, architecture, interior design, landscaping, stage and set design, special effects, movie editing, etc. Anyone needing to manipulate, edit, create, and work with images uses photo-editing systems. For instance, photo-editing systems are extensively used in many design industries, such as in architectural, lighting, urban, and landscape design. Such systems are especially useful when designers wish to include the surrounding context along with their designs — for example, how will a new building look in the current urban context?

Most high-end photo-editing systems are capable of displaying *layers* of images, which are used to superimpose and composite multiple images, as shown in Figures 2-1, 2-2, and 2-3. Layers are a powerful way of organizing and decomposing images. Layers include the alpha channel, also known as the transparency channel, in combination with the color channel [Porter and Duff 1984]. For each pixel in a layer, the user can specify a degree of transparency that displays blending effects with the occluded layer behind. This is a simple, yet powerful framework, and the standard in photo editing.

Unlike vector-based drawing systems, such as the Adobe Illustrator [Illustrator], the image representation in photo-editing systems is sample based, i.e. pixels, making it more amenable for editing photographs. This simple representation allows practical, intuitive, and powerful operators to be built, and the user has the freedom to directly and interactively modify and edit the layers of images with tools and filters. There are a wide variety of tools and filters available for high-end photo-editing packages. Some note worthy classes of tools are selection, brushes, and filters. *Selection* allows the user to choose a region of the image, i.e. a set of pixels, that will be affected by the tools and filters. In other words, only the pixels that are selected may be edited. By default, all pixels are selected and, therefore, editable. Selection is also used for copying and pasting parts of the image. *Brushes* allow the user to interactively "paint" over regions of the image. Some examples of the brush operators are paint brush, air brush, the smudge tool, and the clone brush. *Filters*,

Figure 2-3: Photo editing used for landscape design. Photos on the left column show the original and the right column shows a possible design scheme.

unlike the brushes, work on selected parts of the image according to user-specified parameters of the filter. They do not have an interactive brush but operate evenly on the entire selected parts of the image. They are usually more computationally expensive, and thus, not interactive. Some examples of filters are blur, sharpen, denoise, bump, etc.

The appeal and power of these photo-editing systems lies in the immediate feedback they provide to the user after changes are made to the representation, similar to pen and paper, but in a much more powerful framework. When the representation is edited with tools and filters, the system immediately displays the current state of the representation. The user can iteratively edit and modify the image while seeing the changes in real time.

Although photo-editing systems provide powerful tools and filters that can drastically modify an image, there are limitations mainly due to their 2D nature. A photograph can capture the richness and the complexity of the scene, but it is, nonetheless, a flat image. 2D editing tools and filters are not able to consider the 3D scene depicted in the image in their operations. For instance, as previously mentioned, the traditional clone brush cannot take the 3D scene into account due to the lack of geometry information. It is also difficult to convincingly modify the lighting, discount existing lighting, seamlessly change existing materials, or freely transform and modify the objects in a 3D fashion. As we will see, we address these issues in our system.

2.2 Image-Based Modeling and Rendering

In this section, we discuss related work in image-based modeling and rendering. We distinguish three classes of image-based techniques. The first is *image-based rendering*, which is based on sampling. A scene is represented as images with depth, and view warping is employed to render new views from a set of photographs [Chen and Williams 1993; Laveau and Faugeras 1994; McMillan and Bishop 1995; Shade et al. 1998]. Higher-dimensional representations exist, such as [Gortler et al. 1996; Levoy and Hanrahan 1996], but they are beyond the scope of this thesis. The second class concerns related work in *image-based modeling* that builds a more traditional geometric representation from photographs [Faugeras et al. 1995; Debevec et al. 1996; Liebowitz et al. 1999; Poulin et al. 1998; Photomodeler; Canoma; Realviz]. Using photogrammetry techniques and recovering textures from the photographs, these systems can construct photorealistic models that can be readily used with widely-available 3D packages. Finally, we discuss related work in *single image modeling*. This is a special case, where only a single image is available, and the user infers the scene geometry via interactive tools.

2.2.1 Image-Based Rendering

(a) (b)

Figure 2-4: (a) Color channel. (b) Depth channel. Darker pixels are closer to the camera.

Image-based rendering techniques relevant to our work use depth images that store both color and depth (or disparity) information per pixel (Figure 2-4). The depth information encodes the

distance from the 3D position of the projected pixel to the camera. The disparity information is analogous to the depth information, though they are inversely proportional. As we will see, we use a modified version of depth images as the underlying representation for our system.

In these systems, standard cameras are used to capture the scene at various positions and directions, and then used as inputs to build the representation. The most difficult and time-consuming process is recovering depth from the input images. To compute the depth, one must solve the camera registration and the correspondence problems between multiple images. Once the depth has been recovered, image warping [Chen and Williams 1993; McMillan and Bishop 1995] is employed to display the representation. Image warping maps from the representation, in this case the *reference images* with depth per pixel, to the *desired image* — the synthesized new image from the user-specified viewing position and direction. We discuss 3D image warping, as introduced by McMillan and Bishop [1995], in detail in Chapter 3.

Chen and Williams first introduced this data structure to demonstrate their interactive view interpolation system on a synthetic scene, where the depth information was easily computed [Chen and Williams 1993]. They used several regularly-spaced reference images along with the precomputed depth information to enable the user to change viewpoints interactively. To render new views, they used linear interpolation between corresponding points instead of 3D image warping, which provides reasonable estimates with slight viewpoint changes.

Laveau and Faugeras [1994] and McMillan and Bishop [1995] take advantage of the epipolar geometry to compute the depth per pixel. The epipolar geometry is used to solve the correspondence problem in multiple cameras, where it is used to constrain the potential reprojections of the reference image to the desired image. They describe the projective shape of the scene with scalar depth values at each sample point. Laveau and Faugeras also discuss combining information from multiple images to resolve visibility.

McMillan and Bishop further demonstrate their image-based rendering system on real scenes (Figure 2-5). Their representation consists of a set of cylindrical panoramas, which are built from planar images acquired from the real world. The depth map is computed from multiple cylindrical panoramas using stereo correspondence techniques and user-assisted inputs. Once the depth is determined, new views are rendered at interactive rates via 3D image warping — by "unprojecting" the sample points from the reference image to their corresponding 3D location in world space, then reprojecting them to the desired image. They introduce a view-dependent rendering order of pixels from a reference image using a painter's algorithm, which avoids the use of the hardware Z-buffer.

Figure 2-5: Plenoptic Image Editing [McMillan and Bishop 1995]. Two cylindrical images separated by approximately 60 inches. Courtesy of Leonard McMillan.

Shade et al. introduced a data structure, which they call the layered depth image (LDI) [Shade et al. 1998]. This consists of a linked list (or an array) of multi-layered color and depth information per sample point. There are a couple of advantages to this representation. First, the LDI can represent surfaces that are occluded from the viewpoint of the reference image. Second, the representation is amenable to rendering without the aid of the graphics hardware using a technique such as [McMillan 1995].

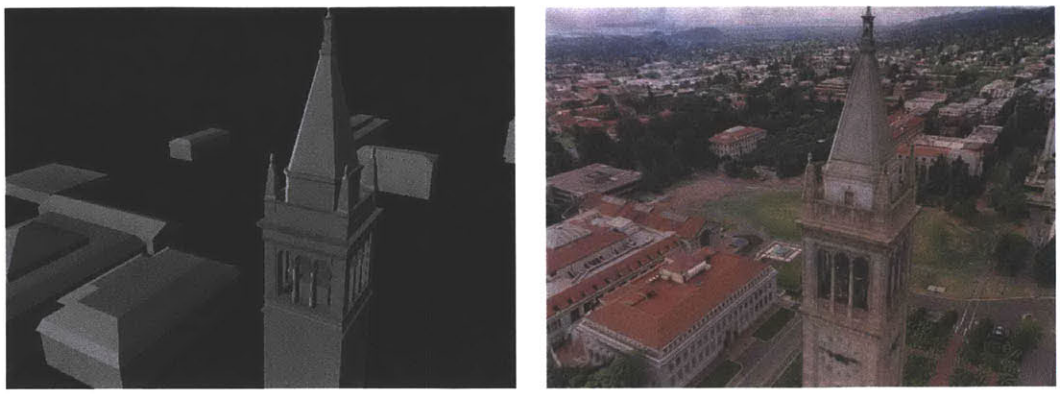
One of the major difficulties in using depth images is in extracting depth information. Despite decades of research, reliable depth extraction remains elusive. McMillan and Bishop note that extracting depth from stereo images of complex real scenes was difficult, time consuming, and required much manual handling of the depth computation process. And Shade et al. use simple planar depth images in constructing the LDI data structure.

In our system, we employ a modified version of depth images, and we organize these images into multiple layers. To extract the depth information, we use a technique similar to the single view modeling systems mentioned in Section 2.2.3.

2.2.2 Image-Based Modeling

Image-based modeling addresses the problem of acquiring polygonal geometric models from photographs. Similar to images with depth, image-based modeling techniques store color and geometry information, but instead of storing sample-based depth per pixel, the geometry is represented using polygons and textures.

Debevec et al. introduced an image-based modeling system called *Façade*, which they developed to reconstruct architectural scenes from a sparse set of photographs [Debevec et al. 1996]



(a) (b)

Figure 2-6: Snapshots from *the Campanile movie* [Debevec 1997]. (a) Recovered geometry. (b) Texture-mapped scene. The scene was modeled using the image-based modeling system, *Façade* [Debevec et al. 1996; Debevec et al. 1998]. Courtesy of Paul Debevec.

(Figure 2-6). The user interactively specifies and matches line segments on the architectural features of the multiple input images. Photogrammetry techniques are then used to match line segments between images [Taylor and Kriegman 1992], and parameterized geometric primitives, such as boxes, prisms and surfaces of revolution, are fitted to reconstruct the geometry. The input photographs are reused as projective textures, and the results are photorealistic 3D models of architectural scenes, where the representation consists of polygons and texture maps.

In recent years, image-based modeling has become an active area of research and development both in the research community as well as in industries, including special effects, games, and architectural design. Image-based modeling techniques have been used to generate special effects for films, such as *The Matrix*, and there are commercial software products available that use photogrammetry techniques to recover geometry and reuse the photo-textures [Photomodeler; Canoma; Realviz].

There are many advantages in producing polygons and photo-textures. First, for architectural and urban scenes, image-based modeling techniques exploit the elements of shape, which provides a much better global geometry approximation than using sample-based images with depth representation. Second, the representation can display photorealistic reconstructions of buildings using current graphics hardware at interactive rates. Interactively displaying a scene is an important, if not critical, component in many applications. Third, the results can be edited and combined with existing 3D modeling systems, such as [AliasWavefront; Discreet]. Being able to combine the re-

sults of these systems with powerful 3D modeling systems opens up numerous avenues for various applications. Finally, due to the polygonal representation, it is possible to employ inverse lighting simulations to recover the material and illumination characteristics from the input photo-textures, to which common illumination or relighting may be applied [Fournier et al. 1993; Drettakis et al. 1997; Debevec 1998; Yu et al. 1999; Loscos et al. 1999].

However, image-based modeling techniques are not without limitations. The use of traditional geometric primitives for approximating geometry can be limiting in the case of non-architectural scenes. In general, it is difficult to model organic shapes. Recovering organic and curved shapes from sparse images for objects, such as trees, statues, and cars, is a much more challenging task especially with basic polygonal primitives. Also, the representation assumes a diffuse environment, similar to the images with depth representation, unless view-dependent texture mapping is used [Debevec et al. 1998; Pulli et al. 1997]. Although specular effects may also be added as a postprocess, they do not faithfully redisplay these effects from the captured scene.

Similar to image-based modeling approaches, we also provide geometry-assignment tools that use planes and polygonal primitives, which are more suitable for architectural and urban scenes. As we will see in Chapter 3, our representation is sample based, similar to images with depth, and thus the polygonal geometry used for assigning depth is temporary.

2.2.3 Single Image Modeling

Single image modeling involves inferring 3D shapes from single images, either automatically or interactively. Automatic 3D reconstruction from a single image has been a long-standing problem in the vision community. Much of the techniques make strong assumptions about shape or reflectance, and are limited to specific classes of images, e.g. [Horn 1990]. More recently, several researchers have used interactive approaches that enable the user to specify the shape from visual cues. These user-in-the-loop techniques, where the geometry acquisition process is user specified, have proven to be highly effective and efficient [Horry et al. 1997; Kang 1998; Zhang et al. 2001]. The depth information is interactively specified by the user via tools and operators, which eliminates the need to extract unreliable depth automatically from multiple images.

Horry et al. introduced their tour-into-the-picture approach, which is one of the inspirational works for our system [Horry et al. 1997]. They use a central perspective interface, which they call a "spidery mesh," to approximate the ground plane, from which flat 2D planes, i.e. billboards, are used and placed relative to the ground. Billboards are simple texture-mapped planes with alpha channels

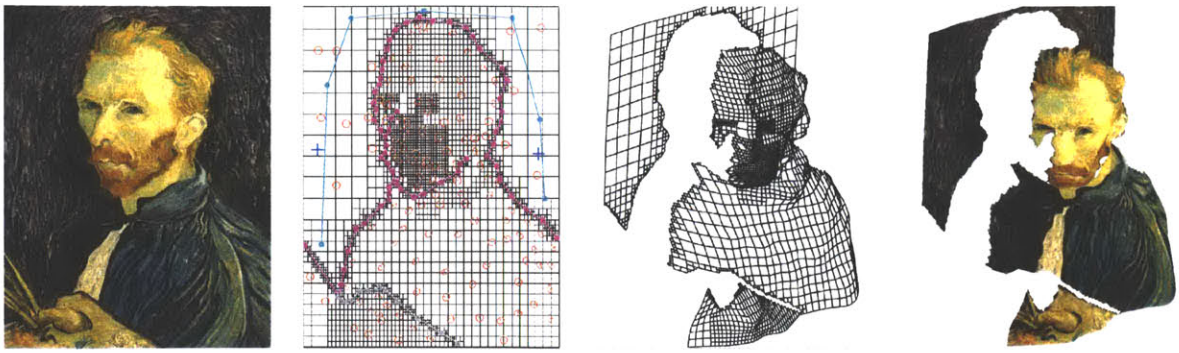
that provide detailed silhouette contours of the objects depicted. The user can then navigate into a 2D image in a 3D manner. Although the resulting scene geometry is limited and most foreground objects are flat, their results are nonetheless convincing, and the system enables the user to change viewing positions and experience parallax effects. Furthermore, modeling a scene is trivial, since simple photo-editing systems are used to separate foreground objects from the background using alpha channels.

Kang and Williams proposed depth-brush interfaces, where they leave the depth assignment task to the human artist [Kang 1998; Williams 1990]. They provide several painting tools that allow the user to adjust the depth of the image, and Kang provides multiple views from different angles to show the changes to the user in real time. Williams also uses level sets from silhouettes and image grey levels [Williams 1998] to compute approximate depth.

Criminisi et al. use vanishing lines and points to determine the measurements and the camera pose from a single image [Criminisi et al. 1999]. They assume that the input image has visual geometric cues, such as parallel lines, to first compute the reference plane using homography. They are then able to take measurements of objects in the image. They also demonstrate from the gathered information that they are able to reconstruct a 3D scene using a similar technique to that of Horry et al.

Zhang et al. present a new class of tools for reconstructing a 3D scene model from a single image [Zhang et al. 2001]. They cast the single-view modeling problem as a constrained optimization problem, where the user specifies the constraints, such as surface positions, normals, silhouettes, and creases. The system, in turn, generates a piecewise continuous hierarchical surface representation that satisfies the constraints (Figure 2-7).

There are many advantages to single-image modeling systems. First, in many cases, only a single view of the scene is available, in which case, techniques that require multiple images to compute depth cannot be used, such as stereo vision or photogrammetry techniques. Another big advantage is that there is no need to solve the difficult, and often unreliable, correspondence problem from multiple input images. There is no need to manually specify correspondence information, fix unreliable depth, especially for complex real scenes, or employ optimization techniques that may be difficult to control. Second, mainly due to the user-driven tools, the accuracy and precision of the inferred geometry are user-dependent parameters. The user has the freedom to directly interpret the depth of the scene from visual cues on the image, and to focus more time and effort on specific parts of the image that are of interest. In essence, the user solves the depth extraction problem. For



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

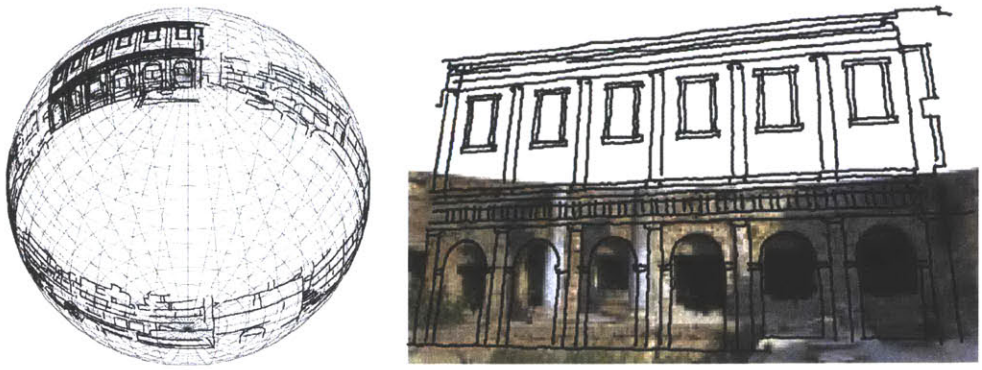
Figure 2-7: Single View Modeling [Zhang et al. 2001]. Courtesy of Steven Seitz.

instance, given an ambiguous painting by Salvador Dali, the user can freely interpret the geometry of the scene (see Figure 7-1). Also, techniques, such as [Criminisi et al. 1999; Liebowitz et al. 1999], may be used on photographs with planar features to compute a more accurate geometric information.

We also use a single image as input to our system. As we will see in the next chapter (Chapter 3), we provide a suite of interactive depth assignment tools that enable the user to infer depth from a single photograph. Some tools we provide are similar to [Criminisi et al. 1999; Liebowitz et al. 1999], where we can accurately infer planar geometry and compute the field of view of the camera; other tools are similar to [Kang 1998; Williams 1990], where the user can interactively refine the depth information via a brush interface.

2.3 Image-Based Editing

In this section, we discuss related work in image-based editing. We first discuss a projective drawing system that enables the user to interactively sketch on a panoramic canvas in an outward-looking manner. We next discuss related work in 3D painting systems that allow interactive painting on 3D polygonal representations. Finally, we discuss two relevant image-based editing systems that allow the user to interactively edit image-based representations via operators similar to those in photo editing.



(a) (b)

Figure 2-8: Projective drawing system [Tolba et al. 2001]. Courtesy of Osama Tolba.

2.3.1 Projective Drawing System

Tolba et al. proposed a projective drawing system that allows the user to draw on a panoramic scene directly [Tolba et al. 2001; Tolba 2001]. They use a spherical map and a projective representation for strokes. The display interactively projects the drawing strokes onto the image plane of the pinhole camera positioned at the center (Figure 2-8). The representation of the user-sketched marks are 2D, but they also provide pseudo-3D effects that enable the user to translate and rotate the drawings. This is accomplished by storing normal information for each depicted planar surface instead of 3D geometry information. They are also able to modify the illumination conditions of the scene, such as casting shadows from the sun and appropriately modifying the surface shading, also accomplished by using the normals.

Although photo-editing systems are often used to edit panoramas, one of the main problems is that unwrapped spherical or cylindrical panoramas display straight edges as sinusoids. Cube maps can maintain the straight edges, but they need to be separated into six images, which makes it difficult to edit the environment as a whole. Although the projective drawing system focuses on the drawing and sketching issues, this idea can be extended to photo-editing systems, where the powerful interactive tools may be applied panoramic images.

2.3.2 3D Painting Systems

3D painting systems allow users to interactively edit the material and geometry of a 3D model via a brush interface. They are adaptations of popular 2D photo-editing systems to the painting of textures and other attributes directly on the 3D model.

Hanrahan and Haeberli first introduced their interactive WYSIWYG (What You See Is What You Get) painting system [Hanrahan and Haeberli 1990], where the system enables the user to interactively paint material attributes, such as color, specularity, roughness, and thickness, onto the 3D model. One of the problems with the paint brush interface is the interpretation of the 2D pointer (e.g. a mouse) on the 3D surface. To determine the mapping from the 2D pointer position to the texture coordinates on the 3D object surface, they use the framebuffer of the graphics hardware to encode the texture coordinates. In Chapter 3, we address a similar issue using inverse warping. They introduce three types of 2D-to-3D interactions: the parameter-space, the tangent-space, and the screen-space brushes. The *parameter-space* brush paints directly to the 2D texture parameter space. It's the simplest mapping technique, but not always intuitive due to texture distortions. The *tangent-space* brush utilizes the normal of the 3D surface to determine the orientation of the brush. The *screen-space* brush is projected directly onto the surface of the 3D model using the inverse viewing transformation. This technique is similar to the painting brush tool used in our system. Once the interactive user specification was done, the system parameterized the painted regions as texture maps to display the updates from any viewing position.

Agrawala et al. describe their 3D painting system for scanned surfaces in [Agrawala et al. 1995]. Their painting system is geared towards densely-meshed scanned 3D objects, and they use the sensor of a Polhemus 6D tracker as a paintbrush. Their paint brush is also three-demensional, where they use spheres, cylinders and cones as brushes. They are then able to interactively paint colors and chisel geometry of the 3D model.

There are also commercial software systems available that provide a powerful framework for modeling and editing 3D models using a brush interface [Pixologic; RightHemisphere; AliasWavefront]. Similar to the 3D painting introduced by [Hanrahan and Haeberli 1990], these systems are capable of interactively creating and editing 3D models.

2.3.3 Interactive Editing Tools for Image-Based Rendering Systems

One inspirational work for our system was Rangaswamy's image-based editing system [Rangaswamy 1998]. This work is one of the earlier attempts to provide an editing framework for images with depth representation, e.g. [Chen and Williams 1993; McMillan and Bishop 1995]. Multiple reference images are used as inputs to the system, and the depth information is assumed to have already been acquired. The author notes that acquiring the depth information for real scenes is difficult and refers to the techniques used by McMillan and Bishop [1995]. He provides a suite of editing tools

Figure 2-9: Plenoptic image editing [Seitz and Kutulakos 1998]. Courtesy of Steven Seitz.

similar to those in photo-editing systems, such as scissoring and painting, to modify the representation. Due to the depth information, these operators may be applied from multiple viewing positions. They also use layers to group parts of the input representation for cutting and pasting.

The differences between Rangaswamy's system and ours is that we take a single image as input instead of multiple images. Our system is more compatible with traditional photo-editing systems, thus providing a consistent and intuitive interface. We also provide a suite of depth assignment tools, since reliable depth extraction using stereo vision technique is challenging. Furthermore, we enable the user to edit the shape of objects, i.e. depth, and also provide tools to photorealistically modify the existing lighting and material conditions.

2.3.4 Plenoptic Image Editing

The plenoptic image-editing technique introduced by Seitz and Kutulakos was another source of inspiration for our system [Seitz and Kutulakos 1998]. They allow interactive editing operations on

multiple reference images via tools are similar to those in 2D photo-editing systems, such as painting, scissoring, and morphing. The editing operator applied on one reference image is automatically propagated to others in a consistent and predictable manner.

They first acquire multiple views of the same object in an outside-looking-in manner, from which they construct a voxel representation to compute the pixel correspondences between multiple input images using the voxel coloring technique [Seitz and Dyer 1997]. The quality of their results is dependent on the volumetric representation that they use. They separate the shape and radiance components, which they call *plenoptic decomposition*, and formulate image editing as a set of operations that act on either the shape or the radiance component. They first estimate the shape of the object by using the voxel carving technique, where the initial blocks of voxels are carved away according to multiple views and the coherence of incoming light [Seitz and Dyer 1997]. Since the camera pose and input images are known, the problem basically solves for a photometrically consistent set of pixels for each voxel. The voxels, therefore, encode the corresponding pixels from the input images. Editing operations applied on one of the views are propagated to other images using the voxel representation, either to modify the shape or the radiance component. For instance, if a pixel is painted from one view, its corresponding voxel is first determined, then the radiance change is propagated to other images.

The differences are that we deal with single images and do not require a complex acquisition setup. Moreover, our system can treat high-resolution images. However, their system is capable of capturing and displaying view-dependent effects. The lower-dimensionality of our representation permits lower storage and computation cost, higher sampling resolution, and simpler editing strategies, at the expense of view-dependent effects.

2.4 Discussion

The depth assignment aspect of our system may be seen as a generalization of Horry et al.'s *Tour into the Picture* [Horry et al. 1997]. As previously mentioned, their work was one of the inspirations for our system. It is also important to note that our system was designed with the objective of enabling users to not only create a much richer geometric representation, but also be able to freely *edit* the scene. We provide a much more flexible data structure amenable for editing, and provide numerous tools for flexible and powerful modifications of the scene.

Systems have been implemented that address various editing issues, but their focus and ca-

pabilities are usually limited. Not all systems address the problem of editing shape, texture, and illumination. Seitz and Kutulakos [1998] and Rangaswamy [1998] are among the pioneers in the area of image-based editing. They address the issues involved in editing shapes, textures, as well as illumination conditions for image-based representations.

Similarly, one of the goals for our system was to provide a complete framework for an image-based editing system. This means that the system should enable the user to edit not only the geometry, but also the texture and lighting of the scene as well. The system takes a single image as input, but unlike typical single view modeling techniques, we provide multiple layers such that occluded regions of the image are easily accessible and the holes may be filled in. The data structure we use is *layers of images with depth*. Furthermore, the simplicity of our representation allows powerful tools to be built for the system. The familiar tools of the photo-editing systems are easily extended to our system, such as painting, cutting and pasting, scissoring, and selection. We provide a suite of interactive tools for the user to intuitively specify and edit depth from visual cues available on the image. New tools have been implemented for clone brushing, as well as tools that enable the user to edit the lighting or the materials, even with a single input image.

CHAPTER 3

System Architecture

In this chapter, we describe the architecture of our system to provide a comprehensive understanding of the data structures, the editing tools, and the user interaction in modeling and editing a 3D scene from the input image. As previously mentioned, the main philosophy of our system is to combine the strengths of both photo-editing systems and IBMR techniques. Thus, our goal is to design a system that meets the following criteria: data structures that can be easily accessed and modified; an interactive display that enables the user to render new views from the input image in real time; tools to edit the scene from any given viewpoint; and a real-time feedback loop that allows dynamic and iterative editing.

The architecture of our system is simple: it consists of a set of tools organized around a common data structure. As illustrated in Figure 3-1, there are three main components in our system: the image-based representation, the interactive display, and the tools. The *image-based representation* is the data structure composed of layers of images with depth (Figure 3-1(a)). The depth channel stores the "geometry," that is, the projective depth information. Note that our representation does not store any 3D vertices or polygons — simply images and depth information. The *interactive*

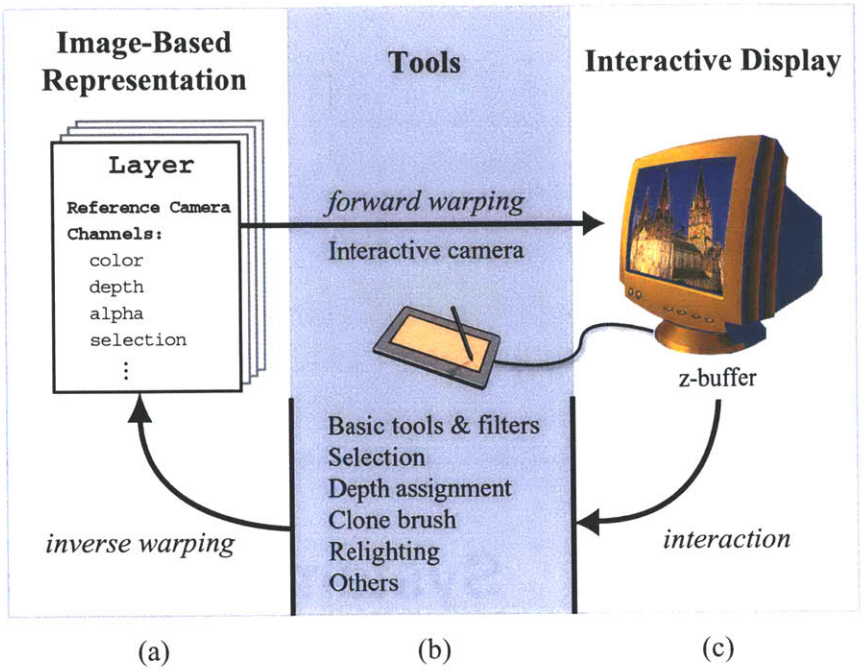


Figure 3-1: System architecture. The main components of the system are the image-based representation, the interactive display, and the tools.

display allows the user to visualize the current state of the image-based representation (Figure 3-1(c)). An image-based rendering technique, denoted as *forward warping* in Figure 3-1, enables the user to interactively navigate and change viewpoints within the scene. Finally, *tools* provide the means to interactively edit and modify the representation from any viewpoint (Figure 3-1(b)). This component consists of basic tools, a set of operators that is similar to those in photo-editing systems, as well as a new class of tools, such as depth assignment, clone brushing, and relighting tools. These tools may be used from any viewpoint, and the image-based representation is updated with the modifications via *inverse warping*. The interactive display, in turn, reflects the modified state of the data structure in real time. This real-time feedback loop enables the user to interactively edit the representation, facilitating an iterative editing process.

In the following sections, we discuss the components of our system in detail. First, we describe the data structures of the image-based representation and a typical workflow scenario, which demonstrates how a scene is modeled and edited from a single photograph (Section 3.2). We then discuss the interactive display, as well as image warping in Section 3.3. Image warping essentially acts as a bridge between the data structure and the interactive display, and enables the user to interactively

visualize and edit the representation from any viewpoint. Finally, we discuss the tools in Sections 3.4 and 3.5. We first describe the basic tools that are natural extensions of existing photo-editing tools (Section 3.4), and then introduce a new class of tools for interactively assigning depth to the input image (Section 3.5). Much of the work described in the depth assignment section was done by Max Chen as a part of his Master's Thesis [Chen 2001]. These tools provide an intuitive means to convert a 2D photograph into a 3D scene. In the following chapters (Chapters 4, 5, and 6), we introduce the new clone brushing and relighting tools.

3.1 User Workflow

Before we discuss the components of the system in detail, we illustrate a simple scenario of the user workflow to provide a better intuition of the system (Figure 3-2). Although the features of the system are presented sequentially, all processes are naturally interleaved: editing can start even before depth is assigned, and the representation can be refined while the editing proceeds. The tools may be used from any viewing position via the interactive display.

Initially, the input image is manually segmented into layers, i.e. *layers of images*, using selection, alpha masks, and traditional photo-editing tools This process is the most time-consuming step. Figure 3-2(a) shows the layers outlined in red. Depth is then specified within each layer using a suite of interactive depth assignment tools (Figure 3-2(b)). The user applies coarse depth initially, and then further refines the depth (Figure 3-3). These tools are discussed in Chen's thesis [2001] and summarized in Section 3.5. Our system is also capable of displaying and editing depth captured through other means, such as laser-range scanning, image-based modeling, and stereo-vision techniques. Next, parts of the scene hidden behind the segmented layers in the input image are manually filled in, typically using clone brushing (Figure 3-2(c)). As previously mentioned in Chapter 1, the clone brush is one of the most powerful tools in photo editing; it allows the user to interactively copy and paste existing parts of the image via a brush interface. If the depth has not yet been assigned, the traditional clone brush or our new 2D clone brush could be used (Chapter 4). If the depth has been assigned, our new 3D clone brush tool may be used (Chapter 5). Finally, the user can employ our texture-illuminance decoupling filter to seamlessly edit the lighting or the materials in the scene (Chapter 6). The user, at any given time, can transform and copy-paste the layers, and apply the tools from any viewpoint.

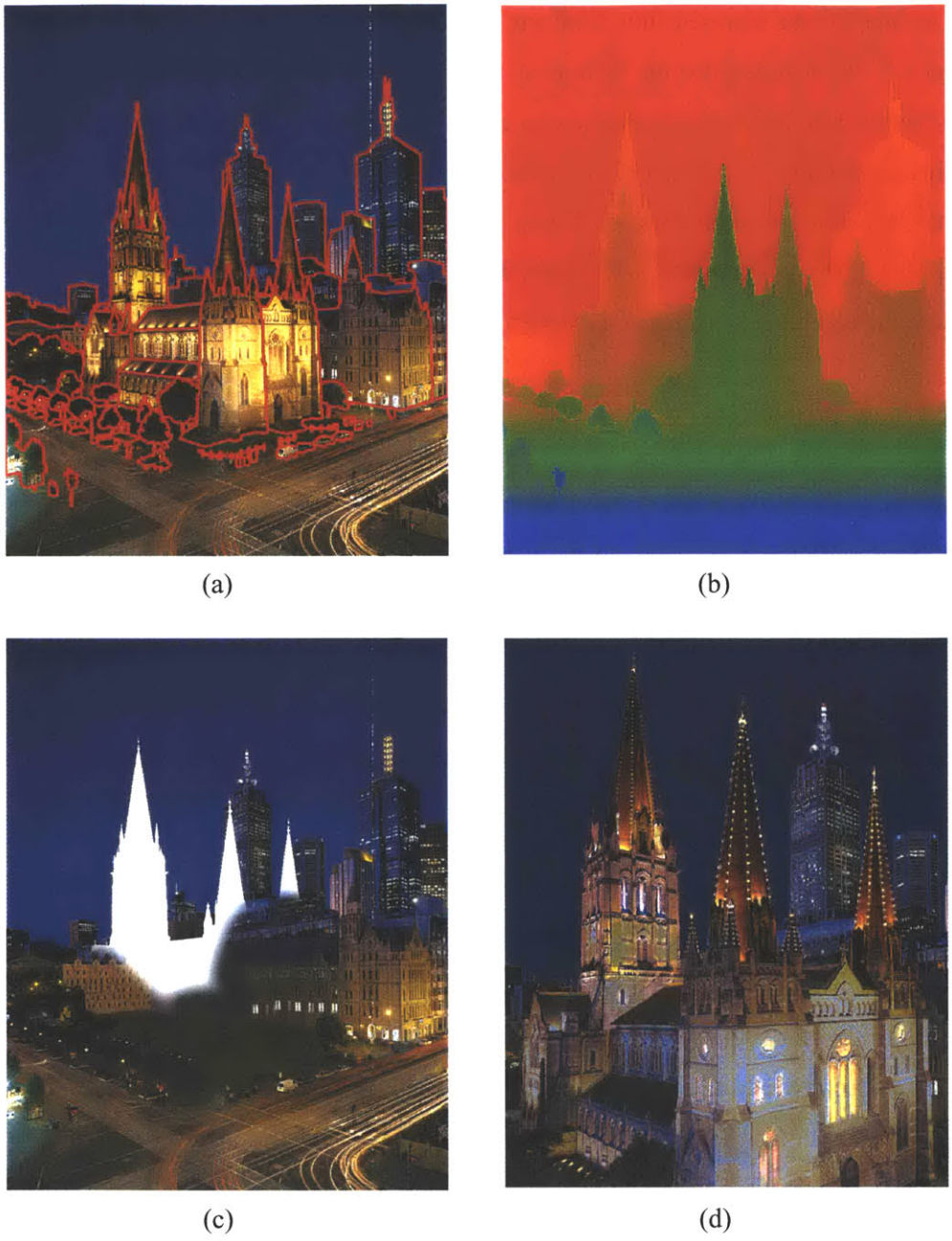
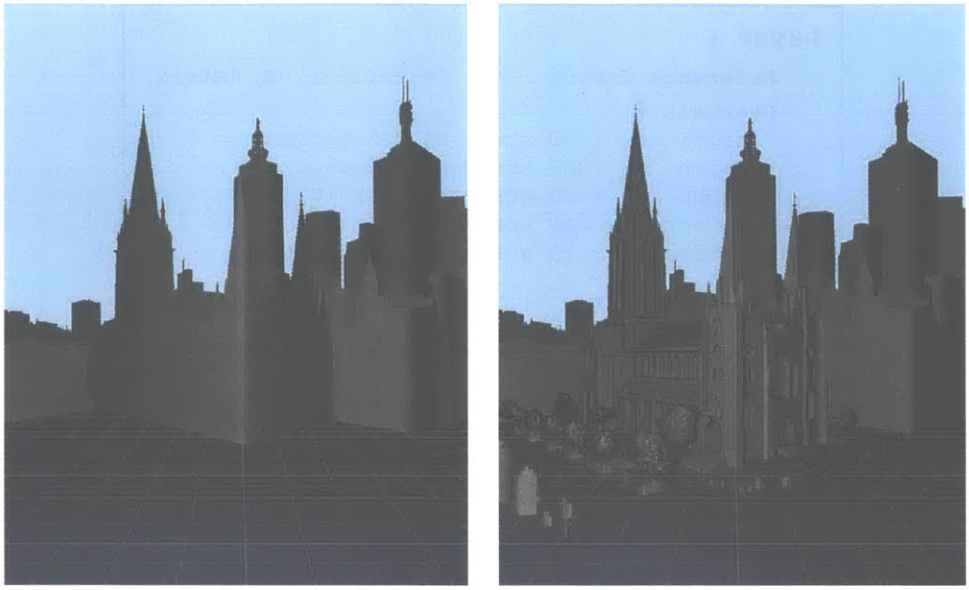


Figure 3-2: A typical user workflow of our system. (a) Image segmented into layers (boundaries in red). (b) Depth assignment tools are used to edit the depth channel (Section 3.5). False-color rendering of the depth of each pixel. (c) Hidden parts clone brushed by the user (Chapters 4 and 5). (d) New viewpoint and relighting of the roof and towers (Chapter 6). A photograph of the St. Paul's Cathedral in Melbourne, courtesy of Barry Webb and Associates.



(a) (b)

Figure 3-3: Depth refinement. (a) Coarse depth. (b) Refined depth.

3.2 Image-Based Representation

All elements of the system operate on the same simple data structure: *layers of images with depth*. In addition to the RGB color component information per pixel, *images with depth* store the projective distances from the camera to the object for each pixel. This scalar depth value is defined up to a global scale factor. This representation permits the use of standard image-based rendering techniques to display the representation [Chen and Williams 1993; McMillan and Bishop 1995; Shade et al. 1998; Mark et al. 1997]. The representation is also organized into *layers of images*, in the spirit of photo-editing systems, and as proposed in computer vision by Wang and Adelson [1994]. The alpha channel, also known as the transparency channel, is used to composite multiple images as layers to form a single image [Porter and Duff 1984]. The layer data structure, which consists of the reference camera, channels, and the bounding rectangle, is shown in Figure 3-4. We discuss each of these components in detail in the following sections.

3.2.1 Reference Camera

Each layer stores a *reference camera*, which is a 4×4 transformation matrix that describes a planar pinhole camera model. The depth channel stores the scalar distance for each pixel, and the reference

```
Layer {
   Reference Camera : 4x4 transformation matrix
   Channels {
     color     : 2D array of RGB floats
     depth     : 2D array of floats
     alpha     : 2D array of char [0..255]
     selection : 2D array of char [0..255]
     others    : 2D array of various types
   }
   Bounding Rect    : top, bottom, left, right
}
```

Figure 3-4: Layer data structure.

camera matrix describes how these pixels are projected from the world space to image space. The notion of projection is discussed in detail in Section 3.3. We use a planar pinhole camera model, and unless other information is available, we assume the optical center is the center of the image. Only the field of view, which can be entered by the user, needs to be specified; alternatively, a default value can be used if accuracy is not critical. Standard vision techniques can also be used if parallelism or orthogonality are present in the image to determine the camera intrinsics [Liebowitz et al. 1999; Guillou et al. 2000].

