View-Dependent Pixel Coloring – A Physically-Based Approach for 2D View Synthesis

by

Ruigang Yang

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2003

Approved by:

Greg Welch, Advisor

Anselmo Lastra, Reader

Guido Gerig, Reader

Henry Fuchs, Committee Member

Gary Bishop, Committee Member

Kostas Daniilidis, Committee Member

ABSTRACT

RUIGANG YANG: View-Dependent Pixel Coloring – A Physically-Based Approach for 2D View Synthesis.
(Under the direction of Greg Welch.)

The basic goal of traditional computer graphics is to generate 2D images of a synthetic scene represented by a 3D analytical model. When it comes to real scenes however, one usually does not have a 3D model. If however one has access to 2D images of the scene gathered from a few cameras, one can use view synthesis techniques to generate 2D images from various viewing angles between and around the cameras.

In this dissertation I introduce a fully automatic, physically-based framework for view synthesis that I call *View-dependent Pixel Coloring* (VDPC). VDPC uses a hybrid approach that estimates the most likely color for every picture element of an image from the desired view, while simultaneously estimating a view-dependent 3D model of the scene. By taking into account a variety of factors including object occlusions, surface geometry and materials, and lighting, VDPC has produced superior results under some very challenging conditions—in particular—in the presence of textureless regions and specular highlights, conditions that cause conventional approaches to fail.

In addition, VDPC can be implemented on commodity graphics hardware under certain simplifying assumptions. The basic idea is to use texture-mapping functions to warp the input images to the desired view point, and use programable pixel rendering functions to decide the most consistent color for each pixel in the output image. By exploiting the fast speed and tremendous amount of parallelism inherent in today's graphics board, one can achieve real-time, on-line view synthesis of a dynamic scene.

ACKNOWLEDGMENTS

Many thanks to my advisor Greg Welch, for making my PhD journey so rich and valuable. In addition to the inspiration and guidance throughout my study, I deeply admire his insistence on excellence and appreciate his care about the students.

I also like to thank my committee members: Anselmo Lastra, Guido Gerig, Henry Fuchs, Gary Bishop, and Kostas Daniilidis. This dissertation could not have be completed without Anselmo and Guido's many useful suggestions to the manuscript. Thanks to Henry and Gary for their vision and insights. Kostas's advice on computer vision topics is invaluable for this dissertation.

I owe tremendous debts of gratitude to many people in this department or out. In particular, I wish to thank Herman Towles for his support in almost all aspects of my research; Marc Pollefeys for contributing some of the original ideas; and Zhengyou Zhang for showing me the wonderful and challenging world of computer vision.

My deepest gratitude goes to the people who have had such a significant impact on my life. My parents have always stood by me, and they have taught me everything I truly value as important. My wife, Huahong, made my PhD journey possible with her unconditional support and love.

Finally, I dedicate this dissertation to my new born son, Evan, who has been and continues to be my constant source of joys and motivations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

View synthesis addresses the problem of generating images of a scene from arbitrary viewing angles. In the Image-Based Modeling and Rendering community, the term *view synthesis* has come to mean to the generation of new images from input images. It differs from traditional computer graphics rendering in that traditional rendering typically assumes a 3D model is known and given *a priori* as input. View synthesis, by contrast, typically has only a number of 2D images as input.

In this dissertation I introduce a new fully automatic, physically-based framework for view synthesis, that I call *View-dependent Pixel Coloring* (VDPC). It is designed to allow direct view synthesis, i.e., assigning a color value for every pixel in the desired view, while maintaining a view-dependent 3D model. By processing a set of input images of a scene gathered from cameras in different positions, VDPC allows continuous viewing of the scene on a computer, during which a user can explore the scene from different angles and/or distances by controlling a virtual camera that can be moved continuously throughout the environment imaged by the cameras.

1.1 Motivation and Goals

View synthesis can be used in many applications. For example, a real-estate agent can use view synthesis techniques to show many houses to her potential customers, allowing them to "visit" each house without leaving her office. Imagine how useful this tool would be if the agent only needs to take a few images of each house instead of getting a full 3D model of it. Or a flight simulation system can use view synthesis techniques to create a *live* airport environment for the trainees using live video streams from a real airport, instead of using canned and often crude animations of 3D models. Ideally, one would

like to have one view synthesis technique that can be used in all situations. However, different characteristics in different scenarios often lead to different, sometimes even contradictory, requirements that hardly any single method can satisfy all of them. For the scope of this dissertation, I concentrate on two technically challenging application areas: interactive visualization of surgical procedures and *Tele-immersion*.

View synthesis techniques allow interactive visualization of three-dimensional objects, which is particularly valuable in teaching and training applications. Because of the ability to change the point of view, interactive visualization provides a much richer user experience than 2D imagery taken from fixed points of view. Benefits include better depth perception and better understanding of spatial relationships. In some applications, rendering new views is relatively simple because a complete scene model is readily available. This is often the case in architectural walkthroughs and virtual prototyping. But there are many applications where the only available information is a few images of the objects of interest. One such technologically challenging application is teaching surgical management of difficult, potentially lethal, injuries (traumas). These events are unpredictable and too complex to be modeled manually. But reconstructing them algorithmically poses significant challenges. As shown in Figure 1.1, many assumptions commonly found in most 3D reconstruction algorithms, such as rich textures and diffuse surfaces, are violated in a surgical environment. One goal of this dissertation is to be able to synthesize novel views under these difficult conditions.

Another driving application for this dissertation is *Tele-immersion*. The basic idea of Tele-immersion is to enable users at geographically distributed sites to collaborate in real time in a shared, simulated environment as if they were in the same physical room. Among the many challenges for tele-immersion is view synthesis. A real-time solution, coupled with stereo immersive display techniques, would allow a user to look at a remote scene from different viewing angles, creating a sense of immersion as if she were physically with the other participants.

In pursuit of these applications, this dissertation research began with the following goals:

1. use a practical number of images as input (around one or two dozens);
2. be fully automatic;
3. be robust and accurate for a wide variety of shapes and surfaces (convex, concave, specular, diffuse, etc.); and
4. work toward real-time, on-line view synthesis (over 10 fps).

In the next section, I describe how no existing method can simultaneously achieve

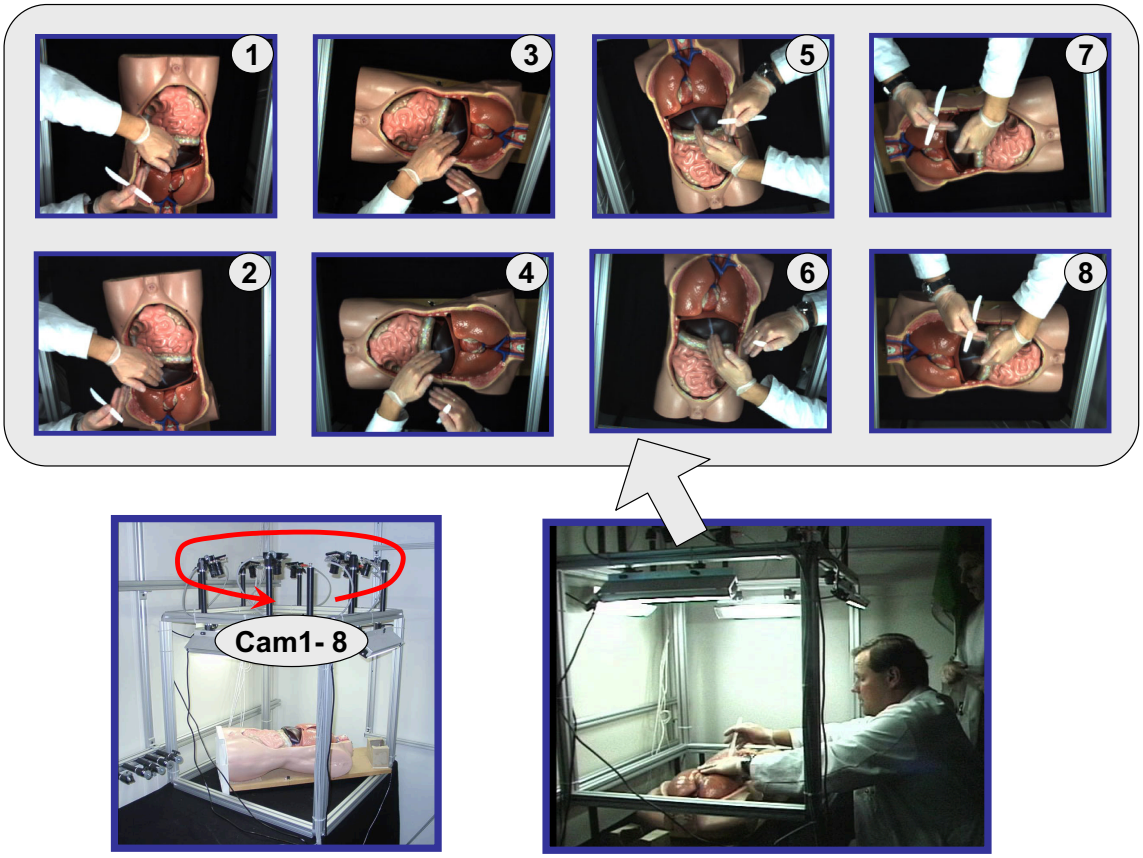


Figure 1.1: A captured scene in a simulated surgical environment. Lower left: *Camera cube*—our scene capture system with eight video cameras on the top, looking down. Lower right: Dr. Bruce Cairns at UNC Medical School was explaining a delicate surgical procedure, which was being captured by the camera cube at the same time. Top: eight images captured simultaneously.

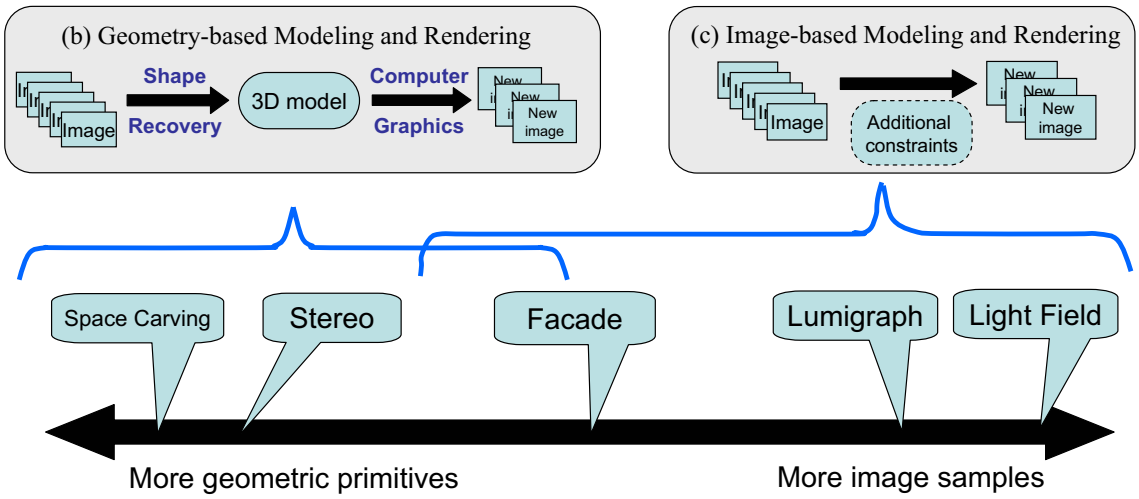
all of these goals.

1.2 A Brief Overview of Existing Methods

In pursuit of a view synthesis method that will satisfy all of the goals set in the previous section, let me first present a brief overview of what has been done for view synthesis in general. A more extensive review of previous work is presented in Chapter 2.

There are many different ways to categorize and introduce the vast variety of existing view synthesis methods. For example, they can be categorized based on the type of scenes they can handle, the number of input images required, or the level of automation. In this dissertation, I choose to categorize existing methods based their

internal representations of the scene. Based on this criterion, there is a continuum of view synthesis methods shown in Figure 1.2. They vary on the dependency of images samples vs. geometric primitives.



(a) The continuum of view synthesis methods

Figure 1.2: The continuum for view synthesis methods. Relative positions of some well-known methods are indicated. The two sub-figures show two main sub-groups for view synthesis. Note that the boundary between these two groups is blurry.

I categorize approaches on the left side of the continuum as *geometry based*. As shown in Figure 1.2(b), given a set of input images, a 3D model is extracted, either manually or algorithmically, and can then be rendered from novel viewing angles using computer graphics rendering techniques. In this category, the primary challenge is in the creation of the 3D model. Automatic extraction of 3D models from images has been one of the central research topics in the field of *computer vision* for decades. Although many algorithms and techniques exist, such as the extensively studied stereo vision techniques, they are relatively fragile and prone to error in practice. For instance, most 3D reconstruction algorithms assume a Lambertian (diffuse) scene, which is only a rough approximation of real-world surfaces.

By contrast, I categorize approaches on the right side as *Image-based Modeling and Rendering* (IBMR)—a popular alternative for view synthesis in recent years. As shown in Figure 1.2(c), the basic idea is to synthesize new images directly from input images, partly or completely bypassing the intermediate 3D model. In other words, IBMR methods typically represent the scene as a collection of images, optionally augmented with additional information for view synthesis. Light Field Rendering (LFR) [LH96,

GGSC96] represents one extreme of such techniques; it uses many images (hundreds or even thousands) to construct a light field function that completely characterizes the flow of light through unobstructed space in a scene. Synthesizing different views becomes a simple lookup of the light field function. This method works for any scene and any surface: the synthesized images are usually so realistic that they are barely distinguishable from real photos. But the success of this method ultimately depends on having a very high sampling rate, and the process of capturing, storing, and retrieving many samples from a real environment can be difficult or even impossible.

In the middle are some hybrid methods that represent the scene as a combination of images samples and geometrical information. Typically they require a few input images as well as some additional information about the scene, usually in the form of approximate geometric knowledge or correspondence information. By using this information to set constraints, the input images can be correctly warped to generate novel views. To avoid the difficult shape recovery problem, successful techniques usually require a human operator to be involved in the process and use a priori domain knowledge to constrain the problem. Because of the required user interaction, these techniques are typically categorized under the IBMR paradigm. For example, in the successful Façade system [DTM96] designed to model and render architecture from photographs, an operator first manually places simple 3-D primitives in rough positions and specifies the corresponding features in the input images. The system automatically optimizes the location and shape of the 3D primitives, taking advantage of the regularity and symmetry in architectures. Once a 3D model is generated, new views can be synthesized using traditional computer graphics rendering techniques.

View synthesis is an active research topic in both the computer graphics and computer vision communities; many new algorithms and techniques are developed every year. Among the existing view synthesis methods, there is no single one that is clearly better than the rest. Certain trade-offs have to be made to select the best method for a desired application. Factors to consider include the desired quality of synthesized images, computational cost, number of input images, and level of automation. Unfortunately, no existing method can satisfy all of the goals set out in the previous section (see Table 2.1 in the end of Chapter 2 for detail).

LFR, for example, meets all of the goals except the first one—using a practical number of images. LFR can generate new images with a quality almost indistinguishable from real photographs, but it typically requires thousands of input images. Stereo vision techniques, by contrast, require only two images, are usually fully automatic, and

can be computed in real time using special hardware or carefully optimized code. But they are also known to be quite fragile in practice and thus fail to meet the third goal. There are many other methods that use a reasonable number of images but require the user to assist the reconstruction process; they do not meet the second and the last goals. For example, the *View Morphing* technique allows very realistic results to be obtained using only two images, or even a single one in some cases. However, a user is required to establish correspondences between a pair of images, which is effectively equivalent to constructing a 3D model by hand—a tedious and time-consuming task that is impractical for real-time applications.

The lack of a existing view synthesis method to meet all of the goals in Section 1.1 prompted this dissertation research, which was aimed at combining the advantages of existing methods while addressing their shortcomings.

1.3 A Brief Historical Note

Before I introduce my dissertation work, let me first present a brief historical note on how I arrived at my dissertation (a more detailed account can be found in Section 5.2).

My first work on view synthesis was motivated by the 3D Tele-Immersion project (http://www.advanced.org/tele-immersion/) in 1999, in which a real-time solution for view synthesis was essential. Together with Prof. Greg Welch, my thesis advisor, we made the observation that for the sake of view synthesis, we did not need to estimate a full 3D model and then render the 3D model to obtain a 2D image, we only needed to estimate a color for each pixel in the desired image. In the following years, I developed a view-dependent formulation for view synthesis that can be accelerated on commodity graphics hardware to achieve real-time performance. The basic idea is to use texture-mapping functions in graphics hardware to warp the input images to the desired view point, and use programable pixel functions to decide the most consistent color for each pixel in the output image. At that time, I called the approach *Sparse Light Field Rendering* (SLFR) in a spirit similar to Light Field Rendering because I was attempting to synthesize novel views directly, bypassing the intermediate 3D model but with fewer images.

While Sparse Light Field Rendering achieved the desired real-time performance, I found a number of complications. It turned out that our first thought that I could completely bypass a 3D geometric model was not entirely correct. Without a rather accurate 3D model, the quality of synthesized images degraded rapidly as the viewpoint

moved away from any of the input viewpoints because of sampling artifacts and occlusion problems. Consequently, the effective view volume was rather limited. Around that time, in collaboration with Brown University, we began a project to visualize surgical procedures (http://www.cs.unc.edu/Research/stc/projects/ebooks/ebooks.htm). Experiments showed that the extensive amount of textureless regions and specular highlights in a surgical scene significantly impacted the quality of images SLFR could synthesize.

At that time, I realized that an image-geometry hybrid approach for view synthesis was probably more appropriate. While my primary goal was 2D image synthesis as opposed to 3D scene modeling, I realized that some amount of 3D scene modeling could lead to a better 2D image. The question was how much understanding of the 3D scene was necessary. I realized that if the 3D scene model was estimated specifically from the perspective of the desired view, relatively little 3D scene understanding was necessary. Further by estimating the 3D scene information from the perspective of the desired view, I could apply spatial constraints that were specifically tailored to the desired view. These realizations led me to a hybrid approach of 2D image synthesis using "just enough" 3D scene information, estimated from the perspective of the desired view. This hybrid formulation, together with novel constraints to improve reconstruction qualities, formed the foundation of this dissertation. I now call the complete approach *View-Dependent Pixel Coloring* (VDPC).

1.4 View-Dependent Pixel Coloring

In this dissertation I introduce a view-synthesis framework, *View-Dependent Pixel Coloring* (VDPC), which meets the goals set out in Section 1.1. In this section, I first briefly provide a high level overview of VDPC, explaining how VDPC is formulated. Then I present the *thesis statement*, followed by a section in which I highlight key innovations.

1.4.1 Approach

VDPC uses a hybrid view-dependent approach. That is, it is designed for *direct* view synthesis while maintaining a *view-dependent* 3D model. For a desired viewpoint C^*, the 3D volume enclosing the objects of interest is discretized into a perspective grid of voxels—volume elements (Figure 1.3 shows a bird's-eye view of a horizontal slice of the

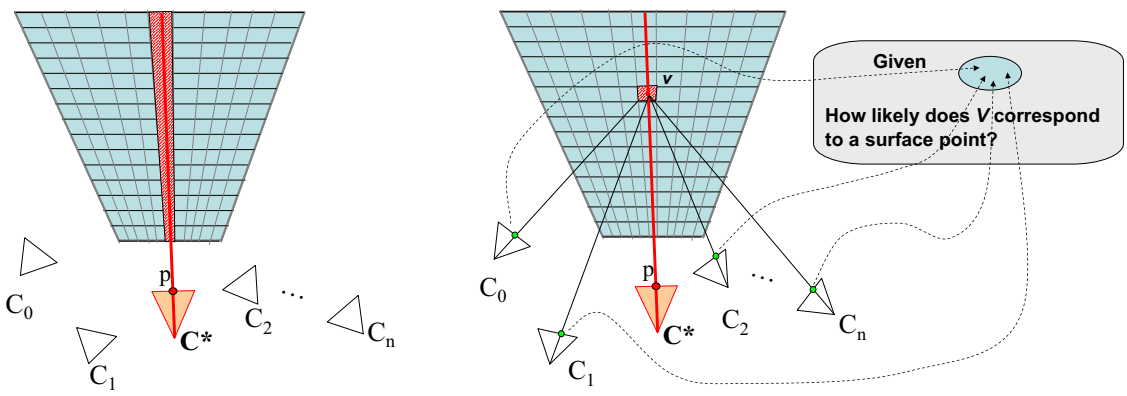


Figure 1.3: The basic formulation of VDPC. $C_0 ... C_n$ represent the viewpoints of input images, while C^* represents the output viewpoint. VDPC uses a view-dependent parametrization of the 3D space—a perspective voxel grid. Left: a bird's-eye view of a horizontal slice of the voxel grid, which corresponds to a scan line in the output image. Right: VDPC determines a voxel's occupancy by its projections in input images.

voxel grid). Each column of voxels (the slant-shaded voxels in Figure 1.3) corresponds to a single pixel in the desired image to be synthesized for C^*. Initially, the volume is filled with voxels. The goal of VDPC is to assign a color to each pixel in the desired image. In order to do that, VDPC attempts to "carve" away voxels in empty space and assign colors to voxels representing scene surfaces. Basically, VDPC examines each voxel v by looking at its projections in all input images. If v represents a surface point in the scene, then the pixels in its projections should be *consistent*—all having the same color if the surface is diffuse. If v is consistent with the input images, it will be assigned a color and remain in the view-dependent 3D model. Otherwise, it will be carved away.

Because of the one-to-one mapping between voxel columns and pixels in the desired image, each pixel's color is set to the color of the first opaque voxel in each column. In the end, VDPC produces a desired image, as well as a surface voxel model from the perspective of the desired viewpoint.

VDPC incorporates a progressive scheme to perform the carving and find the most likely color for each pixel, taking into account occlusions, local shape smoothness, illuminations, and surface materials. This scheme is discussed in Section 3.1.2.

Two novel physically-based constraints are incorporated into this framework. The first is a view-dependent smoothness constraint. Many real world scenes are composed of smooth surfaces almost everywhere except at surface boundaries. This observation can be formulated as a smoothness constraint to provide a more accurate estimation of the scene. The smoothness constraint used in this dissertation is based on the principle

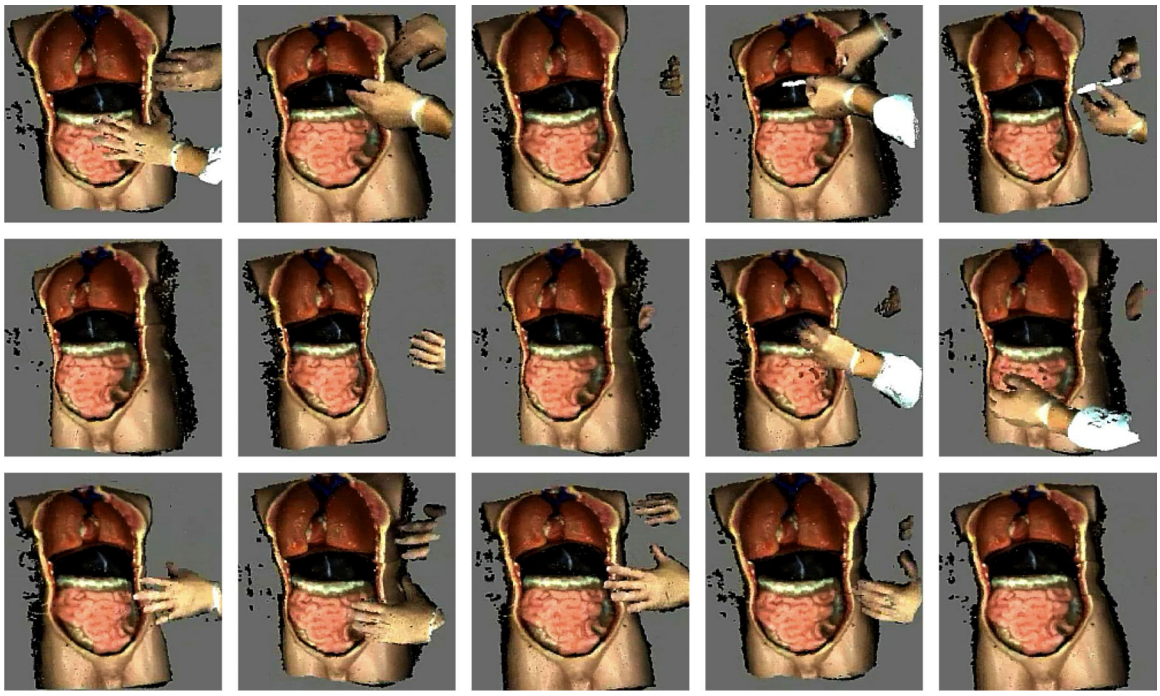


Figure 1.4: Typical results of VDPC: these images are synthesized from eight synchronized video streams captured using the system shown in Figure 1.1, in which a surgeon was explaining a dedicated procedure for trauma management.

of *disparity gradient limit*, which I discuss in detail in Section 3.1.3.

The second constraint is a novel photo-consistency measure, i.e., a measure of color and occupancy likelihood of a voxel given a number of pixel values collected from input images $C_0 ... C_n$. Based on previous studies in photometry, I designed a novel physically-based photo-consistency measure that is valid for both specular and Lambertian surfaces. Details can be found in Section 3.1.4.

In addition, if we choose to make certain simplifying assumptions about the scene, the VDPC framework can be efficiently implemented on commodity graphics hardware, allowing interactive viewing of a dynamic real scene in real time, on line. I present a reduction of VDPC and its implementation on graphics hardware in Chapter 4.

1.4.2 Thesis Statement

> By estimating a *view-dependent* 3D model using physically-based constraints, one can synthesize better 2D images of real scenes than conventional alternatives, such as full 3D reconstruction or 2D image warping. In the process of synthesizing 2D images one can also obtain a local 3D geometric model. In addition, given simplifying assumptions about the scene, the entire framework can be implemented using commodity graphics hardware to achieve on-line, real-time image synthesis of dynamic real scenes.

1.4.3 Innovations

The VDPC framework is comprised of the following conceptual and design innovations, which allow VDPC to meet the goals set in Section 1.1.

Hybrid and view-dependent estimation. The primary goal of view synthesis is to render or predict one or more output views from a set of input images. On the one hand, if we could acquire a priori either enough image samples (as in Light Field Rendering) or additional constraints (such as correspondence information in view morphing), we could simply warp the input images to synthesize new views without understanding the underlying geometry, materials, or lighting of the scene. However, neither requirement can be met for the scope of this dissertation (see goal (1) and (2) on page 2.).