We assign the reference camera matrix *per layer* instead of one reference camera for all layers, which permits each layer to be transformed and edited independently of others. Note that changing the reference camera parameters is equivalent to transforming the objects depicted in the layer in world space. For instance, if we translate the reference camera of a layer, this results in translating the object depicted by that layer. Initially, all layers segmented from the original image have the same transformation matrix, which is arbitrarily set to the default OpenGL matrix (i.e. the identity matrix).

Throughout the thesis, we will deal with two kinds of images: *reference images* that correspond to the original representation as seen from the reference camera, and *interactive images* that are displayed from different viewpoints as seen from the interactive camera and ease user interaction. In terms of implementation, the interactive and reference cameras have the same data structure. The main difference is that the reference camera stores the transformation information for the layer, while the interactive camera is used for interactively viewing the entire image-based representation.

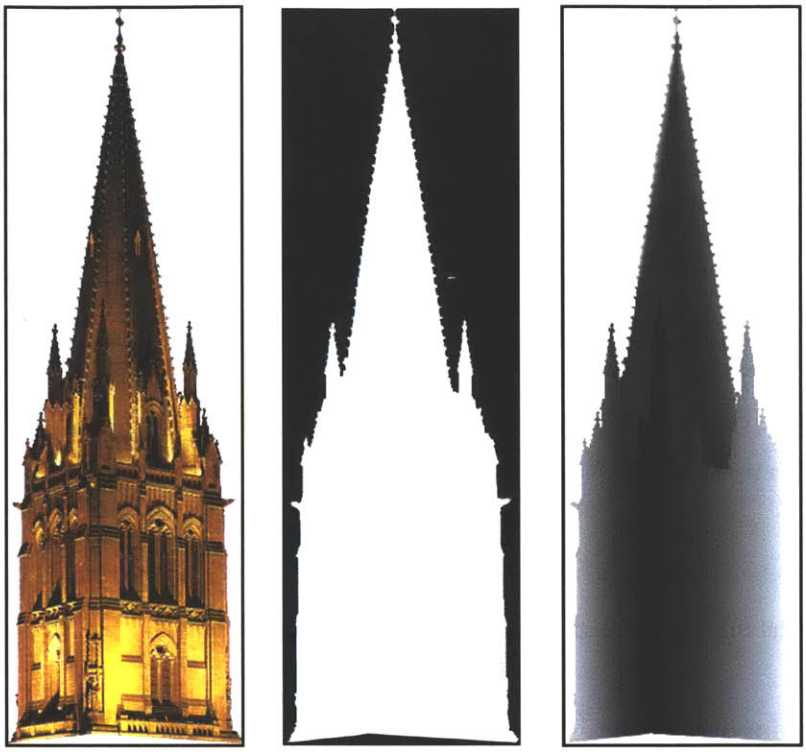


Figure 3-5: Channels and their bounding rectangles. (a) Color, (b) alpha, and (c) depth channels.

3.2.2 Channels

Our main data structure, layers of images with depth, is composed of a 2D array of color, depth, and alpha channels (Figure 3-5). The *color channel* stores the RGB components of the image, which is from the input image. Our system is capable of editing high dynamic-range images, since we store floating-point values per pixel instead of the 24-bit images. We also provide floating-point compression, as described by Ward [1991]. The *alpha channel* is used to represent transparency and object masks [Porter and Duff 1984]. This allows the treatment of semi-transparent objects and fuzzy contours, which are common with fine features, such as trees and hair. Layers of color and alpha channels are used in photo-editing systems to composite multiple images.

The *selection channel* is used for copying and pasting parts of the image, and also for restricting the action of the tools to relevant areas. It is treated as a channel for each layer, and is represented as an array corresponding to the reference image. Acceptable values range between 0 and 255 to

permit feathering. The selection channel is not saved but mainly utilized during the editing process. We provide a set of tools to edit this channel, as discussed in Section 3.4. Additional optional channels may be used for specific applications — for instance, texture and illuminance channels are used for relighting applications (Chapter 6).

Our image-based representation, i.e. layers of images with depth, is related to Layered Depth Images (LDI) [Shade et al. 1998] with one key difference: In an LDI, the layering is done at the pixel level, while in our system, layering is done at a higher level (objects or object parts). LDI's may be better suited for rendering without the aid of graphics hardware, but our representation is more amenable to editing, where the scene is organized into higher-level entities. Furthermore, the reference camera in each layer allows individual layers to be transformed or modified independently of others.

Due to the similarity of data structures, we offer an import/export interface with the Adobe Photoshop format that can be read by many 2D photo-editing systems.

3.2.3 Bounding Rectangle

A *bounding rectangle* is used to store only the necessary sub-image for a layer. Our system is aimed at editing high-resolution images in the mega-pixel range. It is, therefore, necessary to minimize the storage file sizes in practice. Figure 3-5, shows an example of a layer, including the channels and the bounding rectangle.

3.3 Interactive Display and Image Warping

The main purpose of the interactive display component is to allow the user to visualize the current state of the image-based representation (Figure 3-1(c)). As the user controls the interactive camera, the reference image stored in the representation is *warped* according to the the interactive camera parameters and displayed as the interactive image. This enables the user to navigate in the scene and apply editing tools from any viewing position.

Image warping plays an essential role in our system — it defines a mapping of points between the image planes of two cameras — in our case, the reference and interactive images (Figure 3-6). *Forward warping* maps points on the reference image to the interactive image, and is used to display the representation; *inverse warping* maps points back from the interactive image to the reference image, and is used to update the edited parts of the representation.

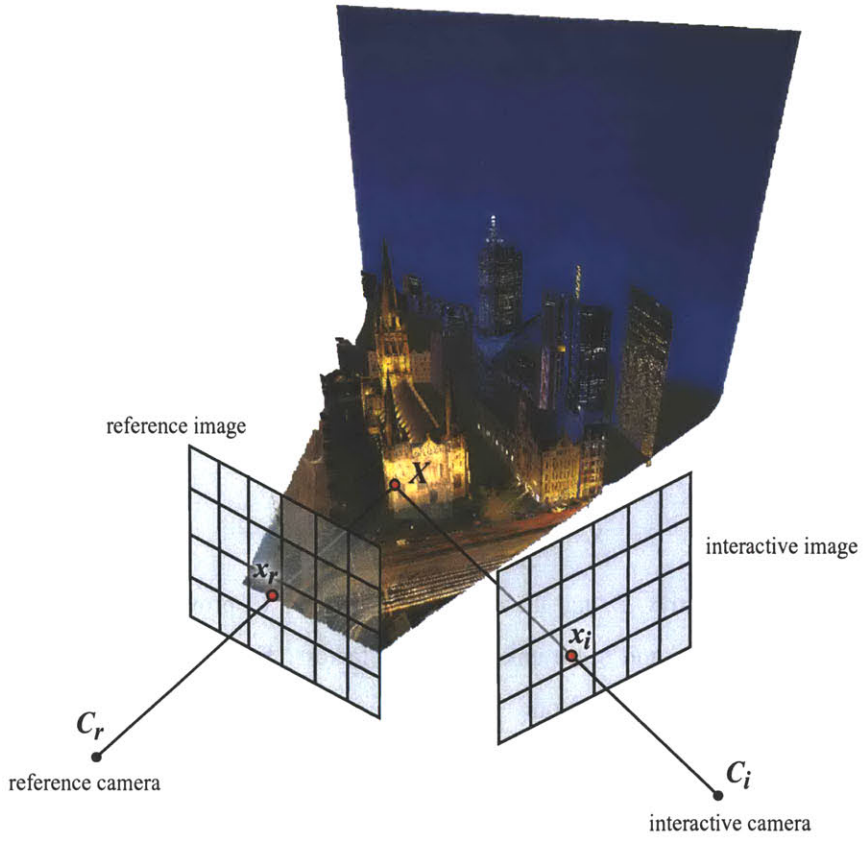


Figure 3-6: Image warping from the interactive camera to the reference camera.

In the remainder of this section, we first review the 3D image warping from McMillan's PhD thesis, in which he formulates the warping equation for the purpose of image-based rendering [McMillan 1997]. This provides the basis for the warping implementation of our system (Section 3.3.1). We then discuss forward and inverse image warping using the graphics hardware and the OpenGL API (Section 3.3.2). Finally, we discuss the interactive display component, where we exploit the graphics hardware to implement the forward warping process — mapping the reference image to the interactive image at interactive rates (Section 3.3.3).

3.3.1 McMillan's 3D Warping Equation

In 3D image warping, two assumptions are made. First, we assume that the two cameras — reference and interactive — are planar-pinhole models and that their parameters are known. Second, we assume that the projective depth is known for the reference camera. Projective depth information

may be gathered in various ways — for instance for real scenes, laser range scanning, stereo vision, and image-based modeling techniques may be used; for virtual scenes, one can use the graphics hardware and read the z-buffer, or use raytracing. Image warping is then utilized to render the new views of the interactive camera by mapping the color pixels on the reference image to their corresponding coordinates on the interactive image.

As shown in Figure 3-6, a 3D point X in world space is visible along rays from two center of projections, C_i and C_r; x_i and x_r are homogeneous 2D points on the reference and interactive image planes, respectively. To determine x_i, we first equate the two planar-pinhole camera models as follows:

$$X = C_r + t_r \mathbf{P}_r x_r = C_i + t_i \mathbf{P}_i x_i, \tag{3.1}$$

where the characteristic matrix, $\mathbf{P}$, and the center of projection, C, provide all necessary information about the camera, and t_i and t_r are unknown scale factors. From Equation 3.1, we obtain the following planar image-warping equation:

$$x_i = \delta(x_r)\mathbf{P}_i^{-1}(C_r - C_i) + \mathbf{P}_i^{-1}\mathbf{P}_r x_r, \tag{3.2}$$

which provides the warped coordinates on the interactive image, x_i. The generalized disparity is expressed by

$$\delta(x_r) = \frac{|\mathbf{P}_r x_r|}{r}, \tag{3.3}$$

where r is a magnitude of the distance between C_r and X. This means that generalized disparity is inversely related to projective depth by a scale factor.

Given the image-space coordinates of the reference camera, the coordinates of the corresponding pixels in the interactive camera are computed as follows:

$$u_i = \frac{\vec{a}_r \cdot (\vec{b}_i \times \vec{c}_i)u_r + \vec{b}_r \cdot (\vec{b}_i \times \vec{c}_i)v_r + \vec{c}_r \cdot (\vec{b}_i \times \vec{c}_i) + (\dot{C}_r - \dot{C}_i) \cdot (\vec{b}_i \times \vec{c}_i)\delta(u_r, v_r)}{\vec{a}_r \cdot (\vec{a}_i \times \vec{b}_i)u_r + \vec{b}_r \cdot (\vec{a}_i \times \vec{b}_i)v_r + \vec{c}_r \cdot (\vec{a}_i \times \vec{b}_i) + (\dot{C}_r - \dot{C}_i) \cdot (\vec{a}_i \times \vec{b}_i)\delta(u_r, v_r)} \tag{3.4}$$

$$v_i = \frac{\vec{a}_r \cdot (\vec{c}_i \times \vec{a}_i)u_r + \vec{b}_r \cdot (\vec{c}_i \times \vec{a}_i)v_r + \vec{c}_r \cdot (\vec{c}_i \times \vec{a}_i) + (\dot{C}_r - \dot{C}_i) \cdot (\vec{c}_i \times \vec{a}_i)\delta(u_r, v_r)}{\vec{a}_r \cdot (\vec{a}_i \times \vec{b}_i)u_r + \vec{b}_r \cdot (\vec{a}_i \times \vec{b}_i)v_r + \vec{c}_r \cdot (\vec{a}_i \times \vec{b}_i) + (\dot{C}_r - \dot{C}_i) \cdot (\vec{a}_i \times \vec{b}_i)\delta(u_r, v_r)}, \tag{3.5}$$

where $x = (u, v)$, vectors $\vec{a}$ and $\vec{b}$ are orthogonal and form a basis for the image plane, and $\vec{c}$ is a vector from C to the origin of the image plane. Equations (3.4) and (3.5) require nine adds,

eleven multiplies, and one inverse calculation per warp. This procedure can be optimized to reduce calculations and apply image warps at interactive rates without using the graphics hardware.

3.3.2 Image Warping with OpenGL

The implementation of image warping in our system is based on the OpenGL API, which is a common industry standard. In this section, we express image warping in terms of the OpenGL notions of camera model, z-buffer and transformations. We do so for explanatory as well as practical implementation purposes. Much of the following is a review of the OpenGL formulation of image warping from Zhang [1999].

A 3D point in world space is converted to a pixel in image space through a series of transformations via model-view, projection, and viewport matrices. Matrix $\mathbf{M}_p$ corresponds to the OpenGL projection matrix, and $\mathbf{M}_m$ corresponds to model-view matrix. We multiply the model-view and projection matrices,

$$\mathbf{M} = \mathbf{M}_p \mathbf{M}_m, \tag{3.6}$$

to compute matrix $\mathbf{M}$ that projects a 3D point in world space onto a 2D point on the image plane as follows:

$$\begin{bmatrix} x'w' \\ y'w' \\ z'w' \\ w' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{3.7}$$

where $[x'', y'', z'']^\top = [\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'}]^\top$. The resulting x'', y'', and z'' components range between $[-1, 1]$, i.e. are in normalized device coordinates. They are then mapped to the viewport according to the width and height of the display screen:

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 + (i + 0.5) * (2/W) \\ -1 + (j + 0.5) * (2/H) \\ -1 + z_{ij} * 2 \\ 1 \end{bmatrix}, \tag{3.8}$$

where W is the width, H is the height of the image plane, (i, j) is the pixel coordinate, and z_{ij} is the depth value of the z-buffer.

To map a pixel from image space to world space, in other words to "unproject" a pixel, we simply invert matrix $\mathbf{M}$:

$$\begin{bmatrix} xw \\ yw \\ zw \\ w \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix}, \tag{3.9}$$

and (x, y, z) are the 3D coordinates.

By combining Equations (3.7) and (3.9) with matrix $\mathbf{M}$ from two different cameras, $\mathbf{M}_i$ and $\mathbf{M}_r$, we can map a point from one image plane to another:

$$\mathbf{M}_{forward} = \mathbf{M}_i \mathbf{M}_r^{-1}, \tag{3.10}$$

where matrix $\mathbf{M}_r^{-1}$ projects a point on the reference image to world space, and $\mathbf{M}_i$ unprojects the point to the interactive image. The combined matrix $\mathbf{M}_{forward}$ is the 4×4 *forward warping* matrix that directly maps point x_r from the reference image to x_i on the interactive image:

$$x_i = \mathbf{M}_w x_r. \tag{3.11}$$

Points x_r and x_i are in normalized device coordinates. This means that given a reference pixel coordinate, (i, j), and its corresponding z-buffer value, z_{ij}, we can compute the corresponding coordinates and z-buffer value on the interactive image.

To compute the *inverse warp*, we simply swap the matrices as follows:

$$\mathbf{M}_{inverse} = \mathbf{M}_{forward}^{-1} = \mathbf{M}_r \mathbf{M}_i^{-1}. \tag{3.12}$$

As previously mentioned, and will be emphasized again in the following sections, inverse warping is an essential part of the tools, since the user applies a tool on the interactive image and the modified pixels are inverse warped to the reference image. The representation is then updated and visualized on the interactive display.

Splat Size

Although forward (or inverse) warping computes the corresponding coordinates of the interactive (or reference) image, it does not provide information about the size of the reprojected pixel. To

compute the approximate reprojected pixel size, i.e. the *splat size*, we use partial derivatives of the pixel on the interactive image. Let the elements of $\mathbf{M}_w$ be m_{ij}, where $0 \leq i, j \leq 3$:

$$\mathbf{M}_w = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix}. \tag{3.13}$$

Then the partial derivatives of the reference pixel to the interactive pixel in x are:

$$\begin{bmatrix} \partial x'/\partial x \\ \partial y'/\partial x \\ \partial z'/\partial x \\ \partial w'/\partial x \end{bmatrix} = \mathbf{M}_w \begin{bmatrix} 1 \\ 0 \\ \partial z/\partial x \\ 0 \end{bmatrix} = \begin{bmatrix} m_{00} + m_{02}\frac{\partial z}{\partial x} \\ m_{10} + m_{12}\frac{\partial z}{\partial x} \\ m_{20} + m_{22}\frac{\partial z}{\partial x} \\ m_{30} + m_{32}\frac{\partial z}{\partial x} \end{bmatrix}. \tag{3.14}$$

We then need to convert the partial derivatives to the normalized device coordinates, where $[x'', y'', z'']^\top = [\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'}]^\top$ as follows:

$$\frac{\partial x''}{\partial x} = (\frac{\partial x'}{\partial x}w' + \frac{\partial w'}{\partial x}x')/w'^2. \tag{3.15}$$

The partial derivatives of y are computed similarly.

3.3.3 Interactive Display

To interactively visualize the current state of the image-based representation, the interactive display utilizes the forward image-warping process — it maps the reference image to the interactive image in real time. In our system, we use triangles in order to exploit the graphics hardware. Alternatively, it would be possible to use other image-based rendering techniques, such as splatting [Chen and Williams 1993; McMillan and Bishop 1995; Shade et al. 1998]. As we will see in the next section (Section 3.4), the inverse image-warping is used to map the modified pixels from the interactive image to the reference image, and splatting is used to determine size of the mapped pixel on the reference image.

The layers of the image-based representation are displayed using triangles [McMillan 1997; Mark et al. 1997]. Equation 3.9 is used to unproject the pixel from the reference image to the world space, and neighboring pixels are then connected into a triangle mesh. The segmentation of the scene into layers eliminates much of the rubber-sheeting triangle artifacts, which are mainly caused

from using a single mesh for a scene with multiple objects. The level-of-detail of the triangle mesh may be varied from layer to layer according to the user, since not all need to be displayed at its finest detail. Hardware projective texture mapping is then used to display the color channel on the triangle mesh [Segal et al. 1992]. The reprojection step, i.e. projecting the 3D triangles back to the 2D interactive view (Equation 3.7), is computed and displayed by the OpenGL API.

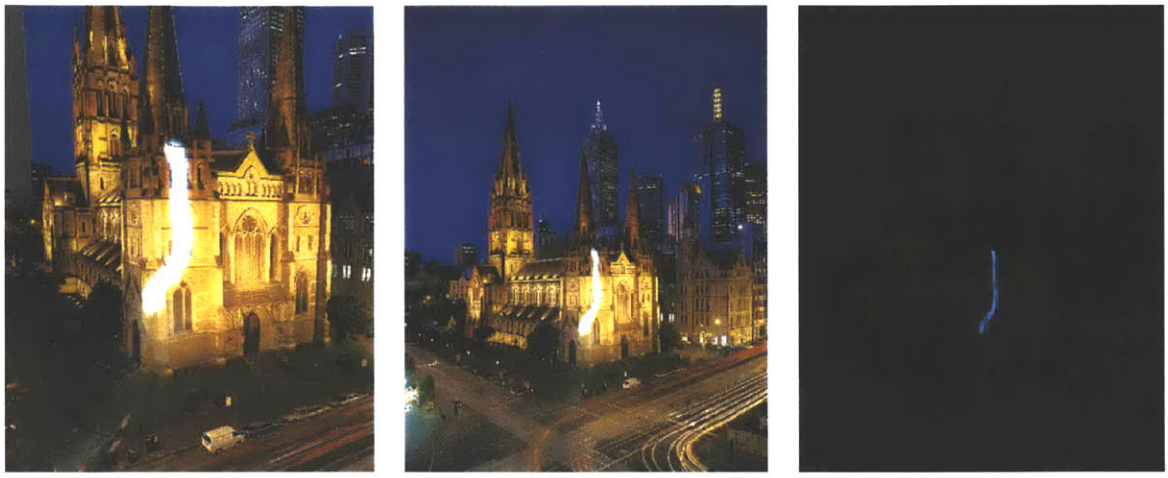
The degree to which the viewpoint can be altered, without artifacts, is dependent on the particular scene, assigned depth, and occluded regions. In our system, the user can toggle the layers as visible or invisible, and the layers may be displayed according to the order list, similar to photo-editing systems, i.e. a layer is in front if it is first in the order list.

3.4 Basic Tools

Tools are operators that enable the user to interactively edit the image-based representation from any viewpoint. The user controls the interactive camera to a desired viewpoint, and applies the tools on the interactive image. The z-buffer of the interactive image is read, and the modifications are then updated on the reference image of the image-based representation via inverse warping (Section 3.3.2).

As shown in Figure 3-1(b), we provide several classes of tools. In this section, we discuss the *basic tools*, such as paint brush, color picker, and selection tools. Because the editing interactions are applied on a 2D image (i.e. the interactive image), existing photo-editing tools are naturally extended and incorporated into our system. The *depth assignment tools*, which we discuss in detail in Section 3.5, are a new class of tools specifically for editing the depth channel. In Chapters 4 and 5, we describe the new clone brush tools, which extend and adapt the classical clone brush. Finally, in Chapter 6, we describe a texture-illuminance filter that allows the user to seamlessly edit the lighting or the materials in the scene.

We developed these tools in the spirit of those available in the photo-editing systems — emphasizing intuitive 2D interactions. Leveraging direct access to the simple data structure, the system provides the user with intuitive and predictable control. Moreover, the system architecture, with its simple data structure and clearly separated components, is easy to implement and augment with new tools and filters.



(a) (b) (c)

Figure 3-7: Paint brush example. (a) Paint brush applied to the interactive image. (b) Edited pixels inverse warped and updated on the reference image. (c) The color selected from the palette was much brighter than the image, because of our floating-point data structure. Although the entire image was darkened, the paint brush stroke remains visible.

3.4.1 Paint Brush and Color Picker

The *paint brush* tool enables the user to paint selected values on the interactive image with a brush interface. The brush shape, size, and color are user-selected from a given palette, and these attributes can be applied on multiple channels simultaneously from any viewpoint. For instance, the user may apply the paint brush tool to edit the color and depth channels at the same time. We also provide a chisel tool, which is equivalent to the paint brush tool in all respects except that it adds or subtracts from the existing values instead of painting over with a selected value from the palette. We discuss the chisel tool, in the context of editing the depth channel, in detail in Section 3.5.1. As illustrated in Figure 3-7, the user paint brushes on the interactive image, and the edited pixels are inverse warped and updated on the reference image.

Since we store a 2D array of floating-point color channel values instead of 8 bits per channel, we essentially have a high-dynamic-range photo editor. As shown in Figure 3-7(c), even when we darken the image, the paint brushed region remains visible since a much brighter color was selected from the palette. We provide simple linear and gamma functions to display the high dynamic-range images, and it is easy to plug in a more sophisticated tone-mapping operator into the system, such as [Tumblin and Turk 1999; Durand and Dorsey 2002].

A complementary tool to the paint brush is the *color picker*. It enables the user to sample the

values of a "picked" point on the interactive image for any channel, including depth. From any given viewpoint, the user picks a desired point on the interactive image to record its values. The interactive image, in this case, essentially functions as the palette. The picked point is inverse warped to the reference image, and the values of the warped point from the reference channels are returned for reuse. The picked values may be repainted into the scene, including the depth value to modify the geometry.

3.4.2 Selection

We provide a set of *selection tools*, such as rectangle, lasso, and intelligent scissoring [Mortensen and Barrett 1995], that modify the selection channel. These tools are a simple, yet essential part of photo editing, since they provide a means to flag which pixels may or may not be edited by other tools. As shown in Figure 3-4, the selection values range from 0 to 255 to permit feathering.[1] The selection tools are also applied on the interactive image, and inverse warped to the corresponding selection channel of the reference image.

Because our system has import/export capabilities from Adobe Photoshop file format, we can take advantage of the latest semi-automatic segmentation tools that they offer. For most of the examples presented in this thesis, we used Photoshop for the initial segmentation of the input image.

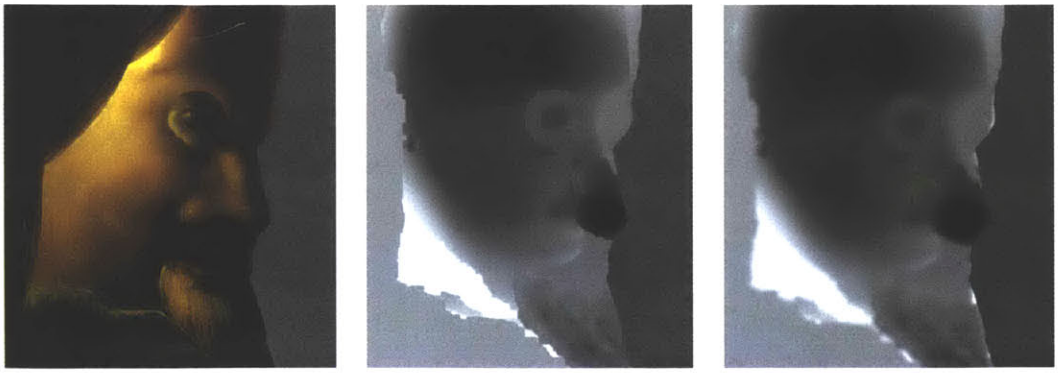
3.4.3 Filters

We also provide a set of filters, such as the Gaussian blur function, the median filter, the anisotropic diffusion filter [Perona and Malik 1990], and the bilateral filter [Tomasi and Manduchi 1998]. These 2D image-processing filters may be applied on any selected channels, including depth, since our data structure stores a set of images. For instance, applying the anisotropic filter on the depth channel is similar to work by Desbrun et al. [Desbrun et al. 2000], since it is analogous to denoising bivariate data or smoothing projective height fields.

3.5 Depth Assignment

We have developed a new class of depth assignment tools to take advantage of the versatility of our representation. As previously mentioned, much of the work here was done by Max Chen for his

[1] The description of feathering from the Adobe Photoshop manual is, "Blurs edges by building a transition boundary between the selection and its surrounding pixels. This blurring can cause some loss of detail at the edge of the selection."



(a) (b) (c)

Figure 3-8: A face painting example. (a) Color channel. (b) Chiseled depth channel. (c) Blurred depth channel.

Master's Thesis [2001].

There are two types of depth assignment tools: the depth painting tools and the hybrid tools. The underlying metaphor of the *depth painting* tools is to paint and draw depth like colors are painted, in the spirit of Kang [Kang 1998]. This provides complete user control, and it also relies on the user's ability to comprehend the layout of the scene. Although the user can easily infer the spatial organization and shapes depicted in the image, it is not always easy to directly paint or chisel the corresponding depth. Hence we have developed the *hybrid tools* that use pre-defined shapes to aid in painting accurate depth, such as the ground plane tool (Section 3.5.2), geometric primitives tool (Section 3.5.3), and the template tool (Section 3.5.5). In the development of our interface, we have emphasized 2D, rather than 3D interaction, the direct use of cues present in the image, and the use of previously-assigned depth as a reference.

Depth can be edited from any interactive viewpoint, which is important in evaluating the effects of current manipulations. Multiple views can also be used [Kang 1998]. We will see that some tools are easier to use in the reference view, where image cues are more clearly visible, while for others, interactive views permit a better judgment of the shape being modeled. The level of detail and accuracy of depth, which can be refined at any time, depend on the application and intended viewpoint variation.

The use of selection also permits us to restrict the effect of a tool to a specific part of the image, providing flexibility and finer control. Since our selections are real-valued, the effect of depth tools can be attenuated at the selection boundary to obtain smoother shapes. In our implementation, we

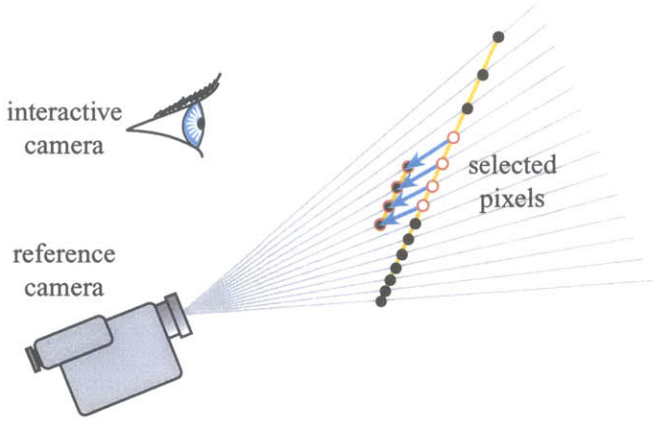


Figure 3-9: Depth translation is performed along lines of sight with respect to the reference camera.

use the selection value to interpolate linearly between the unedited and edited values. Smoother functions, such as a cosine, could also be used.

In contrast to optimization-based photogrammetry systems [Canoma; Debevec et al. 1996; Faugeras et al. 1995], the field of view of the reference camera must be specified as a first step (we assume a pinhole camera). If enough information is available in the image, the field of view can be calculated. The user can also set the focal length manually. Otherwise, the focal length is set to a default value (50mm or approximately 40 degrees for the field of view in practice).

3.5.1 Depth Painting

The user can directly paint depth using a brush interface to set the absolute depth value, or chiselling may be applied to add or subtract from the current depth value (Section 3.4.1). Absolute depth can be specified using the color picker on the depth channel, by clicking on a point of the image to read its depth. The relative brush tool is particularly useful for refining already-assigned depth (Fig. 3-8(b)). The size and shape of the brush can be interactively varied.

The whole selected region can also be interactively translated. Translation is performed along lines of sight with respect to the reference camera: The depth of each selected pixel is incremented or decremented (Fig. 3-9). However, it is desirable that planar objects remain planar under this transformation. We do not add or subtract a constant value, but instead multiply depth by a constant value. Depth-translating planar objects, therefore, results in parallel planar objects.

In the spirit of classical interactive photo-editing tools, we have developed local blurring and sharpening tools that filter the depth channel under the pointer (Fig. 3-8(c)). Blurring smooths

the shape, while sharpening accentuates relief. Local blurring can be used to "zip" along depth discontinuities, as described by Kang [Kang 1998]. A global filtering is also possible, in particular blurring to smooth the 3D shape, noise to add complexity, and median filtering to remove outliers.

Similar to Kang [Kang 1998] and Williams' [Williams 1998] methods, the user can use the color channel to assign depth. This is motivated by cases where darker values correspond to more distant pixels (such as trees) or by atmospheric perspective making distant objects bluer. The user specifies the z_{min} and z_{max} depth values, and the vector specifying color direction $\vec{C}$ (e.g. dark to light or amount of blue), and the effect can be applied absolutely or relatively. In the absolute case, for example, depth is then specified via the color at each pixel $\vec{c}(x,y)$ from:

$$z(x,y) = z_{min} + (z_{max} - z_{min}) * \vec{C} \cdot \vec{c}(x,y).$$

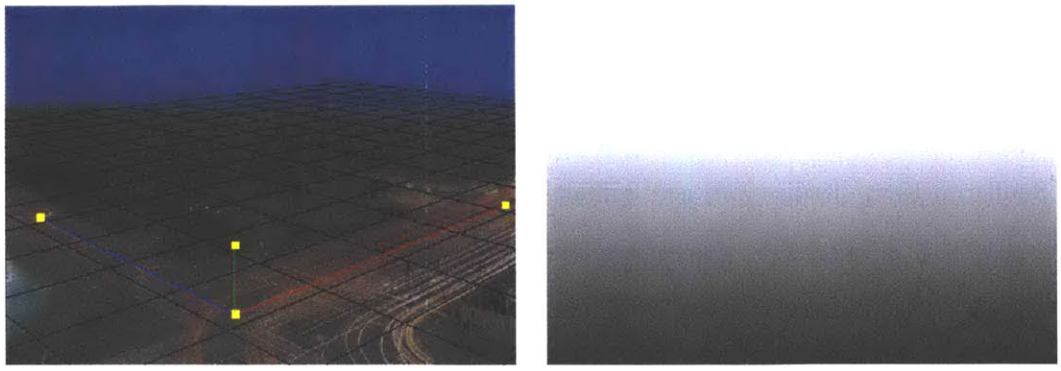
3.5.2 Ground Plane and Reference Depth

The tools presented so far work best if some initial depth has been assigned, or if a reference is provided for depth assignment. Similar to the "spidery mesh" interface by Horry et al. [Horry et al. 1997], we have found that the use of a reference ground plane greatly simplifies depth acquisition and improves accuracy dramatically, since it provides an intuitive reference. The position with respect to the ground plane has actually been shown to be a very effective depth cue [Palmer 1999]. Specifying a ground plane is typically the first step of depth assignment.

The ground plane tool can be seen as the application of a gradient on the depth channel (Fig. 3-10). However, an arbitrary gradient may not correspond to a planar surface. In our system, the user specifies the horizon line in the reference image, which constrains two degrees of freedom, corresponding to a set of parallel planes. The remaining degree of freedom corresponds to an arbitrary scale factor on depth. We can thus arbitrarily set the height of the observer to 1, or the user can enter a value.

We have also implemented the method by Liebowitz et al. [Liebowitz et al. 1999] that allows the acquisition of architectural models and camera parameters from one image using parallelism and orthogonality constraints. This provides both the camera parameters and an accurate reference ground plane. This also allows us to compute the position of the optical axis if it is not in the center of the image. This idea is also used for structure-preserving clone brushing, as discussed in Chapter 4.

Depth picking and depth painting can then easily be used to depth-paint billboards parallel to the



(a) (b)

Figure 3-10: (a) Ground layer and the ground-plane tool. (b) Depth map.

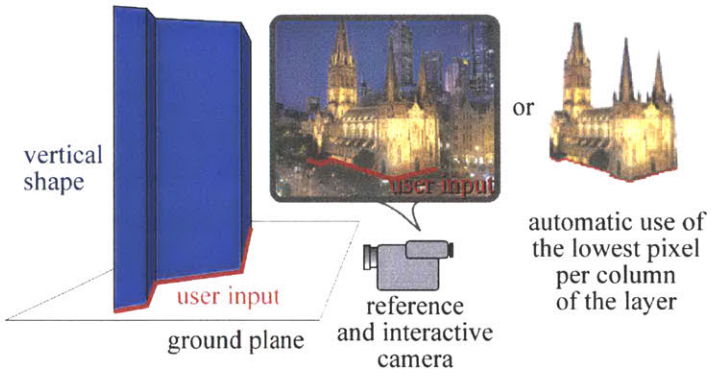


Figure 3-11: Vertical tool. The user draws the contact of the vertical geometry with the ground plane.

image plane. Since most objects in a scene touch the ground, or their projection on the ground can be inferred by the user, this proves to be very efficient. This is similar to the placement of billboards on the "spidery mesh" of Horry et al. [Horry et al. 1997]. However, their system is limited to central perspective and polygonal orthogonal objects, while we can refine our representation and obtain arbitrary shapes.

We have extended the notion of billboards to allow the user to paint the depth of arbitrary vertical objects. This works best when the interactive camera is set to the reference camera, since the contact points between the ground plane and the object are correctly shown from the reference view. The contact or projection of the object on the ground plane is drawn, and a vertical depth is extruded (Fig. 3-11). In practice, the contact drawn by the user is represented as a polyline, the corresponding vertical polygons are rendered using OpenGL, and the z-buffer is read to update the selected pixels.

We are not limited to the use of a planar ground reference. The 3D locations of the points of the contact polyline drawn by the user are read from the interactive z-buffer. This means that if the ground has been chiseled or translated for better terrain modeling, vertical objects will be extruded accordingly.

We have also implemented an automatic version that processes the whole selection or layer at once. It assumes that the layer or selection is in contact with the ground reference at its lowest pixels. Each column of pixels in the reference image is assigned depth corresponding to its lowest visible or selected pixel.

3.5.3 Geometric Primitives

Some geometric shapes, such as boxes, spheres, or cylinders, are hard to depth-paint accurately. We therefore provide geometric primitives that can be drawn transparently as 2D objects. For example, the user draws a circle or clicks on three points to assign spherical depth. We use similar interfaces for cylinders (the user draws the edges), boxes (the user draws three edges of a corner), and pyramids (the user draws the base and apex). These tools work best when used from the reference camera.

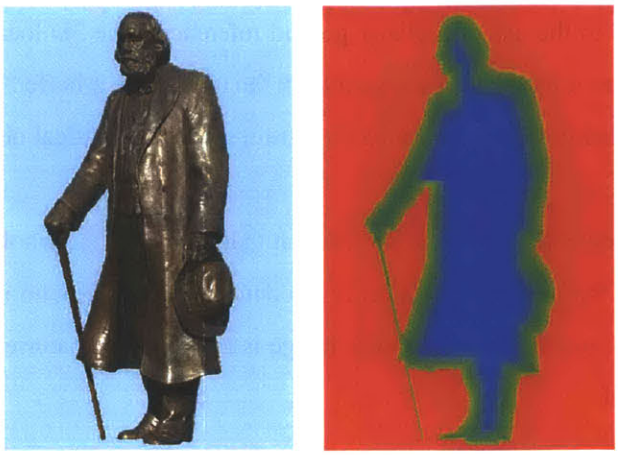
The primitive is rendered from the reference camera using OpenGL, and the z-buffer is read to assign depth. This requires the use of a depth buffer at the resolution of the reference image. Since our system treats images that can be larger than the size of the screen, we use tiling — we divide the reference image into smaller sub-images and treat each accordingly.

Once the image projection of the primitive has been provided, its distance must be specified. The user can use an arbitrary distance as a starting point and then refine it with the translation tool. He can also use the already-assigned depth as a reference by clicking on one point. By default, the first point clicked by the user is used as a reference depth (e.g. corner of a box).

To improve the accuracy of the depth when a ground plane has been assigned, the user can use *primitive snapping* to enforce the verticality of boxes, cylinders or pyramids, or to constrain them along the normal of a given pixel. A least-squares error minimization is then used to optimize the 3D shape position.

3.5.4 Organic Shapes

In the spirit of Williams [1998] and the Teddy fast-modeling system [Igarashi et al. 1999], we propose a tool that assigns depth using level sets [Sethian 1999]. By specifying more distant depth to the camera at the object silhouette and closer depth in the center, this technique is well suited to



(a) (b)

Figure 3-12: Organic Shapes. (a) Statue. (b) False Color Depth Map.

giving organic shapes a bulgy appearance (Fig. 3-12). This approach is relative, and the range r of depth addition is specified by the user.

The initial active interior of the object is defined as the set of pixels of the layer or selection with non-null alpha. The distance to the boundary d_{bound} is initialized to 0, and we iteratively "erode" (e.g. [Sillion and Drettakis 1995]). For each iteration, we discard pixels that have a non-active neighbor, and increment the distance of active pixels by 1.

We use the normalized distance to the centroid:

$$d' = 1 - d_{bound}/d_{bound}max$$

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.

3.5.5 Faces and Geometric Templates

The specific case of human faces is important. The output of the morphable model by Blanz et al. [Blanz and Vetter 1999] could be used to retrieve accurate depth. However, this technique is not easy to implement since it requires a large database of 3D face scans.

We developed a simpler method that trades accuracy and user intervention for speed and simplicity. We use a generic arbitrary 3D model, optimize its 3D position to match the photograph, and

(a) (b) (c)

Figure 3-13: Face Tool. (a) User specified correspondence points. (b) Transformation after optimization. (c) Face after depth is applied.

then use 2D morphing to refine the match (Figure 3-13). This method could be further generalized to a broader class of template shapes.