On the other hand, we could first estimate a full 3D model then render the 3D model from new viewing angles. However, estimating a full 3D model automatically is a difficult task. Existing methods are typically fragile and prone to error in practice. In addition, such an approach is less efficient for dynamic scenes (goal (4)) since usually only part of a full 3D mode is visible for any given viewpoint, the effort to estimate the occluded part is wasted. For the sake of view synthesis, a complete model is unnecessary, we only need to extract enough information to allow us to predict the immediately necessary output view. In other words, a *partial* model may suffice.

There are previous hybrid approaches that make use of the above observations. For example, the *view-dependent texture mapping* technique represents a scene as an approximate geometric model with a set of input images gathered from different viewpoints [DTM96]. Multiple images are projected onto the basic model and blended at

run-time to better simulate geometric and photometric details not captured by the basic model.

Unlike previous hybrid approaches in which the 3D model is fixed and/or known a priori, I propose the notion of *view-dependent estimation*, which aims to construct a 3D model optimized for a given output view. I use this *view-dependent* model to directly improve the quality of the synthesized view.

Based on the above conceptual innovations, VDPC uses a view-dependent parameterization of 3D space (a perspective voxel grid), which allows a direct correspondence between the 3D model and the 2D image. This formulation has a number of advantages. For example, a voxel grid is relatively flexible, and can represent any shape—meeting the geometry part of goal (3). In addition, a view-dependent formulation avoids some common problems of view-independent ones, as detailed in Section 3.3.1.

Robust to textureless regions—a progressive refinement scheme with a view-dependent smoothness constraint. It is difficult to estimate shapes in regions lacking color variations—many possible ambiguous shapes exist. If not resolved correctly, artifacts will become obvious when the model is viewed from an oblique angle, far from any of the input viewpoints. A typical treatment is to apply a smoothness constraint that biases the surface estimation toward locally flat surfaces. However the typical accompanying *smoothing* effect undermines the ability to reconstruct highly complex shapes. Ideally, we want to apply a smoothness constraint *only* when there is ambiguity. To this end, VDPC adopts a *progressive* scheme. Starting from a few reliable voxels, VDPC incrementally refines the underlying shape estimate using photo consistency measures, progressively updated visibility information, and smoothness constraints.

I also recognize that surface smoothness perceived by humans is a *view-dependent* property and apply a smoothness constraint from the perspective of the output viewpoint. The smoothness constraint I use, the *Disparity Gradient Limit*, is based on psychophysical studies in human vision system, which provides a physically meaningful way to fine-tune the parameters since the synthesized images are for humans to view.

This innovation allows VDPC to meet part of goal (3), providing better shape estimates for textureless regions. It is discussed further in Section 3.3.1,

Robust to specular highlights—a novel physically-based photo-consistency measure. Almost all existing 3D reconstruction methods make a relatively simplistic assumption that all scene surfaces are diffuse: light reflected from the same surface

point to various viewing angles is assumed to be the same. While useful as a basis for establishing correspondence, this property does not hold in practice for specular surfaces. There have been some attempts to deal with specular surfaces. They either treat specular highlights as outliers, or require a full calibration of the light source. Based on previous studies in photometry that recognized some distinctive distributions of light reflected from certain surfaces, I have designed a novel physically-based photo-consistency measure that addresses these problems. Using this new measure, I have been able to successfully reconstruct highly specular objects, like the scene in Figure 1.1. In addition, this measure does not require light calibration or surface normal estimation, thus it can be used in many existing reconstruction algorithms to find correspondences in the presence of specular highlight.

This innovation allows VDPC to meet part of goal (3), providing better shape estimates for specular regions. It is discussed further in Section 3.3.1.

The recognition of the potential power in commodity graphics hardware and its application in VDPC. Modern commodity graphics hardware is becoming increasingly powerful and programmable. While most hardware improvements were originally designed for rendering, i.e., generating images from a given geometric and photometric model, I recognized the potential power of using commodity graphics hardware for view synthesis. There is a great synergy between image processing and graphics rendering. Both frequently use simple operations applied thousands or even millions of times. Any application having similar characteristics can exploit the inherent Single-Instruction-Multiple-Data (SIMD) parallel architecture in modern graphics boards to accelerate their computation. Others have recognized this potential power for other uses, for example, matrix multiplication [LM01].

Based on the above recognition, I present in Chapter 4 a reduction of VDPC, real-time VDPC, that can be efficiently implemented on existing commodity graphics hardware to meet goal (4). By exploiting the enormous internal bandwidth and computational power in today's graphics cards, I have been able to develop a real-time on-line system for view synthesis *and* depth estimation (a depth map), thanks to the hybrid nature of our formulation. At the time of this writing (May 2003), my implementation, when operating to output a depth map, is as fast as the fastest commercial software package available. Furthermore, since all the processing is done entirely on the graphics hardware, the CPU can be spared to perform other tasks.

1.5 Dissertation Outline

The remainder of my dissertation is organized as follows:

In Chapter 2, I review related work in detail. I categorize existing view synthesis methods into two classes: *geometry based* and *image based*. They are reviewed in Section 2.1 and Section 2.2, respectively. In addition, with the increasing computational power on inexpensive personal computers, several real-time methods have been proposed to capture and render dynamic live scenes. They are of particular interest to this dissertation and are reviewed in Section 2.3. This chapter ends with a discussion of the major problems with existing methods that prompted this dissertation research.

For the sake of clarity, as well as attempting to cater to different interests among different readers, I break up the VDPC framework into two chapters, one for the complete VDPC framework and the other for a reduction of VDPC that can achieve real-time performance using commodity graphics hardware.

In Chapter 3, I present the View-Dependent Pixel Coloring framework (VDPC). First, the approach VDPC uses is presented in Section 3.1 and Section 3.2. A discussion of VDPC, including its innovations and limitations, is in Section 3.3. This chapter ends with experimental results from various data sets with difference geometry and material properties.

A full implementation of VDPC in software is not real-time yet— each synthesized image presented in Chapter 3 typically requires a few minutes to render on a PC. But in Chapter 4, I present a special reduction of VDPC, real-time VDPC, that can be efficiently implemented on commodity graphics hardware. Modern graphics hardware offers orders of magnitude in acceleration, enabling real-time on-line view synthesis of a live dynamic scene.

I conclude this dissertation in Chapter 5 with some historical notes of this dissertation research and possible directions for future work. Appendix A provides some sample code for real-time VDPC introduced in Chapter 4.

Readers interested in implementing the full VDPC framework could start at Section 3.1 in which I explain VDPC in detail and continue to Section 3.2 in which I discuss some specific implementation issues. Readers looking for the real-time view synthesis method could jump directly to Chapter 4, which is a self-contained chapter describing real-time VDPC.

Chapter 2

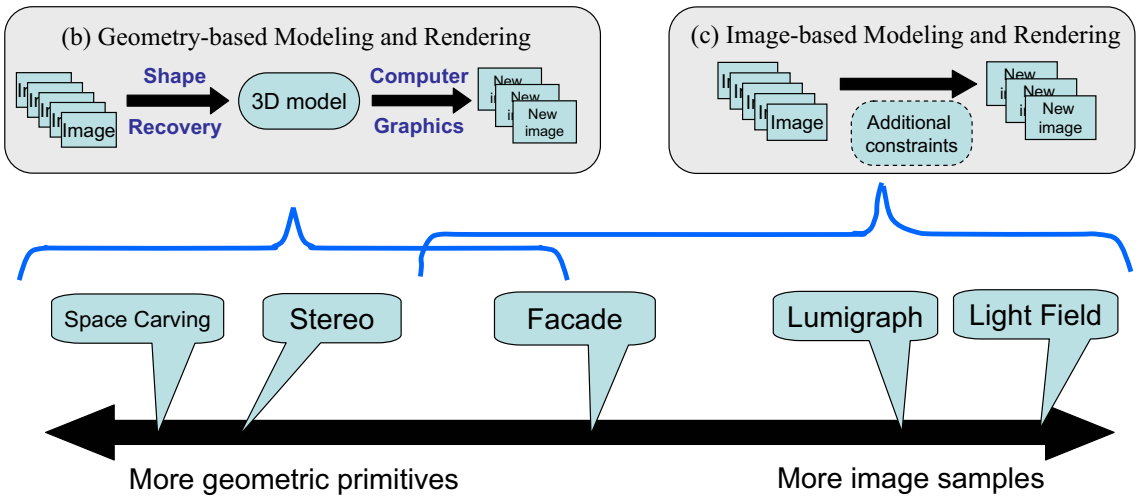
Background and Related Work

In this chapter, I will set the context of this dissertation and introduce several related previous approaches. Being able to look freely through a scene has long been an active research topic in the computer graphics community. Historically, computer graphics research has been focused on *rendering*. That is, *given* a 3D model, how to generate new images faster, better, and more realistically. *View synthesis* addresses a typically more challenging problem. It is aimed to generate new images using only a set of 2D images, instead of 3D models.

There are many different ways to categorize and introduce the vast variety of existing view synthesis methods. For example, they can be categorized based on generality, speed, the number of input images required, or the level of automation. In this dissertation, I chose to categorize existing methods based on their internal representations of the scene. Based on this criterion, there is a continuum of view synthesis methods as shown in Figure 1.2 and repeated in Figure 2.1. They vary on the dependency of images samples vs. geometric primitives.

Roughly, approaches on the left side of the continuum are *geometry based*. As shown in Figure 2.1(b), given a set of input images, a 3D model is extracted, either manually or algorithmically; then the 3D model can be rendered from novel viewing angles using computer graphics techniques. With this type of approach, automatic extraction of 3D shapes from images is a very difficult task. It has been one of the central research topics in computer vision for decades. I will survey a number of computer vision techniques in Section 2.1. They range from basic stereovision techniques requiring only two images to more advanced techniques such as *Space Carving*, which builds a complete 3D model from multiple images.

A recent, popular alternative to geometry-based approaches is *Image-based Modeling*



(a) The continuum of view synthesis methods

Figure 2.1: The continuum for view synthesis methods. Relative positions of some well-known methods are indicated. Each of them will be reviewed in detail in this chapter.

and Rendering, which resides roughly on the right side of the continuum. As shown in Figure 2.1(c), the basic idea is to synthesize new images directly from input images, bypassing the intermediate 3D model. I will review some of these methods in Section 2.2.

With the increasing computational power on inexpensive personal computers and the wide availability of low-cost imaging devices, several real-time methods have been proposed to capture and render dynamic scene. They are of particular interest to this dissertation and are reviewed in Section 2.3.

At the end of this chapter (Section 2.4), I discuss some of the major problems with existing methods that prompted this dissertation research.

2.1 3D Shape Recovery

3D reconstruction of an unknown scene from a set of images is one of the oldest problems in computer vision. One major driving application is autonomous robots, which need to understand the shape of the scene to navigate throughout. Other applications include surveillance, reverse engineering, and human-computer interactions. 3D reconstruction has been and continues to be one of the most active research areas in computer vision; many methods and techniques have been tried and tested. These

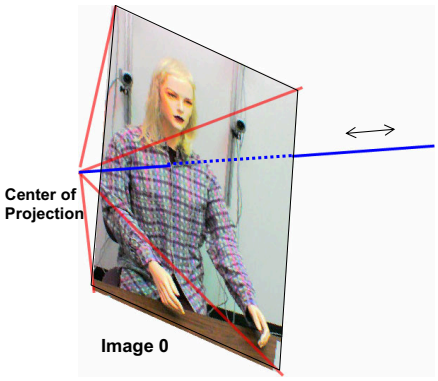


Figure 2.2: Depth from a single image is undefined.

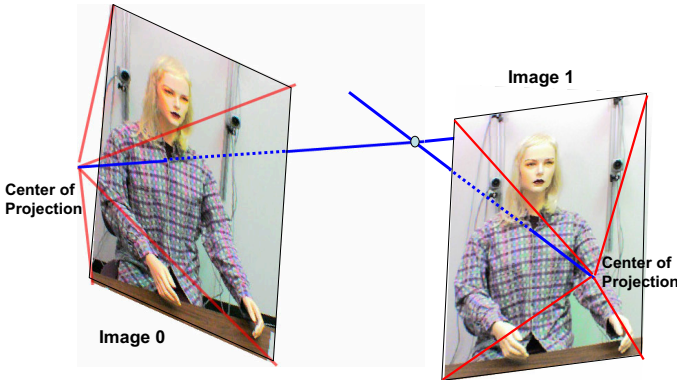


Figure 2.3: Depth from two (or more) images can be determined if correspondences are known.

methods vary vastly as to the image features used and the underlying scene representation. I review two categories of methods: correspondence-based stereo methods and volumetric reconstruction methods. They are the most widely-used methods and are most relevant to this dissertation. Interested readers may refer to computer vision textbooks [TV98, Fau93, KSK98, Hor86, Dav97] for other shape recovery methods, such as shape from shading, sparse feature-tracking based methods, or active sensing techniques.

2.1.1 Stereo Vision Methods

Stereo vision attempts to infer 3D shapes from images. While a single image can tell us a lot about an observed scene, it does not contain enough information to reconstruct the scene's 3D geometry. This is due to the nature of the image formation process, which consists of a projection from a three-dimensional scene onto a two-dimensional

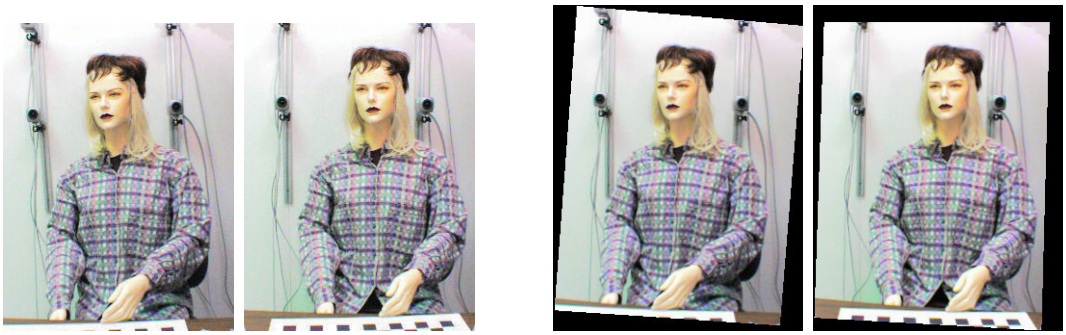


Figure 2.4: A pair of stereo images before and after rectification. The first two are the original images, while the last two are the rectified images. Note that the corresponding features are on the same row of pixels after rectification.

image plane. During this process, the depth is lost [Pol98]. Figure 2.2 illustrates this. From a single image, we know only that the three-dimensional point corresponding to a specific image pixel is constrained to be on the associated line of sight; it is impossible to determine which point on this line corresponds to the image pixel. If two (or more) images are available, then, as can be seen from Figure 2.3, the intersection of the two lines of sight can uniquely determine a three-dimensional point. This process is called *triangulation*. In order to reconstruct a three-dimensional point, we must know the following:

- Corresponding image points (pixels);
- Relative positions and orientations of the cameras from the different views (*extrinsic* camera parameters);
- Relation between image pixels and the corresponding lines of sight (*intrinsic* camera parameters).

While there are several techniques for reconstructing a 3D model with unknown, or partially unknown, camera parameters [TK92, DTM96, PKG98, PG97, LZ99a], I limit the scope of this dissertation, as in many existing reconstruction techniques, to calibrated cases. That is, both the intrinsic and extrinsic camera parameters are known *a priori*. The central problem for stereo vision thus becomes finding the correspondences between two images.

The earliest attempt to solve the stereo problem, by Marr and Poggio, dates back to 1976 [MP76]. Since then, stereo matching has been one of the most active research areas in computer vision. Compiling a *complete* survey of existing correspondence-based stereo methods would be a formidable task. A large number of new methods

are published every year. However, there are a few excellent survey papers collectively covering the history of stereo vision. There are two very good reviews of early vision; the first is by Barnard and Fischler [BF82] and covers the early 70s and 80s. The second is by Dhond and Aggarwal [DA89] and covers the late 80s. More recently, Scharstein and Szeliski [SS02] updated us on the current state of art. They also introduced a taxonomy of two-view, or *binocular*, stereo algorithms that allows the dissection and comparison of individual algorithm component design decisions. First I will provide a more rigorous definition of the binocular stereo problem (Section 2.1.1), and then I will use the taxonomy from Scharstein and Szeliski to review a number of binocular stereo algorithms (Section 2.1.1). Several metrics for quantitative evaluations of binocular stereo algorithms are discussed in Section 2.1.1. I further examine several multi-view stereo methods that use more than two images, in Section 2.1.1.

Binocular Stereo Representation

Most binocular stereo correspondence methods compute a univalued *disparity* function $d(u, v)$ with respect to one of two reference images. The term *disparity* was first introduced in the human vision literature to describe the difference in location of corresponding features seen by the left and right eyes [Mar82]. For a 2D feature s in one reference image, say the left image, its corresponding 3D point is constrained to be on the line of sight (Figure 2.2). This line's projection in the other image is called the *epipolar line*. Thus the corresponding feature s' in the right image must be on the epipolar line. This is the important *epipolar constraint* which reduces the search space of corresponding features to one dimension. In computer vision, input images are usually transformed so that the epipolar lines are aligned horizontally. This process is called *rectification* [AH88, PD96, LZ99b, PKG99][1]. A pair of stereo images before and after rectification is shown in Figure 2.4. For a pair of rectified images, disparity can be treated as synonymous with inverse depth, i.e., a large disparity value means that the 3D point is close, a small disparity value means that the 3D point is further away, and a zero disparity value means that the 3D point is at infinity.

Given a pair of rectified images, let (u, v) be the pixel coordinates in a reference image chosen from the pair, say the left image. The correspondence between a pixel

[1] Readers are referred to the classic textbook by Faugeras [Fau93] for a detailed explanation of these important concepts.

(u, v) in a reference image and a pixel (u', v') in matching image is then given as

$$u' = u + d(u, v), v' = v.$$

The goal of a stereo algorithm is then to produce a univalued function $d(u, v)$ that best describes the shape of the surfaces in the scene [SS02].

A Taxonomy of Stereo Algorithms

Scharstein and Szeliski [SS02] propose a taxonomy based on the observation that stereo algorithms generally perform (subsets of) the following four steps:

1. matching cost computation;
2. cost (support) aggregation;
3. disparity computation / optimization; and
4. disparity refinement.

Matching cost computation A matching cost is a value indicating how likely two pixels are to correspond to the same scene point. The three most common pixel-based matching costs include squared intensity differences (SD) [Han74, Ana89, MSK89], absolute intensity differences (AD) [Kan94], and normalized cross-correlation [Han74, RGH80, BBH93]. They all behave similarly in terms of disambiguating power [SS02].

Besides the above three, there are many other cost criteria that are designed for specific needs. Some costs are insensitive to differences in camera gain or bias; these include, for example, gradient-based measures [Sei89, Sch94] and non-parametric measures such as rank and census transforms [ZW94]. More recently, robust measures such as truncated quadratics and contaminated Gaussians [BA93, BR96, SS98] have been introduced to limit the influence of mismatches during cost aggregation(the next step). Other cost criteria include phase and filter-bank responses and the sample-insensitive cost measure developed by Birchfield and Tomasi [BT98].

All these cost measures assume the scene surfaces are Lambertian, i.e., that their appearance does not vary with viewpoint. This poses a significant restriction on the type of scenes a stereo algorithm is able to reconstruct. Later in this dissertation (Section 3.1.4), I will introduce a novel matching cost that is valid for both specular and Lambertian surfaces.

Matching cost aggregation The matching cost for each pixel is usually ambiguous and noisy. To reduce ambiguity, many stereo algorithms use a local and window-based approach: matching costs are aggregated by summing or averaging over a *support region*. Most stereo algorithms use a two-dimensional support region at a fixed disparity, as in [Arn83, BI99, OK92, KO94a, KSC01, O.V01, BVZ98]. Such approaches favor front-parallel surfaces. By contrast, a three-dimensional support region in x-y-d space, as in [PMF85, Pra85, Gri85], does not have this bias, thus supporting both slanted and front-parallel surfaces.

A different method of aggregation is iterative diffusion, i.e., an aggregation (or averaging) operation that is implemented by repeatedly adding to each pixel's cost the weighted values of its neighboring pixels' costs [SH85, Sha93, SS98]. The work presented in this dissertation uses a similar iterative strategy in applying a view-dependent smoothness constraint (Section 3.1.2).

Disparity computation and optimization There are two broad classes of methods used at this stage, *local* methods and *global* methods. In local methods, the disparity computation at a given pixel depends only on the intensity values within a finite window. Computing the final disparities is trivial: one simply chooses for each pixel, the disparity associated with the minimum cost value. Thus, these methods perform a local "winner-take-all" (WTA) optimization for each pixel. A major limitation is that the uniqueness of matches is enforced for only one image (the reference image), while pixels in the other image might have multiple matches.

Global methods, by contrast, are usually formulated within an energy-minimization framework [PTK85]. The objective is to find a disparity function d that minimizes the global energy, such as:

$$E(d) = E_{data}(d) + \alpha E_{smooth}(d) + \beta E_{visibility}(d). \tag{2.1}$$

The data term, $E_{data}(d)$, measures how well the disparity function d agrees with the input image pair. The smoothness term, $E_{smooth}(d)$, encodes the smoothness assumptions made by the algorithm. To make the optimization computationally tractable, the smoothness term is often restricted to measuring only the differences between neighboring pixels' disparities. The last visibility term is often a bi-valued term—zero if the visibility constraint is satisfied, infinity otherwise.

Once the global energy has been defined, a variety of algorithms can be used to find

a (local) minimum. Traditional approaches associated with regularization and Markov Random Fields include continuation [BZ87], simulated annealing [GG84b, MMP87, Bar89], highest confidence first [CB90], and mean-field annealing [GG84a]. More recently, max-flow and graph-cut methods have been proposed to solve a special class of global optimization problems [RC98, IG98, BVZ01, Vek99, KZ01, BGCM02]. Each of these methods constructs a graph such that the maximum flow or minimum cut on the graph also minimizes the energy. These approaches are more efficient than simulated annealing and have produced good results. Kolmogorov and Zabih have characterized the energy functions that can be minimized by graph-cut [KZ02b].

While the optimization of Equation 2.1 can be shown to be NP-hard for common classes of smoothness functions [Vek99], dynamic programming can be used to find the global minimum for each scanline independently in polynomial time [Bel96, BM92, GLY92, CHRM96, BI99]. A major limitation of this approach is the difficulty of enforcing inter-scanline consistency.

While global methods tends to produce better results than local methods, they are typically sensitive to the precise definition of the global energy. Usually these parameters are tuned empirically, and parameters that work well for one data set may not necessarily be good for others. In contrast, local methods typically have fewer parameters and generate consistent, albeit not optimal, results. Local methods are also computationally feasible for real-time implementations.

Disparities refinement Most stereo algorithms compute a disparity map using only integer values. To improve quality, the disparity values can be interpolated for sub-pixel accuracy. Usually, the profile of matching costs for a pixel is fitted into a parabolic curve, then the local maximum is found as the refined disparity value [LK81, TH86, MSK89, KO94b, MID02]. This increases the resolution of the disparity map with little computation.

Other types of disparity post-processing are also possible, such as applying a median filter to remove spurious mismatches or filling holes due to occlusion using surface fitting or distributing neighboring disparity estimates [BT98, Sch99].

Stereo Quality Measures

As there are so many stereo algorithms, several different criteria have been developed for evaluating their performance. Evaluation is a challenging task because it is difficult to obtain the "ground truth" depth data.

Leclerc et.al. [LLF98] proposed a novel methodology that estimates the accuracy and reliability of the results of any multiple-image point correspondence algorithm, without the need for ground truth. The key concept behind their methodology is what they call the *self-consistency* of an algorithm across independent experimental trials. In other words, for a given collection of N input images ($N > 2$), apply the algorithm being evaluated to subsets of the input images and check the consistency of the outputs from these independent runs. This *self-consistency* test does not require knowledge of the "ground truth", which makes it more applicable to images captured from real scenes. However, it should be noted that the self-consistency test is only a necessary condition for the correctness of a point correspondence algorithm, though the authors explained in their paper why they conjectured the proposed test provides a reasonable approximation of the absolute error distributions.

Recently, *Image-Based Rendering* techniques have gained popularity, and two papers [MID02, Sze99] have developed new criteria for evaluating the performance of dense stereo algorithms from a view synthesis standpoint. The basic idea is to measure how accurately the depth estimates (combined with the original image) can *predict* the appearance of an image from a *novel* view. That is, render the depth map using one of the original images as a texture, and compare the synthesized image with a real image taken from the same viewpoint. Since this metric simply compares two images, it is much easier to collect data sets for which algorithms can be evaluated. However, view-dependent effects, such as specular highlights, undermine the accuracy of image-based metrics. Even if there is no view-dependent effect, there is still the difficulty of measuring the fidelity of synthesized images. The usual method is to compute the root mean square (RMS) difference between the two images. However, it is known that the RMS error can be easily fooled. In [Sze99], the authors used *residue flow* as the error metric. Residue flow uses a conjunction of the RMS intensity error and the per-pixel offset (flow) to register the rendered image with the actual image. Details are in their paper.

More recently, Scharstein and Szeliski [SS02] provided a very comprehensive quantitative evaluation of stereo algorithms. Using several stereo data sets with ground truth, they evaluated a large set of today's best-performing stereo algorithms. They also set up a web site to allow researchers to submit new algorithms and results to be evaluated. Interested readers are encouraged to visit `www.middlebury.edu/stereo` for the most up-to-date evaluation results. Note that while some of their data sets include multiple frames, their evaluation is intended solely for binocular stereo algorithms.

Multi-view Stereo

While there has been significant progress on stereo algorithms during the last two decades, from simplistic but fast local methods to sophisticated global methods based on energy minimization, the problem of computing depth from two images has not been entirely solved. Indeed, some even deem it an ill-posed problem in general [MMP87, PTK85, Bas92]. There simply is not enough information to distinguish correct correspondences from false positives in many practical cases. Various constraints and assumptions have to be imposed to make the problem tractable.

To ameliorate the above problems, Okutomi and Kanade in 1993 proposed the use of more than two images in stereo [OK93]. Using one of the input images as the reference image, the matching costs from all the other images are summed up to a final matching cost. This new cost is more salient to image noise and false positives. Since the search for matches is still performed within the disparity space, there is a major restriction on the camera arrangement: the cameras have to be co-planar or even co-linear.

In 1996, Collins [Col96] proposed a plane-sweep algorithm, which projects all images onto a series of planes in 3D space that correspond to different disparity values. Matching is performed in 3D space. This allows more flexible camera configurations. But an important problem—occlusion—is still not addressed. This is particular important in the multi-view case, since as more and more cameras are added, it becomes less and less likely that all the cameras will see the same surface. In the next section, I will introduce several reconstruction methods based on a volumetric representation of the scene. These methods can deal elegantly with the multi-view reconstruction problem.

2.1.2 Volumetric Methods

Instead of searching in image space as in stereo algorithms, an alternative approach to scene reconstruction is based on computations in three-dimensional scene space, in which a volumetric representation of the scene can be inferred from input images. Volumetric methods usually assume there is a known, bounded volume in which the objects of interest lie. The most common approach to representing this volume is as a regular tessellation of cubes, called *voxels*, in Euclidean 3D space. The task of a reconstruction algorithm is to decide which voxels belong to the objects of interest and which do not. Most methods also assign a color to each voxel based on its projections into input images.

Compared to the disparity space representation used in most stereo algorithms, a

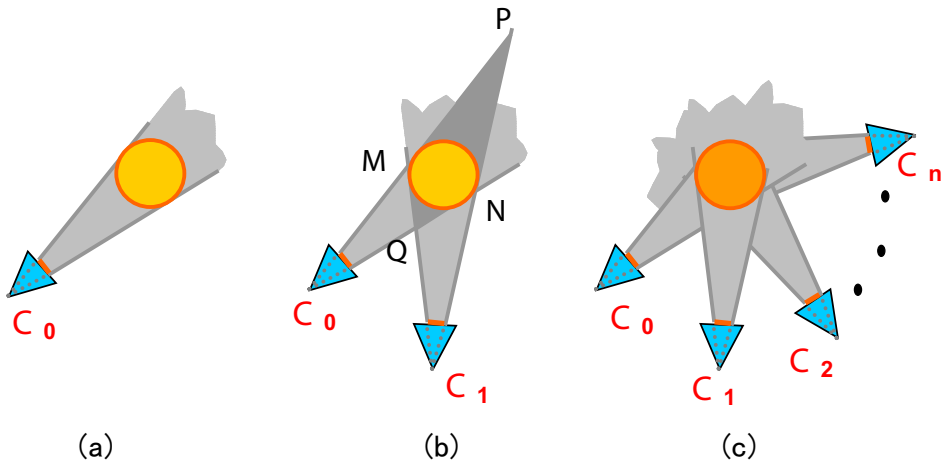


Figure 2.5: 2D Visual Hull. From left to right, (a) A single camera defines a cone area in space containing the foreground object; (b) Two cameras refine the intersection region (MPNQ); (c) As the number of cameras goes to infinity, the intersection region converges to the object's visual hull.

volumetric representation is much more flexible and general. In particular, it allows a much wider range of camera arrangements, and more elegant modeling of visibility in 3D scene space.

Recently there has been considerable progress in developing techniques that build volumetric scene models. There are two broad classes of volumetric reconstruction methods; one uses only silhouette information to compute the *visual hull* of the original shape; while the other uses *photo-consistency* measures to compute the *photo hull* of the original shape. I will review these two classes in the sections that follow. Interested readers are also encouraged to read two recent reviews of different volumetric scene reconstruction methods by Slabaugh et al. [SCMS01] and Dyer [Dye01], respectively.

Shape from Silhouettes—*Visual Hull* Reconstruction

An object's contour (or profile) provides important clues about the object shape. Suppose a 3D object is viewed by a camera. The object's silhouette image, which can be obtained using segmentation algorithms or blue-screen techniques, contains values that distinguish regions where the object is or is not present. Combined with calibration information for the camera, each pixel in a silhouette defines a ray in scene space that intersects the object at some unknown depth. The union of these visual rays for all pixels in the silhouette defines a generalized cone within which the 3D object must lie, as in the shaded area in Figure 2.5(a). If we are presented with multiple views of the

object, the intersection of these generalized cones from all views defines a volume of the scene space that must contain the original object. As the number of the reference views goes to infinity to include all the views possible from all locations, the intersection volume converges to the shape known as the object's *visual hull*, a term defined by Laurentini [Lau94]. The visual hull is guaranteed to contain the object. In 2D, the visual hull is equal to the convex hull of the object. For 3D scenes, the visual hull is a tighter fit than the convex hull. In a visual hull, hyperbolic regions are removed, but concavities are not.

In practice, since one has access to only a finite number of views, one can only construct approximate visual hulls. Given a set of n silhouette images from different views, the approximate visual hull is the best conservative geometric description one can achieve based on silhouette information alone. In Figure 2.5(b), I show an example of the volume (the dark-shaded area $MPNQ$) computed from only two views. From now on, I will use the term "visual hull" to mean the approximate volume computed from n views.

Central to visual hull reconstruction is the intersection test. If the silhouettes are described using a polygonal mesh, the visual hull representation can be constructed using a series of 3D constructive solid geometry (CSG) intersections [PS85]. But it is well known that polyhedral CSG operations are very hard to perform in a robust manner due to numerical inaccuracy [LTH86].

A more common approach is to reconstruct a quantized representation of the visual hull [Lau94, Lau95, Lau97, AV89, MA83, MBR+00, Pot87, Nie97, NFA88, SA90, Sze93, MKJ96, CKBH00, SVZ00]. Starting from a bounding volume that is known to enclose the entire scene, the volume is discretized into voxels. Voxels falling outside of the back-projected silhouette cone of any given view are eliminated from the volume. This can be done efficiently by projecting each voxel into 2D images and testing whether it is contained in every silhouette. In the end, only voxels that are in the intersections of back-projected silhouette cones from all views are retained.

To make the voxel traversal more efficient, most methods use an octree representation and test voxels in a coarse-to-fine hierarchy [CA86, Pot87, SA90, Sze93]. The volume enclosing the entire scene space is initialized to a single voxel. The current voxel is projected into all the views and tested to determine whether it is inside the silhouette in each view. If the projected voxel is outside the silhouette in at least one view, the voxel is removed (marked transparent). If the projected voxel is inside the silhouette in every view, the voxel is retained (marked opaque). Otherwise, the voxel

intersects both background and silhouette points in some views, so it is subdivided into octants and each sub-voxel is processed recursively. This process terminates when no subdivision is necessary or when the size of the subdivided voxels reaches a user-defined threshold.

As in the energy minimization framework for stereo reconstruction (see Equation 2.1), Snow et al. formulated the shape-from-silhouette problem as an optimization problem in order to find a global minimum for an energy function [SVZ00]. In their formulation, the data term of the energy function is based on the images' intensities. In general, the cost is high if there are large intensity differences among a voxel's projections in different views. The smoothness term controls the degree of spatial smoothness; there is a penalty for assigning different labels (opaque/transparent) to a pair of adjacent voxels. Because of the introduction of the smoothness term, this method allows more robust reconstruction under noisy conditions than other methods based on direct intersection tests.

Space Carving—*Photo Hull* Reconstruction

Shape-from-silhouettes methods avoid the difficult correspondence problem associated with stereo vision, and are thus quite robust if the segmented silhouette images are accurate. However, concavity cannot be preserved by the visual hull presentation. In practice, image segmentation is not always possible. In addition, the color information about the objects is not used in any shape-from-silhouettes method, except to assign color to a voxel model already reconstructed.

In 1998 Seitz presented a volumetric reconstruction method, *voxel coloring*, that makes full use of the photometric information contained in input images [SD99]. The basic idea is to reconstruct the 3D scene model that best reproduces the input images when rendered from the perspectives that correspond to the input images. Starting from a regular 3D voxel grid that encloses the scene, the goal is to assign colors and binary transparency values to voxels so as to achieve *photo-consistency* with a set of input images (shown in Figure 2.6). That is, rendering the colored voxels from each input viewpoint should reproduce the original image as closely as possible. Assuming a Lambertian scene, a voxel on the scene surface is photo-consistent with a set of images if, for each image in which it is visible, the voxel's color is equal to the color of its corresponding image pixel.

Conversely, in order to determine a voxel's photo-consistency, we can project it onto un-occluded images and test whether or not the pixels in its projection have equal

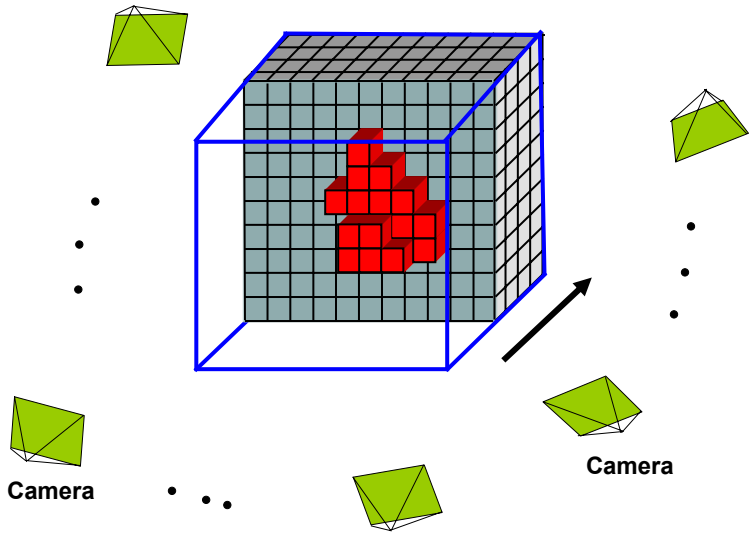


Figure 2.6: Volumetric Reconstruction Using Photo Consistency

color values. In the presence of image noise or quantization effects, we can evaluate the correlation of the pixel colors to measure the likelihood of voxel consistency. Let s be the standard deviation of the pixel colors. One possibility is to threshold the color space error. If s is smaller than a user-defined threshold λ, the voxel is considered to be photo-consistent and is assigned a color—the mean of the pixel colors. Otherwise, the voxel is considered to belong to the free space and is marked as transparent. Alternatively, a statistical measure of voxel consistency can be used [SD99]. This has been done using an F test [Joh00], where a photo-consistency score is computed by the ratio of the variances of pixels' colors and the colors of pixels associated with a known homogeneous surface. A threshold on this score determines the photo-consistency of the voxel. Experiments using other definitions of photo-consistency have also been conducted [BC00, KS00, SCMS01, Chh01].

Note that the photo-consistency test should only be applied to these *visible* pixels. To avoid a combinational search of all visibility configurations, efficient methods for determining the visibility of each voxel are essential. A voxel y can occlude a voxel x if and only if y intersects at least one of the line segments connecting x to the optical centers associated with input views. If there exists a topological sort of voxels in which occluders must be before the voxels that are being occluded, i.e., if y occurs before x in the ordering, then the visibility test becomes tractable. We can traverse voxels in that order and guarantee that when a voxel is visited, all possible occluders for this voxel have been visited. Seitz pointed out that such an order, which is referred to as

the *occlusion-compatible* order, does exist whenever no scene point is contained within the convex hull of the camera centers [SD99]. For instance, if the cameras and the scene are separated by a plane, voxels can be sorted by increasing distance from the plane, resulting in a sequence of voxel planes. We can traverse these planes from near to far, and only voxels in the previous planes, which have been processed and labelled, can occlude the voxels in the current plane, enabling a single-pass algorithm. With an appropriate photo-consistency measure and a voxel grid that has been sorted in an occlusion-compatible order—for instance, a sequence of voxel planes for simplicity—the complete voxel coloring algorithm can be outlined in Algorithm 1.

Algorithm 1 Pseudo code for the voxel coloring algorithm.

```
// Iterate through the planes
for (i = 1 to n) {
    //Iterate through voxels in the plane D_i
    for every voxel v in D_i {
        Ψ = ∅ ; // Ψ is the visible pixel set
        for (k = 1 to m) {// m is the number of input images
            Project v to image I^k
            for each pixel p_j^k in v's footprint in I^k
                if (p_j^k is not marked) { // still visible
                    Ψ = Ψ ∪ p_j^k
                }
        }
        s = consistency(Ψ);
        if (s > λ) { // λ is the photo-consistency threshold
            v = transparent
        } else {
            v = mean(Ψ);
            mark p ∈ Ψ; // keep track of pixels that have been used;
        }
    } // end of the voxel loop within a plane
} // end of the voxel plane loop
```

In Algorithm 1, the threshold λ is supplied by the user, depending on the noise level of the input images and the voxel resolution, and Ψ is the set of visible pixels for a voxel v. Initially, all pixels in all input images are unmarked. When a voxel is declared consistent, the pixels of its projection in the input images will be marked. They will not participate in subsequent photo-consistency tests. Because of the occlusion-compatible order for voxel evaluation, it is guaranteed that all pixels in Ψ are visible from v.

The voxel coloring algorithm is most closely related to the plane-sweeping algorithm first proposed by Collins [Col96]. However, with its explicit modeling and handling of visibility, the voxel coloring algorithm allows more flexible camera configurations and typically generates superior reconstruction results, especially for highly complex scenes in which occlusions and dis-occlusions frequently occur.

In 1999, Kutulakos and Seitz [KS00] extended the voxel color algorithm to allow even more flexible camera configurations. In essence, all views are partitioned into subgroups, so that within each subgroup, an occlusion-compatible order exists. The basic voxel coloring algorithm is applied successively for each subgroup to refine the voxel model from previous passes. They called this method the *Space Carving* algorithm.

Culbertson et al. [CMS99] presented another variation of the basic voxel coloring algorithm that also allows more flexible camera placement. At the expense of computer memory, a visible list of surface voxels for every pixel in every image is maintained. Thus visibility for a voxel can be quickly determined using all input images without the need for an occlusion- compatible order.

To deal with practical issues such as inaccurate calibration, Kutulakos [Kut00] defined an *Approximate Space Carving* algorithm using a weakened photo-consistency test. Each voxel v is projected onto every input image at $p_1, \ldots, p_n$, respectively. If there is a common color for pixels within a small radius of p_i in all input images, that voxel v is declared to be photo-consistent. While this approach was designed to recover shapes from images with inaccurate calibration information, it can also be used to build a series of coarse-to-fine scene models from multiple views.

As always, voxel reconstruction can also be formulated as an energy minimization problem. Slabaugh et al. [SCMS00] used an iterative method, which they presented as a post-processing step, to add or remove surface voxels until the sum of the squared differences between the input images and the scene model rendered in each camera was minimized. Simulated annealing and greedy methods were used for optimization. This refinement effectively produces a spatially varying consistency threshold. More recently, Kolmogorov and Zabih [KZ02a] used graph-cut to optimize volume reconstruction directly. In order to make optimization tractable, the visibility test was approximated in their formulation. While strong results have been obtained for a few standard data sets with very small baselines (originally used for stereo), the effectiveness of their method is yet to be evaluated using other multi-view data sets in which visibility changes are substantial.

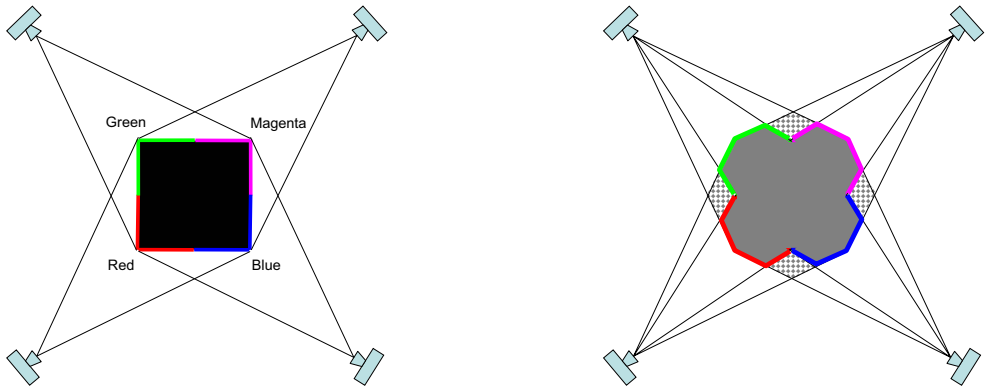


Figure 2.7: Illustration of the Photo Hull [KS00]. Left: the scene consists of a square whose sides are painted diffuse red, blue, magenta, and green. It is observed by four cameras. Right: the *photo hull* of the scene is the gray-shaded area. The photo hull's projection in every input image is indistinguishable from that of the true object. The union of the photo hull and these check-board-shaded areas is an approximation of the *visual hull*, which can be reconstructed using silhouette information only. The use of photometric information leads to a tighter bound of the true scene.

Photo Hull Theorem Kutulakos and Seitz in [KS00] made an important theoretical contribution to volumetric reconstruction methods based on photo-consistency. They presented the formal notion of the *photo hull*:

> For a given shape V in 3D space, there is a unique photo-consistent shape that subsumes, i.e., contains within its volume, all other photo-consistent shapes in V. This unique shape is called the photo hull.

An illustration of the photo hull theorem is shown in Figure 2.7. Without additional information or biases, the photo hull is the tightest bound based on direct comparison with images. They further proved that if the photo-consistency function is monotonic, their proposed space carving algorithm is guaranteed to generate the photo hull. The monotonicity of the consistency function is crucial to the theoretical correctness of the photo-hull theory. Kutulakos and Seitz formally expressed it as follows [KS00]:

> Given N input images, an algorithm $consist_K()$ is available that takes as input at least $K \leq N$ colors $col_1, \ldots, col_K$, K vectors $\xi_1, \ldots, \xi_k$, and the light source positions (non-Lambertian case), and decides whether it is possible for a single surface point to reflect light of color col_i in direction ξ_i simultaneously for all $i = 1, \ldots, K$.
>
> Furthermore, $consist_K()$ is assumed to be *monotonic*,

i.e., $consist_K(col_1, \ldots, col_j; \xi_1, \ldots, \xi_j)$ implies that
$consist_K(col_1, \ldots, col_{j-1}; \xi_1, \ldots, \xi_{j-1})$, for every permutation of $1, \ldots, j$.

It should be noted that in practice, *no monotonic consistency function has been discovered so far.*[2] The most widely used consistency function, which is based on sample variance, is not strictly monotonic, but is a fair approximation.

Probabilistic Space Carving The original space carving framework and its variations [KS00, SD99, CMS99, BC00] use a single global threshold to carve out the shape. While this technique is very efficient and easy to implement, there are two potential problems. First, the noise level and sampling artifacts in input images are not always the same. Too small a threshold may lead to over-carving, i.e., too many voxels are carved away. This could have a catastrophic ripple effect, since once a correct surface voxel is erroneously carved away, there will be no inside voxels that would pass the photo-consistency test. So an entire object could be lost. On the other hand, a large threshold would cause false-positive photo-consistency for voxels that are in front of the true surface. Secondly, once a decision is made about a voxel, there is no easy way to change it. Again, an erroneous decision in a early stage could have a significant impact on subsequent carving operations.

To overcome these problems, several probabilistic space carving methods have been proposed [dBV99, BDC01, AD01, BFK02]. Instead of making these "hard" decisions about voxels, each voxel is assigned a probability, computed by comparing the likelihoods of the voxel existing and not existing.

Bonet and Viola [dBV99] formulated the voxel reconstruction problem in a fashion similar to Computed Tomography (CT), assuming that the objects in the scene are all semi-transparent and that an image is a projection of *all* the objects within the volume. Starting from an initial voxel grid that is assumed to be semi-transparent, an iterative approach is used to refine the transparency and color values of voxels. A unique property of this method is that it can also model semi-transparent objects. The work of Agrawal and Davis [AD01] is very similar, except that they assumed an opaque scene and used different optimization procedures. Visibility is approximated in their formulation.

Broadhurst et al. [BDC01] presented a one-pass probabilistic method for use with data sets in which an occlusion-compatible order exists. Using a proximate function as

[2]The practical issue of monotonicity is a point conceded in [KS00]. Kutulakos discussed it only during his oral presentation at ICCV99 [KS99]. It is also discussed in [Bro01].

the a priori probability distribution function for visibility testing, they derived a closed-form solution to compute the probability of existence for voxels. Alternatively, Bhotika et al. [BFK02] chose to access a voxel's visibility exactly. To deal with the potentially combinational search for visibility configurations, they used a stochastic variant of the space carving algorithm. That is, a random configuration of the voxels is generated and evaluated for photo-consistency. This process is repeated many times to generate a set of photo hulls, and the most consistent one is selected as the final output. The results presented in [BFK02] were obtained through several hundred iterations.

In theory, a probabilistic formulation should consider *all* possible visibility configurations for a voxel. For a set of m images, there are 2^m possible configurations for each and every voxel. To avoid the combinatorial search, existing probabilistic methods either approximate the visibility tests based on heuristics or convenience, or solve them in a stochastic manner through hundreds of iterations. I believe that an accurate treatment of visibility is crucial for any multi-view reconstruction algorithm. If an efficient and accurate probabilistic model of visibility can be found, a probabilistic approach will be very promising.

2.2 Image-based Modeling and Rendering

Despite decades of research, 3D shape recovery remains one of the most difficult unsolved problems in computer vision. As computers have increased memory and bandwidth, computer graphics researchers have begun exploring an alternative approach to view synthesis—*Image-based Modeling and Rendering* (IBMR). Pioneered by McMillan and Bishop in 1995 [MB95, McM97], the basic idea is to use a large number of input images to (partly) circumvent the difficult 3D reconstruction problems described in Section 2.1. IBMR has been one of the most active research topics in computer graphics for the past few years.

Within the IBMR paradigm, some methods use many images to generate novel views without relying on geometric information [LH96, GGSC96, IPL98, CLF98, IMG00, SCG97, SH97, LKHZ98, SVSG01, SH99, SHSd00]. They are collectively called *Light-Field* style rendering techniques and are discussed in Section 2.2.1. Other methods use geometric constraints, in the form of correspondence or a crude geometric "proxy" for a full model, to correctly generate new views by warping a few input images [SD96, Che95, DBY98, DTM96, SGHS98, CBL99]. I call them *geometry-assisted* IBMR techniques; they are discussed in Section 2.2.3. Note that the line between these two categories is

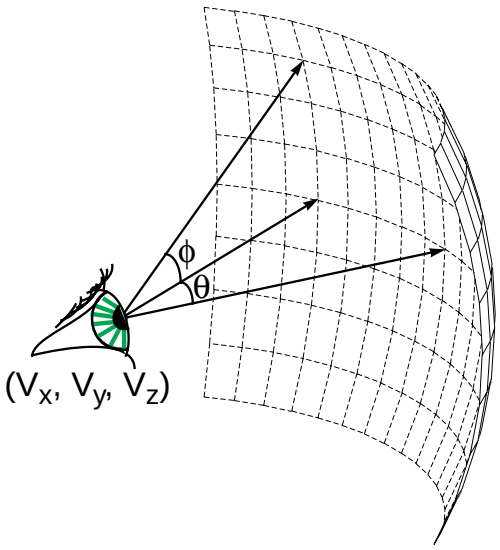


Figure 2.8: The plenoptic function describes all of the image information visible from any given viewing position (from [MB95] with permission).

rather vague. For example, the lumigraph method [GGSC96], which is usually classified in the first category, also uses rough geometry to reduce the artifacts that result from insufficient sampling.

2.2.1 Light-Field Style Rendering Techniques

Light-Field style rendering techniques are formulated around the *plenoptic function* (from the Latin root *plenus*, meaning complete or full, and *optic*, pertaining to vision). The notion of plenoptic functions was first proposed by Adelson and Bergen in 1991 [AB91]. A plenoptic function describes all of the radiant energy that can be perceived by an observer at any point in space and time. Adelson and Bergen formalized this functional description by providing a parameter space over which the plenoptic function is valid, as shown in Figure 2.8. Imagine we want to look at a scene freely. The location of an idealized eye can be at any point in space (V_x, V_y, V_z). From there one can perceive a bundle of rays defined by the range of the azimuth and elevation angles (θ, ϕ), as well as a band of wavelengths λ. The time t can also be selected if the scene is dynamic. This results in the following form for the plenoptic function:

$$\rho = P(\theta, \phi, \lambda, V_x, V_y, V_z, t) \tag{2.2}$$

Adelson and Bergen used this seven-dimensional function to develop a taxonomy for evaluating models of low-level vision. McMillan and Bishop first introduced the plenoptic function to the computer graphics community in 1995 [MB95]. They claimed that all image-based modeling and rendering approaches can be cast as attempts to reconstruct the plenoptic function from a sample set of that function.

From a computer graphics standpoint, one can consider a plenoptic function as a scene representation that describes the flow of light in all directions from any point at any time. In order to generate a view from a given point in a particular direction, one would merely need to plug in appropriate values for (V_x, V_y, V_z) and select from a range of (θ, ϕ, λ) for some constant t.

This plenoptic function framework provides many venues for exploration, such as the representation, optimal sampling, and reconstruction of the plenoptic function. In the following sections, I will discuss several popular parameterizations of the plenoptic function under varying degrees of simplification.

5D Light Field

Assuming that there is a static scene and that the effects of different wavelengths can be ignored, McMillan and Bishop [MB95, McM97] discussed the representation of the reduced 5D plenoptic function as a set of panoramic images at different 3D locations. In their implementation, computer vision techniques are employed to compute stereo disparities between panoramic images to avoid a dense sampling of the scene.[3] After the disparity images are computed, the input images can be interactively warped to new viewing positions. Visibility is resolved by forward-warping [Wes90] the panoramic images in a back-to-front order. This hidden-surface algorithm is a generalization of Anderson's visible line algorithm [And82] to arbitrary projected grid surfaces.

4D Light Field

Levoy and Hanrahan pointed out that the 5D representation offered by McMillan and Bishop can be reduced to 4D in free space (regions free of occluders[4]) [LH96]. This is

[3] I was inclined to put McMillan and Bishop's method in the second category, in which geometry information is used. However, their work is presented here, since the plenoptic function theory presented in their original paper paved the way for subsequent light-field rendering techniques.

[4] Such a reduction can be used to represent scenes and objects as long as there is no occluder between the desired viewpoint and the scene. In other words, the effective viewing volume must be *outside* the convex hull of the scene. Thus a 4D representation cannot be used, for example, in architecture workthroughs to explore from one room to another.