The user specifies correspondence points between the image and the 3D model. These points are used to find the rotation, scale and position of the 3D model using Levenberg-Marquardt optimization [Press et al. 1992]. Rotation and scale are optimized independently to avoid shear in the resulting transformation. The 3D face is rendered and the z-buffer is read back.

We then use the same correspondence points to morph the z-buffer and obtain a better match with the image. We use triangle-based morphing and linear interpolation [Gomes et al. 1998].

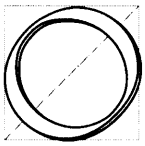
3.6 Discussion

In this chapter, we have described the architecture of our system. There are three main components: the image-based representation, the interactive display, and the tools. The image-based representation is based on layers of images with depth, and is the main data structure of the system. The depth information enables the user to navigate and edit the 2D input image from any viewing position, and the layering representation supports a wide class of editing tools that are intuitive, familiar, and powerful. The interactive display enables the user to visualize the current state of the image-based representation interactively via the forward warp, where we exploit the graphics hardware and the OpenGL API. Finally, a suite of tools enable the user to edit the representation from any viewing position. Inverse warping is utilized to map from the interactive image, where the editing takes place, to the reference image, where the modifications are updated in the data structure. Since the system architecture is simple and has distinct components, it is easy to implement and plug in new

tools, either for modeling or photo editing.

CHAPTER 4

A Structure-Preserving Clone Brush



ne of the most powerful and widely-used tools in photo editing is the *clone brush*, also known as the "clone stamp." The clone brush permits interactively copying and pasting from one region of an image to another via a brush interface. It is often used to remove undesirable portions of an image, such as blemishes or distracting objects in the background, or to replicate parts of a photograph. The brush interface facilitates fine user control, and interactively copying and pasting existing pixels enables the user to maintain photorealism even with drastic modifications. An example is shown in Figure 4-1, where the pole and wires have been clone brushed from the initial image.

Despite its utility and versatility, the clone brush suffers from several important limitations. First, only regions with similar orientation and distance with respect to the camera can be brushed effectively. Perspective foreshortening present in most photographs, and various shapes of the objects in the scene make it difficult to clone brush effectively, as illustrated in Figures 4-2(a) and (b). Although the traditional clone brush works well for regions of the image that do not have a strong geometric structure, such as clouds or vegetation, many structured features are not amenable for the traditional clone brush due to perspective foreshortening, such as buildings. We show an example in

(a) (b)

Figure 4-1: Traditional clone brush example. (a) Initial image. (b) The pole and wires have been clone brushed out of the image, by copying and pasting parts of the existing image to another.

Figure 4-1, where a traditional clone brush was used to remove the pole and wires. Clone brushing the sky and clouds was much easier than removing the pole overlapping the building, mainly due to the perspective foreshortening on the building textures. Second, intensity variations due to existing lighting in the photograph further limit the effectiveness of the clone brush. Artifacts appear when the intensity of the source and destination regions do not match. Only regions with similar intensities can be clone brushed convincingly using the traditional clone brush (Figures 4-2(c) and (d)).

In this and the next chapter, we introduce two new clone brush tools that address these problems. In this chapter, present a structure-preserving clone brush in a purely 2D photo editing context, where no depth information is necessary. This tool can easily be integrated into existing 2D photo-editing systems, such as Adobe Photoshop, to improve the traditional clone brush. In Chapter 5, we then present a 3D non-distorted clone brush tool that can be employed when depth information is available.

We introduce three enhancements to improve the 2D version of the traditional clone brush. First, we present a technique that allows the user to correctly copy and paste pixels according to the perspective foreshortening present on planar surfaces of the input image. The user provides perspective information by tracing parallel lines in the image. Second, we provide a simple color-correction technique that allows the user to clone brush seamlessly between parts of the image that have different lighting conditions, by using a multiplicative factor to compensate for the intensity variation. Finally, we provide a "snapping" feature that allows the user to initialize the source and destination positions more precisely. A precise initialization of source and destination points is often

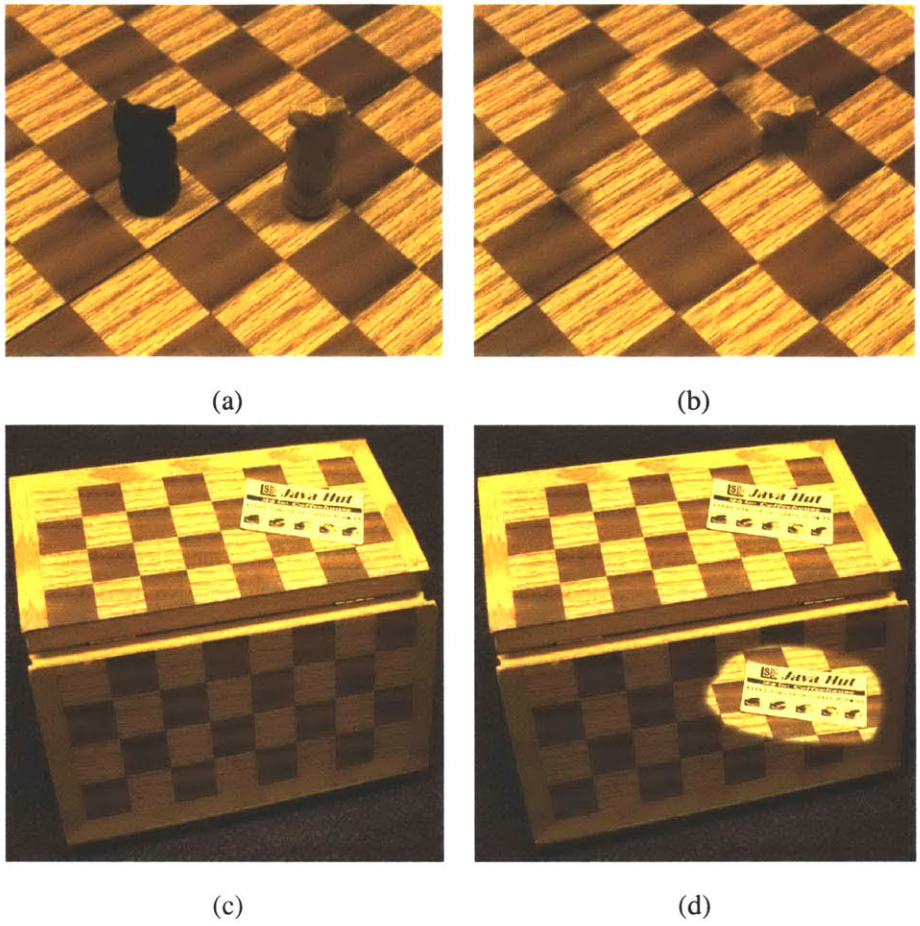
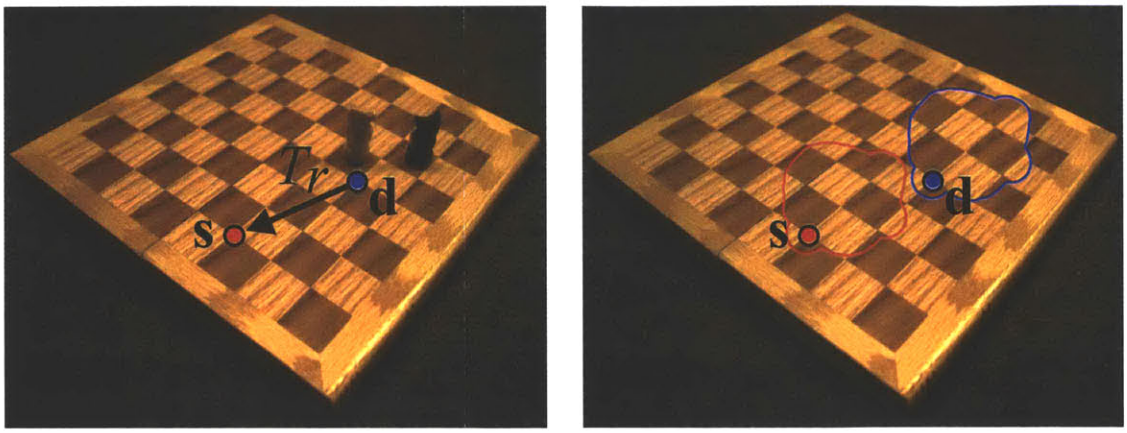


Figure 4-2: Traditional clone brush limitations. (a) Initial input image. (b) Patterns on the chessboard are not aligned due to perspective foreshortening. (c) Initial input image. (d) Although the two halves of the chessboard consist of the same material, the clone brushed region is highly noticeable due to lighting intensity variations between the source and destination regions.

necessary, especially for images that have structured features, such as edges. Misalignment between source and destination features leads to noticeable artifacts at the limit of the clone-brushed region. It is common for the user to undo and repeat the initialization step until the points are sufficiently accurate. The new "snapping" feature optimizes the initial destination point to match features of the



(a) (b)

Figure 4-3: Traditional clone brush. (a) The user first clicks on **s** to specify the initial source position, and then clone brushes starting at the initial destination point **d**. This assigns the relative translation matrix T_r on the image plane. (b) Pixels from the source region (in red) have been copy-pasted to the destination region (in blue) via a brush interface.

initial source point, thus providing precise initial points.

4.1 Background

4.1.1 Traditional Clone Brushing

To use the clone brush, the user first selects a *source region* of the image and then paints over a *destination region* using a brush interface that copy-pastes pixels from the source to the destination.

An important concept of the traditional clone brush is the *relative translation*. Before clone brushing starts, the user selects the initial source and destination points, **s** and **d** respectively, to define the relative translation, T_r, computed in the image plane (Figure 4-3(a)). As the clone brushing progresses, T_r determines the position of the source region with respect to the currently-being-brushed destination region. The equation to determine the location of the source pixel, $\mathbf{x}_s$, relative to the current destination pixel, $\mathbf{x}_d$, is

$$\mathbf{x}_s = T_r \mathbf{x}_d. \tag{4.1}$$

The color of pixel $\mathbf{x}_s$ is then copied to the pixel $\mathbf{x}_d$.

A more general formulation of Equation (4.1) is

$$\mathbf{x}_s = M_r \mathbf{x}_d. \tag{4.2}$$

The matrix M_r is a similarity transformation matrix, where $M_r = T_r R_r S_r$. T_r is the relative translation, R_r is the relative rotation, and S_r is the relative uniform scaling that would allow the user to clone brush rotated and scaled pixels from the source to the destination region in the image plane. This feature is not available in traditional photo-editing software; in Section 4.2, we develop this idea further for the perspective-correction feature.

4.1.2 Related work

We build upon the work on homography and image rectification. In particular, Liebowitz et al. present algorithms for plane rectification, plane orientation, and camera calibration for creating architectural models from images [Liebowitz et al. 1999].

Alternatives to the clone brush include image inpainting, which seamlessly fills in specified regions of the image [Bertalmio et al. 2000]. Unlike clone brushing, this approach only works for non-textured areas. Texture synthesis can also be used to fill-in holes or to mask objects in an image, e.g. [Igehy and Pereira 1997]. These techniques are automatic, while clone brushing is interactive — providing greater control and flexibility.

Gleicher introduced image snapping, which can be used to improve the selection of source and destination points [Gleicher 1995]. However, the clone brushing context is slightly different from traditional snapping, as the goal is not to click precisely on a feature, but rather to match features in the source and destination regions.

4.2 Perspective Correction

As noted previously, one of the main problems in clone brushing stems from perspective foreshortening. To remedy this, we extend the traditional clone brush to correct the perspective distortions on planar and quasi-planar surfaces. As we have seen, the traditional clone brush is based on relative translation in image space. In the case of foreshortened geometry, we would like the translation to take place in the world plane (Figure 4-4). The general idea is to first define the world plane using a *homography* — a linear 2D-to-2D mapping between the world and image planes. Once the homography is determined, points on the image plane can be mapped to the world plane, and vice

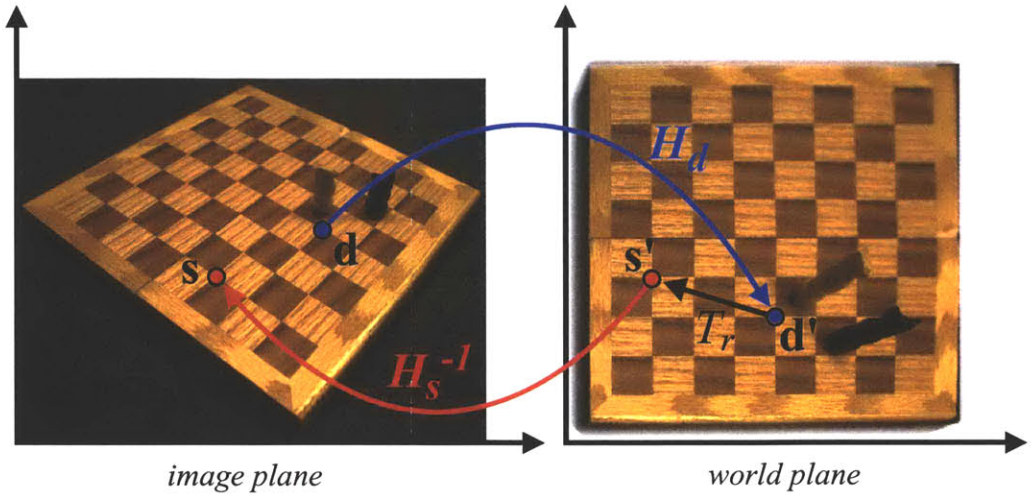


Figure 4-4: Perspective correction. As illustrated, the general idea is to apply the relative translation, T_r, in the world plane.

versa. It is important to note that our goal is not to rectify images — we only compute the *transformations* that determine the perspective-correcting coordinates of source pixels given destination pixel locations. The pixel copy of source color to the destination position then occurs in the image plane.

We discuss two types of clone-brushing scenarios: 1-plane and 2-plane. The 1-plane scenario is when the source and destination regions reside on the same world plane. For instance, to remove the chess pieces from the chessboard in Figure 4-6, we only need to specify a single homography. The 2-plane scenario is when the source and destination regions reside in different world planes, e.g. clone brushing from one side of a building to another. In this case, homographies for both source and destination planes need to be specified, and additional parameters are necessary to determine the relationship between the two planes.

4.2.1 Homography

As illustrated in Figure 4-5, points on the world plane are projections of points on the image plane, and this linear mapping between the two planes is defined through a *homography*, i.e. a 2D projective transformation [Semple and Kneebone 1952; Liebowitz et al. 1999]. More specifically, the homography is represented by a 3×3 homogeneous matrix H. Points on the image plane $\mathbf{x}$ are mapped to the points on the world plane $\mathbf{x}'$ as $\mathbf{x}' = H\mathbf{x}$ up to a scale factor, where $\mathbf{x}$ is a homogenous

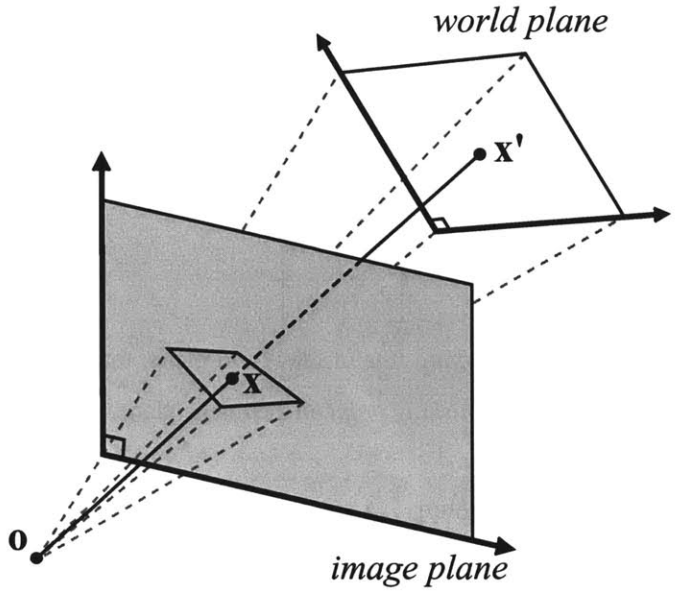


Figure 4-5: A homography determines a linear 2D-to-2D mapping between the image and world planes, where the world plane is projected onto the image plane with respect to perspective camera **o**.

column 3-vector $\mathbf{x} = (x, y, 1)^\top$. Similarly, the inverse homography, H^{-1}, maps points from the world plane back to the image plane, $\mathbf{x} = H^{-1}\mathbf{x}'$.

The homography matrix H can be decomposed into the following transformation matrices:

$$H = MN. \tag{4.3}$$

The matrix M is a similarity transformation, which is also known as the *metric* component of the homography,

$$M = T(t_x, t_y)R(\theta)S(s), \tag{4.4}$$

which has four degrees of freedom: translation matrix T with translation vector $\mathbf{t} = (t_x, t_y, 1)^\top$, rotation matrix R with angle θ, and scale matrix S with uniform scale factor s.

The second *non-metric* component, N, determines the rectified world-plane coordinates. The matrix N can be further decomposed into $N = AP$, where A is an affine transformation,

$$A = \begin{bmatrix} a & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.5}$$

which has two degrees of freedom: a determines the width-to-height aspect ratio, and h determines the shear transformation along the x-axis.

Matrix P represents a "pure projective" transformation:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & 1 \end{bmatrix}, \tag{4.6}$$

where vector $\mathbf{l} = (l_1, l_2, 1)^\top$ is the *vanishing line* of the world plane, that is, the horizon. Parallel lines on the world plane intersect at a *vanishing point* on the image plane, and these vanishing points lie in $\mathbf{l}$ (Figure 4-7).

As described in [Liebowitz et al. 1999], only N is necessary to metric rectify an image, i.e. correctly represent angles and length ratios up to a scale factor in the world plane. Matrix M does not play a role in image rectification, but is applied to rotate, uniform scale, and translate the image in its world plane.

4.2.2 Clone Brushing Using Homography

The perspective-correcting clone brush is similar to the traditional version with one exception: the relative transformation, M_r, takes place in the world-plane coordinates. Similar to Equation (4.2), we use the following formulation to compute $\mathbf{x}_s$ on the image plane:

$$\mathbf{x}_s = H_s^{-1} M_r H_d \mathbf{x}_d, \tag{4.7}$$

where H_s and H_d are the source and destination homographies, respectively. More intuitively, H_d maps $\mathbf{x}_d$ to the destination world plane, M_r applies the relative transformation to the corresponding position on the source world plane, then H_s^{-1} inverse maps the point back to the source region in the image plane. The key concept is that the relative transformation takes place in world-plane coordinates. As the clone brushing progresses, the color of pixel $\mathbf{x}_d$ is replaced by the bilinearly interpolated color of the source pixel $\mathbf{x}_s$ on the image plane, which corrects the perspective foreshortening problem.

In using the formulation in Equation (4.7), the metric components of both homographies can be ignored, i.e. $H = N$. The non-metric part alone sufficiently defines the world plane for the perspective correction. Furthermore, the transformation matrix M_r applies the relative translation, rotation, and scale necessary to relate the two planes.

4.2.3 1-Plane Scenario

For the 1-plane scenario, the source and destination homographies are the same, i.e. $H_s = H_d$, so the user only needs to specify one homography. The user simply draws two orthogonal pairs of parallel lines in the input image to specify a homography (Figure 4-6). In most architectural scenes, these visual cues are commonplace. Alternatively, other techniques could be used to determine the homography, such as [Liebowitz and Zisserman 1998; Liebowitz et al. 1999].

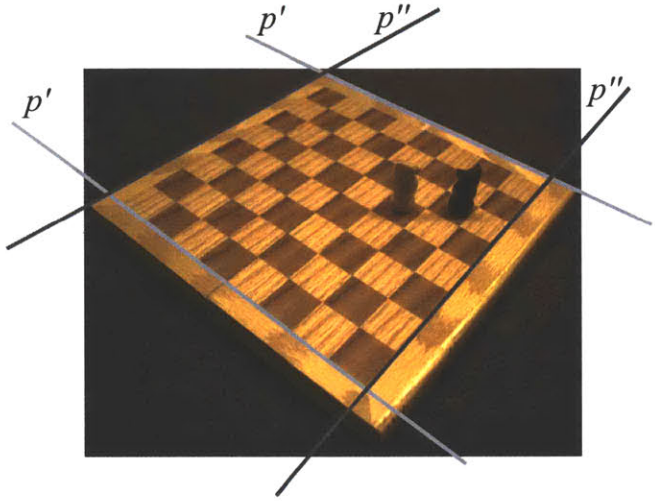


Figure 4-6: The user interactively traces two orthogonal sets of parallel lines, p' and p''.

The two sets of parallel lines drawn by the user specify vanishing points $\mathbf{u}$ and $\mathbf{v}$. As shown in Figure 4-7, the vanishing line $\mathbf{l}$ passes through $\mathbf{u}$ and $\mathbf{v}$, where $\mathbf{l} = (l_1, l_2, 1)^\top = \mathbf{u} \times \mathbf{v}$. Once $\mathbf{l}$ is identified, the matrix P is determined (Equation (4.6)).

The parameters a and h of matrix A (Equation (4.5)) are determined by solving the following quadratic equation for the complex number I:

$$\begin{aligned} &(1 + 2c_x l_1 + l_1^2(c_x^2 + c_y^2 + f^2))I^2 + \\ &2(l_2 c_x + l_1 c_y + l_1 l_2(c_x^2 + c_y^2 + f^2))I + \\ &2l_2 c_y + l_2^2(c_x^2 + c_y^2 + f^2) + 1 \quad = 0, \end{aligned} \tag{4.8}$$

where $I = \alpha - i\beta$, $a = 1/\beta$, and $s = -\alpha/\beta$ [Liebowitz et al. 1999]. We assume that the principal point, $\mathbf{c} = (c_x, c_y)$, is in the center of the image. The only unknown parameter is then the focal length f. The length of the line segment $f = \overline{\mathbf{oc}}$ is determined as follows (Figure 4-8):

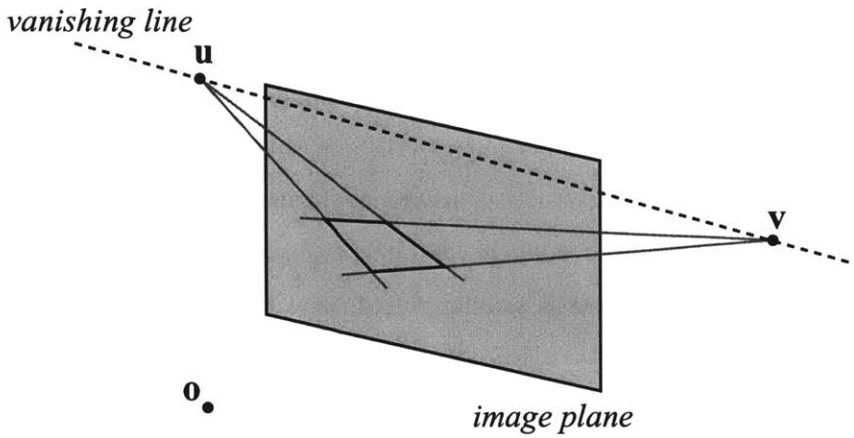


Figure 4-7: Parallel lines determine a vanishing point, and vanishing points determine a vanishing line on the image plane.

$$f = \overline{\mathbf{oc}} = \sqrt{\overline{\mathbf{op}}^2 - \overline{\mathbf{cp}}^2}. \tag{4.9}$$

Point **p** is the orthogonal projection of **c** onto the vanishing line **l** in image-plane coordinates. The length of line segment $\overline{\mathbf{op}}$ is

$$\overline{\mathbf{op}} = \sqrt{\overline{\mathbf{up}} \cdot \overline{\mathbf{pv}}}, \tag{4.10}$$

as shown in [Guillou et al. 2000].

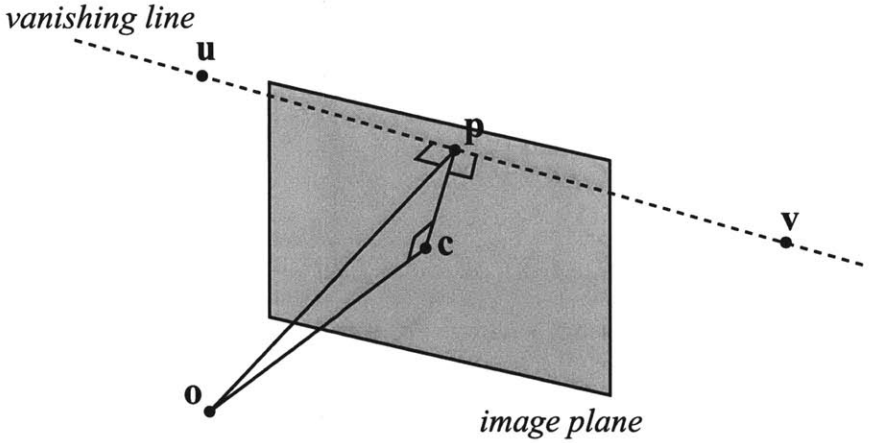


Figure 4-8: Focal length $f = \overline{\mathbf{oc}}$.

As discussed above, specifying two orthogonal sets of lines defines the non-metric part of the homography, $N = AP$. Therefore, the homographies $H_s = H_d = N$. The user then defines the relative translation, T_r, by specifying the initial source and destination points, as is the case with the traditional clone brush. The matrix T_r is now computed in world-plane coordinates, i.e. from $\mathbf{t}_r = \mathbf{s}' - \mathbf{d}'$, where $\mathbf{s}' = H_s\mathbf{s}$ and $\mathbf{d}' = H_d\mathbf{d}$ (Figure 4-4). Matrices R_r and S_r are irrelevant in this case, since $H_s = H_d$, meaning their values do not affect the clone brushing. As a result, $M_r = T_r$, and Equation (4.7) is complete for the 1-plane scenario.

4.2.4 2-Plane Scenario

Similar to the 1-plane scenario, the user first draws a pair of orthogonal parallel lines for both source and destination planes. These determine the non-metric components and, therefore, the respective homographies, $H_s = N_s$ and $H_d = N_d$.

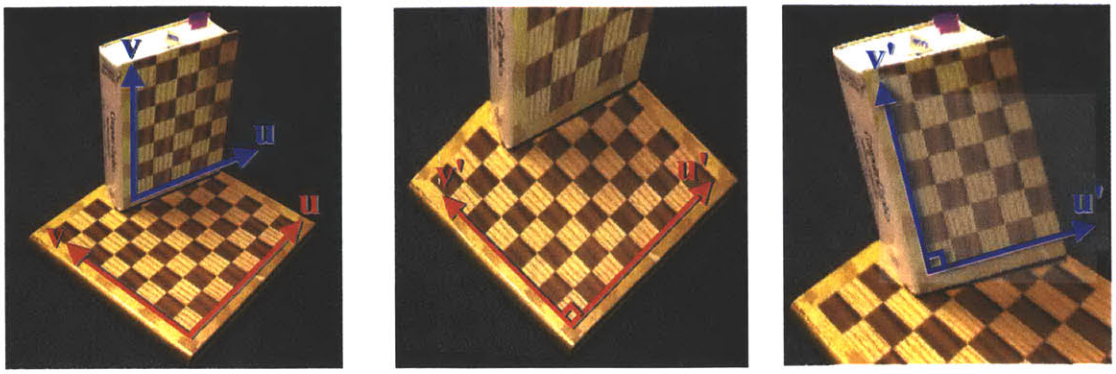
Determining M_r for the 2-plane scenario is more complex since matrices R_r and S_r also need to be specified. To compute the rotation angle for $R_r(\theta)$, we determine the angle between the horizontal axis of the source and the destination in the world plane. A horizontal axis is computed by $\mathbf{u}' = H\mathbf{u}$, where $\mathbf{u}'$ represents the direction of the vanishing point $\mathbf{u}$ at infinity in world-plane coordinates. As shown in Figure 4-9, the rotation angle is then the angle between the source horizontal axis (u'-axis in red) and the destination horizontal axis (u'-axis in blue) in the world plane. In our implementation, we assume that the user specifies these axes while tracing parallel lines for the homography, so no additional user input is necessary. The first set of parallel lines drawn is the u-axis, and the second set is the v-axis. Other methods of interactive specification are also feasible.

To determine the relative scale factor, $S_r(s)$, between the source and the destination planes (Equation (4.5)), we provide a simple interface. The user draws two line segments, and the lengths of the line segments l_s and l_d are computed in the respective source and destination world planes (Figure 4-10). A ratio is then determined that defines the relative scale factor, $s = l_s/l_d$.

Determining T_r is the same as in the 1-plane scenario, and we have all the parameters in Equation (4.7), where $M_r = T_r(t_x, t_y)R(\theta)S(s)$.

4.3 Color Correction

A common problem in clone brushing occurs due to intensity variations in the input image. Even if the materials or textures in the scene are the same, existing lighting conditions influence their



(a) (b) (c)

Figure 4-9: (a) Image plane and user-specified horizontal and vertical axes for source (in red) and destination (in blue). (b) Source world plane. (c) Destination world plane. The rotation angle $R_r(\theta)$ is the angle between the $\mathbf{u}'$-axis in red and the $\mathbf{u}'$-axis in blue.

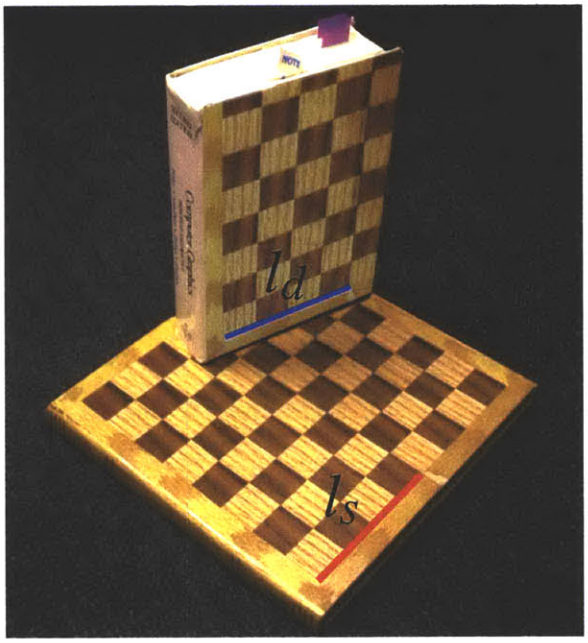
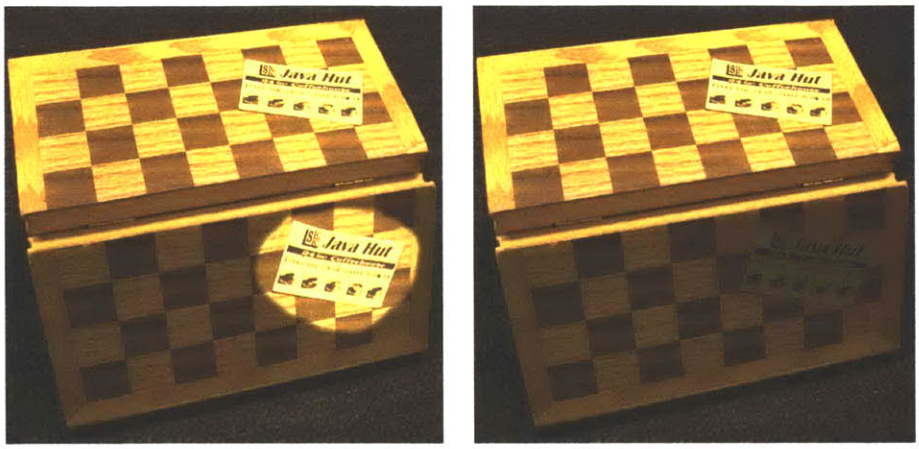


Figure 4-10: The user determines the relative scale factor by drawing line segments on the source plane l_s and the destination plane l_d.

color. As shown in Figure 4-11(a), although the clone-brushed chessboard patterns are aligned, the intensity variations make the clone brushed region appear out of place.

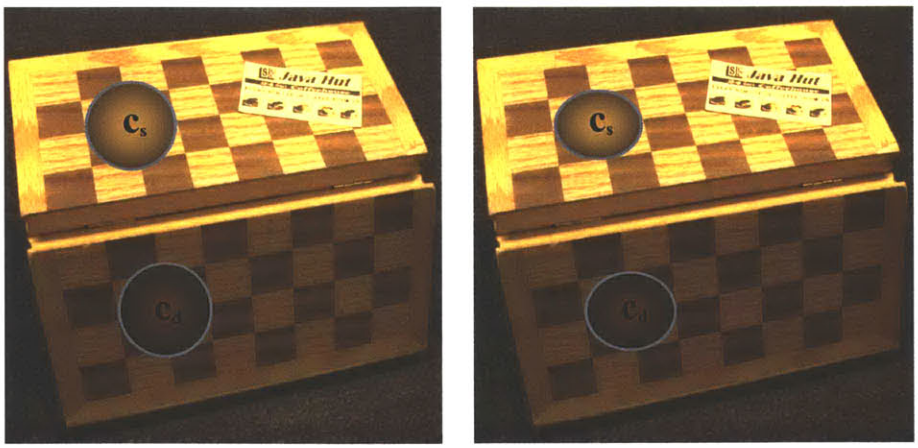
To remedy this problem, we compute a triple of color-correction factors, the RGB ratios from the source to the destination regions, which compensates for the intensity differences. We use a



(a) (b)

Figure 4-11: Color correction. (a) Before. (b) After.

Gaussian-weighted average between sample regions of the source and destination images. The ratios can be computed during the initialization step according to the current brush size. We also allow the user to select sample regions by drawing line segments that determine the position and radius (Figure 4-12). The ratio of the average colors then serves as the correcting factor: For each pixel component copied from the source region, we multiply by $\mathbf{c}_d/\mathbf{c}_s$ to compensate for the color difference. Figure 4-11(b) shows a result of color-corrected clone brushing.

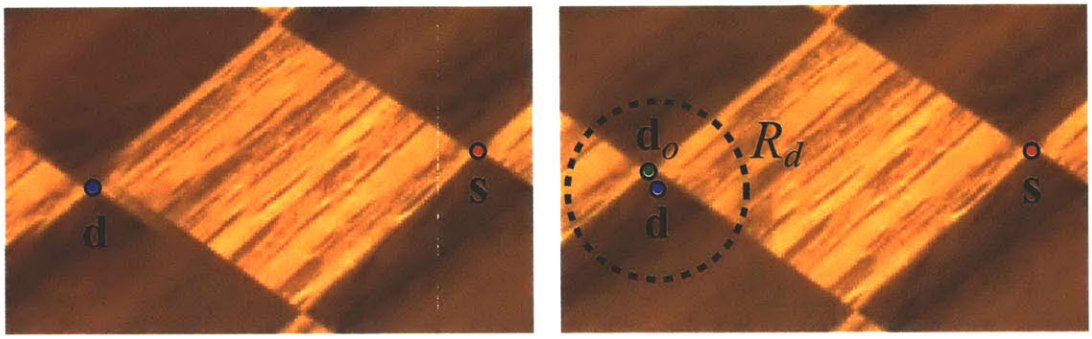


(a) (b)

Figure 4-12: Average color selection. The user simply draws two line segments from which Gaussian-weighted average colors are computed. The color ratio $\mathbf{c}_d/\mathbf{c}_s$ is the correction factor that compensates for the intensity difference. (a) Image plane color selection. (b) World plane color selection.

If the perspective-correction feature is used, the color correction factors are computed in the world plane using the homography matrices. The radius of the color-correction sample region now specifies an area in the world plane. The Gaussian weight is computed with respect to the world-plane distance (Figure 4-12(b)).

4.4 Snapping



(a) (b)

Figure 4-13: (a) The red and blue dots are the initial source position **s** and destination position **d**, respectively. When **s** and **d** are not initialized precisely, misalignment occurs. (b) The green dot $\mathbf{d}_o$ is the optimal destination position computed with our algorithm. The resulting clone brushed region is aligned correctly.

A common problem in clone brushing occurs during the initialization step, when the user manually selects the initial source and destination points to specify the relative translation. Precise selection of these points is necessary, especially for images that have structured features, to avoid misalignment and noticeable artifacts around the seams of the clone brushed region (Figure 4-13(a)).

We improve this process by searching for a more accurate destination point that is most similar to the initial source point. The user only needs to approximately click around the general area, and the closest matching point to the initial source point is automatically computed.

As shown in Figure 4-13(b), we search the local region R_d around the initial destination point **d** for the closest matching point to **s**. We compute a Gaussian-weighted L^2 color difference between a local window centered around each candidate $\mathbf{x}_d \in R_d$ and a local window centered around **s**. The point $\mathbf{x}_d$ with the smallest L^2 difference is used as the new initial destination point, $\mathbf{d}_o$. Since the search space is small, we employ an exhaustive search method, which is found almost instantaneously. In practice, we use $R_d = 20 \times 20$, and the comparison windows around $\mathbf{x}_d$ and **s** are 16×16. It takes 0.15 seconds for an exhaustive search using a Pentium III, 850 MHz machine.

If the homography information is available, the Gaussian-weighted L^2 color difference is computed using Equation (4.7). Also, if the color-correction factor has been computed, color correction is applied while computing the L^2 difference.

4.5 Examples and Discussion

4.5.1 Examples

We show some results using our improved clone brush. In all the examples, snapping was used. In architectural scenes that require precise specification of the relative translation vector, we found this feature to be indispensable. All the examples shown in this chapter took less than 10 minutes of user interaction; most of this time involved brushing. Specifying the homography typically takes less than a minute.

As mentioned, Figure 4-14 shows an example of a 1-plane scenario. Although there are geometric elements that are not necessarily planar to the façade, e.g. the protruding roof and statues, the clone-brushed regions are still convincing.

Similarly in Figure 4-15, even though there are two visible sides of the tower, a single homography was adequate for the result. Clone brushing in Figure 4-16 was a bit more difficult since the lower parts of the windows were, for the most part, occluded.

Figures 4-18 and 4-17 are examples of the 2-plane scenario. For Figure 4-18, actual orthographic photographs of tiles were used. In this case, the source homography was an identity, and the destination homography was the bathroom wall. Since the original wall was white, it was easy to retain existing shadows through simple multiplication with the tile textures.

For the final example in Figure 4-17, we used color correction to clone brush from the dark side (on the right) to the lighter side (on the left) of the building. The original windows on the lighter side were seamlessly blended with the new clone-brushed regions.

4.5.2 Discussion

In this chapter, we have presented techniques to address three main limitations of the traditional clone brush. The new structure-preserving clone brush is capable of correcting the perspective foreshortening problem on planar surfaces, as well as the color differences due to lighting conditions, and can "snap" to a more accurate initial point. These improvements are simple yet effective and broaden the utility and applicability of this powerful photo-editing tool. Our new tool may be used in

Figure 4-14: Clone brushed façade. The façade has been extended from a single photograph outlined in the middle. This example took about a minute, and mainly involved brushing.

our system as well as in any 2D photo-editing system, since the homography specification bypasses the need for depth information.

A possible extension is to allow the user to specify homography using other techniques mentioned by Liebowitz et al. for images that do not have parallel and orthogonal features [Liebowitz and Zisserman 1998; Liebowitz et al. 1999]. For non-planar geometry, the technique presented by Zhang et al. could be used locally to infer orientation of the surface [Zhang et al. 2001]. Our homography specification could also be used with texture synthesis.