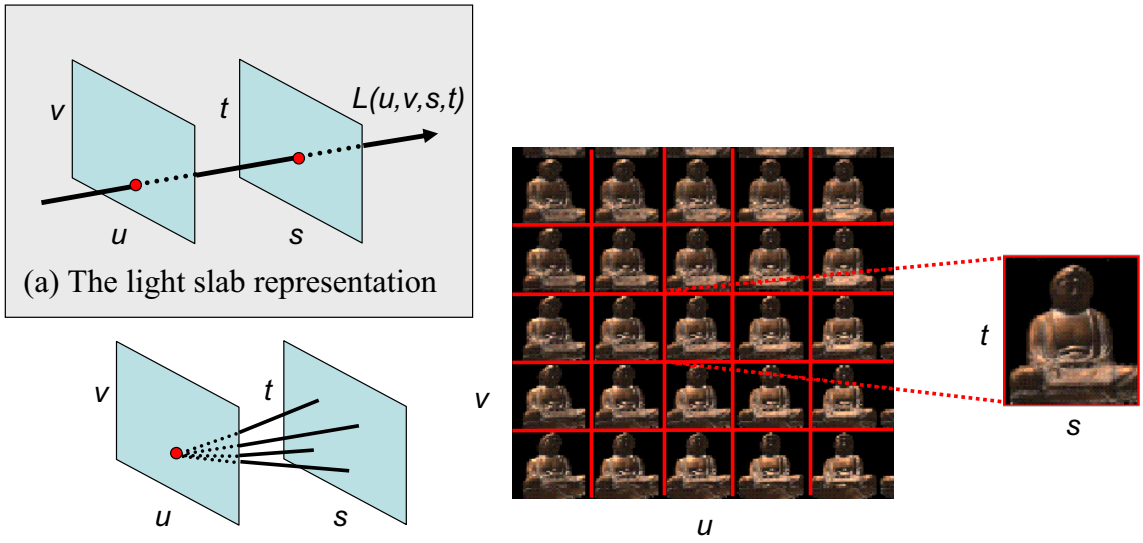


Figure 2.9: Light Field Rendering: the light slab representation and its construction (from [LH96]).

based on the observation that the radiance along a line does not change unless blocked. Levoy and Hanrahan called the 4D plenoptic function the *light field* function, which may be interpreted as a functions on the space of oriented light rays. Such reduction in dimensions has been used to simplify the representation of radiance emitted by luminaries [Lev71, Ash93].

Levoy and Hanrahan parameterize light rays based on their intersections with two planes(see Figure 2.9). The coordinate system is (u, v) on the first plane, and (s, t) on the second plane. An oriented light ray is defined by connecting a point on the uv plane to a point on the st plane. The authors called this representation a light slab. Intuitively, a light slab represents the beam of light entering one quadrilateral and exiting another quadrilateral.

To construct a light slab, one can simply take a 2D array of images. Each image can be considered a slice of the light slab with a fixed (u, v) coordinate and a range of (s, t) coordinates.

Generating a new image from a light field is quite different than previous view interpolation approaches. First, the new image is generally formed from many different pieces of the original input images, and need not look like any of them. Second, no model information, such as depth values or image correspondences, is needed to extract the image values. The second property is particularly attractive since automatically

extracting depth information from images is a very challenging task. However, many image samples are required to completely reconstruct the 4D light field functions. For example, to completely capture a small budda as shown in Figure 2.9, hundreds or even thousands of images are required. Obtaining so many images samples from a real scene may be difficult or even impossible. Gortler et al. [GGSC96] augmented the two-plane light slab presentation with a rough 3D geometric model that allows better quality reconstructions using fewer images. However, recovering even a rough geometric model raises the difficult 3D reconstruction problem.

Isaksen et al. [IMG00] presented a more flexible parameterization of the light field function. In essence, they allowed one of the two planes of the light slab to move. Because of this additional degree of freedom, their method can simulate a number of dynamic photograph effects, such as depth of field and apparent focus. Furthermore, this reparameterization technique makes it possible to create integral-photography-based [Oko76], auto-stereoscopic displays for direct viewing of light fields.

Besides the popular 2-plane parameterization of the plenoptic function, there is the spherical representation introduced by Ihm et al. [IPL98]. Their image-based rendering algorithm is different from previous systems. It is an object-space algorithm that can be easily embedded into the traditional polygonal rendering system. Thus their method can easily be accelerated by 3D graphics boards.

Light field rendering is the first image-based rendering method that does not require *any* geometric information about the scene. However, this advantage is acquired at the cost of many image samples. So what is the minimum number of samples required for light field rendering? That is a very important, yet difficult question—it involves complex relationships among various factors, such as the depth and texture variation of the scene, the input image resolutions, and the desired rendering resolutions. I will discuss this important problem in its own section, Section 2.2.2.

3D Plenoptic Function: Line Light Field and Concentric Mosaic

If we further constrain camera (viewing) motion to a continuous surface or curved manifold, the plenoptic function can be reduced to a 3D function. Sloan et al. [SCG97] derived a 3D parameterization of the plenoptic function. By constraining camera motion along a line, the 4D light field parameterization can be reduced to a 3D function. Such reduction is necessary for time-critical rendering given limited hardware resources. A (u, s, t) representation is used where u parameterizes the camera motion, and (s, t) parameterizes the other plane. Moving along the line provides parallax in the motion

direction. To achieve complete coverage of the object, the camera can move along four connected perpendicular lines, i.e., a square.

Shum and He presented an alternative parameterization for 3D plenoptic function called *concentric mosaics* [SH97]. They constrained camera motion to planar concentric circles, and created concentric mosaics by composing slit images taken at different locations along each circle. Concentric mosaics index all input image rays naturally according to three parameters: radius, rotation angle, and vertical elevation. Compared to a 4D light field, concentric mosaics have much smaller file sizes because only a 3D plenoptic function is constructed. Concentric mosaics allow a user to move freely in a circular region and observe significant parallax without recovering the geometric and photometric scene models.

Compared to Sloan's method, concentric mosaics offer uniform and continuous sampling of the scene. However, rendering with concentric mosaics could produce some distortions in the rendered images, such as bending of straight lines and distorted aspect ratios. Details about the causes of and possible corrections for these problems can be found in [SH97].

2D Plenoptic Function: Environment Map (Panoramas)

An environment map records the incident light arriving from all directions at a point. Under the plenoptic function framework, we can reconstruct from a single environment map a 2D plenoptic function in which only the gaze direction (θ, ϕ) varies. While the original use of environment maps was to efficiently approximate reflections of the environment on a surface [BN76, Gre86], Chen used environment maps to quickly display any outward-looking view of the environment from a fixed location but at a variable orientation. This is the basis of the Apple QuickTimeVR system [Che95]. In this system, environment maps are created at key locations in the scene. A user is able to navigate discretely from one location to another and, while at each location, continuously change the viewing direction.

While it is relatively easy to generate computer-generated environmental maps [Gre86], it is more difficult to capture panoramic images from real scenes. A number of techniques have been developed. Some use special hardware [Mee90, Tec, Nay97], such as panoramic cameras or cameras with parabolic mirrors; others use regular cameras to capture many images that cover the whole viewing space, then "stitch" them into a complete panoramic image [SS97, IAH95, MP94, Sze94, Sze96].

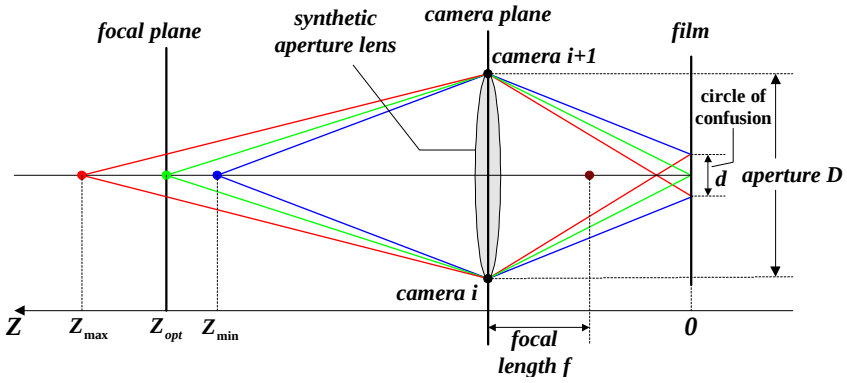


Figure 2.10: A discrete synthetic aperture optical system for light field rendering (from [CTCS00]).

2.2.2 Plenoptic Sampling

At the heart of light field rendering is the reconstruction of the plenoptic function from a set of image samples. In [CTCS00], Chai et al. asked the following important question:

> "How many samples of the plenoptic function (e.g., from a 4D light field) and how much geometrical and textural information are needed to generate a continuous representation of the plenoptic function?"

In their original LFR paper [LH96], Levoy and Hanrahan considered the light field rendering system to be a discrete synthetic aperture optical system [Hal94], as shown in Figure 2.10. There are a number of analyses using this geometric approach [CF99, LS00, FS00]. For the sake of completeness, a brief geometric analysis from [CTCS00] is quoted here.

> This analysis assumes uniform sampling geometry or lattice of the plenoptic function that is parameterized as a 4D light field function. Analogous to the Gaussian optical system, we can define the following optical parameters:
>
> - Focal length f;
> - Smallest resolvable feature (on the image plane) d;
> - Synthetic aperture D, i.e., distance between two adjacent cameras;
> - Circle of confusion $c = d/f$;
> - Hyperfocal distance $D_H = D/c$.

Let the plane of perfect focus be at the distance z_{opt} and the minimum and maximum distance at which the rendering is acceptable be z_{min} and z_{max}, respectively. The following relations exist([Bas95], vol. 1, p.192):

$$z_{min} = \frac{D_H z_{opt}}{D_H + z_{opt}}, \text{ and } z_{max} = \frac{D_H z_{opt}}{D_H - z_{opt}}, \tag{2.3}$$

which lean to

$$\frac{1}{z_{opt}} = (\frac{1}{z_{min}} + \frac{1}{z_{max}})/2 \frac{1}{D_H} = (\frac{1}{z_{min}} - \frac{1}{z_{max}})/2 \tag{2.4}$$

Therefore, to maintain the best rendering quality achievable, the focus should be always at z_{opt}. Moreover, to guarantee rendering without aliasing, D_H has to satisfy certain constraints,[5], i.e.,

$$\frac{d/f}{D} = (\frac{1}{z_{min}} - \frac{1}{z_{max}})/2 \tag{2.5}$$

In other words, given the minimum and maximum distances, the maximum camera spacing can be determined in order to meet the specified rendering quality. The hyperfocal distance describes the relationship among the rendering resolution (circle of confusion), the scene geometry (depth of field), and the number of images needed (synthetic aperture). Intuitively, the minimum sampling rate is equivalent to having the maximum disparity less than the smallest resolvable feature on the image plane, e.g., one camera pixel($d = 1$).

Chai et al. [CTCS00] introduced a mathematical framework for the plenoptic sampling problem. They observed that in the frequency domain, the spectral support of a light field signal is bounded by the minimum and maximum depths of objects in the scene only, no matter how complicated the spectral support may be. Given the minimum and maximum depths, a reconstruction filter with an optimal and constant depth can be designed to achieve anti-aliased light field rendering. However, their analysis of plenoptic sampling was based on the assumption that the surface is diffuse. Important view-dependent effects, such as specular highlights, will significantly increase the sampling rate. Compared to the optical analysis, their analysis also takes into account

[5]There is a typographical error in the equation as originally published, i.e., Equation 15 from [CTCS00]. The left hand side is inverted.

the texture-richness of the scene. Intuitively, under the same geometric setup, a scene without many high-frequency contents will require a lower sampling rate than a scene with rich textures.

In general, if one wanted to build a camera array to capture a general scene, the inter-camera distance would need to be so small that the lenses would touch each other.

2.2.3 Geometry-Assisted Methods

Light field rendering techniques introduced in the previous section fully embody the spirit of IBMR—automatically synthesizing new views without relying on a geometric model. However, they usually require a prohibitively high sampling rate of the scene of interest. For static scenes, the sampling requirement could be satisfied by collecting the image data sequentially, for example, by mounting a camera on a robotic arm to scan the scene from many different viewpoints. But for dynamic scenes, it is currently impractical to employ the thousands of cameras that would likely be needed to completely capture time-varying events.

On the other hand, there are other IBMR methods that get by with fewer images by using some other additional constraints to synthesize new views [SD96, Che95, SGHS98, CBL99, MBR+00, DTM96, DBY98, SHS98, MMB97]. In essence, they use some geometric information about the scene to warp and blend input images to synthesize new views. The geometric information can be represented in many different ways. For example, a crude geometric proxy is a rough representation of the scene, while correspondence information implicitly encodes the scene geometry. Since automatic recovery of these constraints from images is something really hard to achieve, successful methods in this category usually require a human operator and/or a priori domain knowledge.

For example, Seitz introduced *view-morphing* to generate continuous intermediate views from a pair of images [SD96]. Using basic principles of projective geometry, view-morphing is a simple extension of image morphing [Wol90] that correctly handles 3D projective camera and scene transformations. While the ability to synthesize changes both in viewpoint and image structure affords a wide variety of interesting 3D effects via simple image transformations, view-morphing requires feature correspondence known a priori. Typically this correspondence is established manually.

In the successful Façade system [DTM96] designed to model and render architecture from photographs, an operator first manually places simple 3-D primitives in rough

positions and specifies the corresponding features in the input images. The system automatically optimizes the location and shape of the 3D primitives, taking advantage of the typical regularity and symmetry in the architecture. To synthesize new images, the authors presented *view-dependent texture mapping* (VDTM), a method of compositing multiple views of a scene to provide more geometric and photometric details not captured by the basic model. This is implemented by assigning texture coordinates and *view-dependent* blending weights at the vertices of the basic model. Later, Buehler et al. [BBM+01] adopted VDTM as a basis for generalizing many current image-based rendering algorithms.

Other extensions of texture mapping techniques have also been proposed, based on the observation that once a complex scene has been captured/rendered from a viewpoint, the image from a nearby viewpoint is likely to be very similar. In such cases, the original 2D image, or *sprite*, can be slightly altered by a 2D affine or projective transformation to approximate the view from the new camera position [SLS+96, SS96, LS97, HiAA97]. These methods in fact approximate the scene geometry using a plane. Thus the fidelity of the synthesized view depends highly on the depth variation of the scene and the distance from the original viewpoint to the new viewpoint.

The sprite can be augmented with per-pixel depth information to better approximate the views, as in [BSA98, SGHS98, MMB97, DSV97]. But the one-depth-per-pixel limitation causes gaps in the synthesized image due to visibility changes when portions of the scene become un-occluded. To overcome this difficulty, Shade et al. [SGHS98] introduced the Layered Depth Image, or LDI, which contains potentially multiple depth pixels at each discrete location in the image. Instead of a 2D array of depth pixels (pixels with associated depth information), they store a 2D array of layered depth pixels. A layered depth pixel stores a set of depth pixels along one line of sight sorted in front to back order. When rendering from an LDI, the requested view can move away from the original LDI view and expose surfaces that were not visible in the first layer. The previously occluded regions may still be rendered from data stored in some later layer of a layered depth pixel.

More recently Chang et al. [CBL99] introduced the LDI tree, which combines a hierarchical space partitioning scheme with the concept of the LDI. It preserves the sampling rates of the reference images by adaptively selecting an LDI in the LDI tree for each pixel. While rendering from an LDI tree, levels comparable to the sampling rate of the output image are selected to avoid aliasing.

2.3 Real-time On-line View Synthesis Methods

Recently, with the increasing computational power of inexpensive personal computers, and the wide availability of low-cost imaging device, several real-time methods have been proposed to capture and render dynamic scenes. They are of particular interest to this dissertation and are reviewed here.

2.3.1 Stereo Vision Methods

Stereo vision is one of the oldest and most active research topics in computer vision. An overview of stereo vision methods is present in Section 2.1.1, based on a recent survey by Scharstein and Szeliski [SS02]. While many stereo algorithms obtain high-quality results by performing global optimizations, today only correlation-based stereo algorithms are able to provide a dense (per pixel) depth map in real time on standard computer hardware.

Only a few years ago special hardware had to be used to achieve real-time performance with correlation-based stereo algorithms [FHM+93, KYO+96, WH97]. It is only recently that, with the tremendous advances in computer hardware, software-only real-time systems have begun to emerge. For example, Mulligan and Daniilidis proposed a new trinocular stereo algorithm in software [MID02] to achieve 3-4 frames/second on a single multi-processor PC. Hirschmuler introduced a variable-window approach while maintaining real-time suitability [HIG03, Hir01]. There is also a commercial package from Point Grey Research [Inc], which seems to be the fastest one available today. They report 45 million disparity calculations per second (Mdc/s) on a 1.4GHz PC, which extrapolates to 64 Mdc/s on a 2.0GHz PC.

These methods use a number of techniques to accelerate the calculation, most importantly, assembly level instruction optimization using Intel's MMX extension. While the reported performance of 35-65 Mdc/s is sufficient to obtain dense correspondences in real-time, there are few CPU cycles left to perform other tasks such as high-level interpretation of the stereo results. Furthermore, most approaches use an equal-weight box-shaped filter to aggregate the correlation scores, so the result from the previous pixel location can be used in the current one. While this simplifies the implementation and greatly reduces computational cost, the size of the aggregation window has a significant impact on the resulting depth map.

2.3.2 Image-based Methods

Image-based Visual Hull. Matusik et. al. presented an efficient method for real-time rendering of a dynamic scene [MBR+00]. They used an image-based method to compute and shade visual hulls [Lau94] from silhouette images. A visual hull is constructed by using the visible silhouette information from a series of reference images to determine a conservative shell that progressively encloses the actual object (a detailed review of visual hull reconstruction techniques is in Section 2.1.2). Unlike previously published methods, they constructed the visual hulls in the reference image space and used an efficient pixel traversing scheme to reduce the computational complexity to $O(n^2)$, where n^2 is the number of pixels in a reference image. Their system uses a few cameras (four in their demonstration) to cover a very wide field of view and is very effective in capturing the dynamic motion of objects. However, their method, like any other silhouette-based approach, cannot handle concave objects, which makes close-up views of concave objects less satisfactory. Another disadvantage of the image-based visual hull is that it completely relies on successful background segmentation.

Hardware-assisted Visual Hull. Based on the same visual hull principle, Lok presented a novel technique that leverages the tremendous capability of modern graphics hardware [Lok01]. The 3D volume is discretized into a number of parallel planes, the segmented reference images are projected onto these planes using projective textures. Then, he makes clever use of the stencil buffer to rapidly determine which volume samples lie within the visual hull. His system benefits from the rapid advances in graphics hardware and the main CPU is liberated for other high-level tasks. In addition, this approach can be used to detect collisions between virtual and real objects, since the real objects (represented as visual hulls) are already in the graphics hardware in which virtual objects are to be rendered. However, this approach suffers from a major limitation—the computational complexity of his algorithm is $O(n^3)$, compared to the software-based method of [MBR+00] with $O(n^2)$ complexity.

Generalized Lumigraph with Real-time Depth. Schirmacher et. al. introduced a system for reconstructing arbitrary views from multiple source images on the fly [SMS01]. The basis of their work is the two-plane parameterized Lumigraph with per-pixel depth information. The depth information is computed on the fly using a depth-from-stereo algorithm in software. With a dense depth map, they can model both concave and convex objects. Their current system is primarily limited by the quality and the speed of the stereo algorithm (1-2 frames/second).

Goal (1): Use a pratical number of cameras (one to two dozens).
Goal (2): Be fully automatic.
Goal (3): Be robust and accurate for a wide variety of shapes and surfaces.
Goal (4): Work towards real-time, on-line view synthesis.

Method	Goal (1)	Goal (2)	Goal (3)				Goal (4)
			Concave	Convex	Diffuse	Specular	
Stereo (Sec. 2.1.1)	✓	✓	✓	✓	✓	†	✓
Shape-from-sil. (Sec. 2.1.2)	✓	✓		✓	✓	✓	✓
Space carving (Sec. 2.1.2)	✓	✓	✓	✓	✓		
Light field (Sec. 2.2.1)		✓	✓	✓	✓	✓	✓
View morphing (Sec. 2.2.3)	✓		✓	✓	✓		
VDTM (Sec. 2.2.3)	✓		✓	✓	✓	✓	✓
LDI (Sec. 2.2.3)	✓		✓	✓	✓		‡

† Avoid specular highlights using polarization filters or detect and discard pixels in highlight.
‡ Only for rendering. The creation of LDI is off-line.

Table 2.1: Comparison of existing view synthesis methods in the context of the goals of this dissertation.

2.4 Discussion

After a thorough review of existing view synthesis methods, I found that no existing method can satisfy all the goals set out in Section 1.1, as shown in Table 2.1. Note that Table 2.1 does not attempt to compare the relative accuracy and appropriateness of these different techniques on a general basis. Rather, I seek to capture their relative strengths and weaknesses in the context of the goals of this dissertation.

In terms of image quality, light field rendering techniques typically surpass the rest. They work for any surface, any scene. The synthesized images are usually so good that they are barely distinguishable from real photos. However, as shown in Section 2.2.2, the number of input images required is prohibitively high—it would be impractical to deploy the hundreds or even thousands of cameras likely to be needed to capture a dynamic scene of meaningful size. Thus light field rendering techniques fail to meet the first goal—using a practical number of input images. There are other image-based modeling and rendering techniques that require far fewer input images, such as those introduced in Section 2.2.3 including view morphing, View-Dependent Texture Mapping (VDTM), Layered Depth Images (LDI). But these methods usually require a human operator in the loop and/or a priori domain-specific knowledge to provide additional constraints for view synthesis, and thus fail to meet the goal 2: full automation.

In contrast, 3D shape recovery methods are usually fully automatic, but the quality

of the reconstructions they offer is typically less satisfactory. In particular, many existing 3D reconstruction methods assume that the scene is primarily diffuse, a basis for establishing correspondence between different images. This limits the types of scenes they can reconstruct. One notable exception is the shape-from-silhouettes technique (Section 2.1.2), which avoids the correspondence problem completely by performing volume intersections. But the difficulty is that concavity cannot be preserved in the reconstructed model. Overall, existing 3D shape recovery methods have difficulties meeting the goal of being able to deal with a wide variety of shapes and surfaces.

In addition, most 3D shape recovery methods generate a quantized model, be it a depth map or a voxel model. When rendered from different viewpoints, the quantization effects can be disconcerting.

These difficulties prompted my dissertation research, which attempts to find a framework that will keep the advantages while addressing the shortcomings of existing approaches.

Chapter 3

View-Dependent Pixel Coloring (VDPC)

In this chapter, I present the complete View-Dependent Pixel Coloring framework (VDPC). First, I present the basic approach to VDPC in Section 3.1, followed by Section 3.2, which is about specific implementation issues. I discuss VDPC, including its innovations and limitations, in Section 3.3. I end the chapter with experimental results from applying VDPC to various data sets with different geometry and material properties.

3.1 Approach

In this section, I explain in great detail how VDPC works but leave some specific implementation issues to the next section (Section 3.2). I start with the representation VDPC uses in Section 3.1.1, followed by a progressive refinement scheme of VDPC in Section 3.1.2. Then, two important components of VDPC, a view-dependent smoothness constraint and a novel physically-based photo-consistency measure, are presented in detail in Section 3.1.3 and Section 3.1.4, respectively. There is no dependency between these two components, each of these can be applied independently.

3.1.1 Representation

In VDPC, the 3D scene space is discretized into a perspective voxel grid from the desired output viewpoint. Figure 3.1 (which is a repeat of Figure 1.3) shows a bird's-eye view of a horizontal slice of the voxel grid, in which C^* is the output viewpoint. Each column

of voxels (the slant-shaded voxels in Figure 3.1) corresponds to a single pixel in the image to be synthesized for C^*. This image is referred to as the *desired image*. Initially, the volume is filled with voxels. The goal of VDPC is to assign a color to each pixel in the image to be synthesized. In order to achieve that, VDPC attempts to carve away voxels in empty space, and assign colors to voxels representing scene surfaces. As in space carving, a voxel's occupancy is determined by a photo-consistency test. Because of the one-to-one correspondences between voxel columns and image pixels, the color of the first opaque (not empty) voxel in each column is assigned to its corresponding pixel. In the end, VDPC outputs a desired image, as well as a surface voxel model from the perspective of the desired output viewpoint.

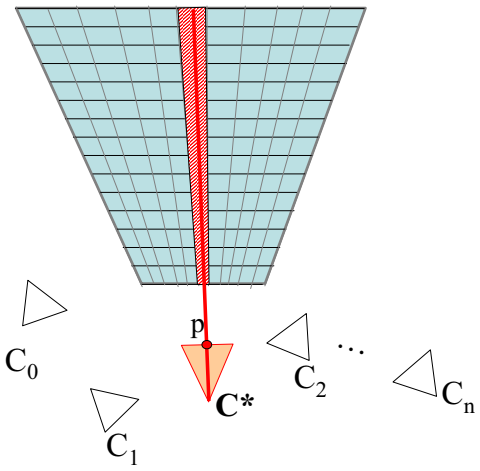


Figure 3.1: VDPC uses a view-dependent parameterization (a perspective voxel grid) of the 3D scene space. C^* corresponds to the desired *output* viewpoint. This figure shows a bird-eye view of a horizontal slice of the grid, which corresponds to a scan line in the output image.

3.1.2 Progressive Refinement

Overview VDPC uses a progressive refinement scheme to carve away voxels and assign color to pixels. For each voxel column, which corresponds to a single pixel in the output image, VDPC calculates a *weighted* photo-consistency value for each voxel. The weight is computed from other constraints or assumptions—local shape smoothness, for example. Then the profile of the weighted photo-consistency values is examined. If, based on the profile, there is a unique voxel whose photo-consistency value is much better compared to the other voxels in the same column, its color is assigned

to the corresponding pixel. If there is ambiguity, the decision will be postponed until enough confidence is accumulated from local shape smoothness and updated visibility constraints.

VDPC starts with a voxel array without any constraints. At each iteration, VDPC first calculates the weighted photo-consistency values for all voxels. Note that the photo-consistency values of some voxels could be "contaminated" due to an incorrect visibility configuration that will be corrected later. Once the calculation is done, voxels are classified as reliable or not, based on their weighted photo-consistency values. These reliable voxels change the visibility configuration, so previously contaminated photo-consistency values of other voxels can be corrected in the next iteration. Based on a local shape smoothness assumption, these reliable voxels also affect the weights for other voxels. The weights propagate recovered depth information to regions lacking color variations; the best-effort reconstruction provides some depth estimate, instead of leaving a hole or making an arbitrary choice.

Details Algorithm 2 (on page 50) outlines a progressive refinement scheme for VDPC. Each voxel can have one of four labels: `UNKNOWN`, `EMPTY`, `SURFACE`, or `INVISIBLE`. At the beginning, every voxel is labelled `UNKNOWN`. In order to deal with occlusions, a visibility mask is created for each input image. A visibility mask is a binary image with the same dimensions as its corresponding input image. It is used to indicate whether or not an input pixel should be included in the photo-consistency evaluation. The visibility masks are initially set to *true*, meaning that every pixel in every input image should be included. Each voxel also has a weight value, which is computed from other constraints, such as a smoothness constraint (I will present one in the next section). The initial weight value is set to 1.