One issue we would like to revisit is the problem of resampling. As we have mentioned, we use bilinear interpolation to determine the pixel color from the source region to the destination region. In case of Figure 4-14, where we extended the façade using the 1-plane scenario, the bilinear interpolation becomes necessary for clone brushed parts that are enlarged (right side of the image). A closer look reveals the enlarged parts of the image are blurred due to bilinear interpolation. Bicubic

interpolation may be an alternative in providing a smoother image. As for the clone brushed façade that is further away (to the left), bilinear interpolation is not sufficient to determine the correct sampled color from the source region. The clone brushed pixel at the destination, when transformed to the source region, encompasses a larger area than the four pixels used for bilinear interpolation. A more accurate super-sampling method would be to use an elliptical weighted average filter to determine the color of the pixel in the destination region [Heckbert 1989].

Although we did not encounter any problems with the software implementation using 2D homography, we would also like to experiment with this tool using the graphics hardware and the OpenGL API. We can use the OpenGL transformation matrices to compute the mapping and use MIP-mapping to resolve the super- and sub-sampling issues.

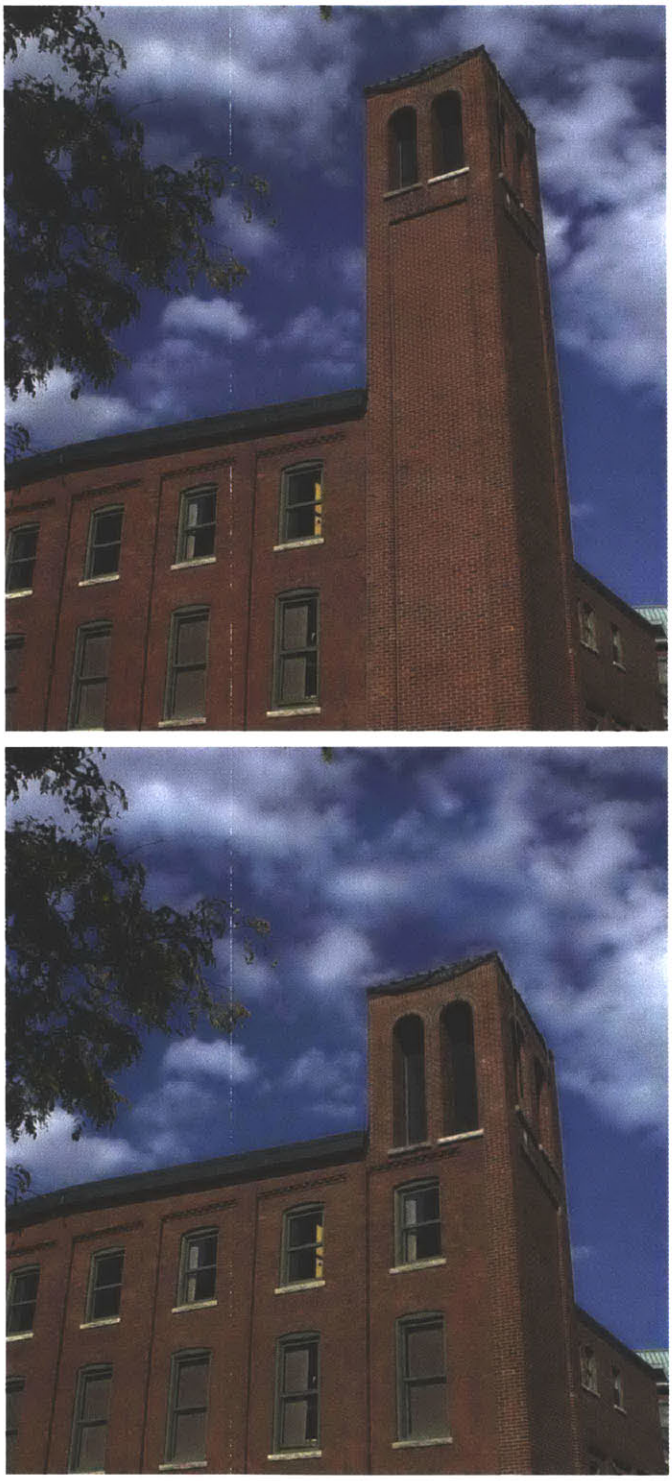


Figure 4-15: Parts of the building façade and the tower have been modified. This example took about 5 minutes.

Figure 4-16: The cars and street details have been removed. This example took less than 10 minutes.

Figure 4-17: The façade from the right side of the building has been clone brushed to the left side using the color-correction feature.

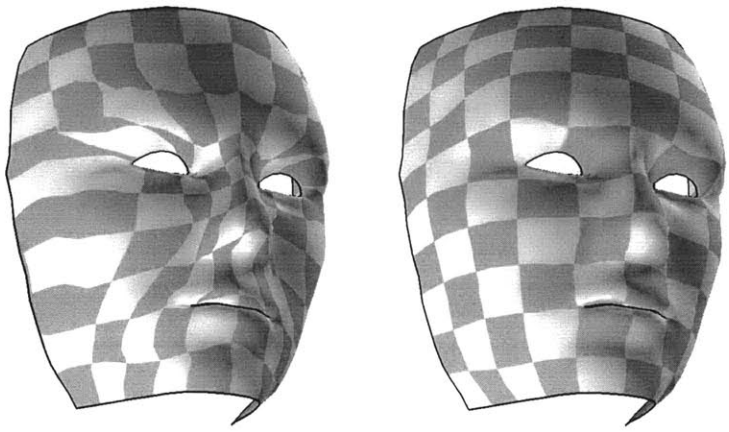
Figure 4-18: Tile patterns have been clone brushed into the photograph. The clone brushing took less than 5 minutes, and it took approximately 10 minutes for selection using Adobe Photoshop.

CHAPTER 5

Non-Distorted Clone Brushing

In the previous chapter, we introduced the structure-preserving clone brush that provides the means to correct the perspective foreshortening on planar surfaces. In this chapter, we discuss a new clone-brushing tool that works on *arbitrary geometry*. In this case, we assume that the depth channel has been assigned using our depth tools or acquired through other means, and do not require that the geometry be flat. Using the depth channel information, we alleviate the distortions due to perspective and surface shape.

The non-distorted clone brushing problem is similar to non-distorted texture mapping: We want to map the pixel color from the source region to the destination, while minimizing distortion. The basic idea is to compute a (u, v) texture parameterization for both the source and destination regions that conforms to the shape of the geometry, and use this parameterization for mapping source to destination regions. To conform the parameterization, we adapt the non-distorted texture mapping approach by Lévy and Mallet [1998; 1989]. Figure 5-1 illustrates an overview of their method, where they use an iterative optimization technique to "smooth" the texture parameters according to the underlying geometric shape.



(a) (b)

Figure 5-1: Lévy and Mallet's non-distorted texture mapping example. (a) Before. (b) After. Courtesy of Bruno Lévy.

As we will see, there are two important differences: our approach needs to run in real time, since the clone brush is an interactive tool, and we have no boundary conditions. As the clone brushing progresses, we parameterize and expand in real time only the necessary parts of the image around the source and destination regions, and we use the non-distorted texture mapping technique to conform the parameterization to the underlying geometry. We then compute the mapping of the matching (u, v) coordinates from the destination to the source, from which we copy the color information from the source back to the destination region.

We first review the key points of Lévy and Mallet's method, then describe the specifics of our adaptation in the non-distorted clone brush context.

5.1 Non-Distorted Texture Mapping

Lévy and Mallet's core algorithm is based on *discrete smooth interpolation* (DSI) [Mallet 1989]. The DSI algorithm measures the global *roughness* of a connected set of discrete nodes, and iteratively minimizes, or smooths, the roughness. It is commonly used in, for instance, automatic contouring of geological sites, where grids of height fields are sampled and interpolated. This algorithm avoids the problem of explicitly computing a function from these samples, and instead produces interpolated and "smoothed" values over the discrete nodes.

In this section, we describe the DSI algorithm in a more specific context, where nodes are vertices of a triangulated surface, and the values interpolated are (u, v) texture parameters at each vertex.

We then discuss how they extend the algorithm for non-distorted texture mapping by constraining the iso-parametric values.

5.1.1 Discrete Smooth Interpolation

Let Ω be the set of vertices of a triangulated surface, and $\mathcal{T}$ the set of triangles formed by Ω. A vertex $\alpha \in \Omega$ has texture parameter $\varphi(\alpha) = (\varphi^u(\alpha), \varphi^v(\alpha)) = (u, v)$. The cost function minimized by the DSI algorithm is the *roughness*, R:

$$R(\varphi) = \sum_{k \in \Omega} \sum_{\rho \in \{u,v\}} \left(\sum_{\alpha \in N(k)} v^\alpha(k) \varphi^\rho(k) \right)^2, \tag{5.1}$$

where $N(k)$ denotes the neighbors of vertex k, including k, ρ is either u or v, and $v^\alpha(k)$ have coefficients defined as follows:

$$\begin{cases} v^\alpha(k) > 0 & \forall \alpha \in N(k) - \{k\} \\ v^k(k) = -\sum\limits_{\substack{\alpha \in N(k) \\ \alpha \neq k}} v^\alpha(k) \neq 0 & \forall k \in \Omega \end{cases}. \tag{5.2}$$

The $v^\alpha(k)$ coefficients act as a weighing function between the current node and its neighbors. Lévy and Mallet use the following simple harmonic weights:

$$v^\alpha(k) = \begin{cases} 1 & \text{if } \alpha \in N(k) - k \\ -degree(k) & \text{if } \alpha = k \end{cases}, \tag{5.3}$$

where $degree(k)$ is the number of neighbors of k.

More intuitively, the roughness function, R, compares the (u, v) parameter values of the current vertex to its neighbors and sums up the weighted differences, i.e. the "roughness," in a least squared sense. The minimum of R is reached when $\partial R(\varphi)/\partial \varphi^\rho(\alpha) = 0$ for each pixel $\alpha \in \Omega$ and for each $\nu \in \{u, v\}$. This results in the following equation,

$$\varphi^\rho(\alpha) = -\frac{G^\rho(\alpha)}{g^\rho(\alpha)}, \tag{5.4}$$

where

$$G^\rho(\alpha) = \sum_{k \in N(\alpha)} \left\{ v^\alpha(k) \sum_{\substack{\beta \in N(k) \\ \beta \neq \alpha}} v^\beta(k) \varphi^\rho(\beta) \right\} \tag{5.5}$$

$$g^{\rho}(\alpha) = \sum_{k \in N(\alpha)} \{v^{\alpha}(k)\}^2. \tag{5.6}$$

The following is the pseudocode for the DSI algorithm that iteratively minimizes R:

Algorithm 5.1.1: DSI($\Omega, \mathcal{T}$)

φ = initialization

repeat

for each $\alpha \in \Omega$

do { **for each** $\rho \in \{u, v\}$ **do** $\left\{\varphi^{\rho}(\alpha) = -\frac{G^{\rho}(\alpha)}{g^{\rho}(\alpha)}\right.$

until more iterations are needed

Algorithm 5.1.1 iteratively computes and assigns $(\varphi^u(\alpha), \varphi^v(\alpha)) = (u, v)$ that minimizes R. The optimization algorithm converges to a unique solution, provided that at least one vertex α has a fixed parameter value, $\varphi(\alpha)$, and the $v^{\alpha}(k)$ coefficients are defined according to Equation (5.2).

5.1.2 Gradient Constraints

To utilize DSI for non-distorted texture mapping, Lévy and Mallet add two constraints to iteratively minimize two classes of distortions: angular (preserve orthogonal angles) and isoparametric distance (make isolines equidistant). The former requires that the gradients of u and v be orthogonal, and the latter requires constant magnitude for their gradients. Different weights can be assigned to emphasize one constraint over another.

They first compute the gradient of each triangle $\mathbf{T} = (\alpha_0, \alpha_1, \alpha_2)$, where $\mathbf{T} \in \mathcal{T}$. As shown in Figure 5-2, the local orthonormal basis of triangle $\mathbf{T}$ is $\mathbf{X}$ and $\mathbf{Y}$, such that

$$\begin{aligned} \mathbf{X} &= \frac{p(\alpha_1) - p(\alpha_0)}{\|p(\alpha_1) - p(\alpha_0)\|} \\ \mathbf{N} &= \frac{\mathbf{X} \times (p(\alpha_2) - p(\alpha_0))}{\|\mathbf{X} \times p(\alpha_2) - p(\alpha_0)\|} \\ \mathbf{Y} &= \mathbf{N} \times \mathbf{X}, \end{aligned} \tag{5.7}$$

where $p(\alpha)$ is the 3D world space coordinates of vertex α, as projected according to its depth channel value and the reference camera parameters.

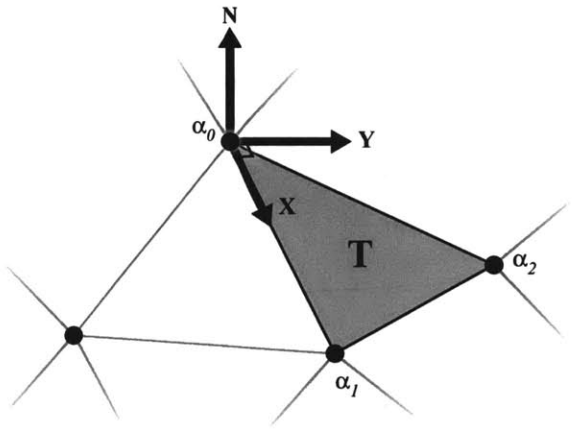


Figure 5-2: Local orthonormal bases, **X** and **Y**, and normal, **N**, for triangle $\mathbf{T} = (\alpha_0, \alpha_1, \alpha_2)$.

The gradient is constant over the triangle **T** and is a linear combination of the values at the three vertices. The partial derivatives for triangle **T** are defined as follows:

$$\frac{\partial \varphi_T^\rho}{\partial X} = \sum_{j=0}^{2} D_X(\alpha_j)\varphi^\rho(\alpha_j) \tag{5.8}$$

$$\frac{\partial \varphi_T^\rho}{\partial Y} = \sum_{j=0}^{2} D_Y(\alpha_j)\varphi^\rho(\alpha_j), \tag{5.9}$$

where

$$\begin{aligned} D_X(\alpha_0) &= (y_1 - y_2)/d \\ D_X(\alpha_1) &= (y_2 - y_0)/d \\ D_X(\alpha_2) &= (y_0 - y_1)/d \\ D_Y(\alpha_0) &= (x_2 - x_1)/d \\ D_Y(\alpha_1) &= (x_0 - x_2)/d \\ D_Y(\alpha_2) &= (x_1 - x_0)/d \end{aligned}$$

$$d = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

$$\left.\begin{aligned} x_j &= (\alpha_j - \alpha_0)\cdot\mathbf{X} \\ y_j &= (\alpha_j - \alpha_0)\cdot\mathbf{Y} \end{aligned}\right\} \forall j \in \{0,1,2\}.$$

The coefficients $D_X(\alpha_j)$ and $D_Y(\alpha_j)$ solely depend on the geometry of the triangle **T**, so they are computed only once (assuming the vertices do not change positions, i.e. the geometry remains

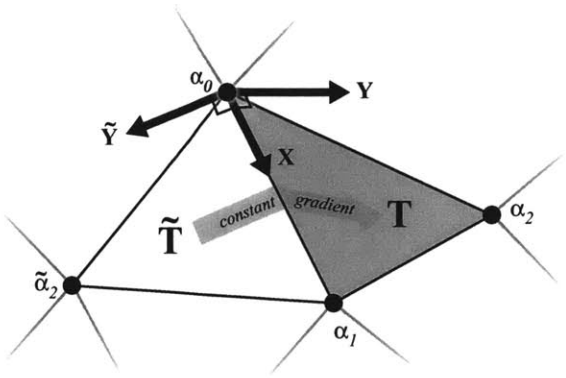


Figure 5-3: Constant gradient across a shared edge of two triangles.

constant).

Orthogonality Constraint

Given the definition of the gradients of φ from Equations 5.8 and 5.9, the orthogonality of iso-u and iso-v curves on triangle **T** is described as follows:

$$\begin{bmatrix} \frac{\partial \varphi_T^u}{\partial X} & \frac{\partial \varphi_T^u}{\partial Y} \end{bmatrix} \begin{bmatrix} \frac{\partial \varphi_T^v}{\partial X} \\ \frac{\partial \varphi_T^v}{\partial Y} \end{bmatrix} = 0. \tag{5.10}$$

In other words, their dot product should be zero to enforce orthogonality.

Constant Gradient Constraint

The second constraint is to maintain a constant gradient across triangles, i.e. a homogeneous spacing of isoparametric curves. To do this, we compute a common local basis for two adjacent triangles, **T** and $\tilde{\mathbf{T}}$, as shown in Figure 5-3:

$$\frac{\partial \varphi_{\mathbf{T}}^u}{\partial X} = \frac{\partial \varphi_{\tilde{\mathbf{T}}}^u}{\partial X} \tag{5.11}$$

$$\frac{\partial \varphi_{\mathbf{T}}^v}{\partial X} = \frac{\partial \varphi_{\tilde{\mathbf{T}}}^v}{\partial X} \tag{5.12}$$

$$\frac{\partial \varphi_{\mathbf{T}}^u}{\partial Y} = -\frac{\partial \varphi_{\tilde{\mathbf{T}}}^u}{\partial \tilde{Y}} \tag{5.13}$$

$$\frac{\partial \varphi_{\mathbf{T}}^v}{\partial Y} = -\frac{\partial \varphi_{\tilde{\mathbf{T}}}^v}{\partial \tilde{Y}}. \tag{5.14}$$

These equations contrain the gradients of the two triangles to be the same along the shared $\mathbf{X}$ and $\mathbf{Y} = -\tilde{\mathbf{Y}}$ directions.

5.1.3 Iterative Smoothing Algorithm

Lévy and Mallet define a *generalized roughness* $R^*(\varphi)$, where they combine the linear constraints together with the roughness factor:

$$R^\star(\varphi) = R(\varphi) + \sum_{c \in C} \left\{ \left(\sum_{\rho \in \{u,v\}} \sum_{\alpha \in \Omega} A_c^\rho(\alpha) \varphi^\rho(\alpha) \right) - b_c \right\}^2, \tag{5.15}$$

where A_c^ρ and b_c are given coefficients *when one parameter, u or v, is assumed constant.* The smoothing on $\varphi^\rho(\alpha)$, where $\rho \in \{u,v\}$, is interleaved, which means that the values of A_c^u depend on v, and vice-versa.

This generalized roughness $R^\star$ reaches its minimum when its partial derivatives, $\frac{\partial R^\star}{\partial u}$ and $\frac{\partial R^\star}{\partial v}$, are null for each vertex, or in our case pixel, α. The following equation then iteratively minimizes $R^\star$ with the two constraints factored in:

$$\varphi^\rho(\alpha) = -\frac{G^\rho(\alpha) + \Gamma^\rho(\alpha|\varphi)}{g^\rho(\alpha) + \gamma^\rho(\alpha)}, \tag{5.16}$$

where

$$\Gamma^\rho(\alpha|\varphi) = \sum_{c \in C} A_c^\rho(\alpha) \left(\sum_{\beta \neq \alpha} A_c^\rho(\beta) \varphi^\rho(\beta) - b_c \right) \tag{5.17}$$

$$\gamma^\rho(\alpha) = \sum_{c \in C} (A_c^\rho(\alpha))^2. \tag{5.18}$$

The final algorithm for the non-distorted texture mapping is:

Algorithm 5.1.2: NON-DISTORTED TEXTURE PARAMETERIZATION($\Omega, \mathcal{T}$)

φ = initialization (1)

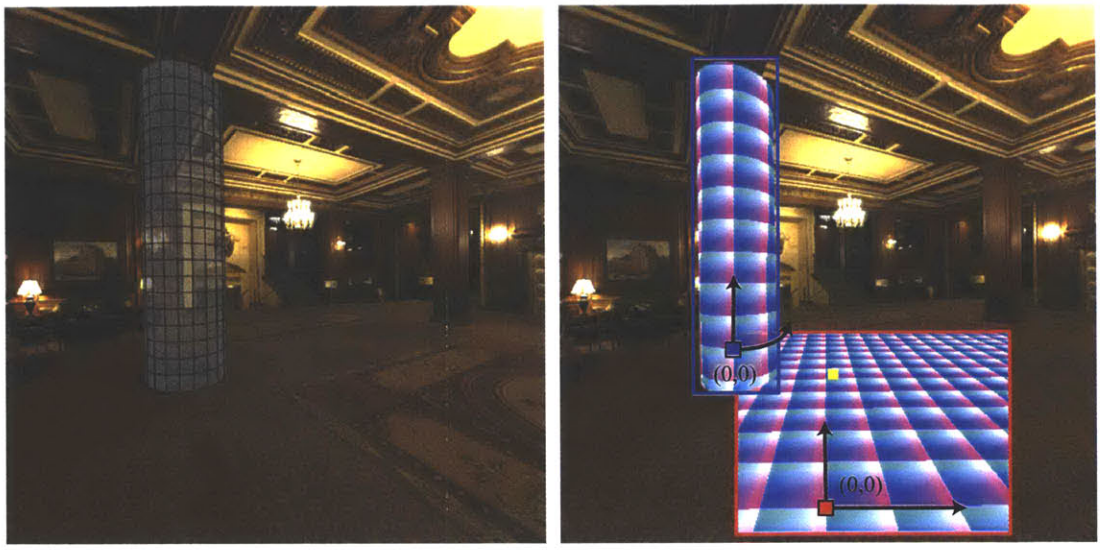
repeat

for each $\rho \in \{u,v\}$

do { **for each** $\alpha \in \Omega$

do $\left\{ \varphi^\rho(\alpha) = -\frac{G^\rho(\alpha)+\Gamma^\rho(\alpha|\varphi)}{g^\rho(\alpha)+\gamma^\rho(\alpha)} \right.$

until more iterations are needed



(a) (b)

Figure 5-4: Underlying visualization of the optimized (u, v) parameterization. (a) Initial image with a cylindrical column in wireframe to show the geometry. (b) The source active region is in red, the destination active region is in blue. The initial source and destination points selected by the user are the origins of the parameter space. Note that the parameterization optimization conforms to the existing geometry.

5.2 Clone Brushing with Non-Distorted Parameterization

The non-distorted texture mapping technique is too slow to optimize over an entire image in an interactive context. Moreover, it requires boundary conditions. We adapt the algorithm to work in our interactive context using two strategies: a flood-fill approach with a simple but effective initialization, and a special treatment of the boundary pixels. The parameterization and optimization only take place locally around the clone brushed region, where we "flood-fill" and initialize with parameters that are already close to the solution. This initialization step (Algorithm 5.1.2, line (1)) quickly converges the optimization process to its solution.

Figures 5-4 and 5-5 provide an intuitive overview of the non-distorted clone brushing process. In this example, the user clone brushes from the ground plane to the cylindrical column. Similar to the traditional clone brush, the user first selects the initial source and destination points. The initial source and destination points are parameterized as $(u_0, v_0) = (0,0)$. As illustrated in Figures 5-5(a) and (b), the (u, v) parameters of a small region around the initial points have been "flood-filled" and initialized. As the user clone brushes around the destination region (overlayed in yellow), the parameterized region expands and is optimized accordingly (Figure 5-5(c)). It is necessary to

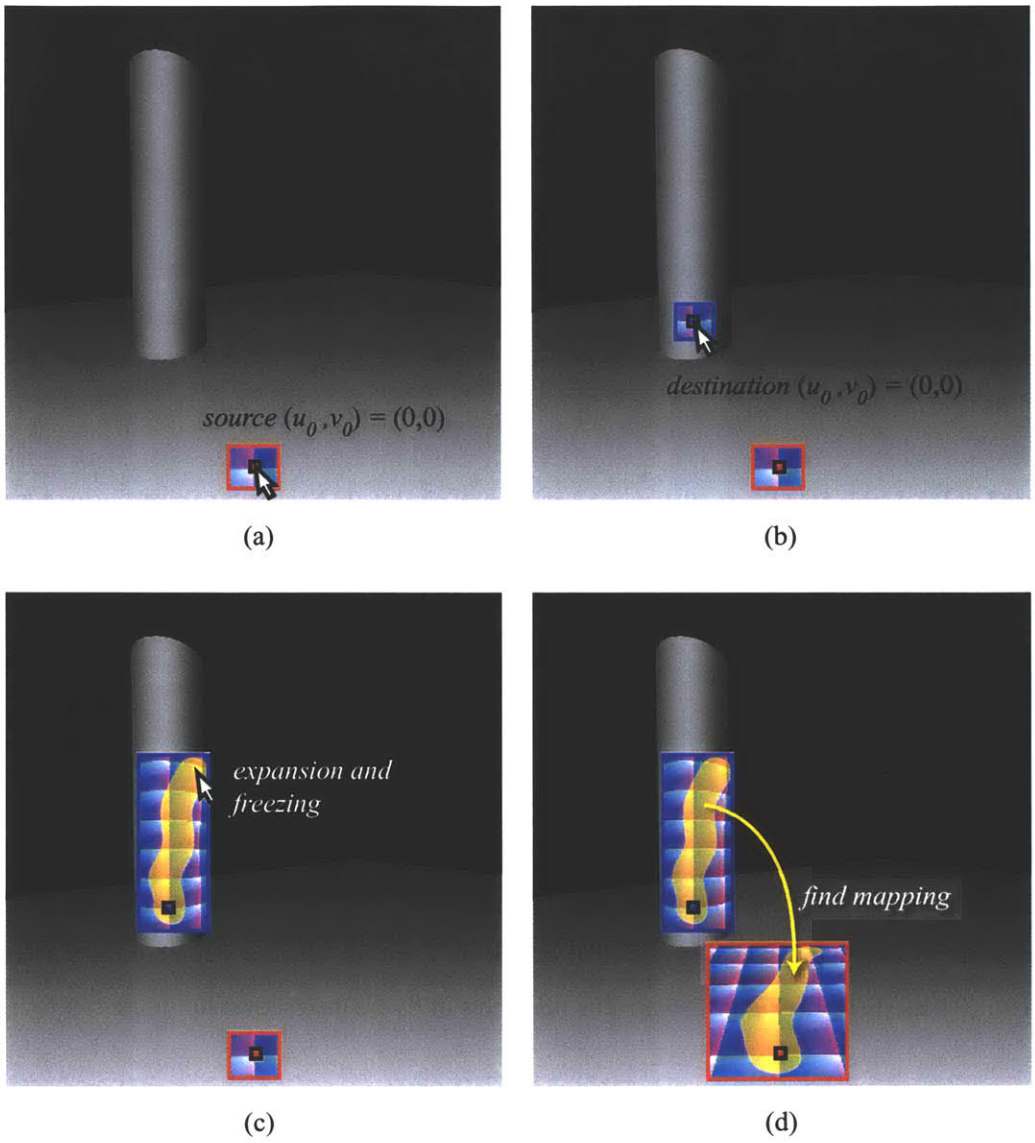


Figure 5-5: Non-distorted clone brushing overview. (a) As the user first selects the initial source point, an area around the source region is parameterized (in red outline). (b) Similarly, as the user selects the initial destination point, the destination region is also parameterized (in blue outline). (c) As the clone brushing progresses, the parameterization of the destination region expands, and iterative optimization runs concurrently only within the parameterized region. (d) The parameterization of the source region also expands, as the correlating parameterizations of the pixels are found from source to destination regions. The clone brushed regions overlayed in yellow are *frozen* pixels, since they are clone brushed and their values should not change.

parameterize and optimize only for the bounded subregion to maintain user interactivity. We then find matching (u, v) coordinates of brushed pixels from the destination to the source (overlayed in yellow). Figure 5-5(d) shows the expanded parameterization and matching (u, v) coordinates of the

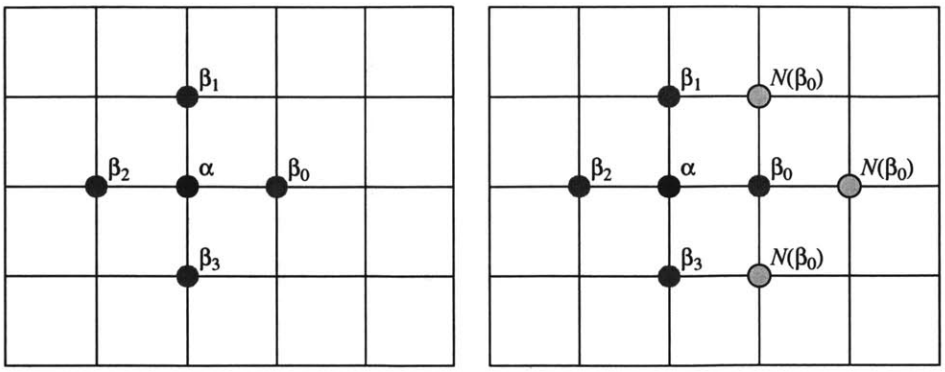


Figure 5-6: Pixel grid of our representation. (a) Neighbors of pixel α. (b) Neighbors of pixel β_0 (for Equation 5.5).

source region. Once the destination parameters of each pixel have been mapped to the source, we copy the pixel color from the source and replace it on the destination region.

5.2.1 Pixel Grid

In our case, since our representation consists of layers of images with depth, we have a regular grid of pixels as our vertices, and (u,v) parameterization of the pixels is optimized using the DSI algorithm. To compute $G^p(\alpha)$ using Equation 5.5, we consider the four neighboring pixels, β_k, $k \in \{0,..,3\}$, as shown in Figure 5-6(a). The neighbors of β_0 are then shown in Figure 5-6(b).

5.2.2 Subsampling

The parameterization proceeds as the user interactively clone brushes. It must be faster than the speed of the brushing to ensure smooth interaction. In practice, we have found that subsampling the layer is necessary in order to obtain real-time feedback. We compute (u,v) values for every 4×4 pixels and interpolate bilinearly. This process does not take into account the local bumps in the geometry, but fits the global shape.

5.3 Real Time Flood-Fill Parameterization

We adapt the DSI method in a "flood-fill" manner to optimize the parameterization around the clone-brushed region in real time. Although the method is described for a single layer, it runs

concurrently for both the source and destination layers. Parameterizations for only a subset of pixels are computed, called the *active pixels*, as shown in red (source) and blue (destination) boundaries in Figure 5-5. These regions are expanded as time progresses and the user drags and applies the clone brush. We interleave *optimization* steps, where the (u,v) coordinates are refined, and *expansion* steps, where new pixels are declared active and initialized. Finally, the coordinates of already-brushed pixels are *frozen* since the colors should not change once clone brushed. Figures 5-5(c) and (d) shows the frozen pixels in overlayed yellow.

To initialize the process, we use the first point clicked by the user as the origin of the parameter space, assign it the coordinates $(u_0, v_0) = (0,0)$, and set the gradient of u, ∇u, orthogonal to the vertical direction and to the pixel normal. The gradient ∇v is then orthogonal to ∇u, and is computed similarly to Equation (5.7).

The set of pixels at the boundary of the active region is called the *active front*. As shown in Figure 5-4, the active front is outlined in red and blue for source and destination regions, respectively. More formally, a pixel is declared in the active front if it is active and if one of its neighbors has an inactive neighbor (Figure 5-7). Intuitively, the active front corresponds to pixels that lack neighbors necessary to compute smoothing in Equations (5.1) and (5.15). Active pixels not in the active front are said to be *fully active*.

5.3.1 Optimization

The optimization follows the steps of the DSI algorithm (Algorithm 5.1.2). Optimizations on u and v are interleaved, and active pixels are treated as vertices of a mesh and smoothed accordingly using linear operations.

The active front requires special care. Due to the absence of some neighbors, Equation (5.16) cannot be directly used. If these neighbors are simply discarded from the formula, the parameterization will "shrink," because the roughness term R in Equation (5.15) is no longer balanced. We thus only optimize the gradient constraints for these pixels and omit the mapping of the roughness term. A good initial value for the active-front pixels is then the key to ensuring stability of the process.

5.3.2 Expansion and Initialization

The expansion step extends the active region by one pixel in the direction of the current clone brush location. The active front is accordingly updated, and each new active pixel receives initialized parameter values. This is done according to each pixel's active neighbors, by using a locally-planar

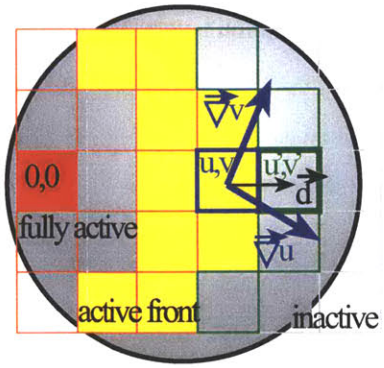


Figure 5-7: Flood-fill parameterization. Fully active pixels are in red. The active front is in yellow. The green pixel is set to active, and its initial (u', v') parameters are computed using the gradient of its active neighbors.

approximation of the geometry. For each neighbor, the coordinates (u', v') of the new pixel are computed using the current gradients of an active neighbor, $(\nabla u, \nabla v)$, and the object-space vector $\vec{d}$ between the two pixels (Figure 5-7):

$$(u', v') = (u + \vec{d} \cdot \nabla u, v + \vec{d} \cdot \nabla v). \tag{5.19}$$

The average of the values computed from the active neighbors is then used. This formula results in an optimal initial value, provided the geometry is planar.

We use a bounding rectangle to expand the active regions. Since the clone brushing takes place in the destination region, expanding the active destination region is simple. The active source region is expanded according to the parameter mapping of the current clone brush position in the destination region. If the current (u, v) parameters of the destination region exist outside of the current active source region, the active source region is then expanded accordingly. The parameterization is monotonic in the gradient directions, $(\nabla u, \nabla v)$, so querying whether to expand the active source region or not is straightforward. The source region expansion is illustrated in Figures 5-5(c) and (d), where the source parameterization is only expanded to find the appropriate (u, v) mapping. This notion is further explained in Section 5.4.

5.3.3 Freezing

As soon as a pixel has been clone brushed, its (u, v) coordinates must be flagged as *frozen*, which means that the pixel's (u, v) coordinates are no longer optimized. This avoids artifacts that would occur if the same pixels were re-used with different coordinate values due to subsequent optimization

iterations. Only fully active pixels may be frozen.

5.4 Source and Destination Mapping

At this point, both the source and destination regions are parameterized using our non-distorted texture mapping technique that conforms to the geometry of the scene. As the clone brushing progresses, we must now find which pixel colors from the source region should be copy-pasted to the clone brushed pixels in the destination region. To do this, we search for the pixel, α_s, in the source region that has the same parameter coordinates as the current pixel, α_d, being clone brushed in the destination region, such that $\varphi_s(\alpha_s) = \varphi_d(\alpha_d)$, where $\varphi_d(\alpha_d) = (u_d, v_d)$ is known. This is an inverse problem, where we search for the pixel location given the following parameterization:

$$\alpha_s = \varphi_s^{-1}(\varphi_d(\alpha_d)). \tag{5.20}$$

We use Newton's method, an iterative search method that linearly approximates a function around the local region to compute the next guess, to find the mapping, φ_s^{-1}. In our case, the function is φ_s, and the next iterative guess brings us a step closer to α_d, such that $\varphi_s(\alpha_s^k) - \varphi_d(\alpha_d) = 0$. At each iteration, $k = 0, 1, \ldots$, we compute

$$\alpha_s^{k+1} = \alpha_s^k + \delta^k, \tag{5.21}$$

where δ^k is the step vector that is a linear approximation at $\varphi_s(\alpha_s^k)$. The exact solution is found in a single iteration when the geometry is planar. Any active pixel in the source region may be used as the initial position, $\varphi(\alpha_s^0) = (u_s^0, v_s^0)$. We use the last known brush position as α_s^0, since the current brush position is assumed to be close.

The step vector, δ^k, is computed from the local gradients, $\nabla u_s = [\frac{\partial u}{\partial X}, \frac{\partial u}{\partial Y}]$ and $\nabla v_s = [\frac{\partial v}{\partial X}, \frac{\partial v}{\partial Y}]$, and the kth iterative approximation is defined as follows:

$$\delta_X = \frac{(u_d - u_s^k)\frac{\partial v}{\partial Y} - (v_d - v_s^k)\frac{\partial u}{\partial Y}}{\frac{\partial v}{\partial X}\frac{\partial u}{\partial Y} - \frac{\partial u}{\partial X}\frac{\partial v}{\partial Y}} \tag{5.22}$$

$$\delta_Y = \frac{(u_d - u_s^k)\frac{\partial v}{\partial X} - (v_d - v_s^k)\frac{\partial u}{\partial X}}{\frac{\partial v}{\partial X}\frac{\partial u}{\partial Y} - \frac{\partial u}{\partial X}\frac{\partial v}{\partial Y}}, \tag{5.23}$$

where $\delta^k = (\delta_X, \delta_Y)$.

If the parameterization $\varphi_s(\alpha_s^k)$ is outside of the current active source region, then we expand the active source region in the direction of pixel $\alpha_s^k + \delta^k$. Remember that the expansion process initializes and activates the not-active pixels, i.e. initializes their (u,v) parameterization values. If the source region parameterization expands to the extent of the layer, then the parameters are not found within the source layer and we return a default color value, black.

In general, no pixel parameterization in the source region will have the exact (u_d, v_d) parameter values, such that $\varphi_s(\alpha_s^k) - \varphi_d(\alpha_d) = 0$. We stop the iterations when (u_d, v_d) is between the parameter values of four surrounding pixel neighbors in the source region. The final step is to compute the bilinearly-interpolated color value from the four neighboring pixels, and to copy-paste it to the destination pixel, where clone brushing is currently taking place.

5.5 Examples and Discussion

5.5.1 Examples

We show a simple example of the non-distorted clone brush in Figure 5-8. The initial image is a grid of repeated texture, and a sphere tool was applied at the dotted area, as shown in Figures 5-8(a) and (b). Our non-distorted clone brush takes the underlying geometry into consideration, as shown in Figure 5-8(d), and conforms the texture with the geometry. The "stretching" of the texture in Figure 5-8(c) is alleviated in (d), and the orthogonality and constant-gradient constraints are maintained.

A more practical example is shown in Figure 5-9, where the rectangular column from the original image has been replaced by a cylindrical column. We clone brushed the rectangular column's original texture onto the new geometry. We then clone brushed the carpet to the ceiling, and clone brushed the floor with the existing carpet patterns. This example took about 15 minutes, which is comparable time to the traditional clone brush. Note that this example could not be accomplished using the traditional clone brush or the structure-preserving clone brush, due to the non-planar geometry.

5.5.2 Discussion

In this chapter, we have discussed a new non-distorted clone brush tool that works on arbitrary geometry. We first parameterize the source and destination regions such that the parameterization conforms well with the underlying geometry. Only the clone brushed regions are parameterized with a simple (u,v) initialization, and expanded as needed to maintain the real-time feedback to

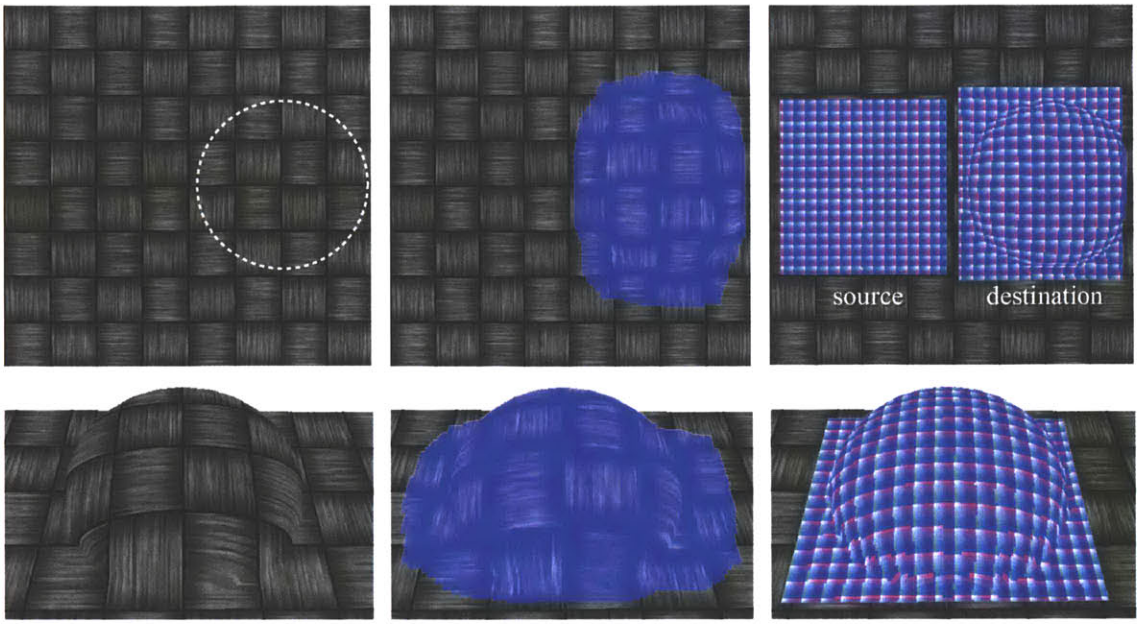


Figure 5-8: Simple example. (a) Initial image. The top image is the initial reference image with dotted lines to show the hemisphere. The bottom image is at an angle to show the geometry. Note that due to the projective texture, the reference view on top shows no hint of the underlying hemisphere geometry. (b) Non-distorted clone brushed image. The same texture has been clone brushed using our new approach onto the hemisphere. The clone-brushed region is overlayed in blue. (c) Visualization of the parameterization for both the source and the destination. Note that the parameterization conforms to the underlying geometry.

the user. We take special care on the boundaries of our expanding regions to avoid the "shrinking" effect. Finally, we use Newton's method to map the parameterizations from destination to source, which then determines the pixel color copied from the source to the destination.

Our optimization technique is not as accurate as the method of Lévy and Mallet [1998; 1989], due to subsampling, but it provides an exact solution in the case of planar geometry, and has worked well in practice for curved geometry. This is because our case is simpler than the general mesh-parameterization problem. The geometry data, i.e. the depth channel, is a height field transformed by a perspective matrix, which greatly decreases potential distortions. Moreover, the layers are segmented by the user into different spatial objects that prevent strong discontinuities.

We would like to conduct a more formal study on the stability issues regarding the optimization process in the context of clone brushing. Our initialization and special care at the boundary pixels stabilize the shrinking of the parameterization due to the lack of boundary conditions, but a formal study may support and suggest possible improvements.

(a) (b)

Figure 5-9: (a) Original image. (b) Non-distorted clone brushing. The column geometry has been changed to a cylinder, and the carpet has been removed and clone brushed onto the ceiling.

Finally, our current implementation only performs the optimization during the clone brushing process. Since our system is interactive, it would make sense to optimize the parameterizations during idle time spans, or spawn separate threads to continue the optimization process.

CHAPTER 6

Texture-Illuminance Decoupling

ne of the major limitations in photo editing is the lack of tools to convincingly alter and modify the existing lighting conditions or the materials in a scene. In this chapter, we introduce a new technique to seamlessly factor the existing lighting from the materials of the input image by decoupling these components into texture and illuminance images. Although there are photo-editing tools that allow the specification of pseudo lighting effects that brighten or darken parts of a flat photograph, removing the existing illumination is extremely difficult and tedious. Manually darkening highlights or brightening shadowed regions to factor out the lighting effects on the texture typically results in artifacts near the illumination discontinuities. Similarly, seamless modification of materials affected by the existing lighting conditions is challenging. Painting or applying different textures on the image results in loss of existing illumination, and manually blending in illumination effects can be tedious, time consuming, and difficult for non-artists. The existing illumination conditions also limit the effectiveness of the clone brush.

We introduce a texture-illuminance decoupling filter that attempts to separates the intrinsic "texture" and environmental "illuminance" channels from the input image (Figure 6-1). The decoupled

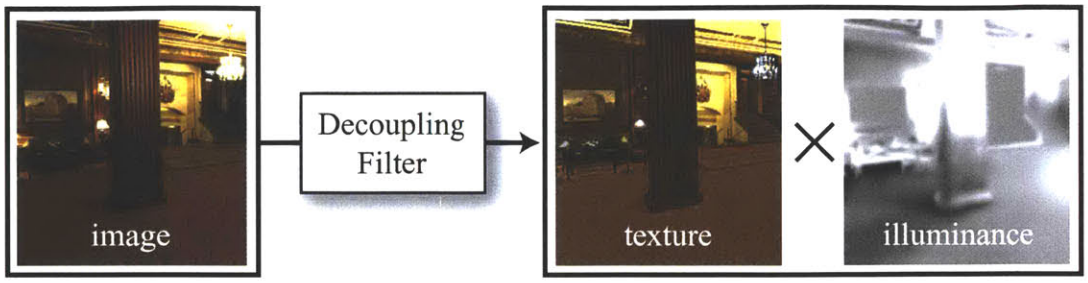


Figure 6-1: Texture-illuminance decoupling filter overview. From an initial image, the decoupling filter separates texture and illuminance channels, as a result of a simple assumption that illuminance varies slowly over a surface, while texture varies more rapidly. Multiplying the two decoupled channels results in the initial input image.

channels are not physically-based texture or illuminance values, but the filter is easy to implement, and provides rapid and convincing results that are easy to edit. It is a non-linear image-processing filter that decouples the lighting effects from uniformly textured images, and thus enables the user to easily edit either the materials or the lighting characteristics of the input image. Multiplying the two separated channels results in the original input image. Applying lighting effects on the illuminance channel modifies the illumination of the input image, while maintaining the existing materials; editing the texture channel changes the materials in the scene, while maintaining the existing illumination. Clone brushing on the uniform texture channel works well, since the illumination effects are now separated.

Figure 6-2 shows an example, where (a) is the initial photograph of a hotel lobby and (b) is the material and lighting changes as a result of using the decoupling filter. Note that the existing illumination is maintained (the shadows of the left column) even when an entirely different floor pattern has been clone brushed into the scene. These results would be difficult to achieve, if possible at all, even with high-end photo-editing systems.

The texture-illuminance decoupling filter is based on a simple assumption: that illumination consists of low frequencies and varies slowly, while the texture generally consists of high frequencies that vary rapidly. A naive idea would be to use a low-pass filter to separate the input image into small- and large-scale featured images, for texture and illuminance, respectively. But, not surprisingly, this naive idea fails due to two limitations. First, the low-pass filter operates on the 2D image space, thus it only takes the colors of the neighboring pixels into account. The actual 3D scene depicted in the image, such as the object shape and the perspective foreshortening, are not taken into

(a) (b)

Figure 6-2: (a) Initial image. (b) Result from using our texture-illuminance decoupling filter. Note that we were able to replace the original floor carpeting with a different material while maintaining the original lighting condition, including the shadows from the near-by column.

consideration in separating illuminance from texture. Second, sharp lighting discontinuities, such as shadow boundaries, produce haloing artifacts, since they contradict our assumption that illumination consists only of low frequencies. In the Fourier domain, the sharp illumination discontinuities introduce frequencies that lie in the high-frequency domain, thus, they are decoupled incorrectly as part of the texture channel.

In the following sections, we first discuss some related work that uses inverse lighting simulation techniques to recover reflectance characteristics. We then discuss our basic idea and assumption in detail, which entails separating the input image into texture and illuminance channels according to a user-specified feature size (Section 6.2). We then improve the basic idea by addressing the two limitations mentioned above — first by using depth correction that takes the depth channel into consideration to compute the low-pass filter (Section 6.3), and then using an edge-preserving filter to preserve lighting discontinuities in the illumination channel (Section 6.4). Finally, we show some examples and discuss future work and possible improvements.

6.1 Previous Work

The goal of our approach is not physical accuracy, but rather to provide a means to seamlessly relight and change the materials in the input image. Much of the previous work relies on physically-based inverse light transport simulation to remove the effect of existing illumination and recover the reflectance model [Fournier et al. 1993; Drettakis et al. 1997; Yu and Malik 1998; Yu et al. 1999; Loscos et al. 1999; Loscos et al. 2000]. In general, these techniques require knowledge of the scene geometry, light sources, and the camera information. The reflectance model is then estimated by comparing the lighting simulation solution of the virtual scene with the actual captured photographs.

Fournier et al. [1993] and Drettakis et al. [1997] address the problem of compositing synthetic objects in a real scene that share common illumination. They approximate the geometry and the lighting of the scene, then compute the diffuse reflectance estimates using the progressive radiosity technique [Cohen et al. 1988]. Drettakis et al. further present a method for fast illumination updates for synthetic objects that move within the real scene.

Yu et al. recover the diffuse and specular reflectances of an indoor scene with the assumption that the geometry, the light sources, and the camera pose are known [Yu et al. 1999]. Multiple high dynamic-range images are taken to model a fairly accurate scene geometry and camera pose using the *Façade* system [Debevec et al. 1996]. At least one specular highlight must be visible in the input image to estimate the specular component. They also use omni-directional lights to simplify the selection of virtual light source models. Finally, an inverse light transport simulation is employed to minimize the parameters of Ward's anisotropic BRDF reflectance model [Larson 1992] to match the high dynamic-range images. The system is capable of removing indirect lighting effects, such as color bleeding.

Debevec also estimates the reflectance characteristics to seamlessly composite virtual objects into a real photograph [Debevec 1998]. He divides the scene into the distant scene, the local scene, and the synthetic objects. An approximate geometry of the local scene is constructed with the *Façade* system, and the distant scene is treated as an environment-mapped incoming light, which is captured using the light probe [Debevec]. To seamlessly composite the synthetic objects with the local scene, the authors employ a differential rendering technique, where they use RADIANCE [Ward 1994] and minimize the difference between the reflectance and the input photographs taken from one or more viewpoints.

Loscos et al. present work in interactive relighting and computer-augmented reality [Loscos

et al. 1999; Loscos et al. 2000], which builds on the pioneering work of Fournier et al. [1993]. They approximate the scene geometry from a set photographs, and the light source positions are assumed to be known. However, they limit themselves to a single view, where the camera position is fixed. They compute diffuse texture reflectance estimates using the known geometry, light source, camera position, and the high dynamic-range initial image obtained from the iterative radiosity solution. They are then able to relight and modify the scene at interactive rates, using the interactive radiosity technique by Drettakis and Sillion [1997]. They are also able to remove real objects by applying a texture-filling algorithm to their reflectance estimate.

Boivin and Gagalowicz similarly estimate a reflectance model given a single input photograph, the 3D geometry of the scene, the light sources, and the camera information [Boivin and Gagalowicz 2001]. They also use a differential technique, where they compare the virtual scene rendered using a global illumination technique with the input photograph, and update the reflectance estimate to a more complex model to minimize the difference.

Although these various techniques are capable of relighting or rendering synthetic objects into real scenes, they are impractical in the context of interactive user systems for several reasons. First, recovering accurate geometry, light source positions, and the camera pose from photographs of arbitrary scenes is difficult and time consuming. Image-based modeling systems have been implemented and are available, such as *Façade* [Debevec et al. 1996], [Photomodeler], and [Realviz], but recovering complex geometry, such as people, plants, and non-rectilinear and curved geometric objects, using these systems can prove challenging. Second, applying an inverse lighting simulation is not only a difficult implementation task, but also time consuming and difficult to validate. It also requires full 360×180 degree information of the captured environment to compute the direct and indirect lighting. Furthermore, in many real-world scenes, the geometry and lighting may be too complex to estimate accurately. Inverse simulations require not only precise geometric modeling, but also precise modeling of the light sources, i.e. their distribution, color, intensity, and geometry. The accuracy of the recovered reflectance depends on the precision of the modeled geometry, light sources, and the recovered camera pose. Finally, at lighting discontinuities, e.g. shadow or highlight boundaries, artifacts may occur due to a series of possible errors — from camera misregistration to geometry approximations. If there are sharp highlights or shadows present in the scene, the precision of the predetermined information is paramount to correctly separating the reflectance from the illumination. Loscos et al. use texture synthesis to remove artifacts along shadow boundaries, but still require an initial physical simulation [Loscos et al. 2000].

In contrast, our filter requires far less preprocessing time, and produces convincing results much more rapidly, without requiring full knowledge of the environment — even with a single photograph. There are many advantages and possibilities in trading physical accuracy for fast, intuitive, and visually convincing results. We further discuss this issue in Section 6.5.

6.2 Small- and Large-Scale Feature Separation

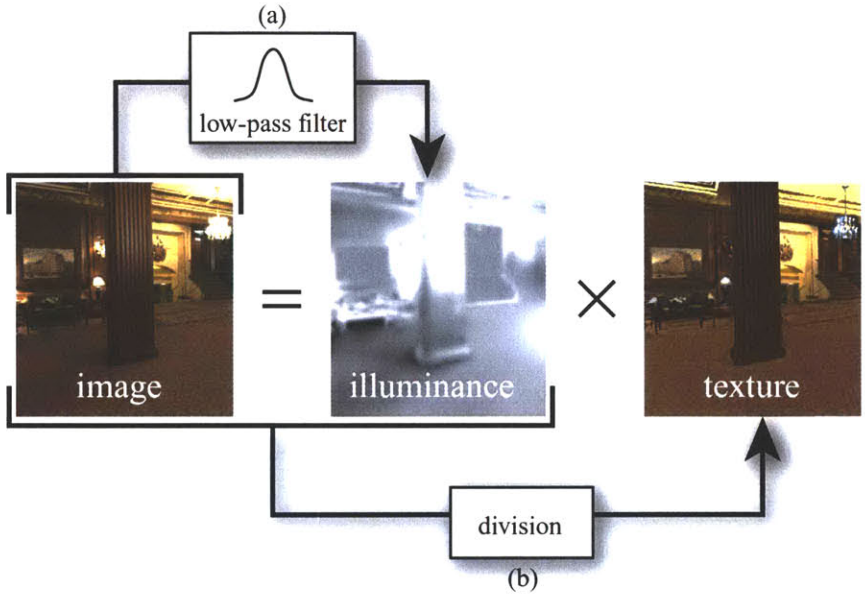


Figure 6-3: Basic idea. (a) Low-pass filtering of the initial input image to obtain the illuminance channel. (b) Division of color and illuminance channels to compute the texture channel.

To decouple the texture and illuminance channels from the input color channel, we make the following simple assumption: Large-scale illuminance variations are due to the lighting, which changes slowly over a surface; small-scale details are due to the texture, which varies more rapidly. The basic idea is to first use a low-pass Gaussian filter to blur the initial input image to obtain the illuminance channel, and then separate the texture channel from the initial and illuminance images (Figure 6-3).

The basis of our decoupling filter idea is similar to the retinex theory introduced by Land and McCann [1971]. The human visual system has an attribute called the *lightness constancy*, where a human subject can perceive the sameness of reflectances under different illuminations. For instance, we can tell a difference between a light surface in dim light versus a dark surface in bright light,

although photometer outputs may be the same. From this, they argue that reflectance from an observer's viewpoint tends to be constant across space except at the abrupt transitions between objects and materials, and illuminance tends to change gradually over space. From this argument, one can separate the reflectance and illuminance by simply computing the spatial derivatives: high derivatives are from reflectance and low derivatives are from illuminance. This technique is often used for dynamic range compression [Jobson et al. 1997], for displaying high dynamic-range images on low dynamic-range display devices (e.g. computer monitors), sharpening images, and for color constancy, i.e. removing illumination from the object in the image.

Our basic idea is also related to homomorphic filtering [Lim 1990] (*pp.* 463-465), where illuminance and reflectance are separated using a similar technique. A low-pass and high-pass filtering on the input image in log intensity domain is used to separate the reflectance and illuminance images.

The low-pass Gaussian filter is defined as follows:

$$I_1(p) = \frac{1}{k(p)} \int_0^\infty \mathcal{G}_s(p,p')\ I_0(p')\ dp', \tag{6.1}$$

where I_0 is the input image, p and p' are pixel locations, $\mathcal{G}_s$ is the Gaussian function in the spatial domain, and I_1 is the resulting Gaussian-blurred image. The normalization factor for the Gaussian function, $k(p)$, is defined as

$$k(p) = \int_0^\infty \mathcal{G}_s(p,p')\ dp'. \tag{6.2}$$

The Gaussian function, $\mathcal{G}_s$, is radially symmetric in image space:

$$\mathcal{G}_s(p,p') = e^{-\frac{1}{2}\left(\frac{\|p'-p\|}{\sigma_s}\right)^2}. \tag{6.3}$$

$\mathcal{G}_s$ applies a convolution in the *spatial* domain, since the difference $\|p' - p\|$ depends on the Euclidean distance between the two pixels. The geometric spread, σ_s, determines the degree of low-pass filtering applied to the image — a large σ_s blurs the image more, and vice versa.

We illustrate the basic idea in Figure 6-4, where we separate the lighting effects on the carpet (the illuminance channel) from the carpet pattern (the texture channel). First, the user interactively specifies a *feature size* in the input image by drawing a line segment over the pattern in the image. The feature size determines the scale at which small- and large-scale features are separated, and it is directly proportional to the geometric spread, σ_s. In practice, we use *feature size* $= \sigma_s$. As shown in Figure 6-4(a), the specified feature size is the size of the repeated pattern on the carpet. Directly

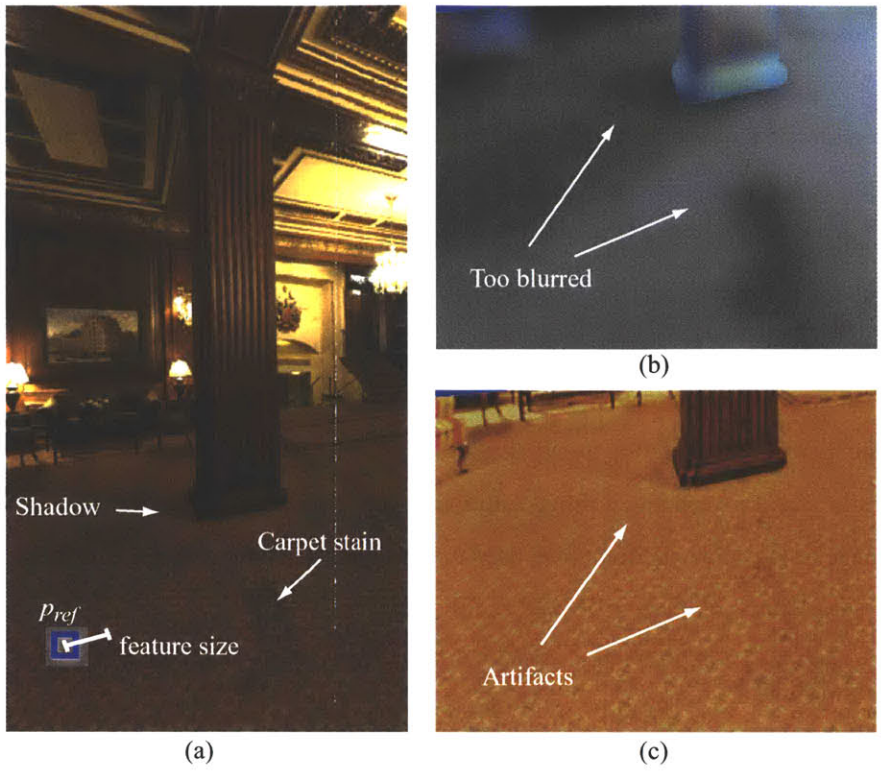
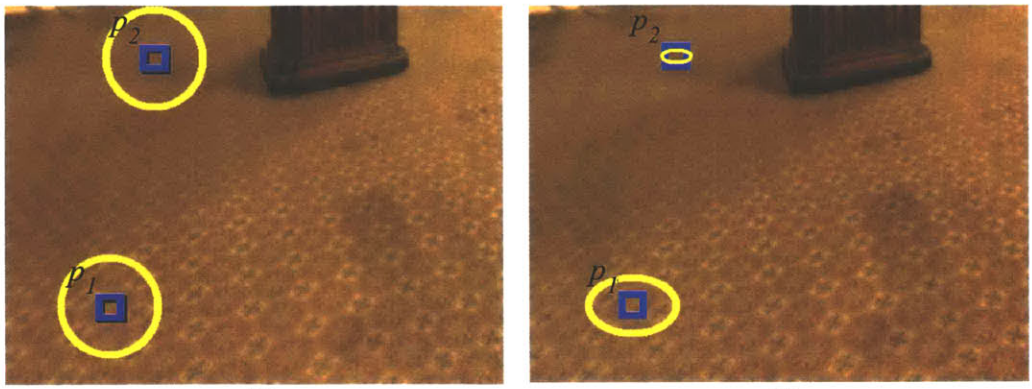


Figure 6-4: Texture-illuminance decoupling. (a) Input image with user-specified feature size and the reference pixel, p_{ref}. The left arrow points to a shadow from the column, and the right arrow points to a carpet stain. (b) Initial illuminance estimation using simple Gaussian filtering. (c) Initial texture estimation. Note the artifacts corresponding to boundaries from both the shadow and the carpet stain.

specifying the feature size in the image to separate the large- and small-scale features provides an intuitive control of the filter. We use the Gaussian filter (Equation 6.1) to compute the illuminance channel, I_1 (Figure 6-4(b)). We assume that the average color comes from the texture component, so we divide the illuminance obtained by the normalized average color value, $k(p_{ref})$, where p_{ref} is the initial pixel selected by the user while specifying the feature size (Figure 6-4(a)). A uniform texture component is then factored out by dividing the input image with the blurred image (Figure 6-4(c)).

This approach works well if the illumination varies slowly in the given input image. But, in general, our basic idea has two shortcomings. First, perspective foreshortening and the scene geometry need to be taken into account to make consistent use of the feature size. A radially-symmetric Gaussian kernel in 2D image space does not reflect the actual geometry of the scene — the feature size should be consistent in world space, not in image space, and the kernel shape should comply



(a) (b)

Figure 6-5: Depth correction. (a) Convolution kernel of a Gaussian filter (in yellow circle) for pixels p_1 and p_2 (in blue box) in image space. (b) An ellipse is used as the kernel depending on distance and orientation of the surface relative to the reference camera.

with the object orientation in the scene. We can exploit the depth information to blur the image according to the scene geometry. The second shortcoming occurs at illumination discontinuities, where haloing artifacts occur (Figures 6-4(c)). The shadow boundaries introduce frequencies that are in the scale of the feature size, so they are treated as texture frequencies by the Gaussian blur function.

In the next two sections, we address these mentioned limitations. First, we compute a more accurate kernel size and shape for the Gaussian function using the depth channel (Section 6.3). Second, we use a non-linear edge-preserving filter to handle the problems at lighting discontinuities (Section 6.4).

6.3 Depth Correction

Due to perspective foreshortening and surface orientation of the objects in the scene, the convolution kernel size of the Gaussian filter should not be constant in the image space, and its shape should not be radially symmetric. However, by definition, the Gaussian filter applies the same radially-symmetric convolution kernel throughout the pixels to blur the image, since it does not have the geometry information to take the depicted 3D scene into consideration. As illustrated in Figure 6-5(a), although pixel p_2 is further away than p_1, their kernel shapes and sizes are the same. This results in pixel p_2 being blurred more than the specified feature size. As shown in Figure 6-5(b), we want the kernel shape and size to conform to the orientation of the geometry, as well as to

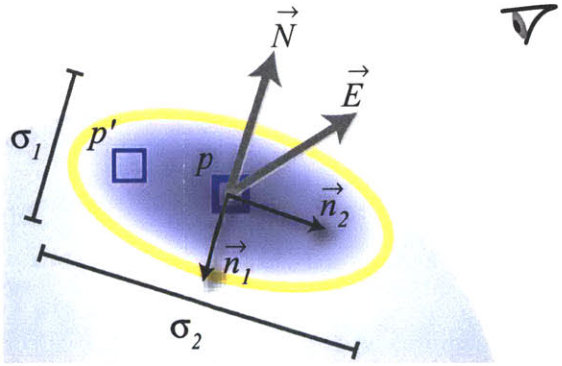


Figure 6-6: Depth correction of the spatial Gaussian using an ellipse.

compensate for the perspective foreshortening.

We use the depth channel to scale the size of the kernel to handle foreshortening, and normals to compute an elliptical kernel shape to handle orientation, similar to the Elliptical Weighted Average filter [Heckbert 1989]. The user specifies a feature size at the reference pixel p_{ref}, with depth z_{ref}. The spatial kernel size for other pixels is then scaled by $\frac{z_{ref}}{z}$. To compensate for orientation, we use a locally-planar approximation of the surface: Gaussian ellipses orthogonally oriented to the surface normal (Figure 6-5(b)). The normals are computed from the neighboring pixels in world space.

As shown in Figure 6-6, let $\vec{N}$ be the unit normal and $\vec{E}$ the unit viewing direction in world space. The small axis of the ellipse is along $\vec{n_1}$, which is a 2D unit vector of $\vec{N}$ projected onto the image plane. Vector $\vec{n_2}$ is the long axis of the ellipse, which is a unit vector orthogonal to $\vec{n_2}$. The small-to-large ratio $\frac{\sigma_2}{\sigma_1}$ is given by the dot product $\vec{N} \cdot \vec{E}$, where $\sigma_1 = \frac{z_{ref}}{z(p)}\sigma_s$ as described above. We then have:

$$\mathcal{G}_s(p', p, \sigma_s) = \mathcal{G}_1(\vec{pp'} \cdot \vec{n_1}, \sigma_1)\mathcal{G}_2(\vec{pp'} \cdot \vec{n_2}, \sigma_2). \tag{6.4}$$

The Gaussian function in the spatial domain, $\mathcal{G}_s$, now compensates for perspective foreshortening by scaling its kernel size according to z_{ref}, and handles surface orientation using the ellipse.

6.4 Edge-Preserving Filter

In the previous section, we have solved the first problem of perspective foreshortening using the depth-corrected Gaussian kernel. In this section, we deal with the second problem: haloing artifacts that occur at lighting discontinuities. If sharp illumination discontinuities, such as shadow bound-

aries, are present in the input image, the basic idea introduces frequencies that are in the scale of the feature size, and are incorrectly decoupled as the texture channel. We use a non-linear edge-preserving filter to address this problem. This problem is related to image denoising, but the "noise" in this case is the texture information that we want to retrieve. Edge-preserving anisotropic diffusion could be an option [Perona and Malik 1990], but since we have information about the feature size, we prefer to use bilateral filtering introduced by Tomasi and Manduchi [1998], which is more controllable. The key idea is to use an additional Gaussian in the *intensity* domain, also known as the *range* domain, to convolve the pixels of similar intensity. Therefore, the Gaussian kernel now considers the product of the spatial and intensity domains.

6.4.1 Bilateral Filter

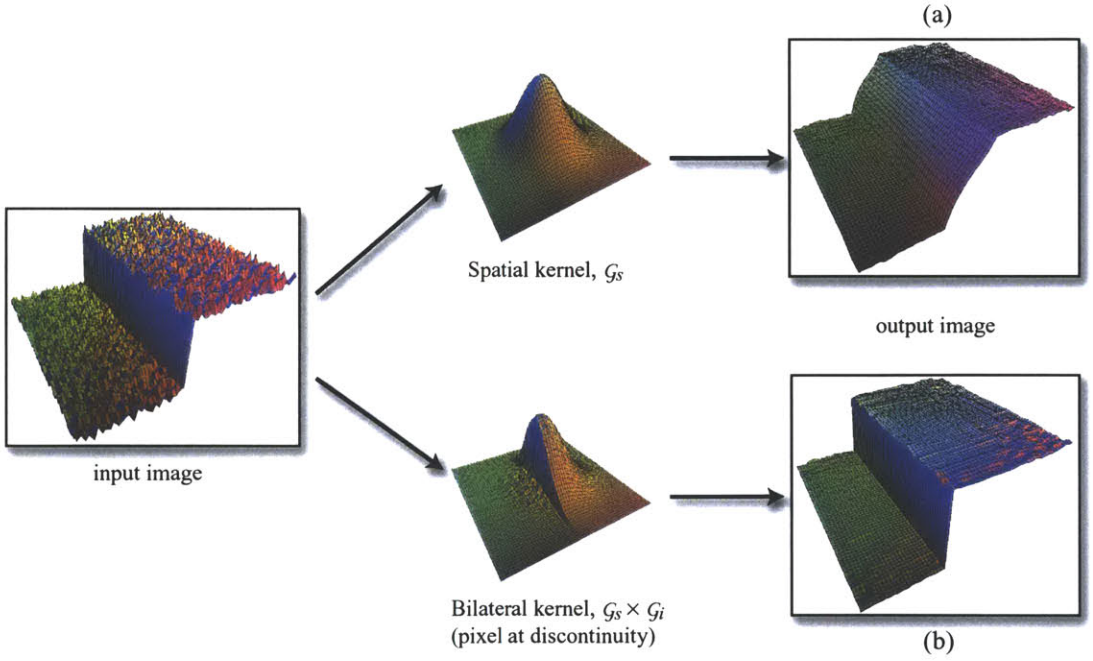


Figure 6-7: (a) Gaussian filter with spatial kernel, G_s. Note that the discontinuity from the input image is not preserved. (b) Bilateral filter with spatial and intensity kernels, $G_s \times G_i$. The discontinuity is preserved while the high frequencies are blurred. Images courtesy of Frédo Durand.

The general idea of the bilateral filter, as presented by Tomasi and Manduchi, is to not only consider the geometric closeness of a Gaussian kernel in the spatial domain, but also the similarity in the intensity domain [Tomasi and Manduchi 1998] (Figure 6-7).

The bilateral filter is defined as follows:

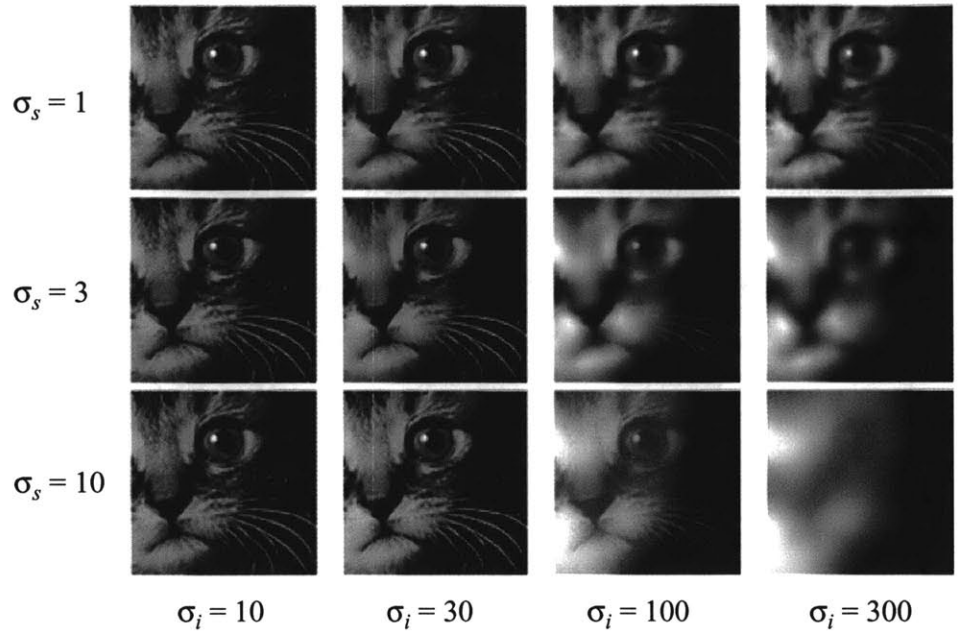


Figure 6-8: Bilateral filtering example with various spatial and intensity parameter values. Courtesy of Carlo Tomasi.

$$I_1(p) = \frac{1}{k(p)} \int_0^\infty \mathcal{G}_s(p,p') \;\; \mathcal{G}_i(I_0(p), I_0(p')) \;\; I_0(p') \;\; dp' \tag{6.5}$$

where $\mathcal{G}_s$ and $\mathcal{G}_i$ are spatial and intensity Gaussian kernels, respectively. The normalization, $k(p)$, is defined by

$$k(p) = \int_0^\infty \mathcal{G}_s(p,p') \;\; \mathcal{G}_i(I_0(p), I_0(p')) \;\; dp'. \tag{6.6}$$

The difference between Equations 6.1 and 6.5 is the additional intensity Gaussian, $\mathcal{G}_i$, which is defined as follows:

$$\mathcal{G}_i(I_0(p), I_0(p')) = e^{-\frac{1}{2}\left(\frac{\|I(p)-I(p')\|}{\sigma_i}\right)^2}. \tag{6.7}$$

Tomasi and Manduchi define the scalar difference between two colors, $\|I(p) - I(p')\|$, in CIE-Lab color space [Wyszecki and Styles 1982], where the Euclidean distance measures the perceptual similarity of colors. An example of the bilateral filter is shown in Figure 6-8.

6.4.2 Our Approach

We adapt the bilateral filter for the purpose of decoupling texture and illuminance channels. Our filter is iterative, and the initial Gaussian estimate of the illuminance, I_1, is used to drive the intensity Gaussian:

$$I_{j+1}(p) = \frac{1}{k(p)} \int_0^\infty \mathcal{G}_s(p, p', \sigma_s)\ \mathcal{G}_i(I_j(p), I_j(p'))\ I_0(p')\ dp'. \tag{6.8}$$

Note the combination of the spatial Gaussian, $\mathcal{G}_s$, where the depth correction takes place, and the intensity Gaussian, $\mathcal{G}_i$, produces the proper results for lighting discontinuities in the illuminance channel.

The main difference between this approach and standard bilateral filtering is that we always filter the initial image I_0. Unlike denoising approaches, we are interested in factoring the initial image, not in removing noise. Because the convolution kernel averages only pixels of similar estimated illuminance, the filter captures shadow boundaries (Figure 6-11). The process converges quickly, and we use I_3 as the final illuminance estimate. The only hard-coded parameter is the variance of the intensity Gaussian. In practice, we have found that $\sigma_r = 0.01 * max(I_1)$ provides good results. Figure 6-9 illustrates an overview of our method.

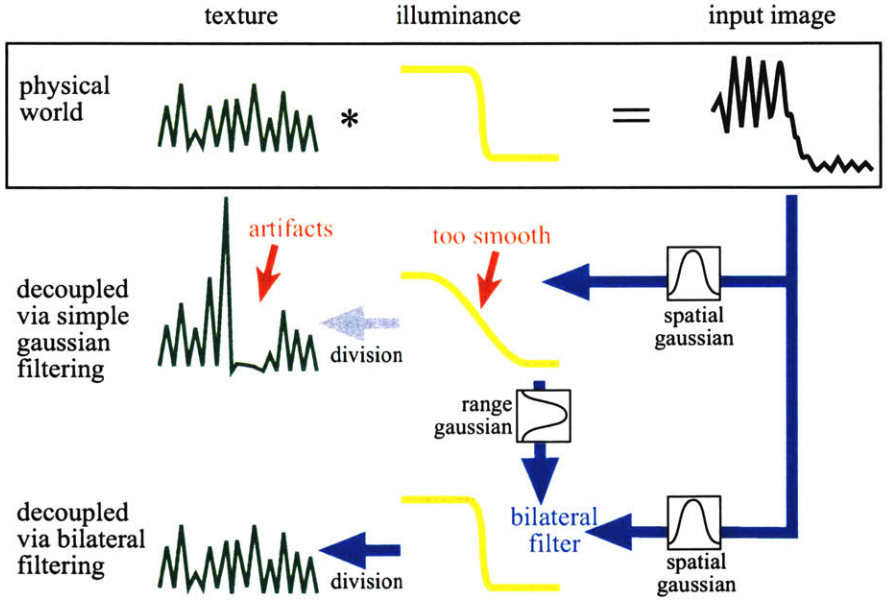


Figure 6-9: Texture-illuminance decoupling in 1D. The example exhibits sharp illumination variations, which cannot be captured using simple Gaussian filtering. This first pass is, however, used for the bilateral filtering pass, to average only pixels with similar illumination, due to the use of an additional intensity Gaussian.

6.5 Examples and Discussion

6.5.1 Examples

In this section, we show two examples that illustrate the application of the decoupling filter. The first is an interior scene of a hotel lobby (Figures 6-10 to 6-13). In Figure 6-10, we show a simplified scene with only two layers displayed, the column and the carpet (the full example is shown in Figure 6-13). Figures 6-10(b) and (c) show the first iteration, I_1, where only the depth-corrected Gaussian was applied. The shadow boundaries from the left column are still too blurred in the illuminance channel, so the texture channel still shows haloing artifacts. Figures 6-10(d) and (e) show the results from I_3, where the bilateral filtering was applied and the resulting shadow boundaries are sharper. The closeup is shown in Figure 6-11, where the resulting texture channel significantly reduces the artifacts and is more even. The filter took about 15 seconds for this particular example, which consists of a 512×300 resolution image. Figure 6-12(a) shows an example where the texture channel has been modified while maintaining the existing lighting — we used our clone brush to edit the floor texture. Figure 6-12(b) shows the editing of the illuminance channel by placing spot lights. We implemented some light source tools for the system, such as spot lights, point lights, and general IES light sources [of North America 2000], that modify the decoupled illuminance channel. More results are shown in Figure 6-13, showing the relighting results of different parts of the hotel lobby.

The second example involves relighting a church façade at night (Figure 6-14). We removed the spot lights on the roof and applied up lights and point light sources on the turrets of the church. The decoupling of the roof and the turret layers took about 10 minutes (including all the interactions), and modeling and applying of the light sources took about 20 minutes of editing. We also made subtle changes to the flood lights applied on the church walls, and painted lights on the windows.

6.5.2 Discussion

In this chapter, we have discussed our non-linear filter that decouples texture and illuminance channels from the input image. Our basic idea is to first use a low-pass filter to compute the illuminance channel, then divide the input and illuminance images to compute the texture channel. From this basic idea, we further improve the filter by exploiting the depth channel to correct the convolution kernel according to the geometry and perspective foreshortening of the scene, and use bilateral filtering to handle illumination discontinuities.