At each iteration of the progressive scheme, there are three stages of calculations: *scoring, selection*, and *update*. At the scoring stage, a *weighted* photo-consistency value (the product of the current weight and photo-consistency value) for each `UNKNOWN` voxel is caculated. This value indicates how likely it is that the given voxel belongs to an object in the scene, rather than to empty space. If a voxel cannot be seen from at least two views, it is immediately labelled `INVISIBLE`. (Note that we can traverse the voxels in any order, since the visibility configuration is fixed during an iteration.)

Once the scoring for all voxels has been done, VDPC enters the *selection* stage, trying to find the one voxel in each voxel column that is most likely to be a surface voxel. VDPC examines at a column of voxels that all correspond to a single pixel, say

Algorithm 2 Pseudo Code for a Progressive View-Dependent Pixel Coloring Scheme.

```
Clear visibility mask M_I for every image I;
for every voxel v {
    label[v] = UNKNOWN;
    weight[v] = 1.0;
}

while (true) {

    // the scoring stage
    for every UNKNOWN voxel v {
        s = ∅ ; // s is the visible pixel set
        for every image I {
            {p_I} =  v's projection in I
            if (p_I visible) s = s ∪ p_I
        }
        if (||s|| < 2)  // v is visible in less than 2 views
            label[v] = INVISIBLE; // its occupancy can't be determined.
        else {
            score[v]=photo_consistency(s)
                *weight[v];
        }
    }

    // the selection stage
    n = select_consistent_voxels();
    // if no more voxels can be selected, stop
    if (n == 0) break;

    // the update stage
    update weight;
    update visibility masks;
}
```

p, in the output image. These voxels are denoted as $\{v_p^i\}$. Assuming an opaque scene, only one voxel in $\{v_p^i\}$ is the surface voxel reflecting light to p; thus that voxel should have the best photo-consistency value if visibility is solved correctly. Considering the photo-consistency curve of $\{v_p^i\}$, if there is a single local maximum (assuming a larger photo-consistency value means better consistency), then the corresponding voxel v is considered consistent and labelled `SURFACE`, and its color is assigned to pixel p. In addition, any voxels in $\{v_p^i\}$ that are in front of v will be labelled `EMPTY`, i.e., carved away. If no `SURFACE` voxel can be found, all voxels in $\{v_p^i\}$ are ambiguous, and their occupancies are left to be resolved in later iterations.

At the *update* stage, all `SURFACE` voxels are projected back into every view to update the visibility mask. Pixels in `SURFACE` voxels' footprints will be marked as *false* (not visible), and they will not participate in future photo-consistency computations. In addition, the weights for `UNKNOWN` voxels are updated based on a smoothness constraint that I will explain in the next section (Section 3.1.3).

After the update stage, the progressive scheme goes back to the scoring stage to start a new iteration. The entire process is repeated until all pixels in the output view are assigned a color. In the end, VDPC outputs a synthesized image, as well as a surface voxel model from the perspective of the output viewpoint.

In the following two sections, I explain two central components of VDPC: a view-dependent smoothness constraint and a novel physically-based photo-consistency measure. The smoothness constraint is used to compute the weights for voxels in the *update* stage. The photo-consistency measure is used to compute the photo-consistency scores in the *scoring* stage. There is no dependency between these two components, each can be applied independently or replaced by other constraints or measures.

3.1.3 View-dependent Smoothness Constraint

While there are many possible smoothness constraints [SH85, OK92, KZ02a, PTK85], I chose to use the *disparity gradient limit* [BJ80] because of its relevance to the human vision system, its simplicity, and its successful use in stereo algorithms. Before I get into the details of our formulation, I first present a brief overview of the principle of the disparity gradient limit.

Disparity Gradient Principle Disparity is defined between a pair of rectified stereo images. Given a pixel (s, t) in the first image and its corresponding pixel (s', t') in the second image, disparity is defined as $d = s' - s$ (where $t = t'$ since the image

is rectified). Disparity is inversely proportional to the distance of the 3D point to the cameras. A disparity of zero implies that the 3D point is at infinity.

For two 3D points whose pixel coordinates are (s_1, t_1) and (s_2, t_2) in the first image and (s'_1, t'_1) and (s'_2, t'_2) in the second image, their disparities are $d_1 = s'_1 - s_1$ and $d_2 = s'_2 - s_2$, respectively. Then the disparity gradient (DG) can be defined as

$$DG = \left| \frac{\Delta d}{\Delta s - \Delta d/2} \right| \tag{3.1}$$

where $\Delta s = s_2 - s_1$ and $\Delta d = d_2 - d_1$. Experiments in psychophysics have provided evidence that there is an upper bound on the disparity gradient DG that humans can perceive. In [BJ80] the limit $DG < 1$ was reported. The theoretical limit for opaque surfaces is 2, to ensure that the surfaces are visible to both eyes [PPMF86]. Also reported in [PPMF86], is that under normal viewing conditions most surfaces are observed with a disparity gradient well below the theoretical value of 2.

Applying the Disparity Gradient Limit Manipulating Equation 3.1 we can arrive at

$$-\frac{\Delta s \cdot DG}{1 - DG/2} + d_1 \leq d_2 \leq \frac{\Delta s \cdot DG}{1 - DG/2} + d_1, \tag{3.2}$$

This equation tells us that if one pixel's disparity, say d_1, is known and DG is limited, then the disparity range (the possible values of d_2) of a neighboring pixel is also limited. The closer these two pixels are, the smaller the disparity variation can be. Thus the disparity gradient principle has been used as a smoothness constraint in several stereo matching algorithms [BJ80, PPMF86, ZS01, YZ02]. However it has not been applied in a volumetric representation. One problem is that the disparity gradient is defined between a pair of images, so it is not directly applicable in a multi-view volumetric setting. Here, thanks to our view-dependent formulation, I apply the disparity gradient principle from the perspective of the *output viewpoint*, and introduce a way to relate the disparity gradient to the 3D voxel grid.

For a given output viewpoint C^*, we can assume that there is an imaginary image, taken from a parallel viewpoint $\tilde{C}^*$ some distance away, as shown in Figure 3.2. Assuming there is a pixel m_1 whose depth is known at v_1, the allowable disparity range for a neighboring pixel m_2 given a limit on DG can be obtained from Eq. (3.2). Then we can back-project the disparity range onto the set of voxels that intersect the line of sight from m_2, denoted as $\{v_2^i\}$. A voxel that is both in the disparity range and in $\{v_2^i\}$ is more likely to be the voxel reflecting light to m_2 (the slant-fill voxels in Figure 3.2). In

other words, because v_1 is known to be a surface voxel, the disparity gradient principle tells us that neighboring voxels to v_1 are also likely to be surface voxels.

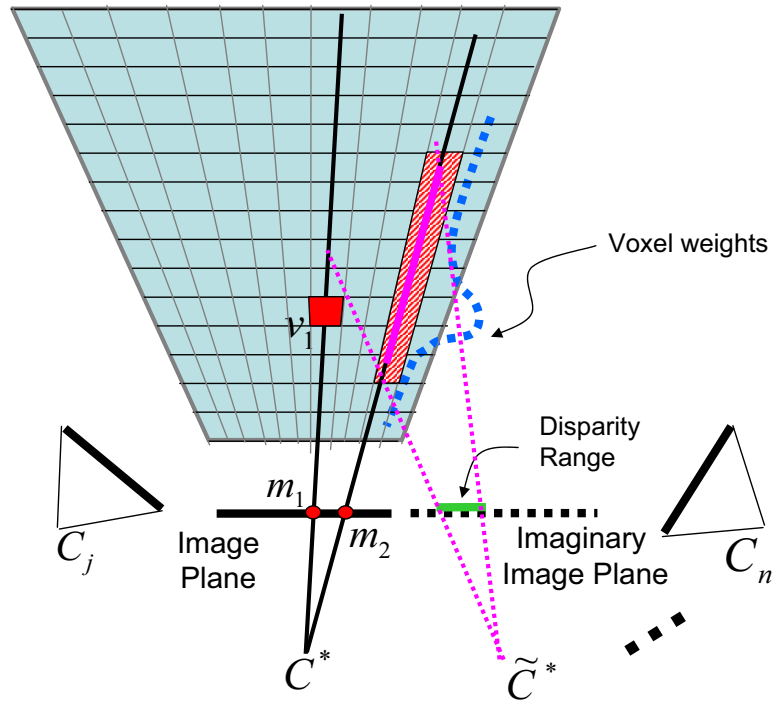


Figure 3.2: Applying the disparity gradient principle in a volumetric setting. An imaginary image $\tilde{C}^*$ is introduced to define the disparity range. The back-projection of the disparity range provides a limit about where surface voxels are more likely. The blue dotted curve illustrates the weights for voxels.

In practice, we want to favor voxels that have a similar depth as v_1, as well as allow a small possibility that voxels may fall beyond the disparity range at occlusion boundaries. So I use a weighting function similar to a normal distribution. Let v_2 be a voxel in the voxel column corresponding to pixel m_2 in the output image; m_1 is the closest pixel to m_2 with a known depth at v_1; the disparity difference Δd can be obtained by projecting v_2 and v_1 into the imaginary image $\tilde{C}^*$. Eq. 3.2 can be re-written as

$$\| \Delta d \| \leq \frac{\Delta s \cdot DG}{1 - DG/2} \tag{3.3}$$

Thus the weight for v_2 is

$$W_{v_2} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(\Delta d/\Delta s)^2}{2\sigma^2}\right), \tag{3.4}$$

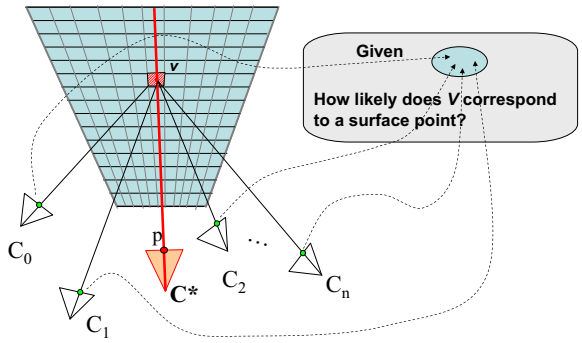


Figure 3.3: An illustration of a photo-consistency measure.

where $\sigma = \frac{DG}{1-DG/2}$.

3.1.4 Physically-based Photo-consistency Measure

Central to VDPC and many space carving algorithms (presented in Section 2.1.2) is the photo-consistency measure. As shown in Figure 3.3, for a given voxel v, we project it into all images in which v is visible. A photo-consistency measure is designed to estimate that, given all pixels in v's footprints, how likely v corresponds to a surface point instead of empty space in the scene and what the most likely color of v is.

Almost all existing photo-consistency measures assume the surface is diffuse [SD99, KS00, Joh00]. In this section, I present a new photo-consistency measure that is valid for both specular and diffuse surfaces.

Studies in photometry have shown that the reflected light (C) from many real-world surfaces can be approximated as the sum of diffuse and specular components [FP03, Pho75]. This can be modeled as

$$C = \text{diffuse}(C_l, \hat{C}, N, L) + \text{specular}(C_l, \hat{C}, R, V) \tag{3.5}$$

where C_l is the color of a light source, $\hat{C}$ is the object color, N is the normal vector of the surface, L is the lighting vector—where the light comes from, V is the viewing vector, and R is the reflection vector which is the mirrored vector of V about the surface normal N (as in Figure 3.4). Usually colors are expressed in the RGB color space, so C, C_l, and $\hat{C}$ are all three-vectors, i.e., $C = [I_r, I_b, I_g]$ where I_r, I_g, and I_b are scalars corresponding to the red, green, and blue channels in the RGB color space, respectively.

Now let us examine the change of radiance for a given surface point under different viewing directions. As a start, we assume that the scene and lighting are both

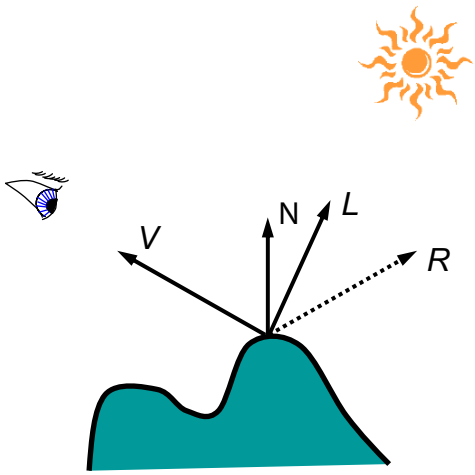


Figure 3.4: Light reflection under a fixed point light source

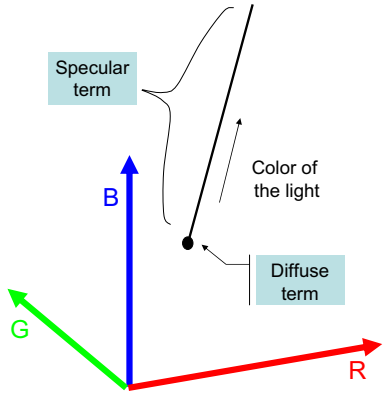


Figure 3.5: Light reflected from a specular surface point forms a line in the RGB color space.

static when images are taken, i.e., N and L are constant. Under this condition, from Equation 3.5, we can see that the diffuse term remains a constant from any view direction. If the specular effect can be ignored, the color samples (pixels) from input images will cluster into a point in the color space. That is the basis for the original photo-consistency check proposed by Seitz and Dyer [SD99]. To check if a voxel exists or not, we simply need to compute the variance of the color samples. I call this the *variance* measure. A large variance indicates they are not likely to be from the same surface point, and thus that voxel should be carved away.

On the other hand, if the specular highlights cannot be ignored, then for a broad class of materials known as *dielectric* materials, including plastic and glass, the specular reflection is only modulated by the incident light [FP03]. In other words, Eq. 3.5 for

dielectric materials can be simplified to

$$C = \text{diffuse}(C_l, \hat{C}, N, L) + C_l \text{ specular}(R, V). \tag{3.6}$$

For these materials the color values observed from different viewpoints are co-linear in the color space. As shown in Figure 3.5, they form a ray originating from the diffuse term of the reflected light (which is a constant independent of viewing angles) and extending toward the color of the light C_l. Note that the direction of the line is independent of the object color. The basic idea of my photo-consistency measure is to detect such a "signature" in the color space. If the surface is diffuse, its signature will be a point; if the surface is specular, its signature will be a line. Because I do not have *a priori* knowledge about whether a voxel represents a specular point or a diffuse point, I want to design a measure that is valid for both cases, while simultaneously providing as much disambiguating power as possible.

Maximum Likelihood Estimation Based on the above observations, I cast our novel photo-consistency measure as a maximum likelihood estimator aimed at estimating the most likely color for a given voxel, and its likelihood of being a surface voxel.

I assume that a color sample can be classified in one of three ways: a diffuse color C_d , an "onset" color C_o, or a saturated color C_s. Each case has a different *a priori* likelihood, denoted P_d, P_o, and P_s respectively. I also assume that the color samples from the images are corrupted by zero-mean gaussian noise with a variance of σ_s^2. For simplicity and robustness, I assume the color of the light is known. (It can be measured by imaging a white object.)

Thus given an object color $\hat{C}$, the likelihood of observing a particular color sample C_j is

$$p(C_j|\hat{C}) = \max \begin{pmatrix} N(C_j|\hat{C}, \sigma_s^2)P_d, \\ N(\text{distance}(C_j, \text{line}(\hat{C}))|0, \sigma_s^2)P_o, \\ N(C_j|C_s, \sigma_s^2)P_s \end{pmatrix}, \tag{3.7}$$

where N denotes a normal distribution, C_s is the saturation color, usually [1, 1, 1] for normalized RGB images, and distance$(C_j, \text{line}(\hat{C}))$ is a function to compute the distance between C_j and a ray defined by $\hat{C}$ and the color of the light.

Given a set of m color samples, the maximum likelihood estimation for an object

color $\hat{C}$ is

$$max(\prod_{j=1}^{m} p(C_j|\hat{C})). \tag{3.8}$$

The photo consistency "cost" is the residual after maximum likelihood estimation. The $\hat{C}$ with the maximum likelihood is assigned as the diffuse color of the voxel. I call this measure the *MLE* photo-consistency measure.

Note that in Equation 3.7, I assume that the distribution along the line is uniform. This is an approximation derived from the Phong lighting model [Pho75]. I present a detailed derivation here. For simplicity in the derivation and notation, I assume everything is monochromatic (or derive using one of the three RGB channels as an example). Note this does not affect the correctness of the MLE measure. In the derivation here, I follow the common practice in statistics, i.e., $f(x)$ to denote probability density functions, and $F(x)$ to denote cumulative distribution functions. Random variables are denoted in their upper case while the corresponding lower-case letters are used to represent possible values.

If we ignore ambient light and the atmospheric attenuation factor (i.e., no fog), the Phong lighting model becomes

$$I = I_p k_d O_d (NL) + I_p k_s (RV)^n \tag{3.9}$$

where I_p is the intensity of a light source, O_d is the surface albedo, k_d is a diffuse coefficient, k_s is a specular coefficient, n is the object's shininess, and all vectors are normalized. Compared with Equation 3.5, $I_p k_d O_d (NL)$ corresponds to the diffuse term and $I_p k_s (RV)^n$ corresponds to the specular term in a monochromatic world.

For a static scene under fixed lighting, let the constant diffuse term $I_p k_d O_d (NL)$ be $\hat{I}$, and let the angle between the reflection vector R and the viewing vector V be Θ. It is reasonable to model Θ as a random variable with uniform distribution in its valid range, meaning that a surface point is likely to be viewed from any direction in the hemisphere. Thus the probability density function of Θ is

$$f_\Theta(\theta) = \begin{cases} 1/\pi, & -\pi/2 \le \theta \le \pi/2; \\ 0, & \text{otherwise.} \end{cases} \tag{3.10}$$

Now we need to find the density function of the random variable I corresponding to the intensity of the reflected light. This will tell us how the color samples are distributed along the ray. Let us first look at the cumulative distribution function $F_I(i)$. From

Equation 3.9, we know that I has a minimum value of $\hat{I}$ and a maximum value of $\hat{I} + I_p k_s$, thus $F_I(i) = 0$ when $i < \hat{I}$ or $i > \hat{I} + I_p k_s$. When $\hat{I} \leq i \leq \hat{I} + I_p k_s$, we have

$$\begin{aligned} F_I(i) &= P\{I \leq i\} = P\{\hat{I} + I_p k_s \cos^n \Theta \leq i\} \\ &= P\{\cos^n \Theta \leq k\}, \text{where } k = \frac{i - \hat{I}}{I_p k_s} \\ &= P\{\Theta \leq -\arccos k^{\frac{1}{n}}, \text{or } \Theta \geq \arccos k^{\frac{1}{n}}\} \\ &= \int_{-\frac{\pi}{2}}^{-\arccos k^{\frac{1}{n}}} f_\Theta(\theta)\, d\theta \\ &\quad + \int_{\arccos k^{\frac{1}{n}}}^{\frac{\pi}{2}} f_\Theta(\theta)\, d\theta. \end{aligned}$$

Plug in $f_\Theta(\theta) = \frac{1}{\pi}$, and take the derivative of both sides, to get

$$\begin{aligned} f_I(i) &= \left(\frac{d(-\arccos k^{\frac{1}{n}})}{di} - \frac{d(\frac{-\pi}{2})}{di}\right)\frac{1}{\pi} \\ &\quad + \left(\frac{d(\frac{-\pi}{2})}{di} - \frac{d(\arccos k^{\frac{1}{n}})}{di}\right)\frac{1}{\pi} \\ &= -2 \cdot \frac{d(-\arccos k^{\frac{1}{n}})}{di} \cdot \frac{1}{\pi} \end{aligned}$$

So the probability density of the random variable I is

$$f_I(i) = \begin{cases} \frac{2}{\pi} \cdot \frac{k^{\frac{1-n}{n}}}{n\sqrt{1-k^{\frac{2}{n}}}} \cdot \frac{1}{I_p k_s}, & \hat{I} \leq i \leq \hat{I} + I_p k_s; \\ 0, & \text{otherwise}, \end{cases} \tag{3.11}$$

where $\hat{I} = I_p k_d O_d(NL)$ and $k = \frac{i - \hat{I}}{I_p k_s}$.

In Figure 3.6, I plot several density curves of I under different shininess (n) settings. If we have these curves a priori, we can formulate a photo-consistency measure in terms of Bayesian inference. However, that would require an accurate estimation of the surface material properties, as well as a careful photometric calibration of the lights and the sensing device (cameras). So I chose to use a discrete approximation. From the plot in Figure 3.6, we can see that, indeed, the middle "onset" part is relatively flat. In the meantime, the parts at the beginning and the end have higher probabilities. They represent the diffuse and saturated classes, respectively. The maximum likelihood estimator approximates this distribution nicely.

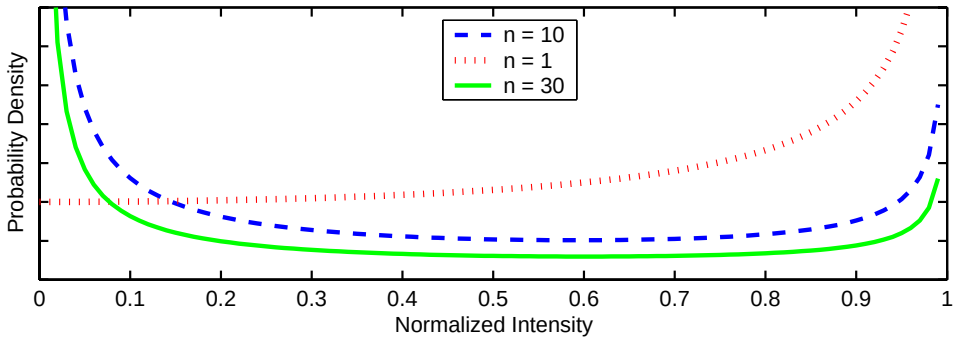
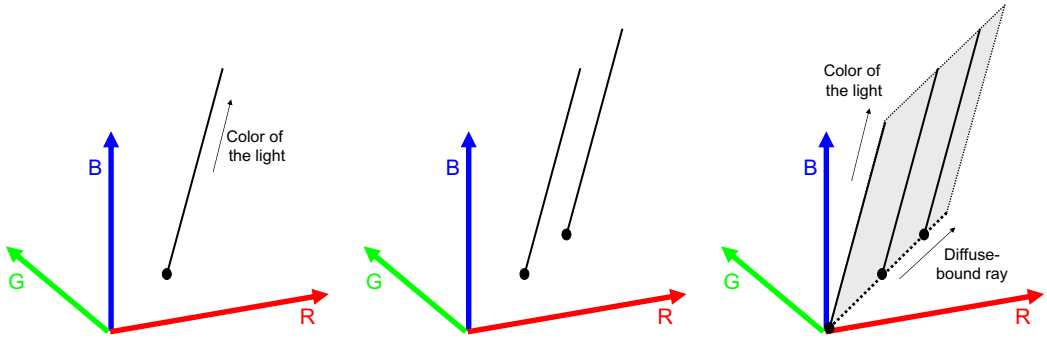


Figure 3.6: Probability Density Functions under different shininess (n) settings. The X axis is the normalized intensity value (k), while the Y axis is probability density with $I_p k_s = 1$.

Simplified Linearity Test The maximum likelihood estimation presented above needs to know the *a priori* likelihoods of the three color sample classes. In case the *a priori* likelihood is unknown or inaccurate, I use a simplified approach by assuming all samples belong to the onset class, i.e., $P_d = 0, P_o = 1, P_s = 0$. In addition, I assume that less than half of the pixel samples are in specular highlights. Thus I can use the median of the color samples as the object color $\hat{C}$, and simply compute the sum of distances to the ray defined by $\hat{C}$ and the color of the light as the photo-consistency measure. Note that this simplification will fail in smoothly shaded diffuse surfaces with uniform colors. In this case, everywhere is consistent, similar to the result of applying a simple variance test to textureless regions. In practice, I find it still works well on scenes with moderate textures. I call this measure the *LMF* (Line-Model-Fitting) photo-consistency measure. Similarly, if we set $P_d = 1, P_o = 0, P_s = 0$, i.e., only diffuse colors are possible, then the MLE measure reduces to the standard variance measure.

Multiple Stationary Lights The above analysis is valid for multiple fixed lights as long as they have the same color. This is true even if their intensities are different or there are area light sources. If the light colors *are* different, then there will be *multiple* lines in the color space, one for each unique light source. In this case, it would be interesting to investigate if a multiple-line fitting and clustering method would be practical.

Moving Lights and Cameras It is also interesting to consider data captured on a turntable. Typically in this case both the lights and the camera are (effectively) moving. For a given static light configuration, corresponding color samples of a sur-



(a) One lighting configuration (fixed lights)

(b) Two lighting config. (lights moved once)

(c) Moving lights

Figure 3.7: Reflected light under moving lights and cameras

face point will fall on the ray defined by the color of the light and the diffuse term (Figure 3.7(a)). For multiple lighting configurations, there will be multiple rays in the color space (Figure 3.7(b)). If the lights do not change their colors in different configurations, all rays will have the same direction. In addition, all rays originate from the diffuse term, if the surface can be approximated using the Phong model, their origins will form a ray starting from the origin of the color space and extending toward $I_p k_d O_d$. Let us call this line the *diffuse-bound ray*. All color samples will form a *plane* spanned by the diffuse-bound ray and the ray pointing toward the color of the light, as shown in Figure 3.7(c). Thus it is possible to reconstruct scenes under moving lights and moving cameras. In particular, if the scene is primarily diffuse, the plane will reduce to the diffuse-bound ray, then we can use a measure similar to the LMF measure to reconstruct diffuse scenes under moving lights and cameras.

3.2 Implementation Details

I use *Powell's method* [Pow78] to optimize the function in Equation 3.8. I also assume the color of light is white. If the light is not white, it can be measured using a camera easily. Note that due to the limited dynamic range provided by the cameras, the ray defined by $\hat{C}$ has three segments in general (left in Figure 3.8). One channel will first saturate, then the second, and finally the last channel. In Figure 3.8, I show a sample probability density distribution on the right. Similarly, I compute the distance to the three line segments in the Line-Model-Fitting (LMF) measure.

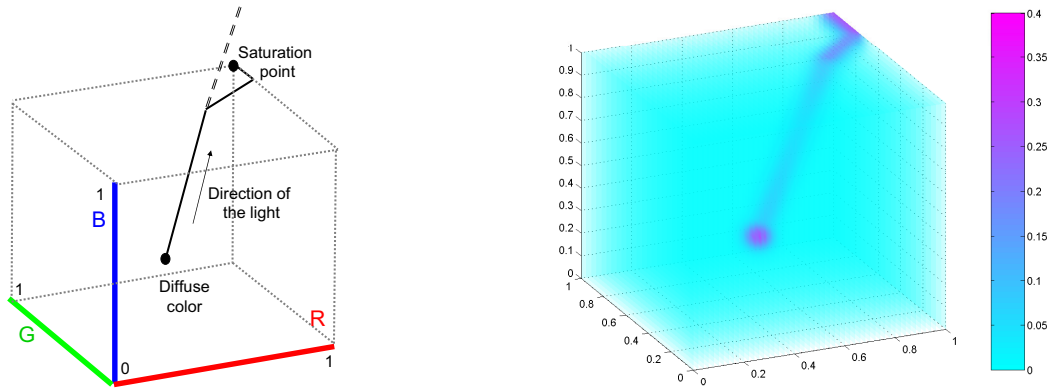


Figure 3.8: Left: Reflected colors from a surface point form a line; due to limited dynamic range, it becomes three connected line segments in the RGB cube space . Right: Sample density distribution with object color [0.3, 0.5, 0.1], $\sigma_s = 10/255$, and a priori probability $[0.4, 0.4, 0.2]$

There are two places where O exploit the graphics hardware to accelerate computation. Note that the way I use graphics hardware here is completely different than the real-time VDPC introduced in the next chapter. The first place is the visibility update. I render each surface voxel as a cube for every input image, thus I can get the exact footprint to update the visibility mask. The second place is the computation of the smoothness weight in Eq. 3.4. For each voxel, I need to find the closest *colored* pixel in the output view, i.e, the closest pixel with known depth. Since each voxel corresponds to a single pixel in the output view, this problem can be reduced to a 2D search, i.e., given a pixel, find the closest colored pixel. I use *Delaunay* triangulation to create a 2D mesh of colored pixels in the output view, each triangle is assigned a unique color (not to be confused with the assigned color to pixels), and render the color-coded mesh. The rendered image serves as a look-up table to find the closest pixel in $O(1)$ time. I found dramatic speedup compared to performing a linear search of all colored pixels in software.

3.3 Discussion

In previous sections, I explained in detail how VDPC works and some specific implementation issues. In this section, I put VDPC in the context of other related work, explaining key innovations over the state of the art. I also discuss the limitations of VDPC.

3.3.1 Innovations

Hybrid and View-dependent Estimation

In this section, I explain the conceptual design choices I have made for VDPC. In particular, two questions are answered: why I chose a hybrid approach, and why I chose a *view-dependent* voxel representation. Results from controlled experiments (using synthetic images) are presented here to support the argument. Results using real images will be presented in the result section (Section 3.4).

Why Use a Hybrid Approach? This dissertation is aimed at view synthesis, i.e., we need to assign a *color* to each pixel in a synthesized view. However, aimed for dynamic scenes captured with a feasible number of cameras (to meet the first goal in Section 1.1), I do not have the luxury we would in light field rendering (Section 2.2.1) of simply *interpolating* the color from input images—rather, I have to *estimate* it. An image is the end result of complex interactions among surface geometry and materials, occlusions, and lighting. Thus maintaining a 3D model that takes into account all these factors is necessary. However creating a full 3D model automatically is a difficult task. Existing methods are usually fragile and prone to error in practice. In addition, usually only part of a full 3D mode is visible for any given viewpoint. For dynamic and live scenes, it is less efficient to first create a full model and then render it from a desired viewpoint. For the sake of view synthesis, I only need to extract enough information to allow us to predict the new view. In other words, a *partial* model may suffice. Thus using a hybrid approach seems logical and natural for the scope of this dissertation.

An added benefit of a hybrid approach is that I can acquire an image *and* a 3D model at the same time. This 3D model can be useful in a number of ways. For example, in an augmented reality application, the 3D model is needed to detect collisions in the interactions between real and virtual objects. The 3D model can also be used to decouple the rendering and update rates. Nearby views can be rendered using the last available 3D model while a new image with a new model is being estimated.

Why Use a View Dependent Formulation? Once I have decided to employ a hybrid approach, the next question is what kind of representation to use. In order to meet the goal of being able to deal with both convex and concave shapes (part of goal (3) in Section 1.1), I chose to use a volumetric representation (a voxel grid), since it is flexible enough to represent arbitrary shapes and allows more flexible camera configu-

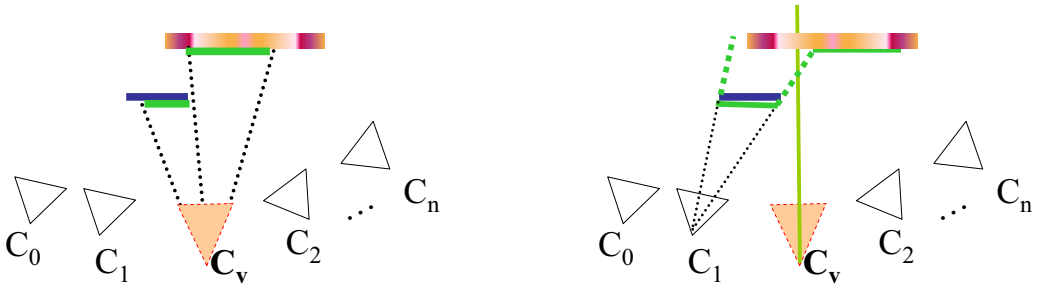


Figure 3.9: VDPC vs. stereo reconstruction. Left: VDPC estimates a depth map from the perspective of the output view (C_v). Right: stereo estimates a depth map from one of the input views, the problem of dis-occlusion leads to holes or "skins" in the output view (C_v).

rations [KS00]. Using a voxel grid also allows the problem of occlusion to be handled elegantly in a fashion similar to space carving ([KS00], reviewed in Section 2.1.2).

In addition, I introduce a view-dependent formulation that has a number of benefits. First, there is a one-to-one correspondence between the 3D model and the final image. Since there is no additional rendering of the voxel grid, quantization effects are minimized.

Secondly, the view-dependent formulation avoids some shortcomings of stereo and space carving. Compared to stereo, VDPC computes a depth map from the perspective of the *output* view, instead of one *input* view, as shown in Figure 3.9. When a depth map from one input view is warped to the output view, there is the problem of dis-occlusion. That is, the depth values for some pixels are not defined, resulting in either a hole or a "skin" stretching over different surfaces. But there is no such problem in a view-dependent formulation, assuming the surfaces are visible in at least two input images.

As shown in [BC00], space carving is usually sensitive to the global threshold for the photo-consistency test. In practice, as a result of random noise and quantization effects, a single threshold rarely achieves optimal results for a complex scene. But in our view dependent formulation, I need only to find the most consistent voxel for a column of voxels corresponding to an output pixel, so a global threshold is unnecessary. In fact, we can think of VDPC as having a varying threshold depending on the relative photo-consistency values for every column of voxels.

Figure 3.10: Five synthetic images rendered from a scene of a cube and a flat back wall. Note the change of perspectives on the sides of the cube.

Results from Controlled Experiments To support the argument for the view-dependent formulation, I ran some controlled experiments using synthetic imagery.[1] I rendered five synthetic images of a scene that consists of a cube and a flat back wall, as shown in Figure 3.10.

For the sake of comparison, I implemented the multi-baseline plane-sweep stereo algorithm [Col96] and the original space carving algorithm [KS00]. Since the scene is perfectly diffuse, I used the standard color variance as the matching metric. I applied both algorithms to the synthetic data set in Figure 3.10. Each created a 3D model, then I rendered the model from different viewpoints. In addition, for every viewpoint, I applied VDPC to the data set using the color variation as the photo-consistency measure but without enforcing any additional constraints, i.e., the weight for every voxel is always one.

Figure 3.11 shows the images generated by VDPC (first row) and the rendered images from an estimated depth map using the stereo algorithm I implemented (second row). Note the holes in the images in the second row. Some of the holes, such as these scattered on the background, are due to quantization effects. In stereo, the depth map has a regular sampling pattern in the image space of the reference camera (from which the depth map is generated). Thus the sampling patter of the depth map in 3D space is not uniform. Depth points around the image center are closer to each other in 3D space than those close to the edges of the image. So when they are viewed from a different viewpoint, they may not cover every pixel in the new image. In my experiment, because the viewpoint actually moves down, depth points in the lower part, which are close to edges in the reference image, move closer to the center area in the new image that requires higher sampling rate. This is the cause of the holes on the background. There are other holes due to dis-occlusions. Parts of the background become dis-occluded as the viewpoint moves, but their depth values are not defined since they are not visible

[1] I also show results using real imagery in Section 3.4.

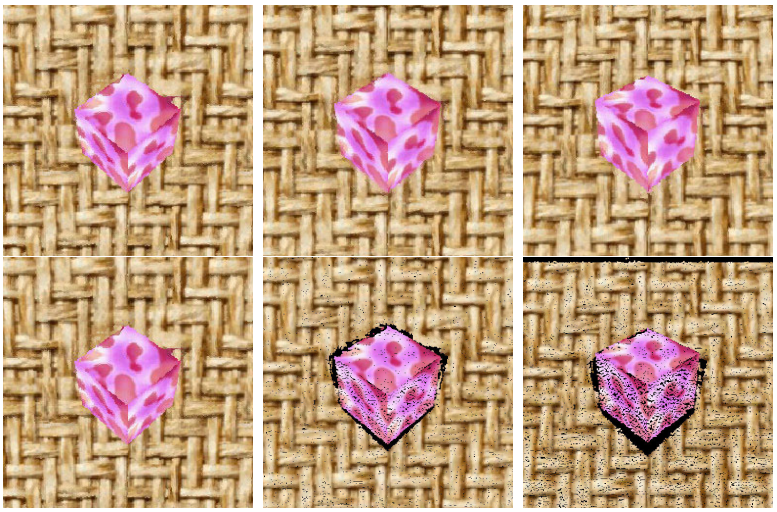
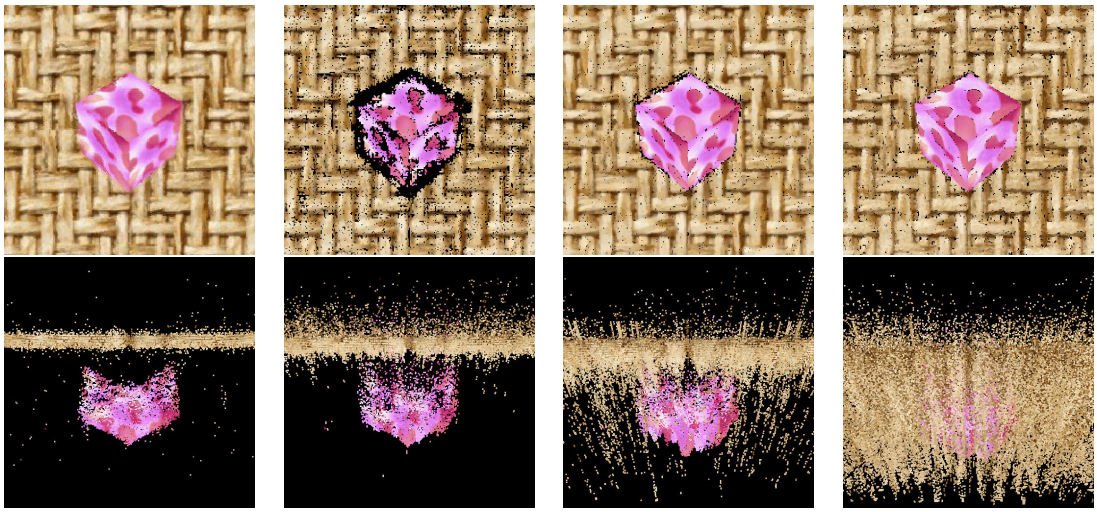


Figure 3.11: VDPC vs. multi-baseline stereo. Images in the top row are synthesized using VDPC. Images in the bottom row are rendered views of an estimated depth map. The depth map, generated by my implementation of [Col96], is from the perspective of the third image in Figure 3.10. As the viewpoint moves away, note the holes in some of the rendered images in the second row.

in the estimated depth map. VDPC addressed both problems, thus there are no holes in the images in the first row.

In Figure 3.12, the first row shows the images generated by VDPC and the images rendered from a model using the original space carving algorithm I implemented. The second row shows the corresponding 3D models rendered from a 90 degree viewing angle. Space carving requires a global threshold for the photo-consistency test. Results from three different thresholds—5, 10, 20—are presented. A threshold of 5 means that if the normalized color variation of a voxel's corresponding pixels is greater than five levels (assuming 8 bit/channel input images), that voxel will be carved away. In Figure 3.12(b) to 3.12(d), we can see that a single threshold can rarely achieve the optimal reconstruction. Too small a threshold leads to holes, too large a threshold creates many false positives (see the oblique views in the second row). VDPC, by contrast, does not rely on a single threshold to determine a voxel's occupancy. It looks at each column of voxels and attempts to find a best one in each column, which can be thought of as using a varying threshold depending on the relative photo-consistency values.



(a) VDPC

(b) Space Carving: var = 5

(c) Space Carving: var = 10

(d) Space Carving: var = 20

Figure 3.12: Comparisons between VDPC and space carving. The top row shows the synthesized image, the bottom row shows the 3D models from a 90 degree viewing angle. Since space carving requires a global threshold, results from three different thresholds are shown. VDPC, by contrast, does not need a global threshold.

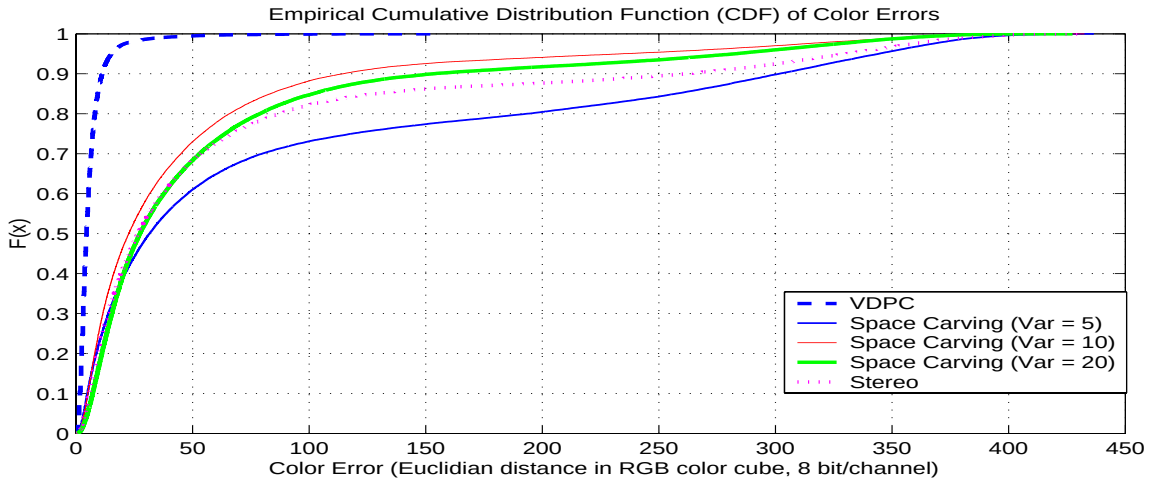


Figure 3.13: CDF of color errors

Progressive Refinement with a View-dependent Smoothness Constraint

Another innovation of VDPC is the progressive refinement with a view-dependent smoothness constraint, which enables VDPC to be robust for textureless regions—part of goal (3) in Section 1.1.

From a shape-recovery standpoint, VDPC can be considered a variation of the space carving framework. Kutulakos and Seitz showed that, without additional constraints the space carving framework provides the tightest reconstruction using color information alone [KS00]. They called the recovered shape the *photo hull*. The idea of using color information *alone* has its pros and cons. On the one hand, in regions with rich textures, arbitrarily complex shapes can be recovered. On the other hand, the lack of additional constraints make space carving more susceptible to image noise and quantization problems.

In addition, space carving has difficulty resolving ambiguity in textureless regions. In regions with low color variation, i.e., different surface points having similar radiance, false positive photo-consistencies typically result in extraneous voxels that "fatten" the reconstruction in *front* of the true surface. This effect is particularly pronounced when the model is viewed from an oblique angle, far away from any of the input viewpoints. Figure 3.14 shows this effect for synthetic images in which an intensity gradient across the image is varied. In regions with low intensity variations, the reconstructed surfaces differ substantially from the original surface.

Additional constraints are often applied in an attempt to resolve the ambiguity. For example a typical approach in stereo vision is to increase the support of the reconstruction kernel. However the accompanying smoothing effect (low-pass filtering) undermines a unique feature of these voxel based methods: the ability to reconstruct highly complex shapes. Instead we want to apply additional constraints *only* when there is ambiguity. To this end, VDPC employs an iterative approach to progressively refine the shape estimates. As explained in Section 3.1.2, the basic idea is to defer decisions about ambiguous voxels until there is enough supporting evidence, including updated visibility information and local smoothness constraints.

I also recognize that smoothness is a *view-dependent* property. Think about a thin sheet of paper: when viewed from the front, the paper is smooth everywhere; when viewed from a 90 degree angle the sheet barely exists. That is probably one of the reasons that volumetric reconstruction methods, which are typically *view-independent*, either do not use a smoothness constraint, or use some simple ones such as averaging of

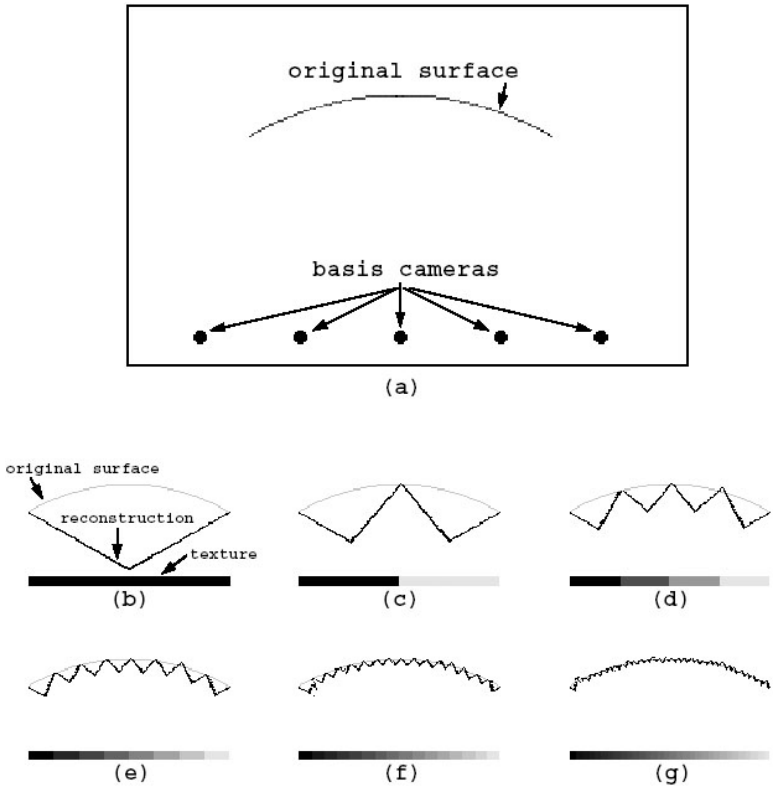


Figure 3.14: Effects of texture variation on reconstruction (From [SD99] with permission). (a) A concave scene surface and the positions of five input cameras. (b) Reconstruction when the surface has constant radiance gives the same result as shape-from-silhouettes. (c – g) Successively better reconstructions result when the surface is textured with an intensity gradient that doubles in frequency with each image.

neighboring voxels. But in my view-dependent formulation, smoothness can be naturally defined from the perspective of the output viewpoint. From the output viewpoint, we are more likely to see frontal-parallel surfaces compared to ones at oblique angles. So it makes sense to use smoothness constraints that favor such frontal-parallel surfaces in the absence of additional information.

Since the synthesized images are to be viewed by humans, I chose to use a smoothness constraint that is based on psychophysical studies of the human vision system–the *disparity gradient principle*. These studies also provide a meaningful range for the parameters used in our smoothness constraint formulation.

In conclusion, the progressive refinement scheme with a psychophysical based view-dependent smoothness constraint is designed to provide better shape reconstruction in low texture regions, leading to more visually pleasing images without sacrificing VDPC's flexibility in recovering highly complex shapes.

Physically-based Photo-consistency Measure

The new photo-consistency measure discussed in Section 3.1.4 is another innovation of VDPC. It allows VDPC to be applied in scenes with specular surfaces—part of goal (3) in Section 1.1.

Most 3D reconstruction algorithms rely on the diffuse surface assumption as a basis to establish correspondence. The effects of specular highlights have largely been ignored. This severely limits applicability of these algorithms in certain scenes, such as surgical scenes in which specular highlights are the norm rather than the exception. The new photo-consistency measure breaks this barrier. Here some related work to deal with specular reflections is discussed and compared to our new photo-consistency measure.

Although the use of more sophisticated lighting models was envisioned in the original space carving work [KS00], almost all existing methods use a photo-consistency measure based on a diffuse (Lambertian) surface assumption. Two notable exceptions are the *Surfel* (surface element) sampling algorithm by Carceroni and Kutulakos [CK01] and the color caching algorithm by Chhabra [Chh01]. The former differs substantially from traditional voxel-based methods. The scene is divided into a very coarse voxel grid, with each voxel represented as a parametric surface referred to as a *surfel*. Under calibrated lighting, additional properties such as surface normal and reflectance parameters can be estimated. Only results from scenes with point light sources were demonstrated. In practice, light calibration is not always possible, especially for area

light sources. In the color caching algorithm, Chhabra tries to characterize the reflected light from specular surfaces in the color space. While this analysis is very similar to my thinking, it is restricted to the case where the reflected light passes through the origin of the color cube. This simplification is only valid in some very limited cases, such as monochromatic surfaces under white light. Based on an analysis of the surface color response under a more generic lighting model [FP03], I recognized that for the sake of matching, i.e., finding corresponding pixels in different images, full light calibration is unnecessary since reflected lights from many real-world surfaces have a certain "signature" in color cube space. A photo-consistency measure only needs to *detect* these signatures independent of lighting. Based on this idea, I arrived at a *general* photo-consistency measure that makes use of the *linear* color response of some typical surface.

There is also some work in the stereo vision literature to recover a disparity map in the presence of specular reflections [Wol89, LB92, BN95, JYS01, LLL+02, LLK+02]. To my knowledge, these all try to first *detect* specular reflections, either through color histograms or using polarized filters, and then *reject* them as outliers or occluders. Instead my method treats specular reflections as "inliers" and accounts for them inherently.

My new photo-consistency measure does not need to know the lights' positions or orientations. It works for both point light sources and area light sources. It requires no surface normal estimation, which is very difficult to obtain under unknown geometry and lighting. Given these relaxed pre-conditions, my photo-consistency measure can be used in *any* existing space carving or multi-view stereo algorithms to extend their applicability to scenes with specular surfaces.

3.3.2 Limitations

Like many, if not all, existing algorithms, VDPC also has its limitations. Here I will discuss these limitations and how we can possibly address them.

While VDPC's view-dependent formulation has a number of advantages, as explained in Section 3.3.1, a major limitation of such a formulation is that the global consistency of the 3D model is not guaranteed. That is, if we merge the 3D models from different viewpoints, the combined global 3D model is not guaranteed to be consistent with all view-dependent sub-models. Note that this problem is not entirely insurmountable. In fact, for situations in which the emphasis is on creating a global 3D model rather than synthesizing an image, I have found a way to reconstruct a globally

consistent voxel model using a variation of VDPC [YPW03]. Changes include a view-independent formulation, which is in-line with space carving, and a more sophisticated "best" voxel selection scheme that considers all visible views simultaneously, instead of just from the output view.

Compared to view-independent reconstruction techniques such as space carving or stereo, VDPC takes considerable computing time. Suppose the complexity of space carving is $O(1)$; then the complexity of VDPC is $O(m)$, where m is the number of output views. So VDPC is best suited for dynamic scenes, when each moment is likely to be viewed only once. From an efficiency standpoint, the variation from [YPW03], in which a view-independent voxel model is first computed, might be desirable for static scenes. Nevertheless the quantization effect of the view-independent voxel model is likely to be visible in the output images rendered.

Another concern is that VDPC uses a progressive refinement scheme that has difficulty recovering from previous errors. The same problem exists for space carving. In fact, VDPC suffers less than space carving since VDPC incorporates additional constraints and decides a voxel's occupancy by looking at a number of related voxels. However, VDPC still makes "hard" decisions about voxels, and once a voxel is carved away, it will not be put back. To overcome this difficulty, several probabilistic space carving methods have been introduced [dBV99, BDC01, AD01, BFK02]. In theory, a probabilistic formulation should consider *all* possible visibility configurations for a voxel. To avoid the combinatorial search, the visibility tests are typically approximated based on heuristics [dBV99, AD01, BDC01] or solved in a stochastic manner through hundreds of iterations [BFK02]. I believe that an accurate treatment of visibility is crucial for any multi-view reconstruction algorithm. If an efficient and accurate probabilistic model of visibility can be found, a probabilistic approach will be very promising.

3.4 Results

I have implemented the full VDPC framework in software. We also constructed a capture rig I call the *camera cube* (Figure 3.15). It consists of eight digital video cameras with VGA resolution (640×480). All cameras are fully calibrated and synchronized. I am able to capture and store VGA videos at 15 frames/second. Lens distortions are removed after capture. In the results shown here, I synthesize images from a viewpoint from the top, looking down. In some cases, I also show the underlying 3D surface voxel

Figure 3.15: Camera Cube: our one meter-cubed camera rig with eight cameras looking down at a human patient model

models. I further define a robust measure to reject potential outliers: if in the color cube space the average distance/pixel to the color ray is over a threshold, the voxel is rejected. I used a very generous number—100 levels (assuming 8 bit/channel), and this number was fixed throughout all experiments. Experiments have shown that VDPC is not sensitive to this threshold.

I captured and reconstructed a variety of real-world scenes. Unless otherwise indicated, all were reconstructed at a resolution of $256 \times 256 \times 128$.

In Figure 3.16, I show synthesized views of a hand at a high resolution (512^3). It is computed using the variance consistency measure through four iterations. Note the textureless regions are nicely reconstructed.