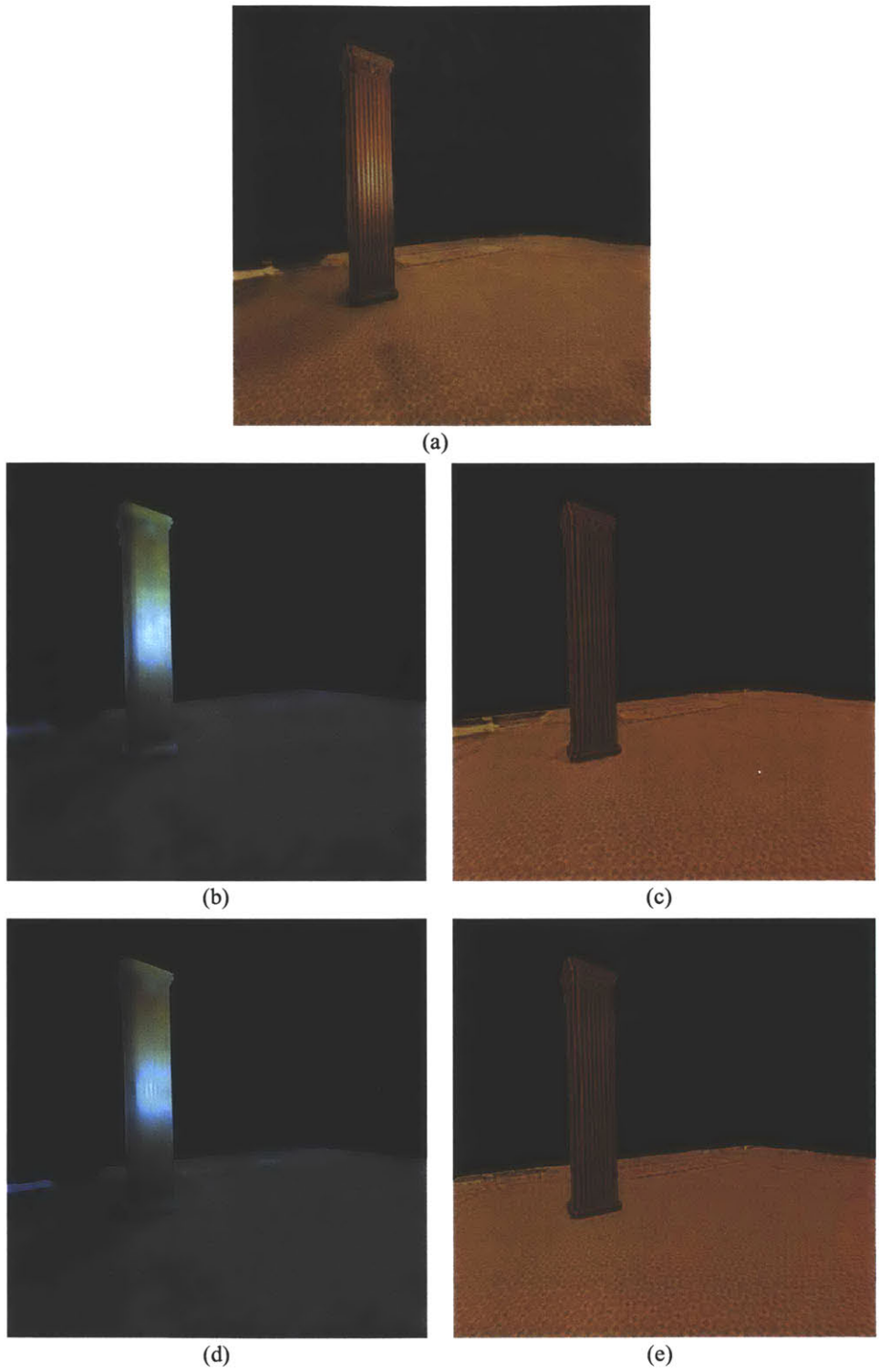


Figure 6-10: Decoupling of column and floor. (a) Initial input image. (b) Illuminance channel from I_1. (c) Texture channel from I_1. (d) Illuminance channel from I_3. (e) Texture channel from I_3.

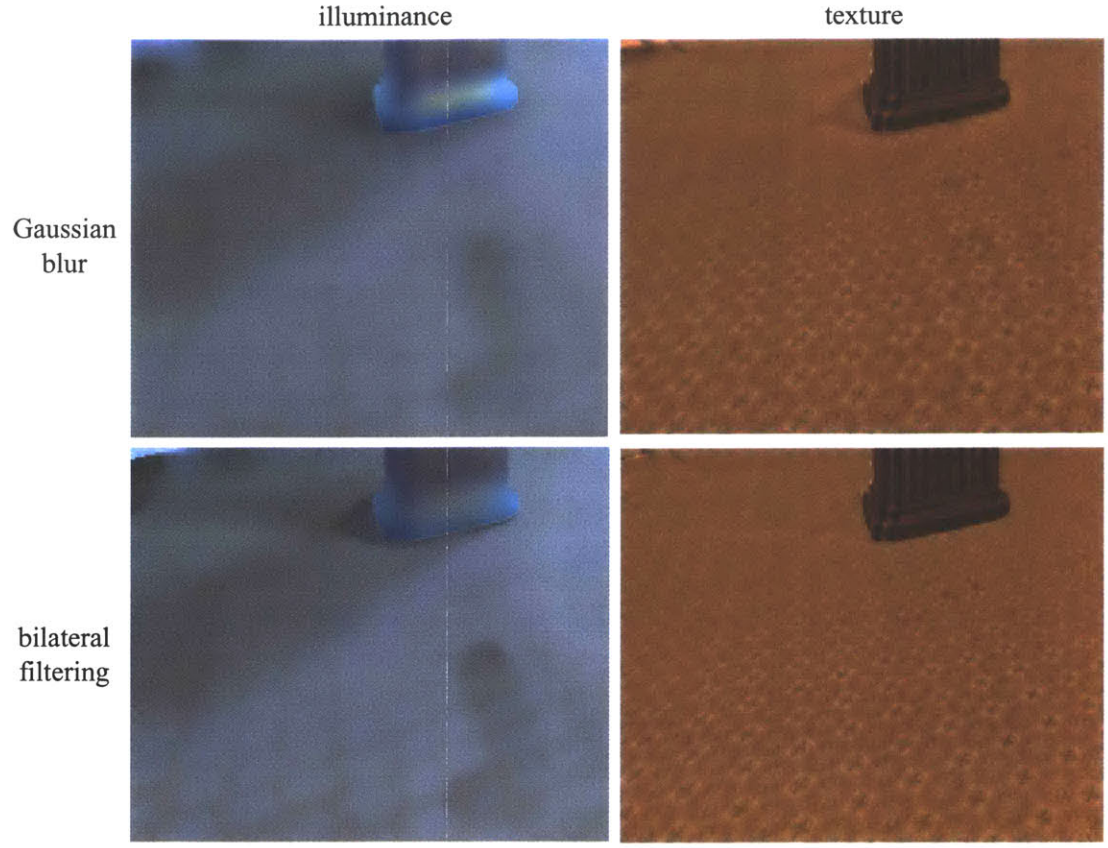


Figure 6-11: Closeup of the comparison between a Gaussian filter and our decoupling filtering. Note that our filter results in a more even texture due to sharper shadow boundaries in the illuminance channel.

There are many advantages in using our decoupling filter. First, because it is a non-linear image-processing filter, the filter does not require precise fully-enclosed 3D geometry or the light source information. In many applications, this is a big advantage since it does not require the costly pre-processing time and effort. Second, the filter is easy to implement, and it is much faster than physically-based inverse simulation. It produces visually convincing results, and does not employ optimization techniques that may not converge. Third, it resolves problematic haloing artifacts that occur at shadow boundaries via the edge-preserving filter. Finally, the two images produced by the decoupling filter are easy to edit, either to modify the lighting or the material.

The filter can easily be extended to a purely 2D photo-editing context. Equation 6.8 may be used without the depth correction, and the user can assign more weight to the intensity Gaussian for finer control. It is also possible to provide an interface, similar to [Zhang et al. 2001], such that the user can interactively specify the normals and the contours on the 2D image to approximate the depth

(a) (b)

Figure 6-12: (a) Changing of the floor material. (b) Applying spot lights with the light source tool.

correction. Homography can also be used on planar geometry, similar to the structure-preserving clone brush (Chapter 4).

We would also like to further experiment by using an additional Gaussian in the *depth* domain:

$$I_{j+1}(p) = \frac{1}{k(p)} \int_0^\infty \mathcal{G}_s(p,p',\sigma_s)\ \mathcal{G}_i(I_j(p),I_j(p'))\ \mathcal{G}_d(N(p),N(p'))\ I_0(p')\ dp', \qquad (6.9)$$

where $\mathcal{G}_d$ is the depth Gaussian and $N(p)$ is the normal. Currently, we linearly approximate the local geometry around pixel p using its normal, and also assume pixels p' in the kernel have the same geometry. By adding an additional Gaussian to adjust for the difference in geometry at pixels p and p', we can further improve the filter, especially for scenes with curved geometry.

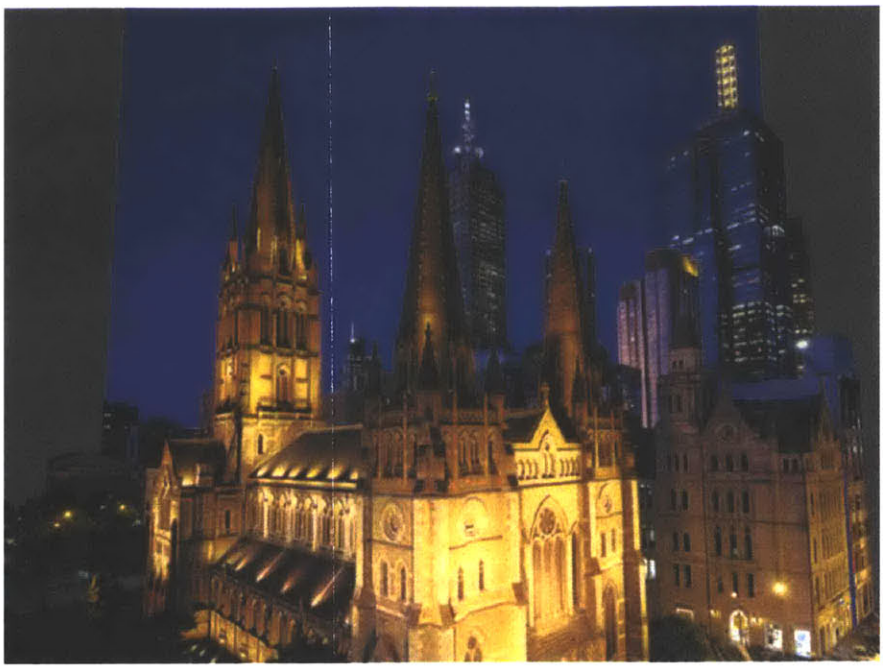
Also, it is possible to generalize the texture-illuminance decoupling to non-uniform textures, in an automatic fashion. The tone and the local texture of an image could then be edited independently, greatly simplifying many photomontage operations.

The real world is highly complex and there are cases where our simple assumption does not hold true — that illumination varies slowly and texture varies more rapidly. Large and slowly varying patterns can be decoupled as illuminance, e.g. Figure 6-4(a) points out a case where the carpet stain (that looks like a shadow) is decoupled as illuminance. Similarly, small and sharp shadows within the texture frequency domain can be decoupled as texture. For example, sharp shadows from the

sun of small patterned objects, such as fences, would be a difficult case. Another limitation arises from the assumption that the texture channel is made of one Gaussian-averaged reflectance value. This is not an issue in cases where the texture patterns and color variations are relatively small in the input image. But in cases where texture patterns are made from large, highly contrasting materials (e.g large checkered patterns) either a large feature size may need to be used, or each pattern may need to be segmented into different layers. In practice, we have found that separating the image into layers, as we do in our image-based representation, lessens the need for further segmentation or user specification.

We do not claim that our filter is a "cure-all" for relighting and related applications. It is important to note that the decoupling filter is a single tool in a larger system. In a typical photo-editing system, there are many ways to produce a desired end result. Our tool, being a part of a similar system, addresses one of the major difficulties and limitations in photo editing, as well as in the image-based editing context. If the goal of the user is recovering a physically accurate reflectance model, our filter would not be appropriate. Yet, we believe that there are many applications for this filter, mainly due to its simplicity and performance speed.

Figure 6-13: Relighting examples. (a) and (c) show the initial images, and (b) and (d) show the relit images.



(a)

(b)

Figure 6-14: Relighting of St. Paul's Church in Melbourne, Australia. (a) Before. (b) After. Courtesy of Barry Webb and Associates.

CHAPTER 7

Results

We demonstrate the flexibility and applicability of our system through the results shown in this chapter. As we have discussed, the simplicity of our representation allows versatile depth assignment, editing operations, and display. Various editing operations similar to those found in 2D photo-editing systems are provided, such as copying and pasting, painting, filtering, selection, and clone brushing. The ability to paint from different viewpoints makes it easy to edit foreshortened surfaces and obtain correct perspective. We show that 3D transformations of objects, such as scaling, translation and rotation, are possible in our system, unlike the 2D systems. The new clone brush tools are used to modify the scene, and the texture-illuminance decoupling filter is used to demonstrate some relighting and design applications. All the examples that we show are in the mega-pixel range.

The most time-consuming part of creating these examples was the precise segmentation process, where we separate the initial input image into layers. Objects in the input image are segmented into separate layers to ease the depth assignment process and editing operations. Much of the segmentation process was done using a semi-automatic tool in Adobe Photoshop [Photoshop]. The user paints around the silhouette boundary of a foreground object with a thick brush, and the tool extracts the

foreground layer by estimating the alpha (transparency) value per pixel within the painted boundary region [Ruzon and Tomasi 2000; Chuang et al. 2001; Berman et al. 2000a; Berman et al. 2000b]. In most cases, this tool was not enough to cleanly segment a layer, mainly due to the complexity of the scene, and much time was spent in further refining the silhouettes. After the segmentation process, the files were imported to our system. The clone brush tools were used to fill in the holes behind the segmented layers, and the depth tools were used to specify the scene geometry. These processes take relatively little time in comparison to the segmentation process.

It is important to stress that the need for segmentation is not unique to our approach. As discussed in Chapter 2, most applications that use photo-editing systems require segmentation of foreground and background objects as a way of organizing the scene for editing. Many design-related applications use photo-editing tools to layer and composite a new scene. Our system can directly import these layered images, and a much more powerful set of editing tools may be applied. The cost of applying depth and editing operators is negligible in comparison to the cost of segmentation.

We enlisted architecture undergraduates to segment, clone brush, and assign depth for some of the results shown in this chapter. After a short tutorial, which took a few minutes for each of the depth assignment and manipulation tools, they were able to experiment and create the shown results. As for the editing itself, we did not rely upon their help, since the modification process was fast, and also since most alterations were performed as the system was being developed.

We demonstrate our system with three types of input images: paintings, single photographs, and panoramas. We first show a painting example to demonstrate the versatility of the depth assignment tools. The depth tools provide the means and the freedom for the user to interpret the shape and marks on the painting. As we will see, although the rules of perspective are not strictly followed in the painting example, the results are still convincing.

Next, we show some results developed from single photographs, where we demonstrate all facets of 3D editing: the shape, material and the illumination of the scene are modified. We demonstrate various depth assignment tools, such as copying and pasting, transforming, and chiselling, that are used to edit the shape of objects in the scene. Relighting applications are demonstrated by first using the texture-illuminance decoupling filter, and then by modifying the illuminance channel. This can be done by either directly painting on the illumination channel or specifying 3D light sources, such as point, spot, and directional lights. We also show seamless and photorealistic editing of materials in the scene. By editing the texture channel, either by using the new clone brush tools or a simple paint brush, the material may be changed while the maintaining the existing illumination,

including soft shadows and highlights.

Finally, we show examples where we use multiple images of a cube-mapped panorama as input. The parallax and occlusion effects that are interactively displayed in our system, which are not possible in typical panorama viewers, dramatically enhance the immersive experience. Our system can greatly improve techniques such as QuickTime VR [Chen 1995; Apple] at a reasonable cost. In addition, enabling the user to edit panoramas has many applications, from interior design, where objects and materials in the scene may be added, removed, or modified, to special effects, where an actor may be placed in various virtual environments.

7.1 Painting by Dali

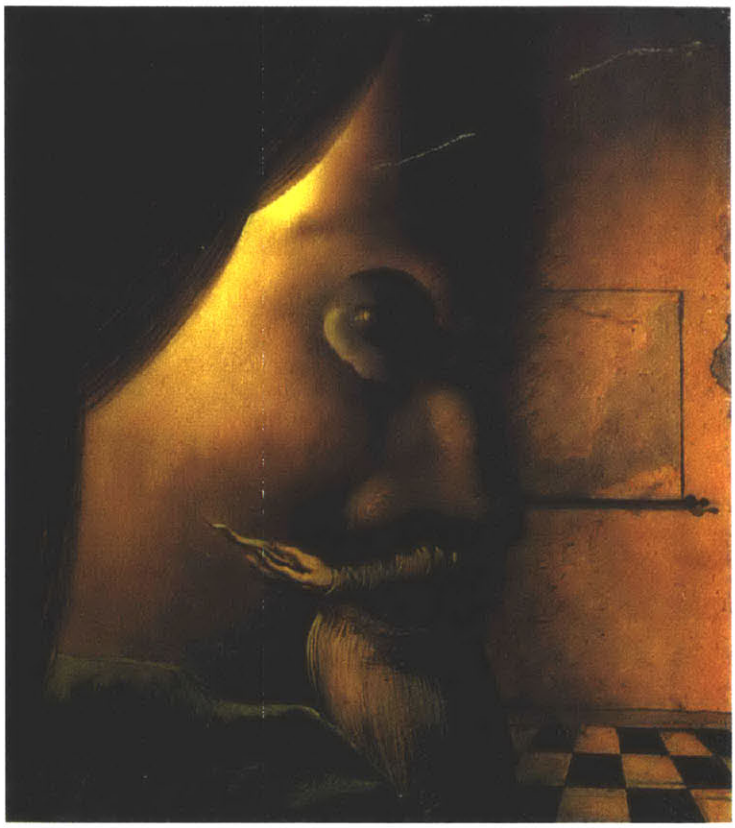


Figure 7-1: A painting by Salvador Dali titled, *The Image Disappears*, 1938.

In this example, we demonstrate the flexibility of our depth tools by experimenting with an ambiguous painting by Salvador Dali (Figure 7-1). The painting has two interpretations: a face or a woman in a room. We show results for both interpretations in Figures 7-2 and 7-3. For the face interpretation, the face template tool was initially used to provide an approximate depth. The depth channel was then refined using the chisel tool and a smoothing filter. It was important to be able to rotate and apply the chisel tool from various angles. A feature that we later added, out of necessity in chiselling the face, was the undo feature. Much of chiselling process was trial and error, where we added or removed depth and needed to view the change from various angles. This is a limitation in user interface, since the only feedback is visual. Using enhanced interface tools, such as the *Phantom* haptic device [Phantom], may improve and ease the depth chiseling process. The refinement process took about two hours (Figures 7-3(a) and (b)).

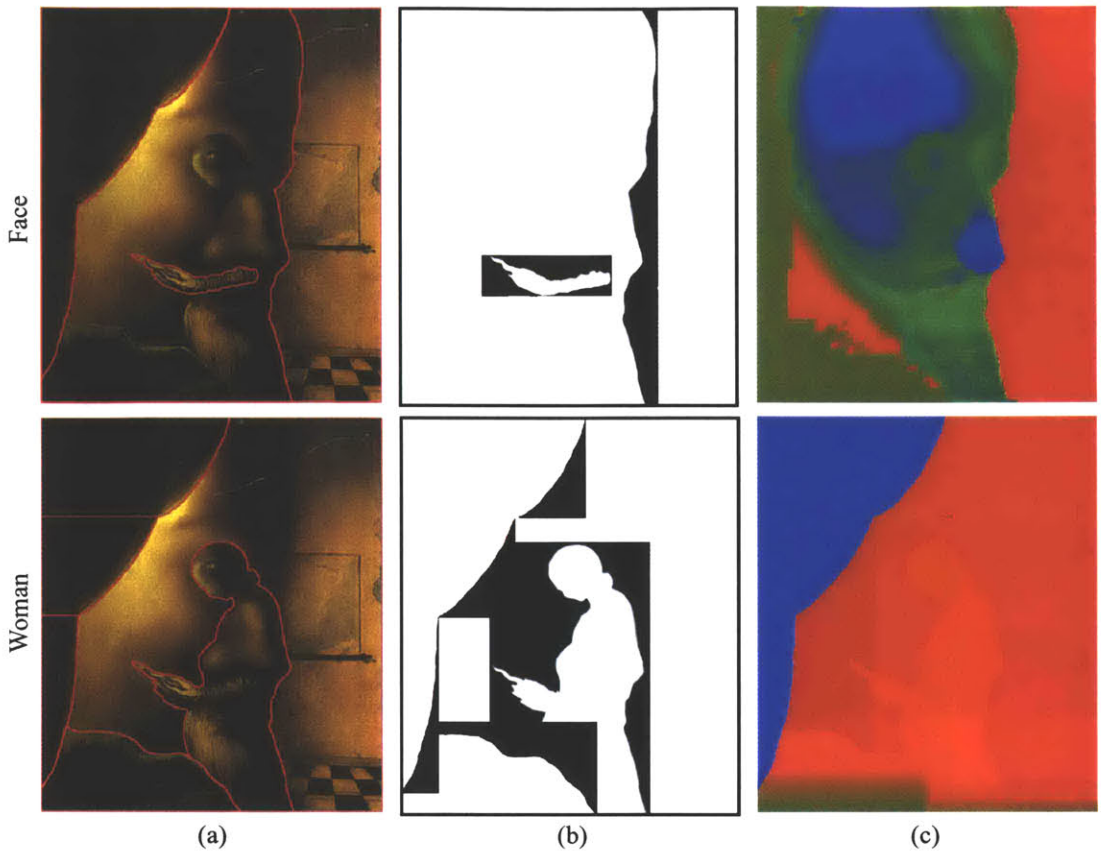
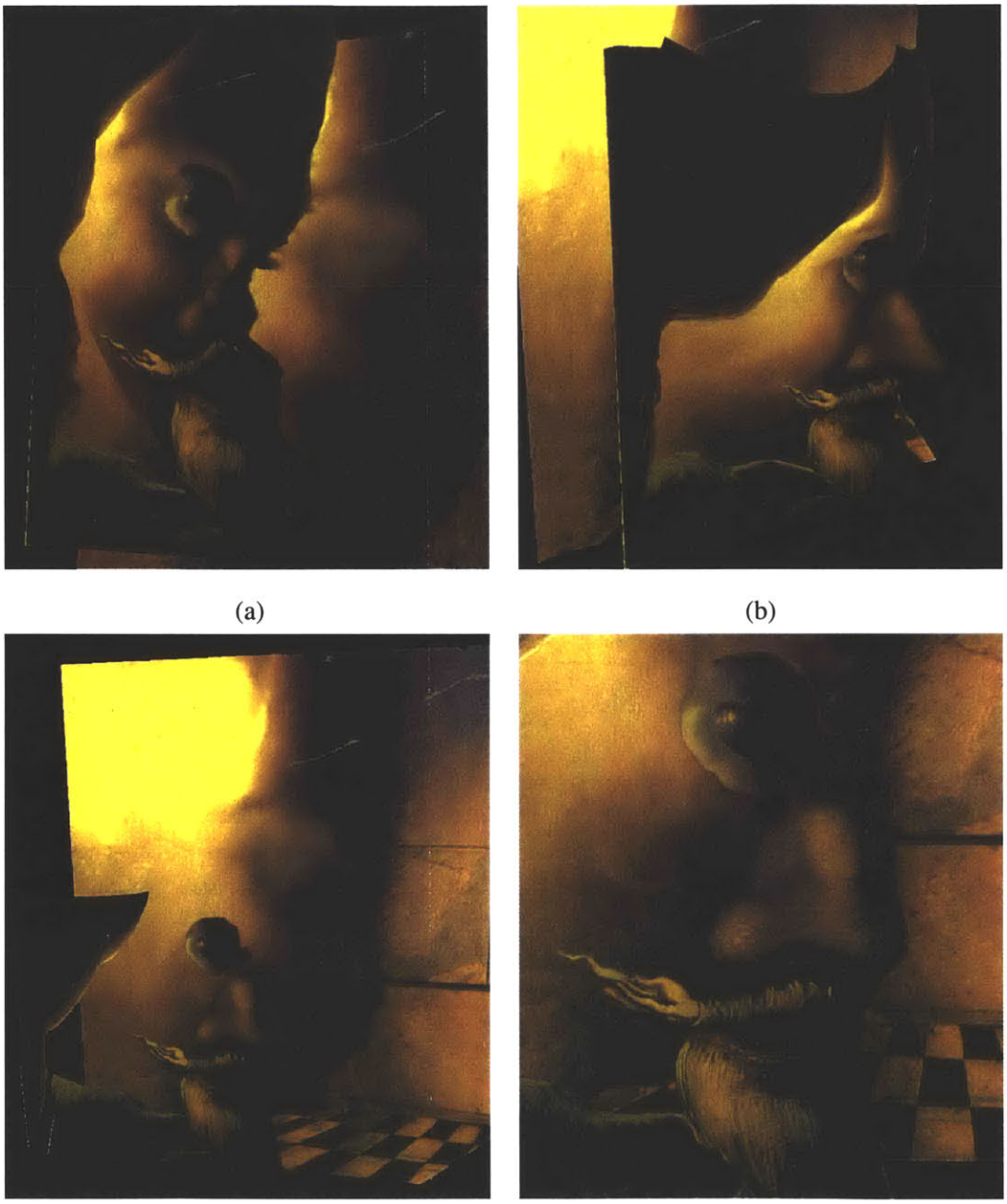


Figure 7-2: The top row shows the "face" interpretation and the bottom row shows the "woman" interpretation. (a) Layers outlined in red. (b) Alpha channel. (c) False color depth channel. The resolution of the initial image is 1000×1124. The face interpretation has four layers; the woman interpretation has six layers. The precise layer segmentation approximately took four hours, and the depth specification took about two hours. Chiselling the face was the most time consuming task.

For the woman interpretation, it only took a few minutes, since the level set tool was used to apply a rounded organic shape (Figures 7-3(c) and (d)), and the chisel tool was used to refine some features. An animation was created that shows smooth transitions from the face to the woman interpretation.

In this example, the ability for the user to specify depth according to his interpretation was crucial. The depth assignment tools provide the means and the freedom to specify these different spatial organizations.



(a) (b)

(c) (d)

Figure 7-3: Different views and close ups. (a),(b) Face. (c),(d) Woman.

7.2 Statue

In this example, the initial photograph is of a statue under a daylight condition. Figure 7-4 shows the initial input image, the segmentation of layers, the alpha channel, and the false-color depth channel. We used the traditional clone brush to fill in the holes behind the statue and the brushes, since a simple planar geometry was specified for the background (Figure 7-5). The level set tool was used to give the statue a bulgy shape, and the face-template tool was used to further refine the head. A simple cylinder tool was used for the base of the statue, even though it is not actually cylindrical. This simplification permitted fast depth specification while yielding a compelling result. The depth assignment process for this example took about five minutes.

In Figure 7-6, we demonstrate a relighting application, where we applied the texture-illuminance decoupling filter to change the initial photograph from day to night. Figures (a) and (c) show the illuminance and texture channels, respectively, decoupled from the initial image. Figure (b) shows the modified illuminance channel, and (d) show the combined image with the new lighting condition. Even though the decoupled channels are not accurate, the approximate separation enables the user to convincingly and rapidly modify the lighting.

To relight the scene, we used spotlights on the illuminance channel of the statue and its base. Parts of the illuminance channel was painted, such as the neck area of the statue, to remove the shadow. Although the geometry of the both the statue and its base are approximate, the relighting results are still convincing. To change the background into a night scene, we applied a simple contrast filter to darken the sky and the tree layers. The whole process took about 30 minutes, including the time for experimentation.

Finally, Figure 7-7 shows the scene after further editing with the chisel tool on the base of the statue, where we engraved the initials of the alma mater. The editing process consisted of applying simple brush strokes on the depth channel, where the modifications are immediately and interactively displayed. The little statue next to the statue base has been copied and pasted, scaled, and rotated, and has also been relit using a simple point light at the base of the statue. The subtle shadow behind the little statue has been added by painting on the illuminance channel. The entire chiselling and copying and pasting operations took a few minutes.

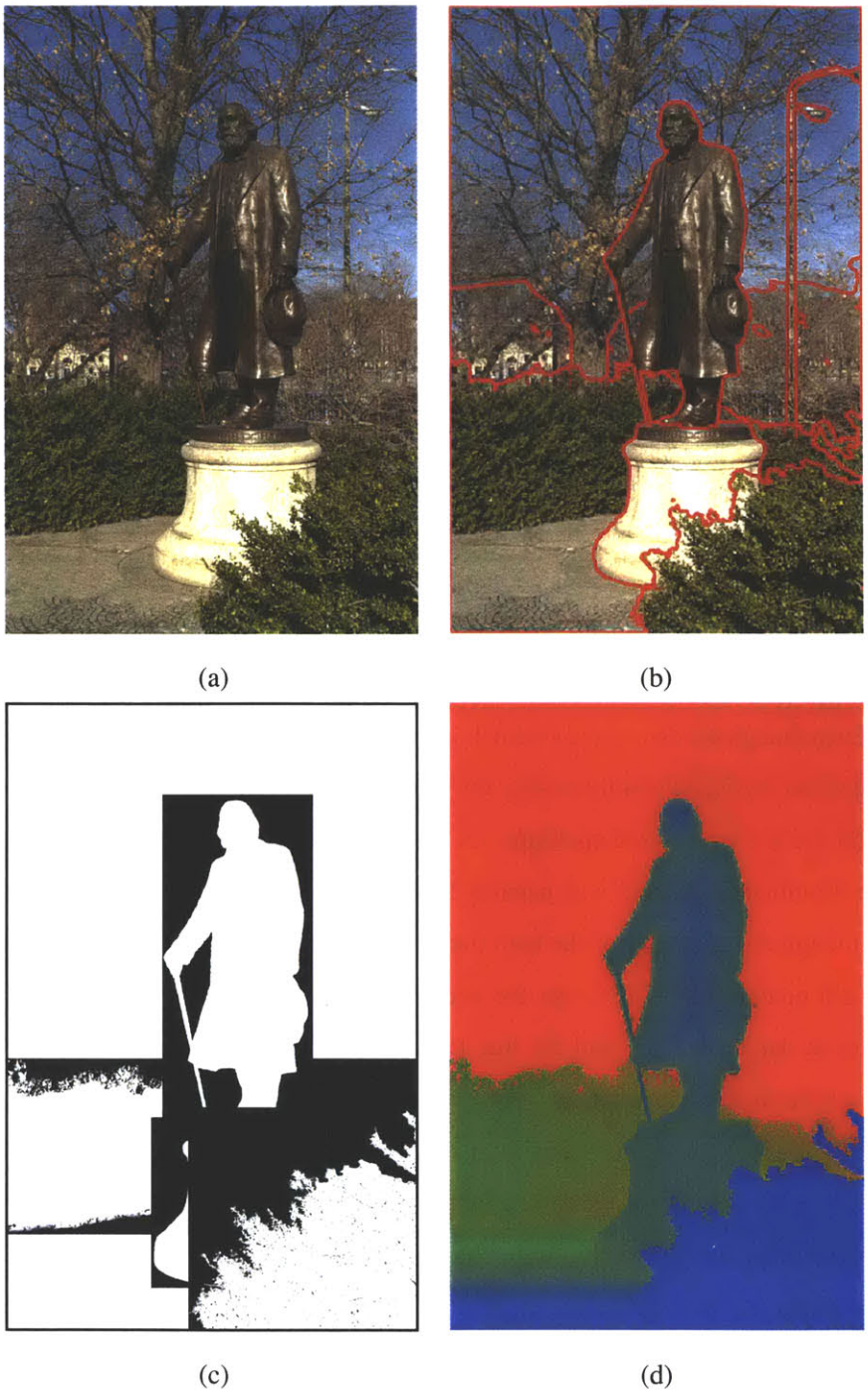


Figure 7-4: A statue. (a) Initial input image. (b) Layers. (c) Alpha channel. (d) False color depth channel. The resolution of the initial image is 1000×1500. There are 6 layers and the precise segmentation process took about 2 hours. Specifying the depth took about 5 minutes.

(a) (b)

Figure 7-5: (a) Side view. (b) Top view.

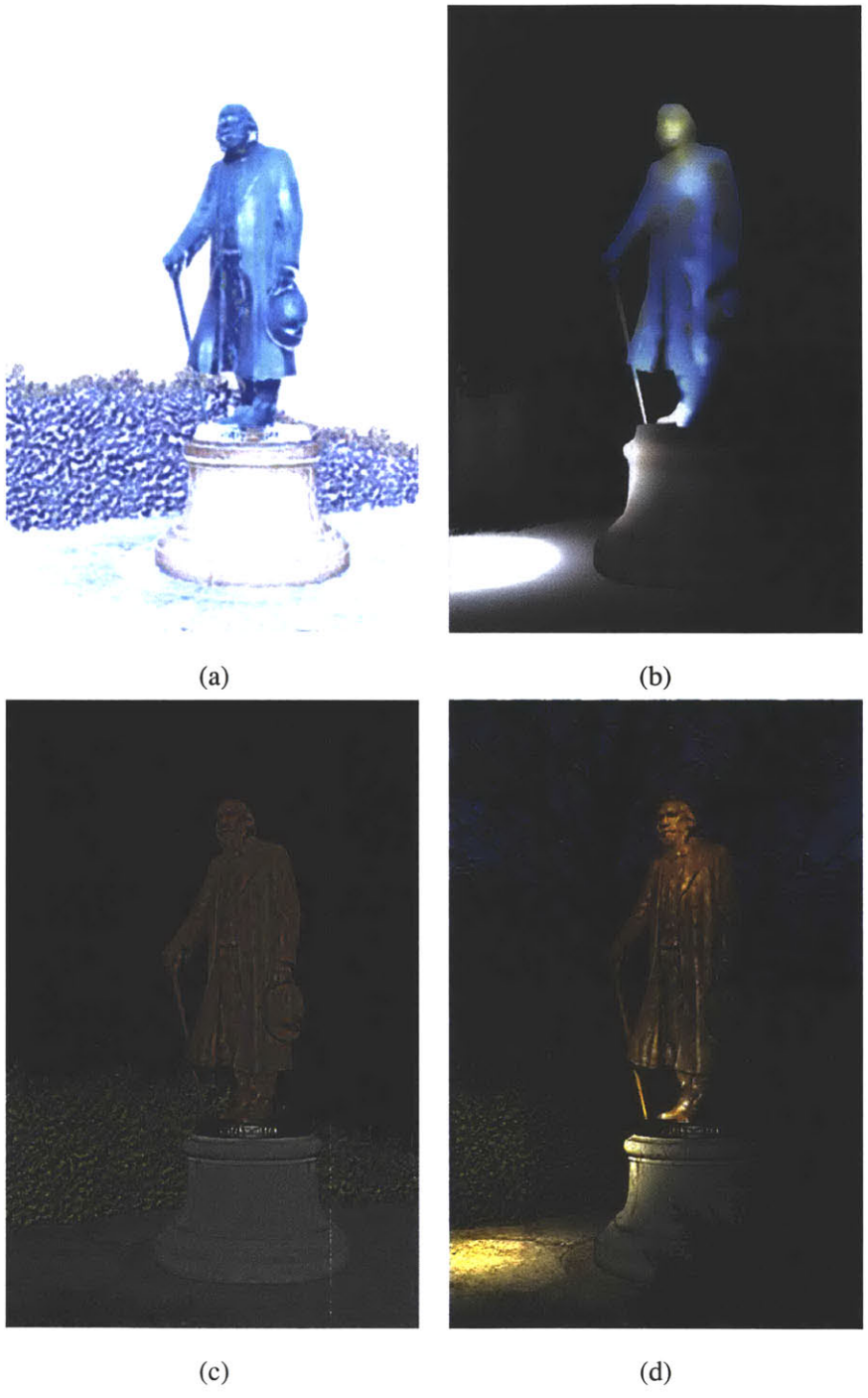
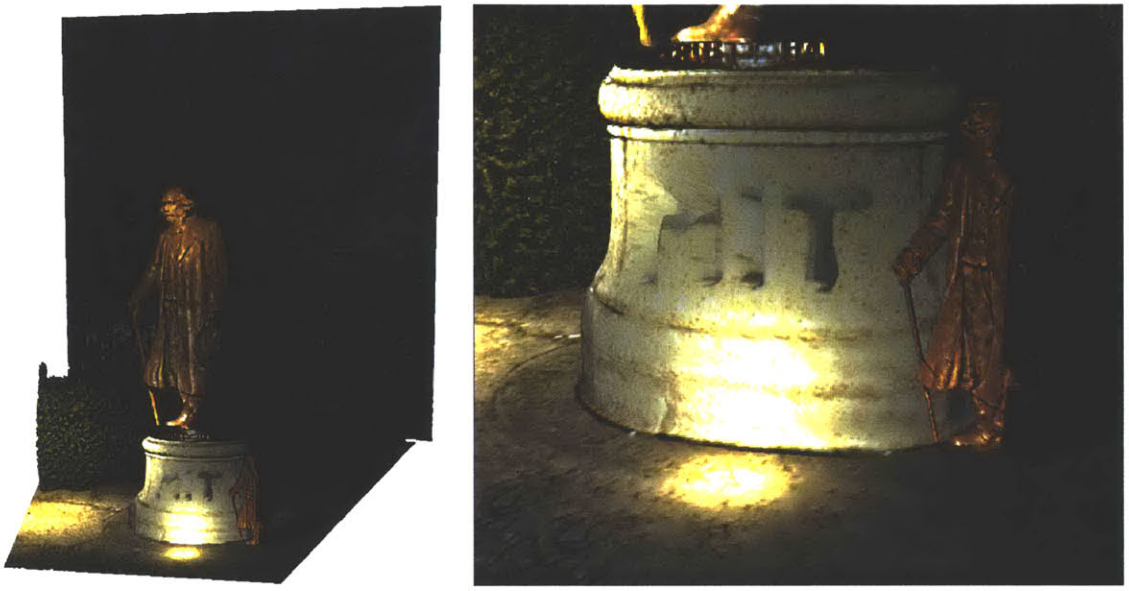


Figure 7-6: Relighting example. (a) Initial illuminance channel. (b) Edited illuminance channel. (c) Texture channel. (d) Combined color channel ((b)×(c)). Decoupling the texture and illuminance channels for the layers shown here took a few minutes. We applied an upward-pointing spot light on the statue, and also a second spot light on the face. We then applied a third light pointing downward on the left near the bush.



(a) (b)

Figure 7-7: (a) The statue has been copied and pasted, scaled, and rotated, and the base of the statue has been chiselled. (b) Close-up view.

7.3 St. Paul's Church in Melbourne, Australia

In this example, we edited an image of the St. Paul's Church, which is a landmark in Melbourne, Australia (Figure 7-8(a)). We use this example to demonstrate the use of our system in design-related applications, such as lighting, urban, and architectural design.

The depth assignment process, as shown in Figure 7-9, took about ten minutes for the coarse geometry, and the following depth refinement process took about two hours, mostly due to experimentation. As shown in the bird's-eye views in Figure 7-10, most of the user time and effort in depth assignment was focused on the church, and coarse geometry was used for the rest of the scene.

Image-based modeling techniques, such as [Debevec et al. 1996; Canoma; Photomodeler], that use polygonal primitives as elements may be used to build a 3D model of the church from photographs, but it would be difficult to model each intricate detail of the church, or the organic shapes of trees, statues, and other street furniture. However, combining image-based modeling techniques with our approach may offer promising possibilities. For example, acquiring a coarse but a more precise geometry from multiple input photographs could provide a more accurate basis for the refinement of the depth channel, especially for architectural scenes. Our layer representation can capture and display the small details shown in the silhouettes of the turrets (Figure 7-11).

To model such a scene using a traditional 3D modeling system, such as [AliasWavefront; Discreet], would take much more time and effort. Modeling the details of the church and the complexity of the existing urban context, and rendering complex night time scene with complex lighting effects represents a difficult challenge for today's systems. It would require an extensive manual acquisition and modeling process, as well as computational resources and techniques to render the scene.

In Figure 7-12(a), we illustrate a simple urban design scenario, where we change the road on the left into a park by copying and pasting and transforming existing trees. In photo-editing systems, the user has to painstakingly and manually apply 2D transformations in order to provide the illusion of 3D modifications. For instance, to place a tree further away, it is scaled down and repositioned on the 2D image. Note that segmentation is also necessary for such applications that use 2D editing systems. Depth assignment is negligible in our system, and once it is accomplished, we are then able to scale, rotate, and translate the trees and the buildings in the background correctly in 3D space. Correct perspective foreshortening of objects from the applied transformations is implicit and automatic to the user.

In Figure 7-12(b), we further edit the scene by modifying the cathedral and the skyline. We

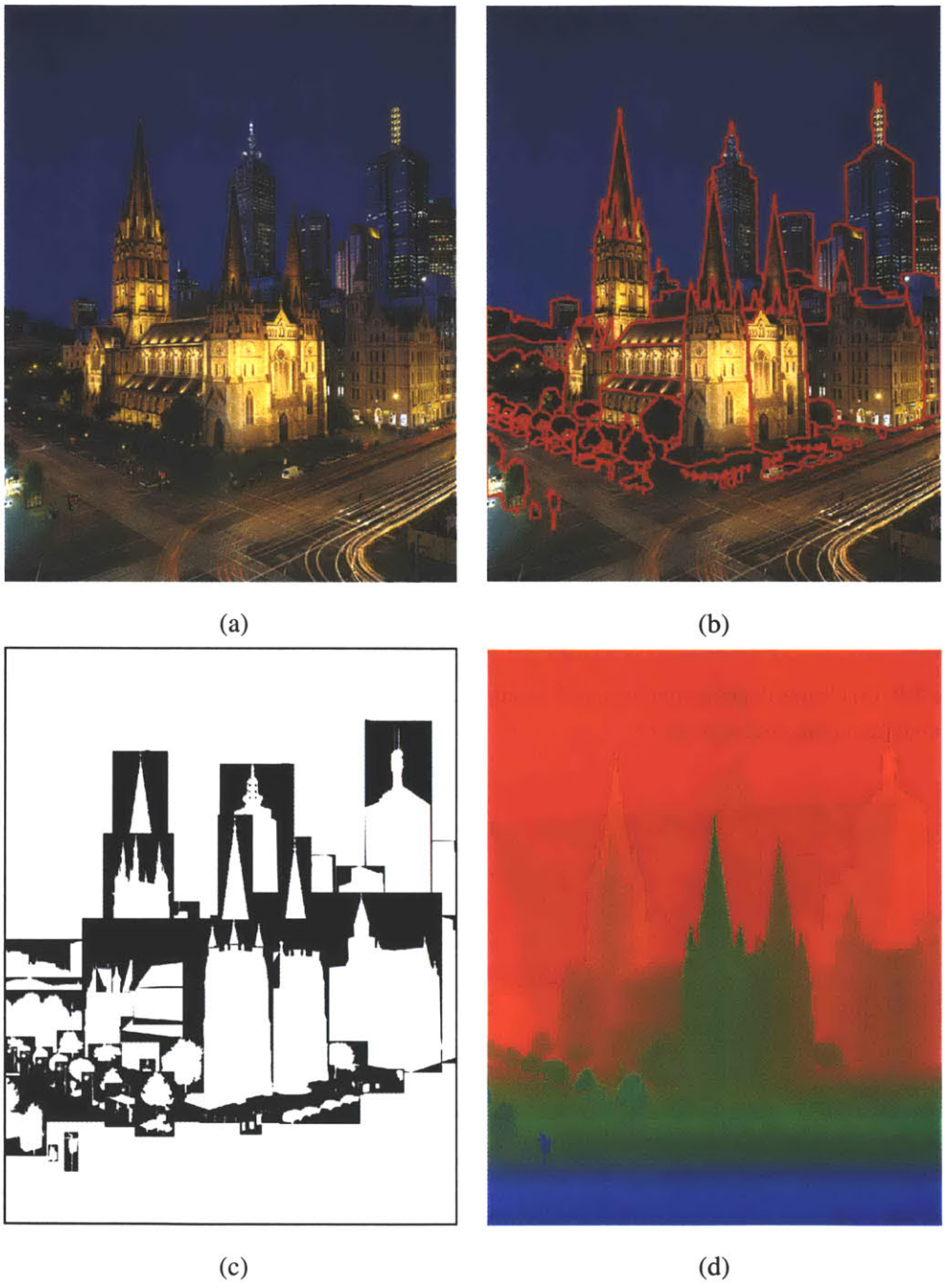
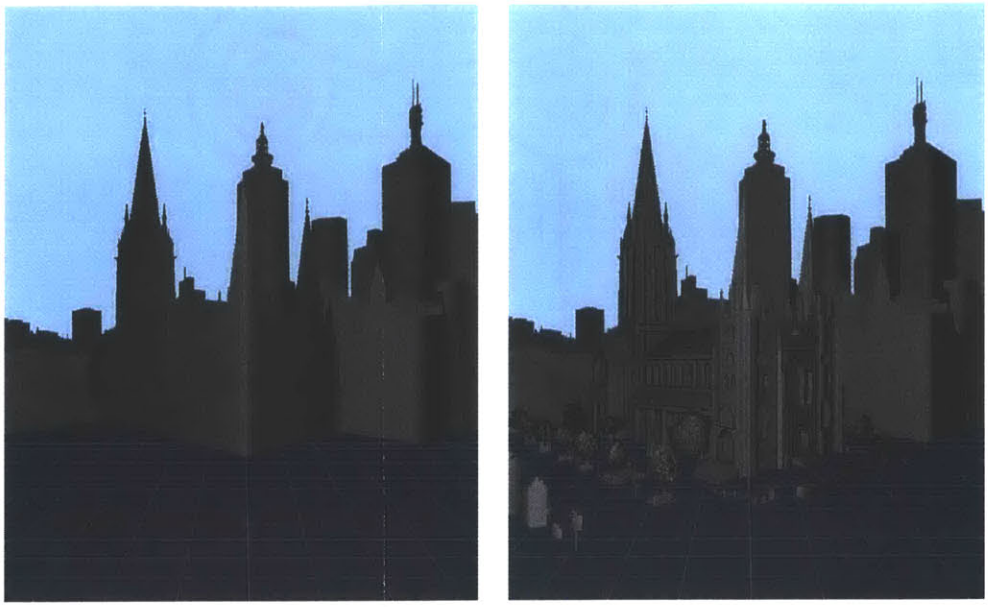


Figure 7-8: St. Paul's Church. (a) Initial input image. (b) Layers outlined in red. (c) Alpha channel in their bounding rectangle for each layer. (d) False color depth channel. The resolution of the initial image is 1000×1280. There are 61 layers, and the detailed segmentation took several days to complete, mainly due to the large resolution of the initial image, and numerous trees, statues, and street furniture. Specifying the depth took two hours. Input image courtesy of Barry Webb and Associates.



(a) (b)

Figure 7-9: (a) Coarse depth assignment took about 10 minutes (b) Refined depth assignment took about two hours total (including trial and error).

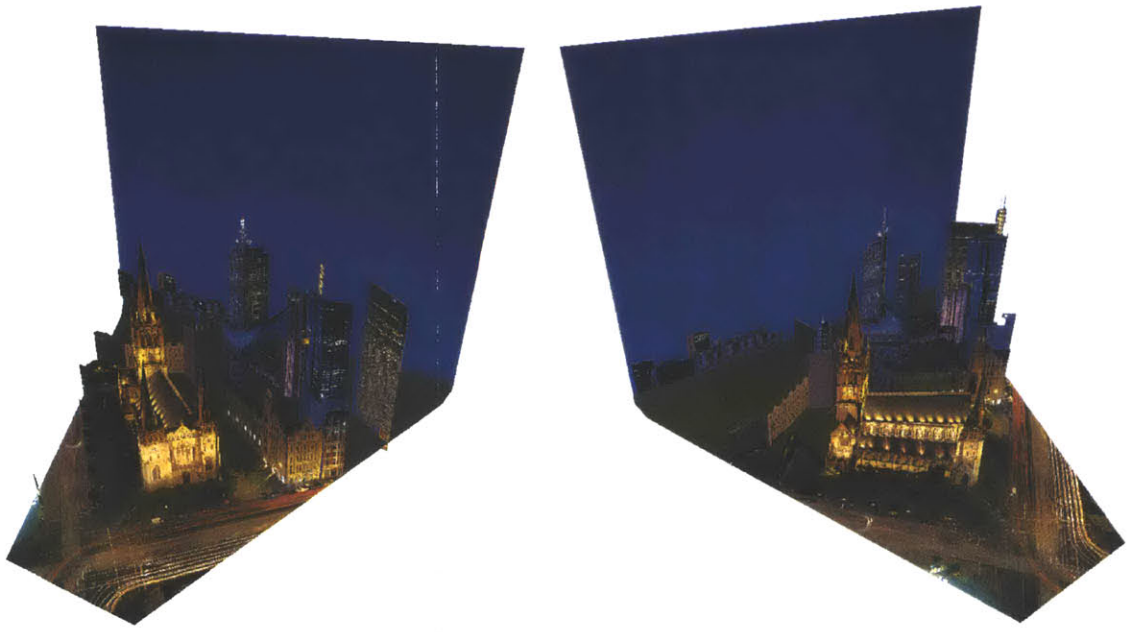


Figure 7-10: Bird's-eye views of the church. Most of the depth specification time was spent on refining the church. As shown, rest of the scene geometry is coarse, but effective.

removed the main tower of the cathedral and placed a new tower on the side, which was built from existing parts of the church. We also copied and pasted, and rotated the skyscrapers in the background. The entire editing process for both examples (a) and (b) took less than ten minutes. Again, these 3D transformations would be tedious and difficult to perform in 2D systems. In order to "rotate" a building, the user would have to manually transform each face of the building using a 2D free-transformation tool. Obtaining a convincing rotation would be tedious and challenging for planar objects, such as buildings, and practically impossible for organic shapes, such as statues and trees.

In Figure 7-15, we show the relighting of the church to demonstrate the use of our system for lighting design applications. We first used the texture-illuminance decoupling filter to separate the texture and illuminance channels for the church layers (Figures 7-13 and 7-14). We painted on the illuminance channel of the windows and the turrets (small point light sources), and used red upward pointing spot lights for the towers. The spotlighting on the roofs has also been removed simply by using the decoupling filter and painting over the illuminance channel with a dark monochrome color. The relighting took about two hours, mainly spent in redesigning an adequate lighting design scenario for the church.

Using traditional photo-editing tools to remove existing lighting would be almost impossible in this case. As discussed in Chapter 6, removing existing lighting is tedious and usually exhibits artifacts at illumination boundaries. In the case of the roof lights of the church, the process would be extremely time consuming and tedious, since the illumination boundaries are frequent and varies from sharp to blurred. Using the traditional clone brush to remove the lighting would also be difficult, since there are not enough uniform source regions to copy from, and also due to perspective foreshortening on the roofs.

Using inverse lighting simulation, as discussed in Chapter 6, would also be difficult and would probably result in artifacts. Due to the sheer magnitude and complexity of such an outdoor urban scenery at night, inferring accurate geometry from photographs, which is necessary for the inverse lighting simulation, is a daunting task. The number of light sources at night, and moving people and cars make this task extremely difficult, if not impossible. In the case of the church roofs, inverse lighting would probably result in haloing artifacts due to misregistration of the spotlights.

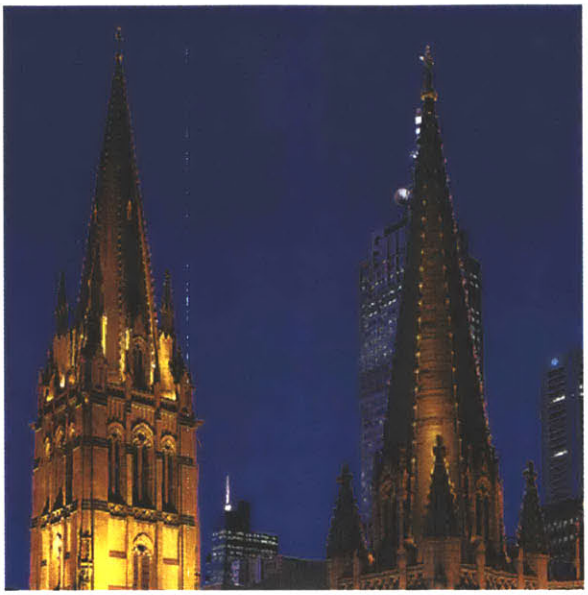
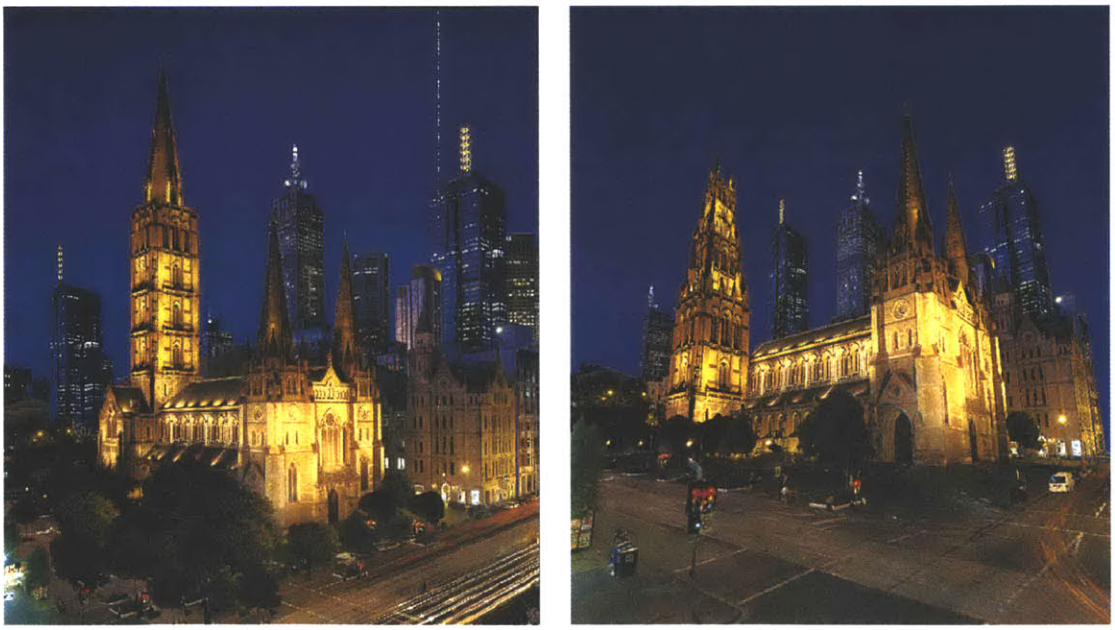


Figure 7-11: Close-up view of the church. Note the intricate details of the church turrets (as shown in their silhouettes) captured and modeled in our representation.



(a) (b)

Figure 7-12: Edited scenes. (a) Trees have been copied and pasted, the church tower has been clone brushed and made taller, and the buildings in the background has been copied and pasted, and rotated. (b) The main church tower has been removed and another tower has been built on the side.

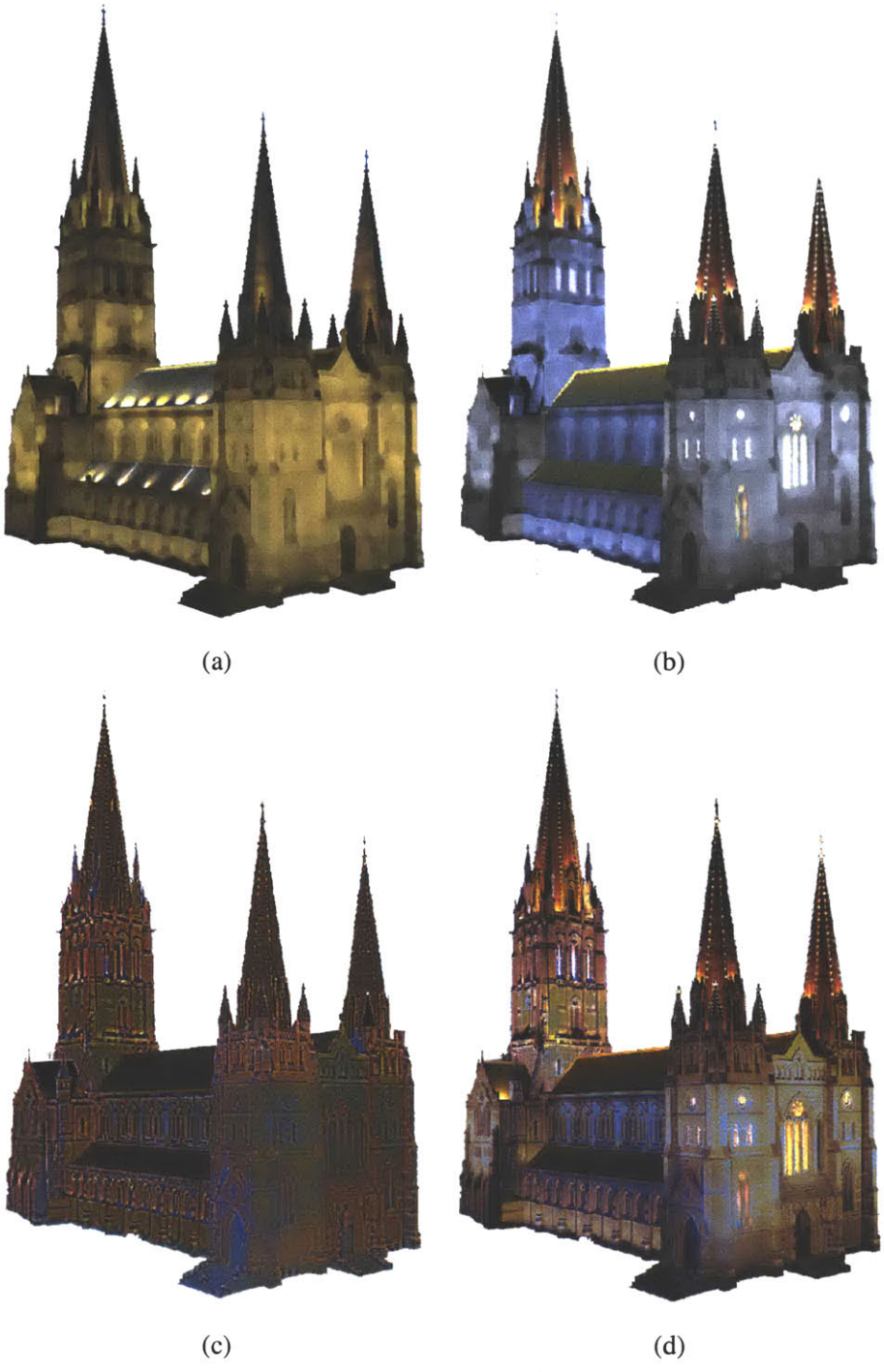
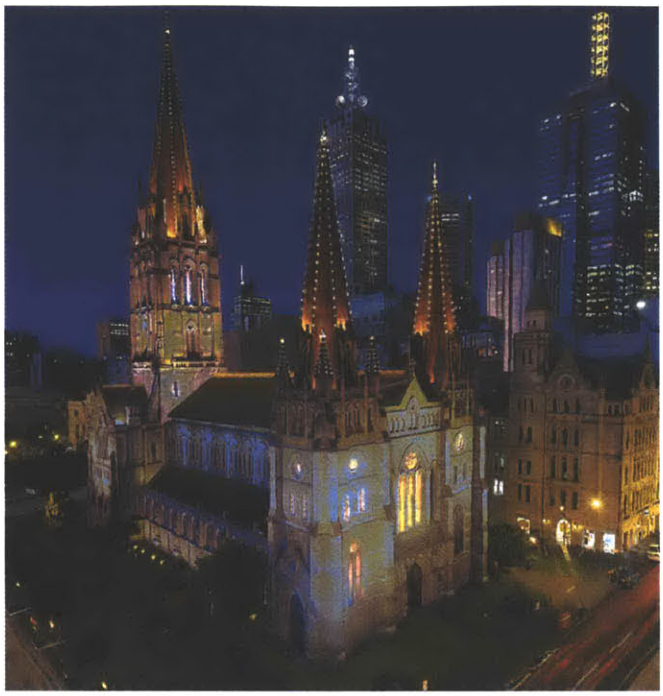


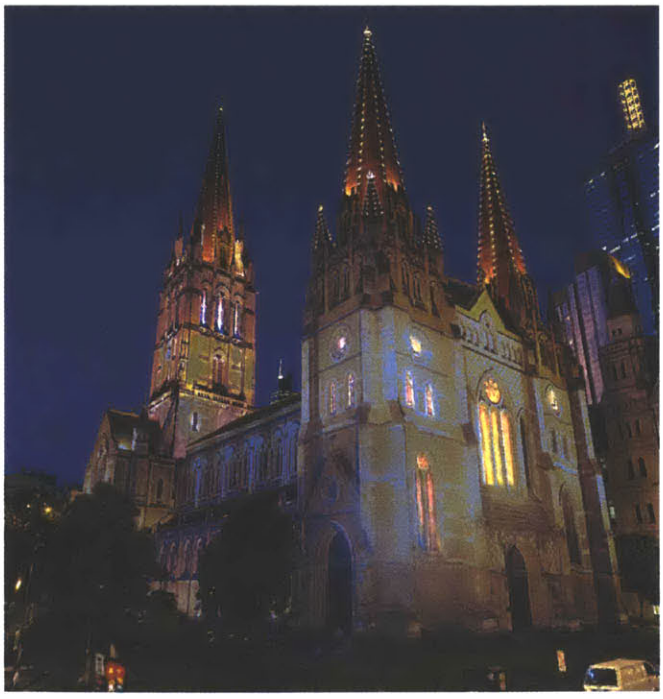
Figure 7-13: (a) Original illuminance channel. (b) Edited illuminance channel. (c) Texture channel. (d) Combined color channel ((b)×(c)).

(a) (b) (c)

Figure 7-14: Close up of the church tower. (a) Texture channel. (b) Edited illuminance channel. (c) Combined color channel.



(a)



(b)

Figure 7-15: Relighting example. (a) The illumination color on the church walls have been changed, and lights have been painted onto the windows. Also, the spotlighting on the roof has been removed. (b) Different view. Relighting took about two hours, mainly due to trial and error.

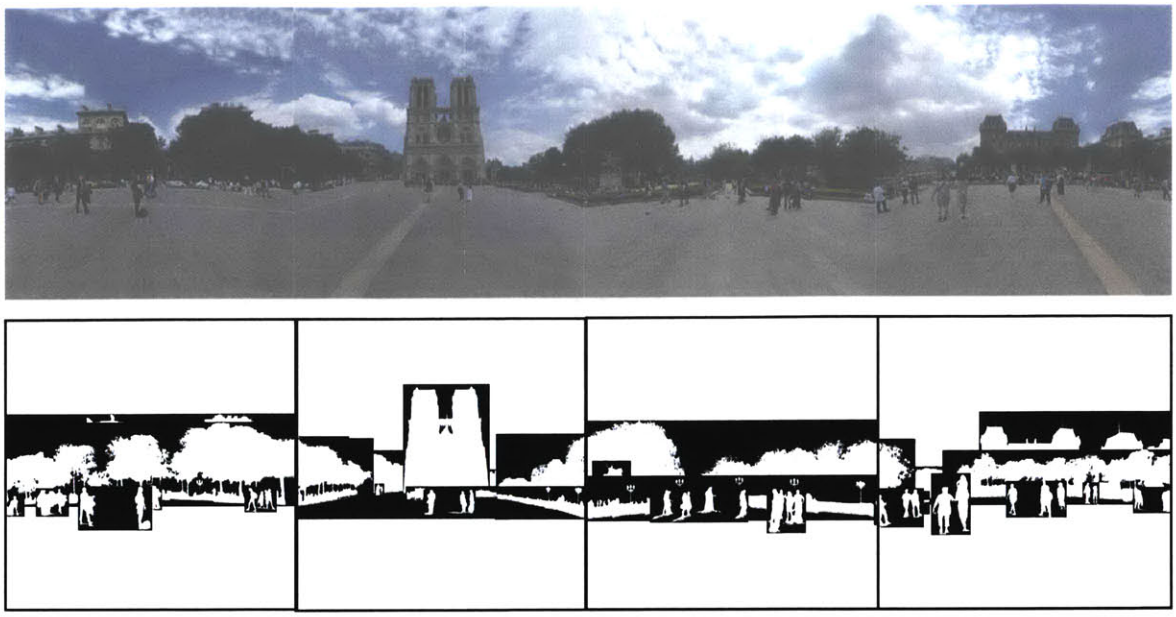
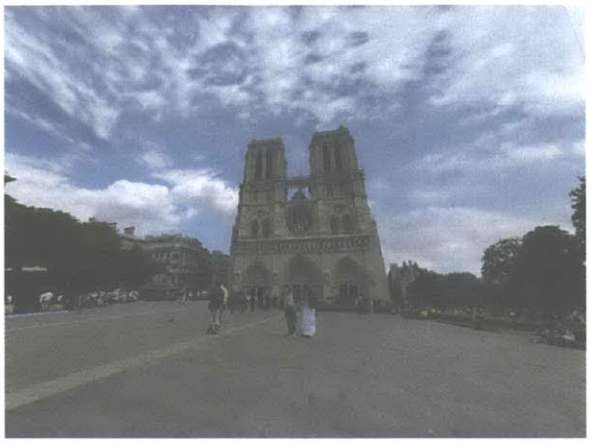


Figure 7-16: Panorama of Notre Dame, Paris (top). Alpha channels (bottom). The resolution of each image is 640×640. There are 43 layers total. The detailed segmentation process took about five hours. Specifying the depth took two hours.

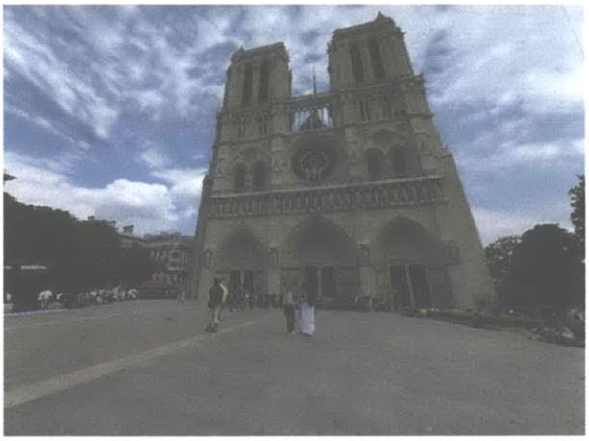
7.4 Notre Dame

In this example, we show a cube-mapped panorama of the Notre Dame cathedral in Paris. Not all six sides have been captured, and much of the sky and the ground was clone brushed. We show that even with an incomplete set of images for the panorama, the system is capable of creating an immersive environment. Unlike typical panorama viewers, such as [Apple; MGI; iPIX], the user can spatially move within the scene, and the system provides convincing parallax and occlusion effects for an immersive experience (Figure 7-17). Some applications, such as virtual tourism, may greatly benefit from our approach.

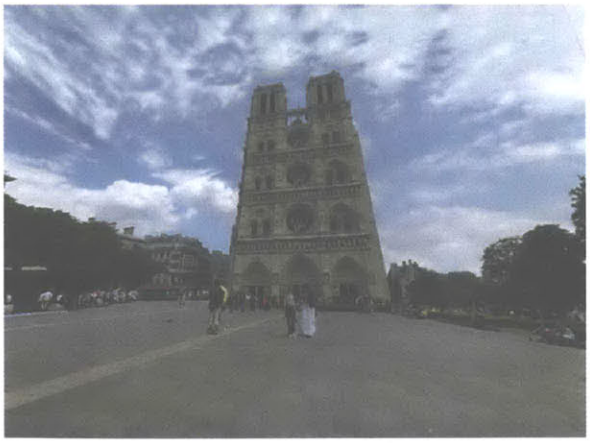
We also demonstrate simple editing on the cathedral, each of which took less than a minute. In Figure 7-17(b) the cathedral has been scaled, and in (c) the structure has been edited with the structure-preserving clone brush. The perspective correction, as discussed in Chapters 4 and 5, is transparent to the user when using the structure-preserving or non-distorted clone brush tools.



(a)



(b)



(c)

Figure 7-17: (a) New view. (b) Notre Dame scaled. (c) Notre Dame clone brushed. The scaling or clone brushing takes less than a minute.

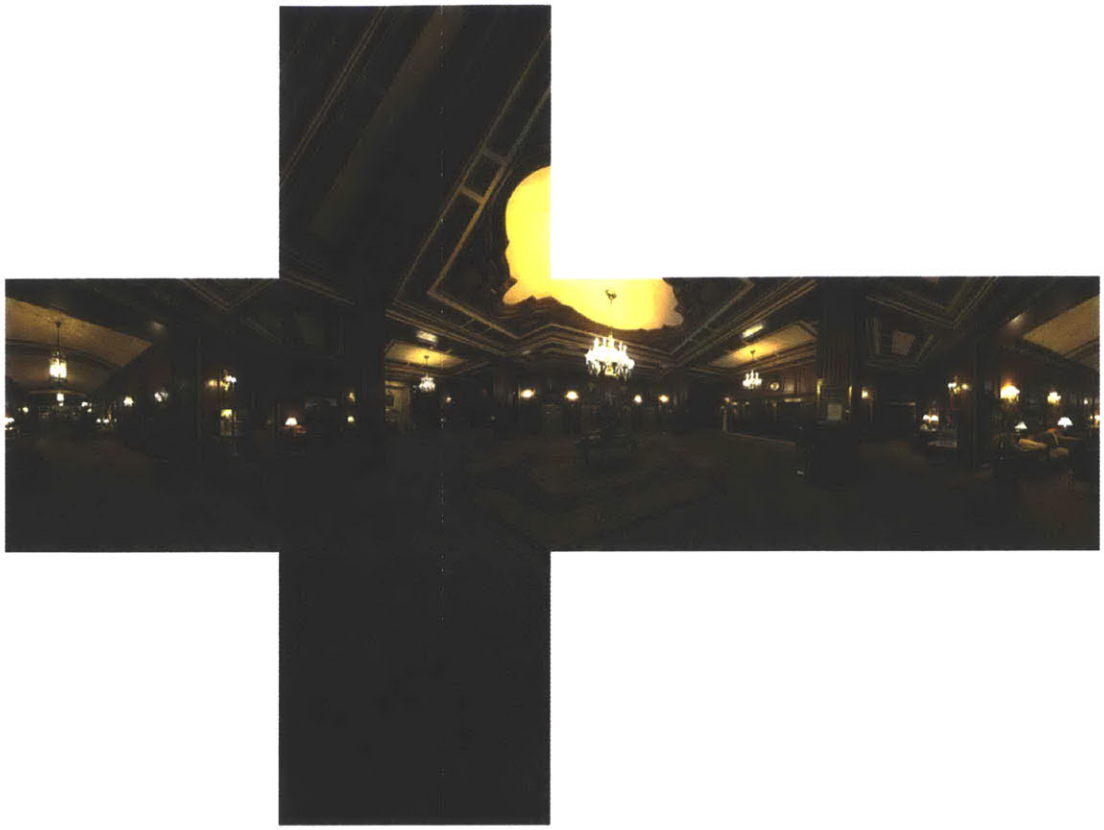


Figure 7-18: Six initial input images — cube map of a hotel lobby. There are six images of 512×512 resolution and 62 layers total. The precise segmentation process took approximately four days. The segmentation was especially difficult for this example, since there were many pieces of furniture, small objects and patterns in the scene. The depth specification took about six hours.

7.5 Hotel Lobby

In this example, we demonstrate our system in the context of interior lighting and design applications. Six images of a cube map are used to create a full 360×180 panoramic environment of a hotel lobby in downtown Boston (Figure 7-18). Figures 7-19 and 7-20 show the outlines of layers and the alpha channels, respectively. Although the overall underlying geometry is coarse, as shown in the bird's-eye view of the scene in Figure 7-21, all the details of the initial photographs are retained. The scene is rendered at interactive rates and the immersive experience is convincing. Note that parts of the scene that are further away from the original position of acquisition, denoted by the grey sphere in the middle of the figure, are more pixelated. Figures 7-22 and 7-23 show various views of the reconstructed environment.

In Figure 7-24, we show the details of the plant layer displayed in Figures 7-22 and 7-23. The

Figure 7-19: Layers outlined in red.

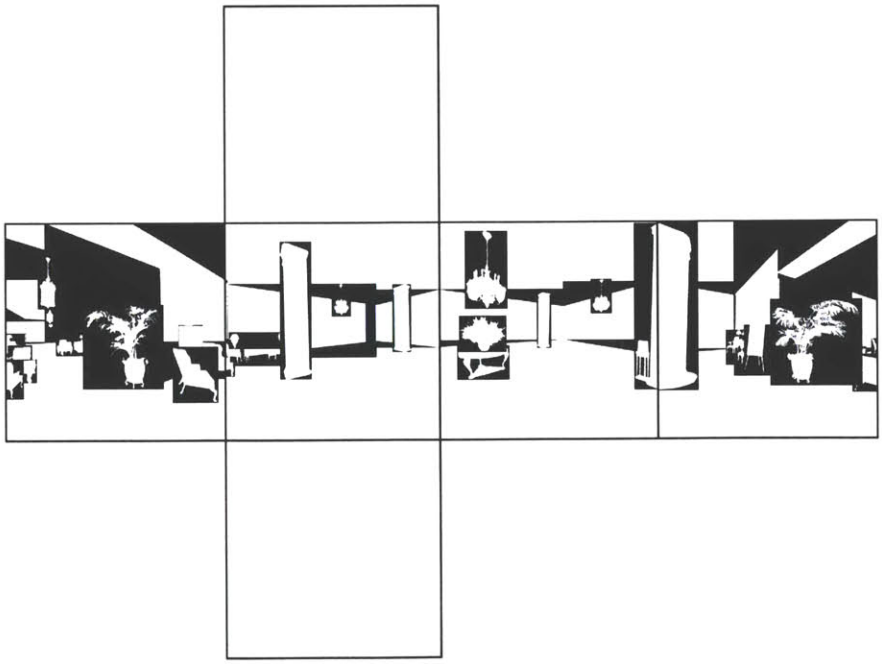


Figure 7-20: Alpha channel.

overall geometry and the number of triangles used to interactively display this layer is simple, yet the complexity of the object is conveyed through the reuse of the photograph and layering data structure.

In Figure 7-25, we demonstrate a seamless and photorealistic change of materials and lighting conditions. The texture-illuminance decoupling filter was the key tool in enabling the user relight and apply different materials in the scene. Important subtleties to note in this example is the change of lights and shadows as various parts of the scene are introduced in the sequence. For instance in image (b), the lamps, furniture, and the column are inserted along with their respective lighting and shadows on the floor. Figure 7-26 further demonstrates a subtle but perceptually important difference when the existing illumination condition is maintained while the floor material has been changed. This type of modification is extremely difficult in photo editing. With our system, removing or retaining existing lighting effects is simple. Relighting results from this example, including the illuminance channel visualization, have also been shown in Chapter 6.

Figure 7-21: Bird's-eye view of the reconstructed hotel lobby. The grey sphere shows the acquisition position of the initial photographs. Note that parts of the scene further away from the acquisition position are pixelated and lose resolution, due to the sample-based representation.

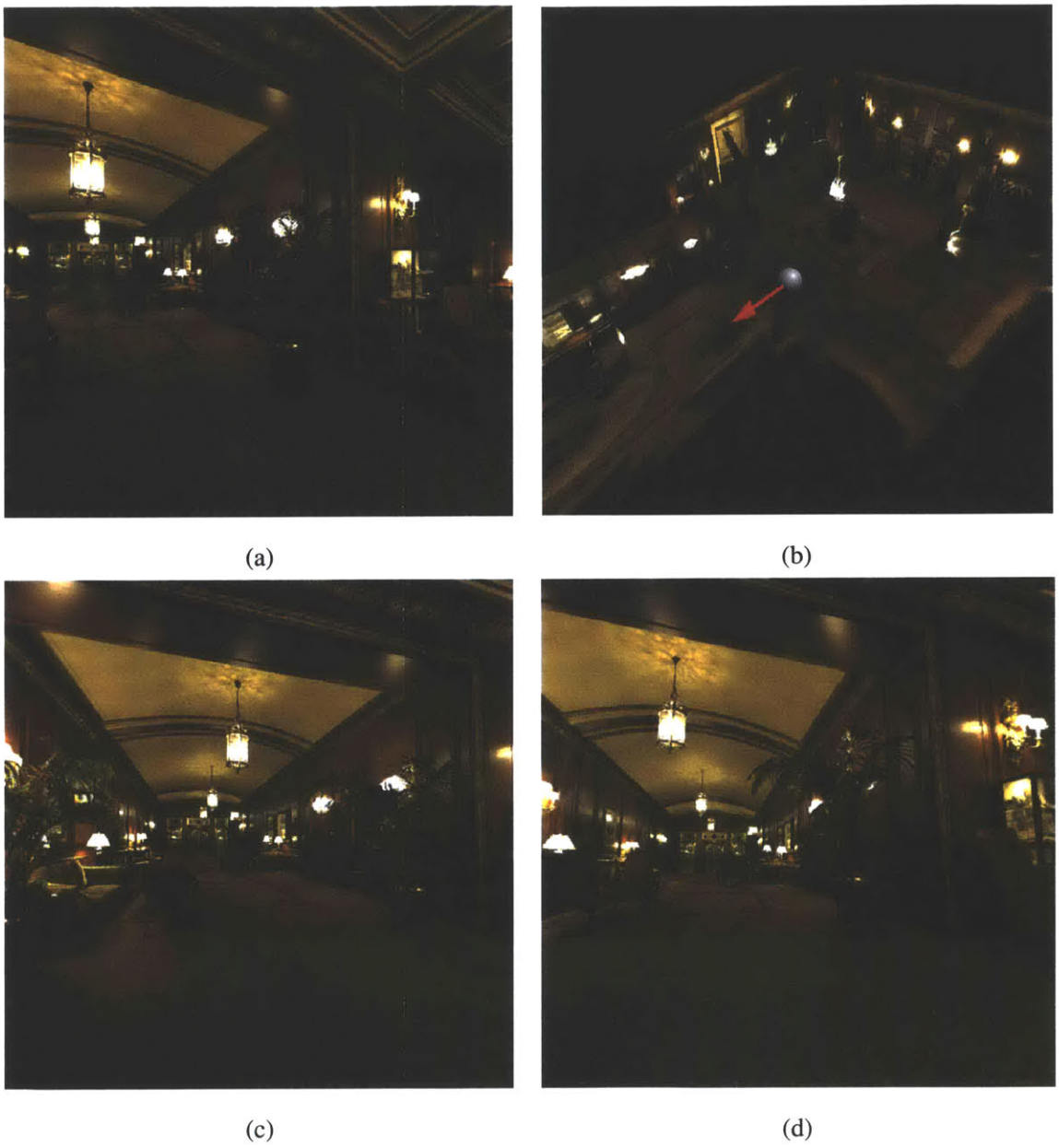
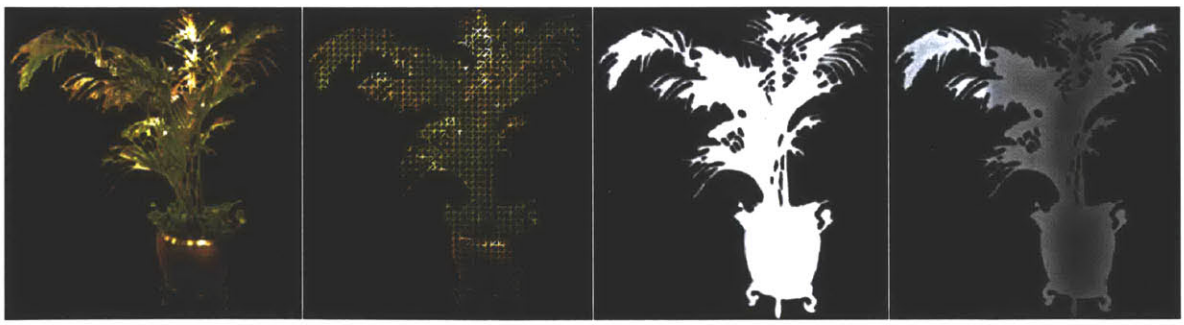


Figure 7-22: Panoramic view of the hotel lobby. (a) Reference image. (b) A bird's-eye view. The red arrow shows the general direction of (a), (c), and (d). (c),(d) Synthetic viewpoints.