My second data set contain a teapot with rich textures (Figure 3.17) and I tested the photo-consistency measure without applying the smoothness constraint. See Figure 3.18. Since I knew nothing about the surface materials or the scene lighting, except that the color of lights is white, I tried different settings of the *a priori* likelihood (denoted as $\vec{P}$) for the MLE measure (introduced in Section 3.1.4). With a 0.1 granularity, there are about 50 different combinations. The most visually pleasing result is shown in Figure 3.18(a), where $\vec{P} = [0.5, 0, 0.5]$. I also show the result with $\vec{P} = [1.0, 0, 0]$ in Figure 3.18(b), which is equivalent to using the standard variance measure. Figure 3.18(c) shows the result with $\vec{P} = [0, 1.0, 0]$, which is very similar to the result from

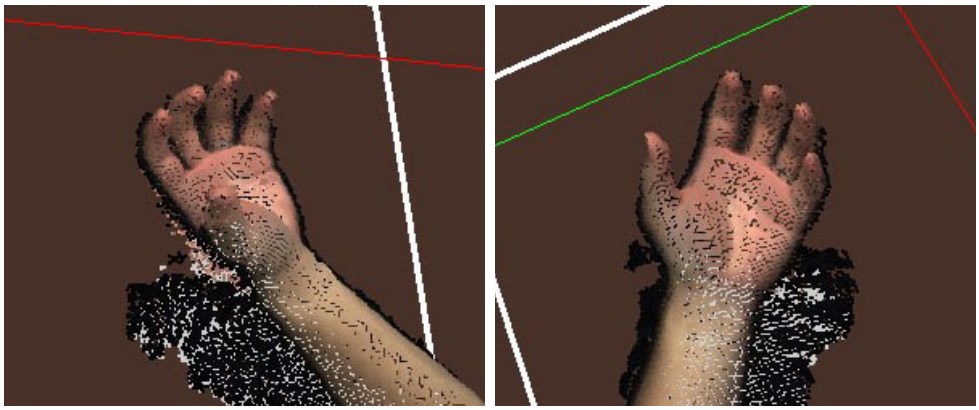


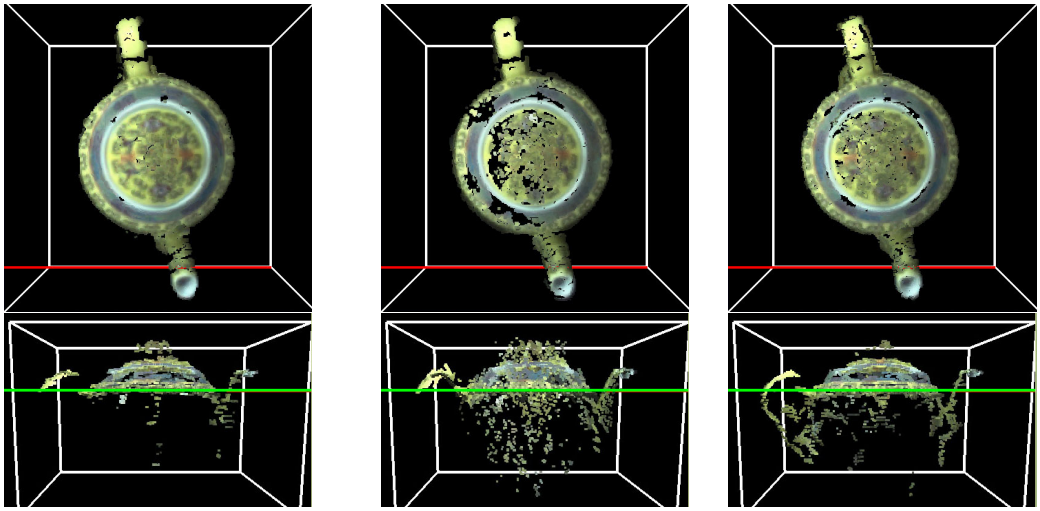
Figure 3.16: Hi-resolution progressive reconstruction on a $512 \times 512 \times 256$ grid using the variance measure.

the LMF measure (introduced in Section 3.1.4). Compared to the best one, it has more stray voxels when viewed from the side.

Figure 3.17: Eight images captured simultaneously by our camera cube. They are cropped to show more details. The teapot roughly took a 300×300 area in every image.

My next data set consists of a teapot and a book with substantial textureless regions (Figure 3.19). I used the LMF measure and applied the smoothness constraint through a few iterations, shown in Figure 3.20. I stopped at the fourth iteration when newly selected `SURFACE` voxels were less than 2% of the total `SURFACE` voxels. I compare it with the result using the variance measure in Figure 3.21.

I further captured a dynamic sequence in which a surgeon was explaining a medical procedure. The torso model was constructed using the LMF measure, while the hand and other moving parts were constructed using the variance measure. They are composited together and shown in Figure 3.22.



(a) $\vec{P} = (0.5, 0, 0.5)$, visually best from 50 combinations of $\vec{P}$

(b) $\vec{P} = (1.0, 0, 0)$ (same as the variance measure)

(c) $\vec{P} = (0, 1.0, 0)$ (similar to the LMF measure)

Figure 3.18: Reconstruction results from data in Figure 3.17. I used the MLE measure under different a priori assumptions, no smoothness weight was applied.

The last data set, courtesy of the Mitsubishi Electric Research Lab, is a teapot captured on a turntable. The light was not static with respect to the teapot. I did not know this when I first tried my method. In Figure 3.23, I show the results using the LMF and the variance measure. On the top of the teapot where highlights exist, neither method produces meaningful results. However on the side of the teapot where there are virtually no highlights, the LMF measure performs much better than the variance measure, since under moving lights the reflected lights of a diffuse point form a line (not a point) in the RGB color space.

Since there is no established data set or standard to evaluate multi-view reconstruction algorithms[2], I have attempted to contact several researchers, asking them if they would be willing to apply their view synthesis or 3D reconstruction methods to my data set or share their implementations with me. I had a very limited success in obtaining such an evaluation; details can be found in Section 5.2.

[2]I will discuss possible ways for quantitative evaluations as future work in Section 5.3.

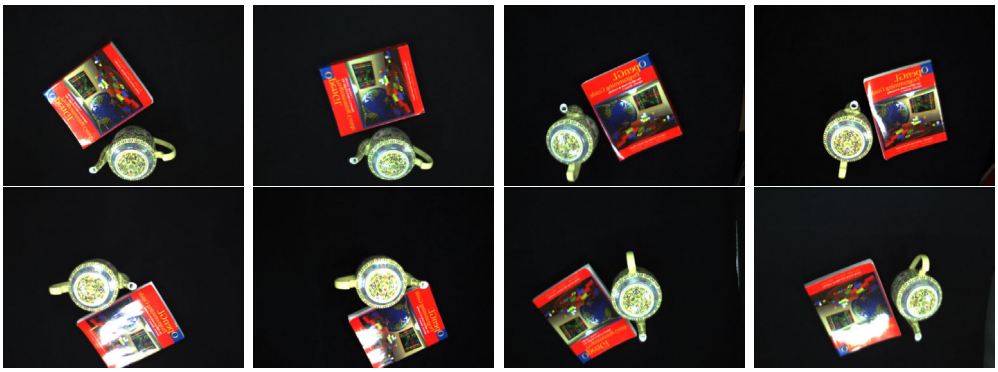


Figure 3.19: Eight images of the teapot-and-book data set.

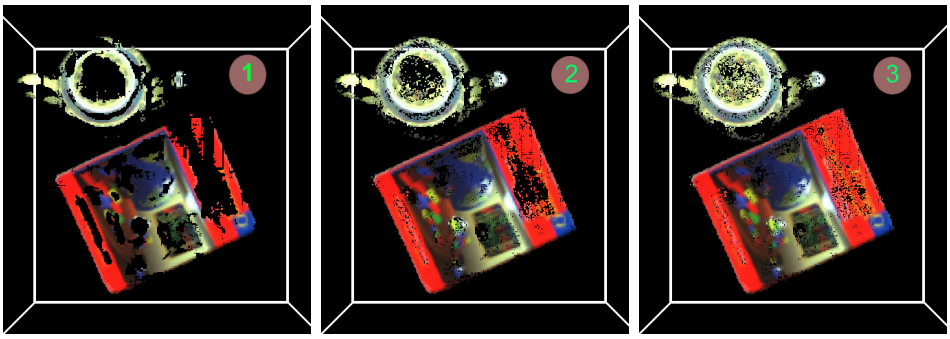


Figure 3.20: Reconstruction results using data in Figure 3.19. From left to right, I show the progressively refined results after iterations one to three. I used the LMF consistency measure and set the disparity gradient limit to 0.8.

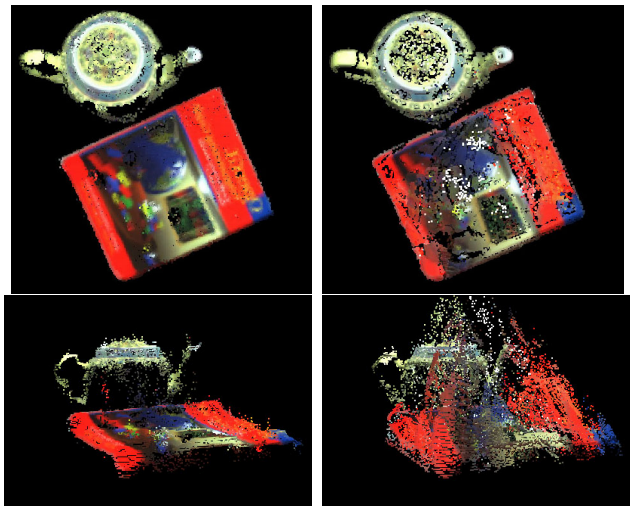


Figure 3.21: Comparison of different consistency measures. Left column (top and side views): LMF measure; right column: variance measure. Both results were obtained after four iterations. All other parameters were kept the same.

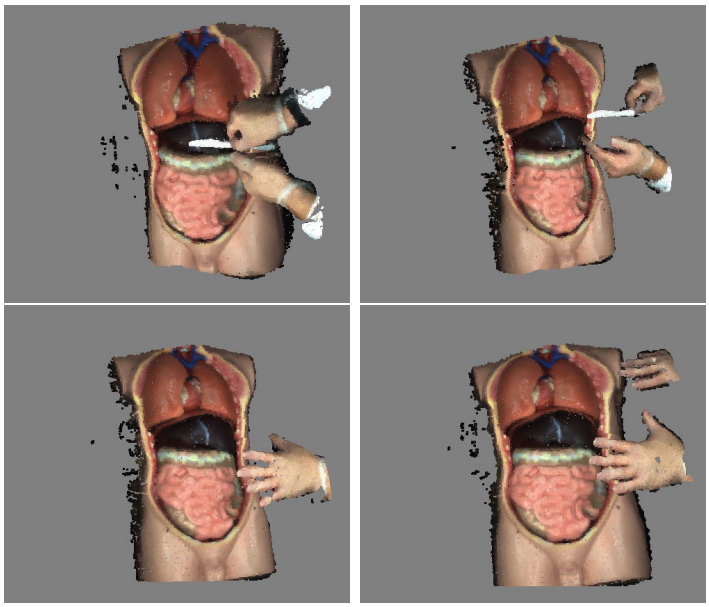


Figure 3.22: A dynamic sequence captured by the camera cube. It was constructed using the LMF measure.

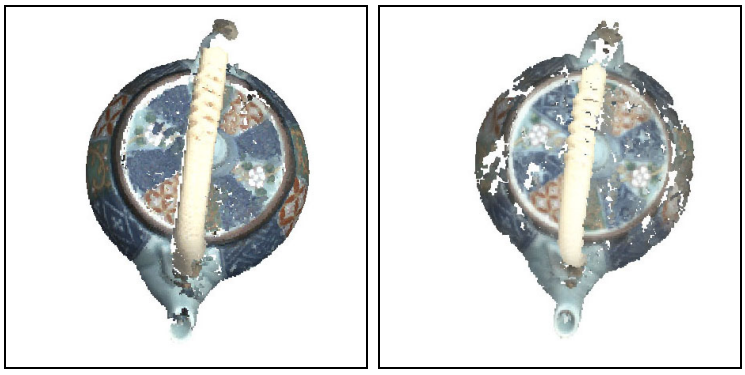


Figure 3.23: Experiment with moving lights. Left: LMF measure; right: variance measure.

Chapter 4

Real-time VDPC on Commodity Graphics Hardware

A full software implementation of VDPC as presented in Chapter 3 is not real-time yet. The results presented in the previous chapter typically require minutes to render. In this chapter, I present a reduction of VDPC that can be efficiently implemented on commodity graphics hardware. Modern graphics hardware offers orders of magnitude in accelerating the view synthesis time, enabling real-time on-line view synthesis of a live dynamic scene. Readers may also want to refer to Section 2.3 for related work in real-time view synthesis.

4.1 Motivation

Modern graphics hardware systems continue to offer increasingly powerful performance in terms of both speed and programmability. Today (as of May 2003), the latest graphics cards on the market achieve over 50 gigaflops, and the programming languages for these cards, Cg for example [NVI02], are almost as flexible as the C language. The consumer graphics boards on the market offer specialized but powerful Single-Instruction-Multiple-Data (SIMD) processing. While such hardware is designed and optimized for computer graphics, there is increasing interest in using the high-performance graphics processing unit (GPU) for tasks other than rendering. For example, Holzschuch and Alonso used the GPU to speed visibility queries [HA00], Hoff et al. to compute generalized Voronoi Diagrams [IKL+99] and proximity information [IZLM01], Larsen and McAllister for fast matrix multiplies [LM01], and Lok to reconstruct an object's visual hull given live video from multiple cameras [Lok01]. Each of these applications

obtained significant performance improvements by exploiting the speed and inherent parallelism of modern graphics hardware.

As described in Section 1.2, I began this work by wondering how I could harness the power in graphics hardware for view synthesis. In the next few sections, I will present a reduction of VDPC under some simplifying assumptions. This reduction can be implemented entirely on commodity graphics hardware. Once the input images are downloaded to the graphics board, the CPU is essentially idle. By utilizing the fast speed of the graphics board, which is already an ubiquitous component in commodity PCs, we can synthesize views of a live scene at interactive rates on one's desktop.

4.2 Approach

While commodity graphics cards are getting more and more programmable in every generation, they are still designed and optimized to render images from geometric primitives. In order to fit VDPC on the *current* graphics hardware architecture, as well as to maximize the utilization of the graphics hardware to achieve real-time performance, I made a few simplifying assumptions about VDPC.

First, I assume, as in the case for multi-baseline stereo, that the problem of occlusion can be ignored. Since there is no visibility change for any voxel, I can compute the photo-consistency values and select the best colors in a single pass. Secondly, I assume that the scene is Lambertian, so I can use the standard variation measure for the photo-consistency test. Thirdly, I use a smoothness constraint that simply aggregates weighted photo-consistency values from neighboring voxels, a practice commonly used in stereo.

Under these assumptions, VDPC can be implemented entirely on commodity graphics hardware. Before I present the detail of such an implementation in the next section, I first briefly introduce the hardware features that are essential to make such an implementation possible.

I make use of *multi-texture mapping*, *the Pixel Shader*, and *the mipmap* functions in graphics hardware. Multi-texture mapping functions allow more than one texture to be rendered onto the same geometry primitive in a single pass. The Pixel Shader, first proposed and implemented by NVIDIA [Nvi], is a texture compositing unit that can perform fixed function arithmetic on a per-pixel basis [Kil00]. The Mipmap [Wil83] was originally developed to deal with the aliasing problem in texture mapping. A base texture is pre-filtered down recursively to create a texture pyramid called mipmap.

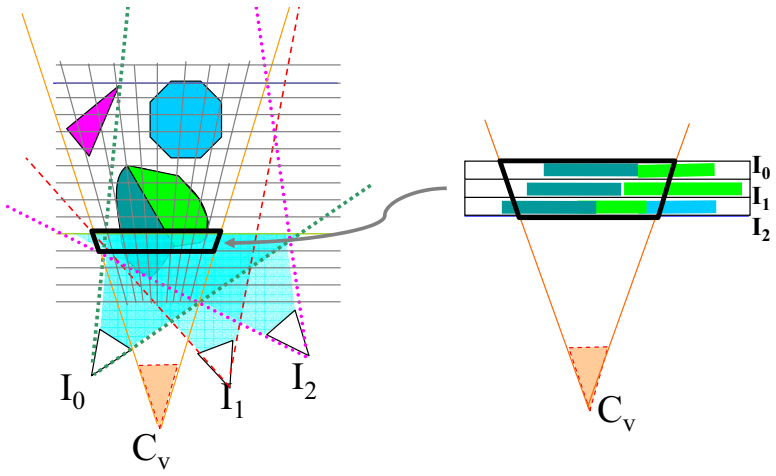


Figure 4.1: Illustration of real-time VDPC on graphics hardware.

At run time, an appropriate level of the mipmap is selected to match the sampling rate desired. Modern graphics boards support automatic generation of mipmaps in hardware and biased selection of a mipmap level.

Using these features, which are available on virtually *every* graphics board on the market today, I can implement real-time VDPC entirely on a commodity graphics board, enabling real-time, online view synthesis of a dynamic live scene.

4.3 Implementation Details

I first outline the entire real-time VDPC algorithm, then discuss several key steps, and how they map to graphics hardware.

Modern graphics hardware typically has multiple processing pipelines to increase throughput. In order to utilize the inherent parallelism, I can compute a voxel plane a time, instead of a single voxel a time. As shown in Figure 4.1, the basic idea is first to project the input images on to each voxel plane, then to use programmable graphics hardware to compute the photo-consistency values of voxels and select the most consistent ones. Essential to this implementation are the multi-texture mapping functions and Pixel Shader, both of which have been available on commodity graphics hardware since 2000. I use multi-texture mapping functions to project input images onto voxel planes, and Pixel Shader to compute photo-consistency values and select the best colors.

In an implementation on a graphics card, there is no need to maintain an explicit

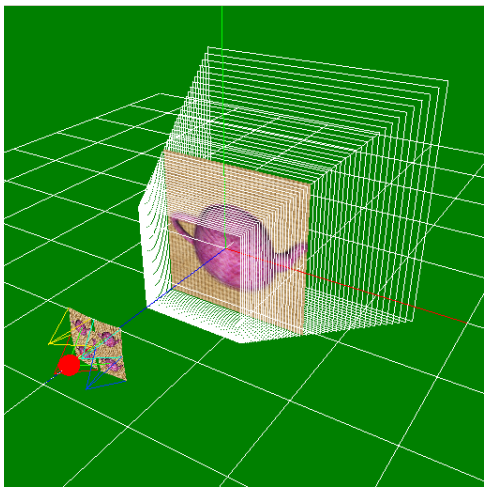


Figure 4.2: A configuration where there are five input cameras. The red dot represents the new view point. Space is discretized into a number of parallel planes.

voxel representation because the rasterization hardware will automatically divide a voxel plane into discrete cells. I call each voxel plane a *depth plane*, to contrast with the explicit voxel representation in a software implementation.

Now I can outline my implementation on a graphics card. From the perspective of a desired viewpoint (the red dot in Figure 4.2), the 3D space is discretized into a number of depth planes. The depth planes are traversed from near to far. At each step (i), there are three stages of operations, *scoring*, *aggregation*, and *selection*. In the *scoring* stage, the reference images are project onto the depth plane D_i, the graphics hardware is programmed to compute the per-pixel mean color and consistency measure. In the second optional *aggregation* stage, a simplified smoothness constraint is applied. That is, consistency values are aggregated from neighboring pixels. In the final *selection* stage, the textured D_i, with a RGB color and a photo-consistency value for each pixel, is rendered into the the frame buffer. At each pixel location in the frame buffer, if the incoming pixel has a better photo-consistency value, it will replace the existing pixel. After all the depth planes are traversed and computed, the frame buffer consists of a synthesized view in which each pixel has the best photo-consistency value.

Comparing real-time VDPC to the full VDPC approach in Section 3.1, there are two major differences. First, the visibility constraint is waived, every input pixel is visible. Secondly, the smoothness constraint is simplified and *optionally* applied *right after* the photo-consistency value is computed. From an algorithmic standpoint, real-time VDPC is similar to the plane-sweeping algorithm by Collins [Col96], but by using the graphics

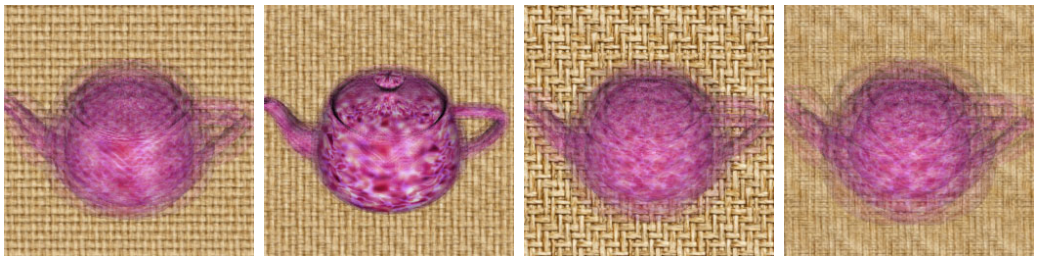


Figure 4.3: Depth plane images from step 0, 14, 43, 49, from left to right; The scene, which contains a teapot and a background plane, is discretized into 50 planes.

hardware my approach supports real-time applications. In the next few sections, I will explain how to implement the *entire* algorithm using the standard OpenGL API, which is supported by virtually all graphics boards. A complete pseudo code and detailed settings in Pixel Shader can be found in Appendix A.

4.3.1 Photo-consistency Evaluation

In the first *scoring* stage, we need to warp the input images and compute the photo-consistency values. For a given depth plane D_i, we project the input images on it, and render the textured plane from the desired perspective to get an image (I_i) of D_i. While it is natural to think of these as two sequential operations, in practice one can combine them into a single homography (plane-to-plane) transformation. In Figure 4.3, I show a number of images from different depth planes. Note that each of these images contains the projections from all input images, and the area corresponding to the intersection of objects and the depth plane remains sharp.

For each pixel location (u, v) in I_i, we want to use the *Pixel Shader* to compute the photo-consistency value and the corresponding most-likely color. Assuming a Lambertian scene, my current implementation uses the sum-of-square-difference (SSD) of the luminance as the photo-consistency value, that is

$$SSD = \sum_i (Y_i - Y_{base})^2 \tag{4.1}$$

where Y_i is the luminance from an input image and Y_{base} is the luminance from the base reference image selected as the input image that is closest to the new viewpoint. The most likely color is the mean color from all images. With a minimum of two multi-texture units (two textures at a time), graphics hardware can compute the SSD score sequentially, an image pair at a time. In this stage, the frame buffer acts as an

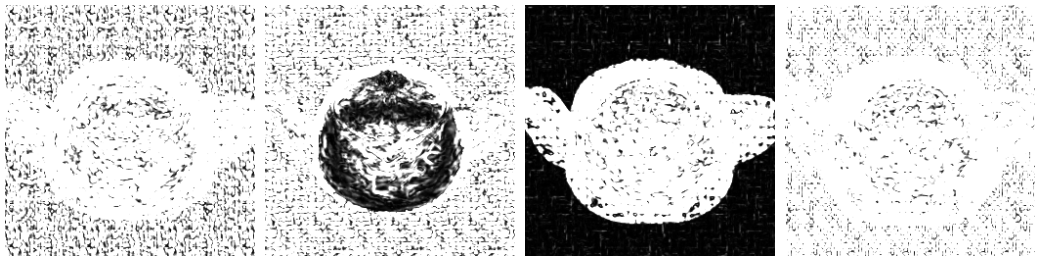


Figure 4.4: SSD scores (encoded in the alpha channel) for different depth planes in the first *scoring* stage. I use the same setup as in Figure 4.3. From left to right, the corresponding depth steps are 0, 14, 43, 49.

accumulation buffer to keep the mean color (in the RGB channel) and the SSD score (in the alpha channel) for D_i. In Figure 4.4, I show the SSD score images (the alpha channel of the frame buffer) at different depth steps. The corresponding color channels are shown in Figure 4.3.

4.3.2 Aggregating Photo-consistency Values

The photo-consistency value for each pixel sometimes can be ambiguous, especially in low-texture regions. In these cases, it is desirable to apply some kind of smoothness constraint. A usual approach is to aggregate the values over a large support window. This approach can be implemented very efficiently on a CPU, and by reusing data from previous pixels, the complexity becomes independent of the window size. However on today's graphics hardware, which uses a Single-Instruction-Multiple Data (SIMD) parallel architecture, it is not so simple to efficiently implement this type of optimization.

One has several options to aggregate the photo-consistency values. One is to use convolution functions in the *Imaging Subset* of the OpenGL Specification (Version 1.3 and above) [1.301]. By convolving a blurring filter with the contents of the frame buffer, one can sum up the consistency measures from neighboring pixels to make color estimates more robust.

If the convolution function were implemented as a part of the pixel transfer pipeline in hardware, as in the OpenGL specification, there would be little performance penalty. Unfortunately, hardware-accelerated convolutions are only available on expensive graphics workstations such as SGI's Oynx2, but these expensive workstations do not have programmable pixel shaders. On commodity graphics cards available today, convolution is only implemented in software because it is not used very often and implementing it in hardware adds cost. In a software implementation, pixels have to be transferred

between the main memory and the graphics board. This is not an acceptable solution since such out-of-board transfer operations will completely negate the benefit of doing computation on the graphics board.

There are ways to perform convolution using standard graphics hardware [WND96]. One could use multiple textures, one for each of the neighboring pixels, or render the scene in multiple passes and perturb the texture coordinates in each pass. For example, by enabling bilinear texture interpolation and sampling in the middle of 4 pixels, it is possible to average those pixels. Note that in a single pass only a summation over a 2×2 window can be achieved. In general such tricks would significantly decrease the speed as the size of the support window becomes larger.

A less obvious option is to use the mipmap functionality available in today's Graphics Processing Units (GPUs). This approach is more general and quite efficient for certain types of convolutions. Modern GPUs have built-in box-filters to efficiently generate all the mipmap levels needed for texturing. Starting from a base image J^0 the following filter is recursively applied:

$$J^{i+1}_{u,v} = \frac{1}{4} \sum_{q=2v}^{2v+1} \sum_{p=2u}^{2u+1} J^i_{p,q},$$

where (u, v) and (p, q) are pixel coordinates. Therefore, it is very efficient to sum values over $2^n \times 2^n$ windows. Note that at each iteration of the filter the image size is divided by two. Therefore, a disadvantage of this approach is that the color consistency measures can only be evaluated exactly at every $2^n \times 2^n$ pixel location. For other pixels, approximate values can be obtained by interpolation. However, given the low-pass characteristics of box-filters, the error induced this way is limited.