Figure 7-23: Various wide-angle views of the panorama.



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

Figure 7-24: Various channels of the plant layer from the reference view. (a) Color channel. (b) Simple triangle mesh used to display the layer. (c) Alpha channel. (d) Depth channel.

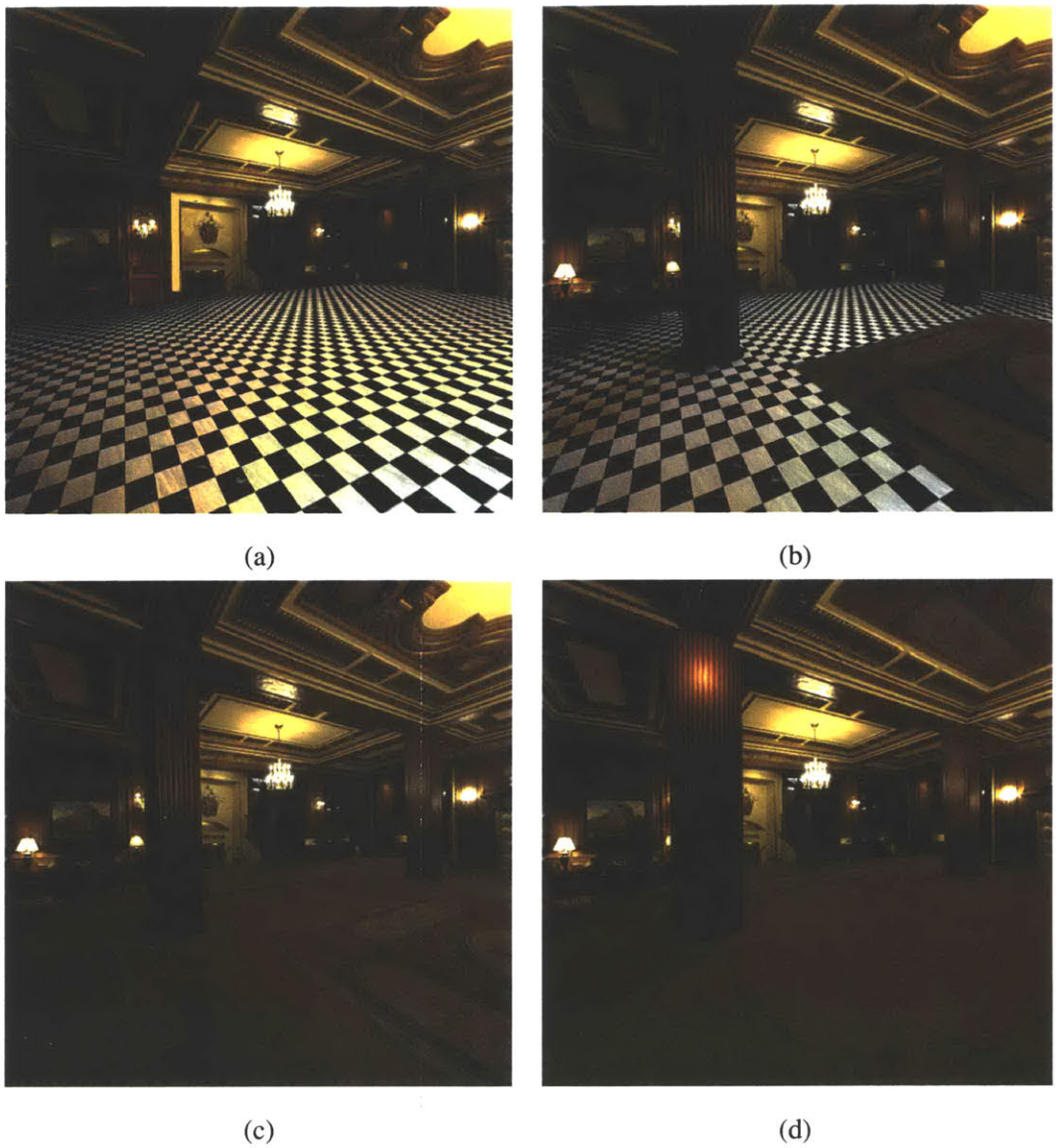


Figure 7-25: A sequence of edited scenes of the hotel lobby. Note in (b), the lighting and soft shadows on the floor are inserted along with the furniture and columns. (c) Original image. (d) The geometry of the column has been changed to a cylinder, the texture has been clone brushed using the non-distorted clone brush, and the carpet has been clone brushed onto the ceiling.

(a) (b)

Figure 7-26: (a) Change of floor material *without* maintaining existing illumination. (b) With existing illumination.

7.6 Discussion

In this chapter, we have shown various results of our system, ranging from paintings to panoramas. We have demonstrated that the system is capable of not only creating a 3D environment from a 2D image, but can edit the scene with the ease of 2D photo-editing tools.

There are many advantages of our system over photo-editing systems. First, we stress the fact that in many applications that use 2D photo-editing systems, the segmentation and layering processes are also necessary for editing. As discussed in Chapter 2, many special effects and design applications already produce amazing results simply by compositing and alpha blending multiple 2D layers. In our approach, after a reasonable time invested in depth assignment, we enable the user to interactively change viewpoints and apply numerous editing operations from any viewing angle. This is not possible in 2D photo editing. Second, objects in the scene may be transformed in 3D space instead of 2D. In the church example (Section 7.3), if we translate, rotate, or scale the trees and the buildings, correct 3D transformations are applied implicitly and automatically. The user can select all the trees with different shapes and positions, and transform them simultaneously with ease without being concerned about the correctness of their transformations. In photo editing, the object transformations are limited to 2D and usually approximated by the artist. A simple operation in our system, such as rotating an object, is a difficult and tedious process in photo editing. Finally, many tools and filters are improved and enhanced due to the depth information, such as the non-distorted clone brush and the texture-illuminance decoupling filter. Even with the coarse depth information, relighting and clone brushing tools are much more powerful and versatile than their 2D counterparts.

Modeling complex environments, such as the church (Section 7.3) or the hotel lobby (Section 7.5), using either high-end CAD systems or image-based modeling techniques would be difficult and time consuming. Sheer complexity of the scene makes it difficult to acquire, model, and render via traditional computer graphics techniques. Building a 3D model of such complex scenes using image-based modeling techniques may be feasible for parts of the scene with planar and rectilinear geometry, but it would be extremely time consuming and difficult for more non-architectural objects, such as the furniture, plants, trees, and detailed decorations, even in the hands of an experienced user.

In our system, the richness and the complexity of shapes are captured through segmentation and alpha blending of multiple layers. The rendering cost is mostly independent of the scene com-

Example	Resolution	Image Pixels	Layer Pixels	Factor
Dali Head	1000 x 1124	1,124,000	2,031,232	1.80
Dali Woman	1000 x 1124	1,124,000	1,956,205	1.74
Statue	984 x 1500	1,476,000	2,742,842	1.86
St. Paul's Church	1000 x 1280	1,280,000	3,119,487	2.47
Notre Dame	640 x 640 x 4	1,638,400	2,937,800	1.79
Hotel Lobby	512 x 512 x 6	1,572,864	2,716,972	1.73

Table 7-1: Table comparing the total number of pixels in the input image versus the pixels in the layered representation.

plexity, and is proportional to the total number of pixels in all the layers. As shown in Table 7-1, most examples that we show have fewer than two times the number of pixels of the input image. Moreover, our system not only enables the user to display the recreated scene, but to further edit the environment with the ease of 2D photo-editing tools. The user-interactions of the tools are based on 2D painting metaphor, and we strongly believe this is an important characteristic of our system for numerous applications.

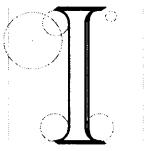
One of the limitations, and a possible extension, of the system is in automatically and coherently providing accurate shadowing effects when new light sources are added or when an object is moved. For instance in Figure 7-26, if we move the column, we do not provide an automatic way to move the shadows according to the existing lighting condition. Projective textures may be used to approximate shadows, but in most cases, full geometric and occlusion information is not available in our system (the back side of the column is nonexistent). Also, if a new light source has been added, or if we move one of the chandeliers, the appropriate change in illumination must be applied manually. In practice, we experienced that painting on the illuminance channel to add or remove shadows is simple, requires little time and effort, and is effective, as demonstrated in Figure 7-25(a) and Figure 7-7.

Another limitation is due to the sample-based representation. As illustrated in Figure 7-21, parts of the scene that are further away from the initial acquisition position are more pixelated and lose resolution. This is a common problem in sample-based representation in general, and in particular for single image modeling techniques. In our system, we use high-resolution images as input, in the

mega-pixel range. We discuss this issue further in Chapter 8.

CHAPTER 8

Conclusion and Future Work



mage-based modeling and rendering is a powerful approach to capture and display existing scenes. Since images are used as the basis for scene representation, they are able to redisplay incredibly complex and intricate object shapes, materials, and lighting effects that traditional computer graphics techniques still struggle with. Photorealism is inherent in IBMR techniques. Yet, editing these representations has received little attention. Image-based editing can prove extremely fruitful for applications that require the manipulation and modification of existing scenes. It introduces an exciting area of research in designing new representations and editing tools.

In this thesis, we have presented a system for image-based modeling and photo editing. We have demonstrated the potential and the power of image-based editing, where the user can not only interactively redisplay the scene from a photograph, but has the means to flexibly and efficiently edit the scene with the ease of 2D photo editing. The simplicity of our system has many important ramifications in designing an image-based editing system. First, we are not limited to polygonal primitives as geometric elements, and the flexible representation enables the encoding of depth, texture, and illuminance channels. The clear separation of these channels allows the user to modify

many facets in 3D editing — the shape, material, and the lighting conditions of the scene. Second, the existing class of photo-editing tools and filters is easily extended to work with our system. Our representation consists of 2D images, so the familiar class of tools, such as clone brushing, selection, cutting and pasting, and filtering, are easily incorporated. We also provide a suite of new tools for depth assignment, clone brushing, and relighting. The interactions of these tools are in the 2D image space, which makes them much more intuitive and user-friendly. Furthermore, the simple system architecture permits new tools and filters to be easily implemented and plugged into the system.

The simplicity of the image-based representation — layers of images with depth — is the cornerstone of our system architecture. The representation is amenable to both displaying as well as editing: it is easy to access and update the data structure, since it consists of simple images. The editing tools are, therefore, powerful and easy to implement. The representation also enables the user to interactively visualize the captured environment — the layers provide rich and accurate occlusion silhouettes of the scene while minimizing visible artifacts, such as rubber-sheet triangles [Mark 1999]. The user interface is intuitive, since many of the operators are extensions of their photo-editing counterparts.

We have proposed three new and powerful editing techniques, some of which have applications beyond the scope of this system. The first new tool, the *structure-preserving clone brush*, is a simple yet effective extension of the traditional clone brush, where it correctly conforms the brush strokes on simple 3D geometric configurations. It is capable of correcting the perspective foreshortening problem on planar surfaces, as well as correcting for the color differences due to lighting conditions, and can "snap" to a more accurate initial point. These improvements are powerful and can broaden the utility and applicability of the traditional 2D clone brush. The structure-preserving clone brush may be used directly in 2D photo-editing systems as a plug-in, since it does not need depth information.

The second new editing tool is the *non-distorted clone brush*. It fully enables the user to clone brush from any part of the scene with arbitrary shapes, and it automatically minimizes the distortions due to foreshortening. It uses the depth information to correct the distortions that appear in the traditional 2D version of the clone brush, and the parameterization optimization technique conforms with the underlying geometry for both source and destination regions. To make the tool interactive, only the clone brushed regions are parameterized with a simple (u,v) initialization, and expanded as needed. The new clone brush tools are crucial for filling in holes that occur from layering, and also for rapidly editing the representation while maintaining photorealism.

Finally, the third new tool is the *texture-illuminance decoupling filter*, which enables the user to discount the existing lighting of the scene, such that the texture and the illumination may be edited separately. The filter is essential for relighting applications, as well as clone brushing on the texture channel — only similarly illuminated regions may be seamlessly and effectively clone brushed. The non-linear filter decouples the input image into texture and illuminance channels, such that they may be edited separately to change the material or the lighting conditions, respectively. The basic idea is to first use a low-pass filter to compute the illuminance channel, then divide the input and illuminance images to compute the texture channel. From this basic idea, we further improve the filter by exploiting the depth channel to correct the convolution kernel according to the geometry and perspective foreshortening of the scene. In addition, we use bilateral filtering to handle illumination discontinuities. This technique can be used in conjunction with the more classical image-based modeling techniques to relight the scene. Furthermore, it can be utilized to retrieve uniform texture that would be a much better input for texture generation algorithms.

In our experience, such tools can greatly facilitate the seamless alteration of image-based representations. The additional information encoded in our representation permits editing capabilities that would be very time-consuming, if not impossible, using 2D photo-editing or 3D modeling systems.

8.1 Limitations and Future Work

We have focused on single images as input, which has both advantages and disadvantages. The single-image case is important because in many applications only a single image of the scene is available. Moreover, dealing with single images allows us to work with a simple and flexible image-based representation that has many benefits in the context of image-based editing. But there are also limitations stemming from having only a single image as input. First, we assume a Lambertian environment, therefore, any captured view-dependent effects, such as specular highlights and reflections present in the input image, are not properly displayed at different viewing positions. Although the texture-illuminance decoupling filter may be used to remove specular highlights, if the captured scene has highly specular reflections, it is often difficult to factor out the reflection from the texture since the specular highlights are in the same frequency domain as the texture. Second, many tools in our system provide the means to build and edit visually convincing reconstructions, but they are not designed for accurate results. Although it is possible to determine accurate depth up to a scale factor from a single image for scenes that have structural cues, such as architectural scenes [Liebowitz

et al. 1999], accuracy was not a paramount goal for our system. Finally, the system is not capable of creating a full 360×180 degree view of an object in an *inward-looking* manner. A full capture and reconstruction of such objects require multiple input images, such as [Debevec et al. 1996].

We would like to extend our system to multiple input images to address the limitations due to the single-image framework. First, it would be possible to provide view-dependent effects by using similar techniques as [Debevec et al. 1996; Debevec et al. 1998]. Although faithfully capturing all view-dependent effects for spatially vast scenes, e.g. outdoor scenes, is difficult, even an approximation would provide a much more visually convincing result. Also, using multiple images would provide a better estimate for the texture-illuminance decoupling filter, particularly for factoring specular effects from texture. Second, multiple images would allow a more accurate modeling of geometry. It would be possible to first employ an image-based modeling technique, such as [Debevec et al. 1996], to build an accurate block model, which is then used as the basis for the refinement tools (e.g. painting and chiselling). Furthermore, alpha channels from the multiple images may be used to extract detailed silhouettes of buildings or trees by employing similar techniques as [Matusik et al. 2000]. Finally, multiple images may be used to not only to model a full 360×180 degree view of an object, but also to allow smooth transitions between multiple capture positions for larger walkthroughs of both indoor and outdoor scenes. Some of the issues involved would be camera registration and blending multiple layers of the same object. Simple manual tests demonstrated in Figure 8-1 show promising results.

Many other techniques could be incorporated into our system to improve its utility. For example, depth assignment could benefit from shape from shading or from multiple images via stereo correspondence and epipolar geometry. The recent Single-View Modeling technique by [Zhang et al. 2001] that relies on normal specification would be a logical extension. Other 2D photo editing techniques could be extended to take into account the depth information, for example for filtering operators as demonstrated by our texture-illuminance decoupling filter.

Another future direction is to use our tools in the context of 3D modeling and 3D painting. Many of the current modeling systems are limited to applying texture maps onto the modeled geometry. By incorporating tools and techniques similar to our system, users can intuitively and rapidly model approximate geometry from images. Further, the texture-illuminance decoupling filter enables the user to edit the materials, which can then be combined with the traditional off-line rendering techniques.

One of the motivations for designing such a system is to apply it to real-world problems. We

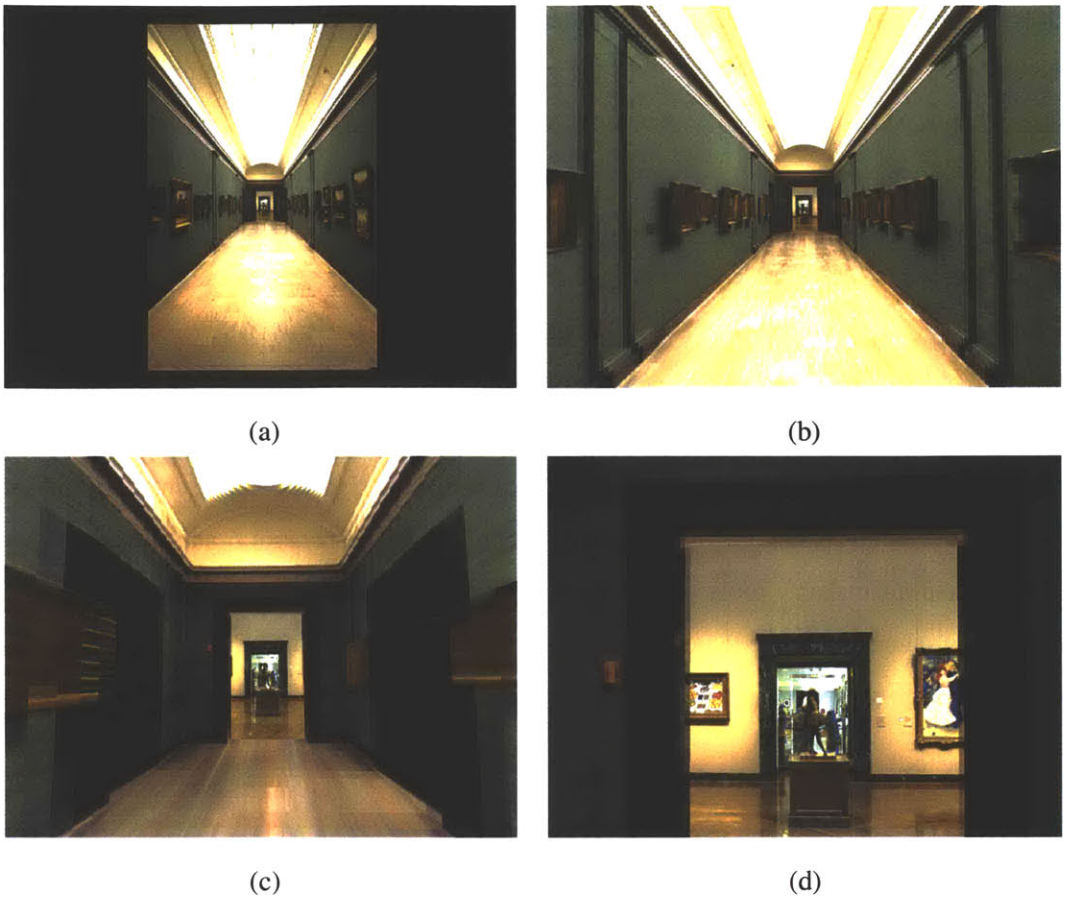


Figure 8-1: Another multiple-image walk-through sequence. The system is capable of displaying multiple scene representations for a wide-ranging walkthroughs. As the sequence of images progresses, the room at the end of the hallway is no longer pixelated.

are interested in looking into various applications for our technique, from architectural, interior and landscape design to games and special effects in the entertainment industry. Understanding the pains involved in design or the process of producing the special effects may lead to the enhancement or invention of editing tools. We demonstrated the system to lighting design consultants and they showed much interest and enthusiasm. Their immediate reaction was that our system looked much more intuitive than traditional 3D approaches. They further commented that it was the first tool offering 3D capabilities they felt they could use during the design process.

Image-based editing, in a more general context, opens up new avenues for future research. There are many different classes of image-based representations, and each may require different implementation of tools and modifications of the representation. An ambitious and challenging area of research is in editing high-dimensional representations, such as light fields [Levoy and Hanrahan

1996] and lumigraphs [Gortler et al. 1996]. A technique similar to that used by Seitz and Kutulakos is necessary: make assumptions and infer the geometry, material, and the illumination conditions [Seitz and Kutulakos 1998]. One possibility is to extend the lumigraph data structure [Gortler et al. 1996], since a volumetric geometry is already computed. For scenes with multiple objects and larger spatial domain, techniques such as [Matusik et al. 2000; Buehler et al. 2001] may be a good starting point.

Another area of research is in editing image-based representations that include the temporal domain. The input could be simple video streams or higher-dimensional image-based data, such as [Yang and McMillan 2002]. Much more sophisticated and coherent editing tools are necessary than simple video-editing tools. Some of the challenges include design of a representation amenable for temporal editing, spatial and temporal registration of editing operations, and the development of flexible and controllable tools. Since the temporal capture devices, such as hand-held video cameras, continuously change temporally as well as spatially, an editing operation applied on one slice of the image must coherently propagate through the rest of the data set.

BIBLIOGRAPHY

AGRAWALA, M., BEERS, A. C., AND LEVOY, M. 1995. 3d painting on scanned surfaces. In *1995 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, pp. 145–150. ISBN 0-89791-736-7.

ALIASWAVEFRONT. http://www.aliaswavefront.com.

APPLE. http://www.quicktime.com.

BERMAN, A., DADOURIAN, A., AND P.VLAHOS, 2000. Method for removing from an image the background surrounding a selected object. U.S. Patent 6,134,346.

BERMAN, A., P.VLAHOS, AND DADOURIAN, A., 2000. Comprehensive method for removing from an image the background surrounding a selected object. U.S. Patent 6,134,345.

BERTALMIO, M., SAPIRO, G., CASELLES, V., AND BALLESTER, C. 2000. Image inpainting. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 417–424. ISBN 1-58113-208-5.

BLANZ, V., AND VETTER, T. 1999. A morphable model for the synthesis of 3d faces. In *Proceedings of SIGGRAPH 1999*, ACM SIGGRAPH / Addison Wesley Longman, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 187–194. ISBN 0-20148-560-5.

BOIVIN, S., AND GAGALOWICZ, A. 2001. Image-based rendering of diffuse, specular and glossy surfaces from a single image. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, pp. 107–116. ISBN 1-58113-292-1.

BUEHLER, C., BOSSE, M., MCMILLAN, L., GORTLER, S. J., AND COHEN, M. F. 2001. Unstructured lumigraph rendering. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, pp. 425–432. ISBN 1-58113-292-1.

CANOMA. http://www.canoma.com.

CHEN, S. E., AND WILLIAMS, L. 1993. View interpolation for image synthesis. In *Proceedings of SIGGRAPH 1993*, Computer Graphics Proceedings, Annual Conference Series, pp. 279–288. ISBN 0-201-58889-7.

CHEN, S. E. 1995. Quicktime vr - an image-based approach to virtual environment navigation. In *Proceedings of SIGGRAPH 1995*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 29–38. ISBN 0-201-84776-0.

CHEN, M. 2001. *Interactive Tools for Acquiring Depth from Single Images.* Master's thesis, Massachusetts Institute of Technology.

CHUANG, Y.-Y., CURLESS, B., SALESIN, D., AND SZELISKI, R. 2001. A bayesian approach to digital matting. *IEEE Computer Vision and Pattern Recognition.*

COHEN, M. F., CHEN, S. E., WALLACE, J. R., AND GREENBERG, D. P. 1988. A progressive refinement approach to fast radiosity image generation. In *Computer Graphics (Proceedings of SIGGRAPH 1988)*, vol. 22, pp. 75–84.

CRIMINISI, A., REID, I. D., AND ZISSERMAN, A. 1999. Single view metrology. *International Conference on Computer Vision*, pp. 434–442.

DEBEVEC, P. Light probes. http://www.debevec.org/Probes/.

DEBEVEC, P., TAYLOR, C. J., AND MALIK, J. 1996. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of SIGGRAPH 1996*,

ACM SIGGRAPH / Addison Wesley, New Orleans, Louisiana, Computer Graphics Proceedings, Annual Conference Series, pp. 11–20. ISBN 0-201-94800-1.

DEBEVEC, P. E., YU, Y., AND BORSHUKOV, G. D. 1998. Efficient view-dependent image-based rendering with projective texture-mapping. In *Eurographics Rendering Workshop 1998*, Springer Wein / Eurographics, Vienna, Austria, pp. 105–116. ISBN 3-211-83213-0.

DEBEVEC, P., 1997. The campanile movie. SIGGRAPH Electronic Theater.

DEBEVEC, P. 1998. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of SIGGRAPH 1998*, ACM SIGGRAPH / Addison Wesley, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, pp. 189–198. ISBN 0-89791-999-8.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 2000. Anisotropic feature-preserving denoising of height fields and bivariate data. In *Graphics Interface*, pp. 145–152. ISBN 1-55860-632-7.

DISCREET. http://www.discreet.com.

DRETTAKIS, G., AND SILLION, F. X. 1997. Interactive update of global illumination using a line-space hierarchy. In *Proceedings of SIGGRAPH 1997*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 57–64. ISBN 0-89791-896-7.

DRETTAKIS, G., ROBERT, L., AND BOUGNOUX, S. 1997. Interactive common illumination for computer augmented reality. *Eurographics Rendering Workshop*.

DURAND, F., AND DORSEY, J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. In *Proceedings of SIGGRAPH 2002*, ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, To Be Published.

FAUGERAS, O., LAVEAU, S., ROBERT, L., CSURKA, G., AND ZELLER, C. 1995. 3-d reconstruction of urban scenes from sequences of images. In *Automatic Extraction of Man-Made Objects from Aerial and Space Images*, A. Gruen, O. Kuebler, and P. Agouris, Eds. Birkhauser.

FOURNIER, A., GUNAWAN, A. S., AND ROMANZIN, C. 1993. Common illumination between real and computer generated scenes. In *Graphics Interface '93*, Canadian Information Processing Society, Toronto, Ontario, Canada, pp. 254–262.

GIMP. The gimp. http://www.gimp.org.

GLEICHER, M. 1995. Image snapping. *Proceedings of SIGGRAPH 1995* (August), pp. 183–190. ISBN 0-201-84776-0. Held in Los Angeles, California.

GOMES, J., DARSA, L., COSTA, B., AND VELHO, L. 1998. *Warping And Morphing Of Graphical Objects*. Morgan Kaufman.

GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. 1996. The lumigraph. In *Proceedings of SIGGRAPH 1996*, ACM SIGGRAPH / Addison Wesley, New Orleans, Louisiana, Computer Graphics Proceedings, Annual Conference Series, pp. 43–54. ISBN 0-201-94800-1.

GUILLOU, E., MENEVEAUX, D., MAISEL, E., AND BOUATOUCH, K. 2000. Using vanishing points for camera calibration and coarse 3d reconstruction from a single image. *The Visual Computer 16*, 7, pp. 396–410. ISSN 0178-2789.

HANRAHAN, P., AND HAEBERLI, P. 1990. Direct wysiwyg painting and texturing on 3d shapes. In *Proceedings of SIGGRAPH 1990*, ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, pp. 215–223.

HECKBERT, P. S. 1989. *Fundamentals of Texture Mapping and Image Warping*. Master's thesis.

HORN, B. K. P. 1990. Height and gradient from shading. *International Journal of Computer Vision 5*, 1, pp. 37–75.

HORRY, Y., ICHI ANJYO, K., AND ARAI, K. 1997. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proceedings of SIGGRAPH 1997*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 225–232. ISBN 0-89791-896-7.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3d freeform design. In *Proceedings of SIGGRAPH 1999*, ACM SIGGRAPH / Addison Wesley Longman, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 409–416. ISBN 0-20148-560-5.

IGEHY, H., AND PEREIRA, L. 1997. Image replacement through texture synthesis. In *IEEE International Conference on Image Processing*.

ILLUSTRATOR. Adobe illustrator. http://www.adobe.com/products/illustrator.

IPIX. http://www.ipix.com/.

JOBSON, D. J., RAHMAN, Z., AND WOODELL, G. A. 1997. A multi-scale retinex for bridging the gap between color images and the human observation of scenes. *IEEE Transactions on Image Processing: Special Issue on Color Processing* (July).

KANG, S. 1998. Depth painting for image-based rendering applications. Technical report, CRL, Compaq Cambridge Research Lab.

LAND, E. H., AND J, J, M. 1971. Lightness and retinex theory. *Journal of Optical Society of America 61*, pp. 1–11.

LARSON, G. J. W. 1992. Measuring and modeling anisotropic reflection. In *Computer Graphics (Proceedings of SIGGRAPH 1992)*, vol. 26, pp. 265–272. ISBN 0-201-51585-7.

LAVEAU, S., AND FAUGERAS, O. 1994. 3-D scene representation as a collection of images and fundamental matrices. In *Proc. of 12th Int. Conf. on Pattern Recognition*, vol. 1, pp. 689–691.

LEVOY, M., AND HANRAHAN, P. M. 1996. Light field rendering. In *Proceedings of SIGGRAPH 1996*, ACM SIGGRAPH / Addison Wesley, New Orleans, Louisiana, Computer Graphics Proceedings, Annual Conference Series, pp. 31–42. ISBN 0-201-94800-1.

LÉVY, B., AND MALLET, J.-L. 1998. Non-distorted texture mapping for sheared triangulated meshes. In *Proceedings of SIGGRAPH 1998*, ACM SIGGRAPH / Addison Wesley, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, pp. 343–352. ISBN 0-89791-999-8.

LIEBOWITZ, D., AND ZISSERMAN, A. 1998. Metric rectification for perspective images of planes.

LIEBOWITZ, D., CRIMINISI, A., AND ZISSERMAN, A. 1999. Creating architectural models from images. *Computer Graphics Forum 18*, 3 (September), pp. 39–50. ISSN 1067-7055.

1990. *Two-Dimensional Signal and Image Processing*. Prentice Hall.

LOSCOS, C., FRASSON, M.-C., DRETTAKIS, G., WALTER, B., GRANIER, X., AND POULIN, P. 1999. Interactive virtual relighting and remodeling of real scenes. In *Eurographics Rendering Workshop 1999*, Springer Wein / Eurographics, Granada, Spain.

LOSCOS, C., DRETTAKIS, G., AND ROBERT, L. 2000. Interactive virtual relighting of real scenes. *IEEE Trans. on Visualization and Computer Graphics 6*, 3.

MALLET, J. 1989. Discrete smooth interpolation. *ACM Trans. on Graphics 8*, 2, pp. 121–144.

MARK, W. R., MCMILLAN, L., AND BISHOP, G. 1997. Post-rendering 3d warping. In *1997 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, pp. 7–16. ISBN 0-89791-884-3.

MARK, W. 1999. *Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping*. PhD thesis, University of North Carolina at Chapel Hill.

MATUSIK, W., BUEHLER, C., RASKAR, R., GORTLER, S. J., AND MCMILLAN, L. 2000. Image-based visual hulls. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, pp. 369–374. ISBN 1-58113-208-5.

MCMILLAN, L., AND BISHOP, G. 1995. Plenoptic modeling: An image-based rendering system. In *Proceedings of SIGGRAPH 1995*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 39–46. ISBN 0-201-84776-0.

MCMILLAN, L. 1995. A list-priority rendering algorithm for redisplaying projected surfaces. Tech. Rep. TR95-005, 14,.

MCMILLAN, L. 1997. *An Image-based Approach to Three-Dimensional Computer Graphics*. PhD thesis, U. of North Carolina, Chapel Hill.

MGI. http://www.mgisoft.com/products/webtools/panorama/.

MORTENSEN, E. N., AND BARRETT, W. A. 1995. Intelligent scissors for image composition. In *Proceedings of SIGGRAPH 1995*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 191–198. ISBN 0-201-84776-0.

OF NORTH AMERICA, I. E. S. 2000. *The IESNA Lighting Handbook, Reference and Applications*, 9th ed. IESNA.

OH, B. M., CHEN, M., DORSEY, J., AND DURAND, F. 2001. Image-based modeling and photo editing. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, pp. 433–442. ISBN 1-58113-292-1.

OH, B. M., DURAND, F., AND DORSEY, J. 2002. Structure-preserving clone brushing. *Submitted for publication.*

PALMER, S. 1999. *Vision Science : Photons to Phenomenology.* MIT Press.

PERONA, P., AND MALIK, J. 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence 12*, 7 (July), pp. 629–639.

PHANTOM. http://www.sensable.com.

PHOTOMODELER. http://www.photomodeler.com.

PHOTOSHOP. Adobe photoshop. http://www.adobe.com/products/photoshop.

PIXOLOGIC. http://www.pixologic.com.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. In *Computer Graphics (Proceedings of SIGGRAPH 1984)*, vol. 18, pp. 253–259.

POULIN, P., OUIMET, M., AND FRASSON, M.-C. 1998. Interactively modeling with photogrammetry. In *Eurographics Rendering Workshop 1998*, Springer Wein / Eurographics, Vienna, Austria, pp. 93–104. ISBN 3-211-83213-0.

PRESS, W., S.TEUKOLSKY, VETTERLING, W., AND FLANNERY, B. 1992. *Numerical Recipes*, 2nd ed. Cambridge University Press.

PULLI, K., COHEN, M., DUCHAMP, T., HOPPE, H., SHAPIRO, L., AND STUETZLE, W. 1997. View-based rendering: Visualizing real objects from scanned range and color data. In *Rendering Techniques 1997 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, Springer Wien, New York, NY, J. Dorsey and P. Slusallek, Eds., pp. 23–34.

RANGASWAMY, S. 1998. *Interactive Editing Tools for Image-Based Rendering Systems.* Master's thesis, Massachuesetts Institute of Technology.

REALVIZ. Image modeler. http://www.realviz.com.

RIGHTHEMISPHERE. http://www.deeppaint3d.com.

RUZON, M. A., AND TOMASI, C. 2000. Alpha estimation in natural images. *IEEE Computer Vision and Pattern Recognition.*

SEGAL, M., KOROBKIN, C., VAN WIDENFELT, R., FORAN, J., AND HAEBERLI, P. E. 1992. Fast shadows and lighting effects using texture mapping. In *Computer Graphics (Proceedings of SIGGRAPH 1992)*, vol. 26, pp. 249–252. ISBN 0-201-51585-7.

SEITZ, S. M., AND DYER, C. R. 1997. Photorealistic scene reconstruction by voxel coloring. In *Proc. Computer Vision and Pattern Recognition Conf.*, pp. 1067–1073.

SEITZ, S., AND KUTULAKOS, K. 1998. Plenoptic image editing. In *Proc. 6th Int. Conf. on Computer Vision*, pp. 17–24.

SEMPLE, J., AND KNEEBONE, G. 1952. Algebraic projective geometry.

SETHIAN, J. A. 1999. *Level Set Methods and Fast Marching Methods.* Cambridge University Press, Cambridge, UK.

SHADE, J., GORTLER, S. J., WEI HE, L., AND SZELISKI, R. 1998. Layered depth images. In *Proceedings of SIGGRAPH 1998*, ACM SIGGRAPH / Addison Wesley, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, pp. 231–242. ISBN 0-89791-999-8.

SHOUP, R., 1972. http://www.alvyray.com/Awards/AwardsMain.htm.

SILLION, F. X., AND DRETTAKIS, G. 1995. Feature-based control of visibility error: A multi-resolution clustering algorithm for global illumination. In *Proceedings of SIGGRAPH 1995*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 145–152. ISBN 0-201-84776-0.

SMITH, A. R., 1975. http://www.alvyray.com/Awards/AwardsMain.htm.

TAYLOR, C., AND KRIEGMAN, D., 1992. Structure and motion from line segments in multiple images.

TOLBA, O., DORSEY, J., AND MCMILLAN, L. 2001. A projective drawing system. In *2001 ACM Symposium on Interactive 3D Graphics*, pp. 25–34. ISBN 1-58113-292-1.

TOLBA, O. 2001. *A Projective Approach to Computer-Aided Drawing*. PhD thesis, Massachusetts Institute of Technology.

TOMASI, C., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. In *IEEE International Conference on Computer Vision*, pp. 838–846.

TUMBLIN, J., AND TURK, G. 1999. Lcis: A boundary hierarchy for detail-preserving contrast reduction. In *Proceedings of SIGGRAPH 1999*, ACM SIGGRAPH / Addison Wesley Longman, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 83–90. ISBN 0-20148-560-5.

WANG, J. Y. A., AND ADELSON, E. H. 1994. Representing moving images with layers. *IEEE Trans. on Image Processing 3*, 5, pp. 625–638.

WARD, G. 1991. Real pixels. In *Graphics Gems II*. Academic Press, Boston, pp. 80–83. ISBN 0-12-064481-9.

WARD, G. J. 1994. The radiance lighting simulation and rendering system. In *Proceedings of SIGGRAPH 1994*, ACM SIGGRAPH / ACM Press, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, 459–472. ISBN 0-89791-667-0.

WILLIAMS, L. 1990. 3d paint. In *1990 Symposium on Interactive 3D Graphics*, vol. 24, pp. 225–233. ISBN 0-89791-351-5.

WILLIAMS, L., 1998. Image jets, level sets and silhouettes. Workshop on Image-Based Modeling and Rendering, http://www-graphics.stanford.edu/workshops/ibr98/Talks/Lance/silhouettes.html, March.

WYSZECKI, G., AND STYLES, W. S. 1982. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley, New York.

YANG, J. C., AND MCMILLAN, L., 2002. Synthetic aperture camera array. http://graphics.lcs.mit.edu/ jcyang/CameraArray/cameraarray.htm.

YU, Y., AND MALIK, J. 1998. Recovering photometric properties of architectural scenes from photographs. In *Proceedings of SIGGRAPH 1998*, ACM SIGGRAPH / Addison Wesley, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, pp. 207–218. ISBN 0-89791-999-8.

YU, Y., DEBEVEC, P., MALIK, J., AND HAWKINS, T. 1999. Inverse global illumination: Recovering reflectance models of real scenes from photographs. In *Proceedings of SIGGRAPH 1999*, ACM SIGGRAPH / Addison Wesley Longman, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, pp. 215–224. ISBN 0-20148-560-5.

ZHANG, L., DUGAS-PHOCION, G., SAMSON, J.-S., AND SEITZ, S. 2001. Single view modeling of free-form scenes. *IEEE Computer Vision and Pattern Recognition.*

ZHANG, H. 1999. A derivation of image-based rendering for conventional three-dimensional graphics. *Journal of Graphics Tools 4*, 2, pp. 27–36. ISSN 1086-7651.