Multi-resolution Approach

Choosing the size of the aggregation window is a difficult problem. The probability of a consistency mismatch goes down as the size of the window increases [Nis84]. However, using large windows leads to a loss of resolution and to the possibility of missing some important image features. This is especially so when large windows are placed over occluding boundaries. This problem is typically dealt with by using a hierarchical approach [FHM+93], or by using special approaches to deal with depth discontinuities [HIG03].

Here I will follow a different approach that is better suited to the implementation

on a GPU. By observing correlation curves for a variety of images, one can observe that for large windows the curves mostly only have a single strong minimum located in the neighborhood of the true depth, while for small windows often multiple equivalent minima exist. However, for small windows the minima are typically well localized. Therefore, I wanted to combine the global characteristics of the large windows with the well-localized minima of the small windows. The simplest way to achieve this in hardware seems to be just adding up the different curves. In Figure 4.5 some example curves are shown for the Tsukuba dataset.

Summing two variance images obtained for windows differing by only a factor of two (one mipmap-level) is very easy and efficient. It suffices to enable trilinear texture mapping and to set the correct mipmap-level bias. Additional variance images can easily be summed using multiple texturing units that refer to the same texture data, but have different mipmap-level biases.

In fact, this approach corresponds to using a large window, but with larger weights for pixels closer to the center. An example of a kernel is shown in Figure 4.6. The peaked region in the middle allows good localization while the broad support region improves robustness.

I will call this approach the Multiple Mip-map Level (MML) method. In contrast, the approach that only uses one mip-map level will be called the Single Mip-map Level (SML) method.

In the actual implementation, I first have to copy the frame buffer, which has the individual photo-consistency values, to a texture, with automatic mipmap generation enabled. Then I use the multi-texture functions to sum up different mipmap levels. Note that all texture units should bind to the same texture object but with different settings of the mipmap bias. This is possible because the mipmap bias, as defined in the OpenGL specification, is associated per texture unit, not per texture object.

Note that if we want to use the SML method, which is equivalent to aggregating the consistency measures over a fixed-size support region, we only need to reduce the rendered image size proportionally (by changing the viewport setting) in the last selection stage. The automatic mipmap selection mechanism will select the correct mipmap level. A sub-pixel texture shift can also be applied in the last selection stage to increase the effective resolution.

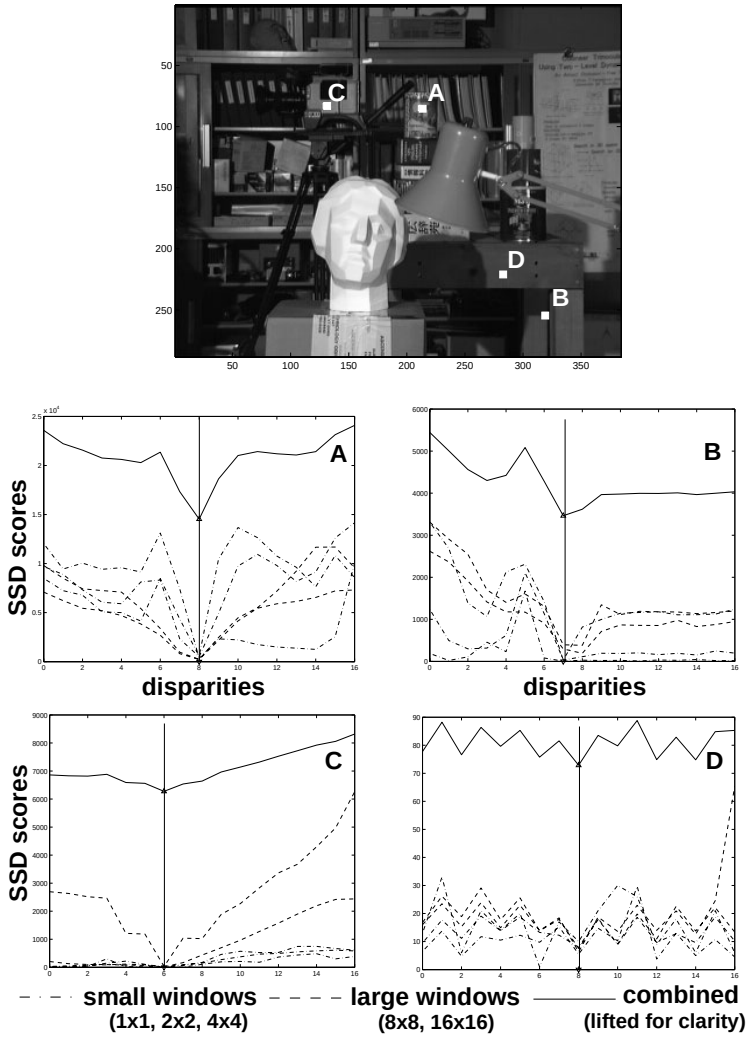


Figure 4.5: Correlation curves for different points of the Tsukuba stereo pair. Case A represents a typical, well-textured, image point for which SSD would yield correct results for any window size. Case B shows a point close to a discontinuity where SSD with larger windows would fail. Cases C and D show low-texture areas where small windows do not capture sufficient information for reliable consistency measures.

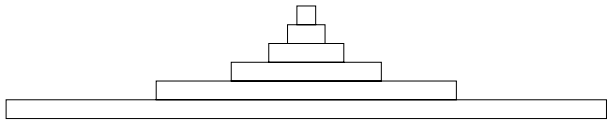


Figure 4.6: Shape of kernel for summing up six levels.

4.3.3 Selecting Best Color

In the last *selection* stage, one need to select the mean color with the smallest SSD score. The content of the frame buffer is copied to a temporary texture (Tex_{work}), while another texture (Tex_{frame}) holds the mean color and minimum SSD score from the previous depth step. These two textures are rendered again into the frame buffer through orthogonal projection. I reconfigure the *Pixel Shader* to compare the alpha values on a per pixel basis, the output color is selected from the one with the minimum alpha (SSD) value. Finally the updated frame buffer's content is copied to Tex_{frame} for use in the next depth step.

My technique is *implicitly* computing a depth map from a desired view point, I just choose to keep the best color estimate for the purpose of view synthesis. If we choose to trade color estimation for depth estimation, we can use almost the same method to compute a depth map in real-time, and liberate the CPU for other high level tasks. The only change necessary is to reconfigure the graphics hardware to keep the depth information instead of the color information in the last selection stage. The color information can then be obtained by re-projecting the reconstructed 3D points into input images. From an implementation standpoint, we can encode the depth information in the RGB portion of the texture image.

4.4 Discussion

Real-time VDPC meets the real-time requirement—goal (4) in Section 1.1. In addition, I believe that real-time VDPC combines the advantages of previously published real-time methods in Section 2.3, while avoiding some of their limitations as follows.

- I achieve real-time performance without using any special-purpose hardware, and our method, based on the OpenGL specification, is relatively easy to implement.
- I can deal with arbitrary object shapes, including concave and convex objects.
- I do not use silhouette information, so there is no need for image segmentation, which is not always possible in a real environment.
- I use graphics hardware to accelerate the computation without increasing the symbolic complexity—our method is $O(n^3)$, the same as most correlation-based stereo algorithms.

- Real-time VDPC is more versatile. I can use two or more cameras in a casual configuration, including configurations where the images contain epipolar points. This case is problematic for image-pair rectification [Fau93], a required pre-processing step for most real-time stereo algorithms.

- The hybrid formulation VDPC uses allows one to acquire a model (a depth map) *in addition to* a synthesized image. So it is possible, though I have not demonstrated this, to decouple the rendering and update loop. That is, we can render new images using the last available depth map at a rate higher than the rate of generating new depth maps. This is important for interactive applications.

Of course, there are certain tradeoffs I have to make when using the graphics hardware. One common complaint about current graphics hardware is the limited arithmetic resolution. My method, however, is less affected by this limitation. Computing the SSD scores is the central task of our method. SSD scores are always non-negative, so they are not affected by the unsigned nature of the frame buffer. (The computation of SSD is actually performed in signed floating point on recent graphics cards, such as the GeForce4 from NVIDIA.) A large SSD score means there is a small likelihood that the color/depth estimate is correct. So it does not matter if a very large SSD score is clamped, it is not going to affect the estimate anyway.

A major limitation of the current hardware acceleration scheme is the inability to handle occlusions. This is a common problem for most real-time stereo algorithms. To address this problem in practice, I use a small baseline between cameras, a design adopted by many multi-baseline stereo systems. However, this limits the effective view volume, especially for direct view synthesis.

The bottleneck in the hardware acceleration scheme is the fill rate. This limitation is also reported by Lok in his hardware-accelerated visual hull computation [Lok01]. In each stage, there is a texture copy operation that copies the frame buffer to the texture memory. I found that texture copies are expensive operations even within the graphics board, especially when the automatic mipmap generation is enabled. I have explored the use of *P-buffer*, an OpenGL extension that allows to render directly to an off-screen texture buffer. There was no performance gain, in fact performance was worse in some cases. I suspect that this extension is still a "work in progress" in the current drivers from NVIDIA. I expect to see a substantial performance boost when this work is done.

In late 2002 NVIDIA introduced the Cg programming language [NVI02]. It is similar to C but has no looping or branching. With its power, many simplifications that had

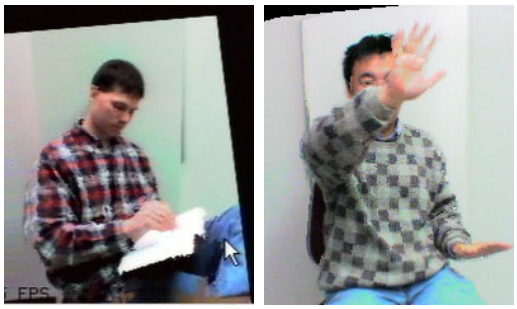


Figure 4.7: directly captured from the screen. They are synthesized using five cameras

to be imposed previously to fit VDPC on graphics hardware are no longer necessary. For instance, the implementation of the LMF measure (introduced in Section 3.1.4) on graphics hardware is entirely possible. In addition, the occlusion problem may be solved using the new texture-look-up functionality to see if a color sample should be included for the photo-consistency evaluation.

4.5 Results

In order to test real-time VDPC's performance, I have implemented a distributed system using four PCs and up to five calibrated 1394 digital cameras (SONY DFW-V500). The camera exposures are synchronized using an external trigger. Three PCs are used to capture the video streams and correct for lens distortions. The corrected images are then compressed and sent over a 100Mb/s network to the rendering PC, in which real-time VDPC is running, to synthesize new views. Some experiments are presented in the next two sections.

4.5.1 Live View Synthesis

I have tested my algorithm on three NVIDIA cards, a Quadro2 Pro, a GeForce3, and a GeForce4, using all five cameras for view synthesis. Figure 4.7 shows some live images computed online in real time. Performance comparisons are presented in Table 4.1 and Figure 4.8. On average, a GeForce3 is about 75 percent faster than a Quadro2 Pro for our application, and a GeForce4 is about 60 percent faster than a GeForce3.

For comparison purposes, I also ran some tests using different support sizes. The results are shown in Figure 4.9. Even a small support window (4×4) can make substantial improvements, especially in low texture regions, like the cheek or forehead

	128^2	256^2	512^2
20	9, 16, 40	18, 31, 55	51, 82, 156
50	20, 31, 85	42, 70, 130	120, 211, 365
100	40, 62, 140	84, 133, 235	235, 406, 720

Table 4.1: Rendering time per frame in milliseconds (number of depth planes vs output resolutions). The numbers in each cell are from a GeForce4, a GeForce3, and a GeForce2, respectively. All numbers are measured with five 320×240 input images.

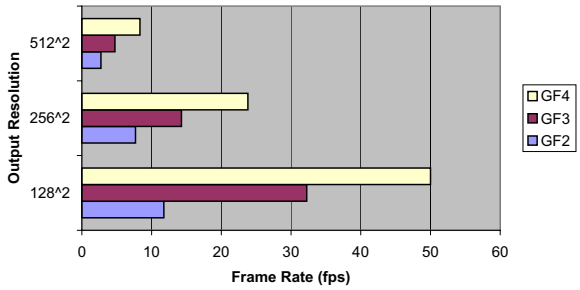


Figure 4.8: Frame rates from three cards at different output resolutions with 50 depth planes—corresponding to the second row in Table 4.1.

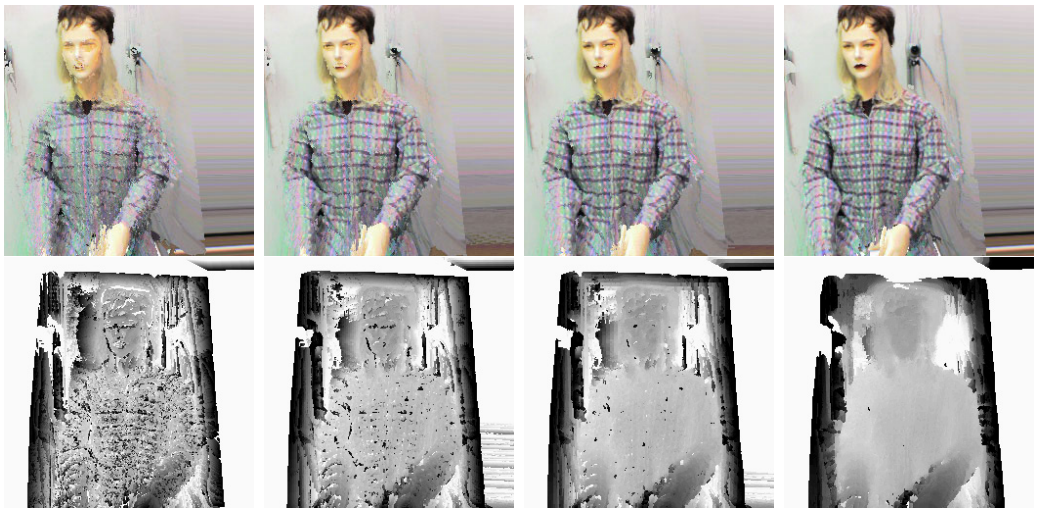


Figure 4.9: Impact of support size on the color or depth reconstruction(using all five cameras); First row: Synthesized images from an oblique viewing angle (view extrapolation) with different levels of aggregation using the MML method. Second row: extracted depth maps using the MML aggregation method. The erroneous depth values in the background are due to the lack of textures in the image. The maximum mipmap level is set from zero to four, corresponding to a support size of 1×1, 2×2, 4×4, and 8×8, respectively.

on the face. The black areas in the depth map are due to the lack of textures on the background wall. They will not impact the synthesized views.

4.5.2 Live Depth Estimation

I also tested real-time VDPC using only a pair of images. If one makes a slight change to enable depth output, real-time VDPC in this case is in fact very similar to standard correlation-based stereo. This flexibility of real-time VDPC may lend itself well for computer vision applications that require real-time depth computation—automatous robotic navigation, for example. So in this section, I only show the depth map generated by real-time VDPC using only two input images. All depth maps shown in this section are encoded based on disparity. So they are referred to as disparity maps from here on.

The first test set is the Tsukuba set, which is widely used in computer vision literature. The results are shown in Figure 4.10, in which I show the disparity maps with different aggregation methods. For disparity maps on the left column, I used the SML method so that the SSD image was only rendered at a single mipmap level to simulate a fixed-size box filter. Note that texture shift trick effectively doubles the size of the filter. So it is equivalent to the use of a 2×2 kernel at mipmap level zero, and a 4×4 kernel at mipmap level one, etc. The right column shows results from using the MML method, i.e., summing up the different mipmap levels. I computed the different disparity maps in each subsequent row by varying the maximum mipmap level (abbreviated as $MaxML$) from zero up to five. We can see that the results using a 1×1 kernel (second row) are almost meaningless. If we use a higher mipmap level, i.e. increase the support size, the results improve for both methods. But the image resolution drops dramatically with the SML method (left column). The disparity map seems to be the best when using the MML method with $MaxML = 4$ (i.e. a 16×16 support). Little is gained when $MaxML > 4$. Results from another widely used stereo pair using $MaxML = 4$ are shown in Figure 4.11.

In terms of performance, I tested my implementation on an NVIDIA GeForce4 Card—a card with four multi-texture units. I found virtually no performance difference when $MaxML$ was changed from one to four. This is not surprising since I can use all four texture units to sum up all four levels in a single pass. If $MaxML$ is set to over four, another additional rendering pass is required, which results in less than 10% increase in calculation time.[1] In practice, I find that setting $MaxML$ to four usually strikes a

[1] I do not use trilinear interpolation in our performance testing, and it seems that in practice setting

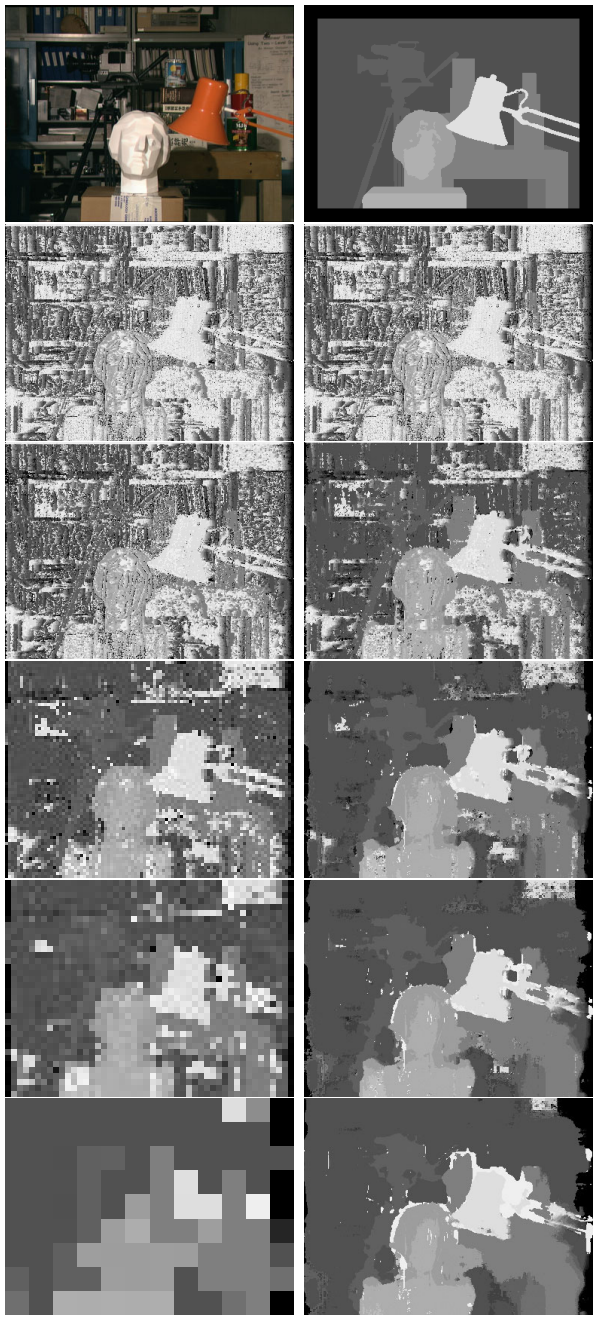


Figure 4.10: Depth results on the Tsukuba data set using only two images. The top-right image shows the ground-truth disparity map. For the remaining rows, on the left column I show the disparity maps from the SML method, while on the right I show the ones from the MML method. The mipmap levels are set to zero to five, corresponding to a support window of size 1×1, 2×2, 4×4, etc.

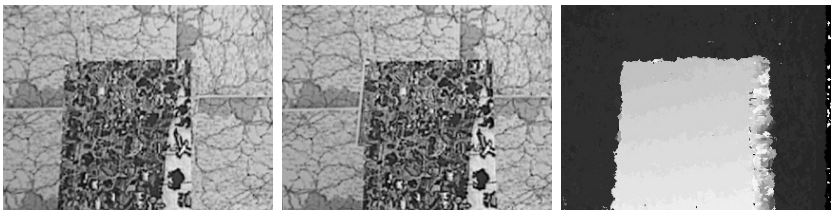


Figure 4.11: Calculated disparity map from another widely-used stereo pair.

Output Size	Search Range	Times (ms)	Times (Hz)	Img. Update (ms)	Read (ms)	Disp. Calc. (M/sec)
512^2	20	71.4	14	(VGA) 5.8×2	6.0	58.9
	50	182	5.50			65.6
	100	366	2.73			68.3
256^2	20	20.0	50	(QVGA) 1.6×2	1.5	53.1
	50	49.9	20			60.0
	100	99.0	10.1			63.2

Table 4.2: Depth estimation performance on an NVIDIA GeForce4 card when summing all mipmap levels. The two input images are 640×480, the maximum mipmap level (MML) is set to 4 in all tests.

good balance between smoothness and preserving small details for stereo. Details of the performance data for the MML method can be found in Table 4.2. On average, the MML method can achieve 50-60 million disparity cacluations/second, including all the overhead to send the images to the graphics board and read back the result. Plotting this data in Figure 4.12, we can see that our algorithm exhibits linear performance with respect to the image size.

I also tested the SML method (results shown in Table 4.3). In this case the frame-rates are higher, especially when going to a higher mipmap level. Note that for higher mipmap levels the number of evaluated disparities per seconds drops because in this case the output disparity map has a lower resolution. This method might be preferred for some applications where speed is more important than detail.

When running under the stereo configuration, my current real-time prototype performs a few additional steps in software, such as radial distortion correction and segmentation.[2] As a proof of concept, these yet-to-be-optimized parts are not fully pipelined with the reconstruction. This overhead slows down the overall reconstruction rate to

$MaxML$ over four has an detrimental effect on the final result.

[2]The cameras are facing a white wall with little texture. So I segment the images to fill the background with different colors.

Base Size	Output Size	Search Range	Times (ms)	Times (Hz)	Overhead (ms)	Disp. Calc. (M/sec)
512^2	128^2 (4×4)	20	2.5	40	12.0	11.7
		50	6.4	15.6		10.7
		100	12.8	7.8		8.86
512^2	256^2 (2×2)	20	28.3	35.3	13.1	31.7
		50	71.4	14.0		38.8
		100	144	6.9		41.7
512^2	512^2	20	40.8	24.5	17.6	89.8
		50	106	9.4		106
		100	207	4.8		117
256^2	128^2 (2×2)	20	12.7	78.7	3.58	20.1
		50	31.6	31.6		23.3
		100	63.1	15.8		24.6
256^2	256^2	20	16.2	61.7	4.7	62.7
		50	40.3	24.8		72.8
		100	80.7	12.4		76.7

Table 4.3: Depth estimation performance on an NVIDIA GeForce4 card when using only a single mipmap level with texture shift enabled. Throughput decreases proportionally to the output resolution because the majority of the time is spent on computing the SSD score. The overhead includes both the image update time and the time to read back the depth map from the frame buffer.

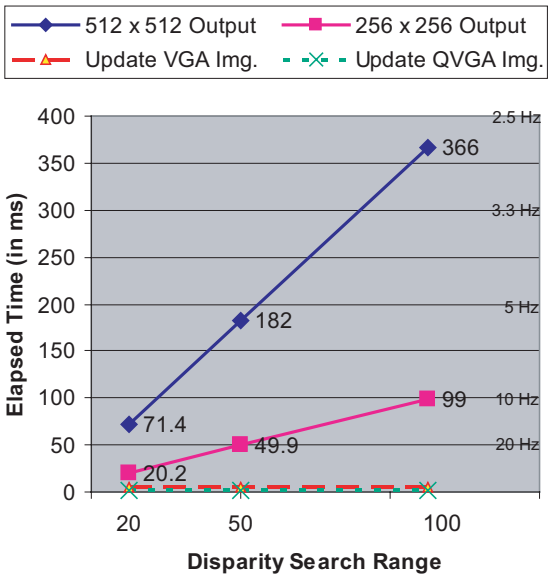


Figure 4.12: Depth estimation performance on an NVIDIA GeForce4 Card. The two image-update curves show the time to update two input images at different resolutions. The data are from Table 4.1.

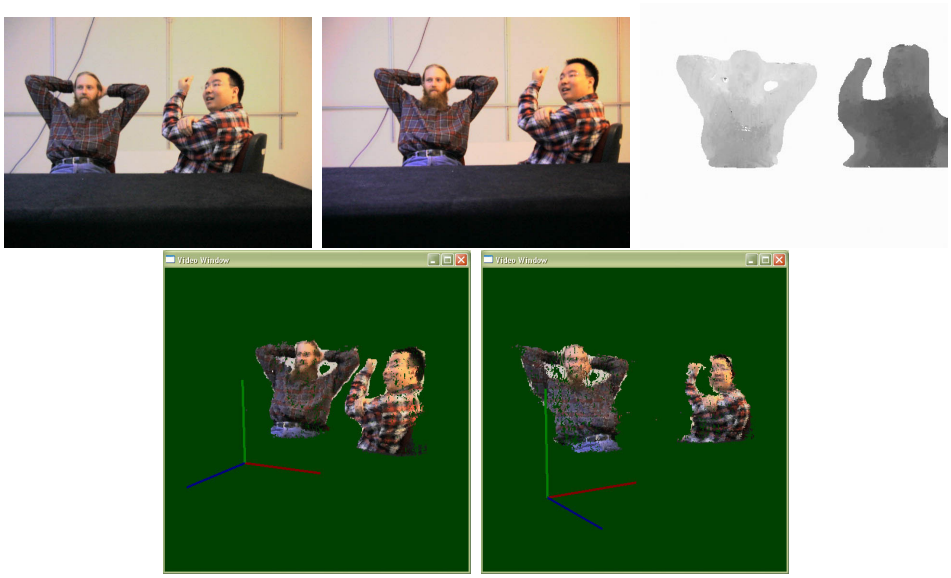


Figure 4.13: Typical results from the real-time online stereo system. The first row shows the two input images and the disparity map. The second row shows the reconstructed 3D point cloud from different perspectives. Some holes in the 3D views are caused by the rendering. I simply render fixed-size (in screen space) points using the GL_POINT primitive.

6-8 frames per second at 256×256 resolution with 100 depth planes. In Figure 4.13, I show a sample stereo pair and a reconstructed depth map. In the real-time system, darker colors in the depth map mean that the object is closer to cameras while brighter colors mean further. To better illustrate the results, I also show the reconstructed 3D point cloud from different perspectives in Figure 4.13. More scenes and their depth maps can be found in Figure 4.14.

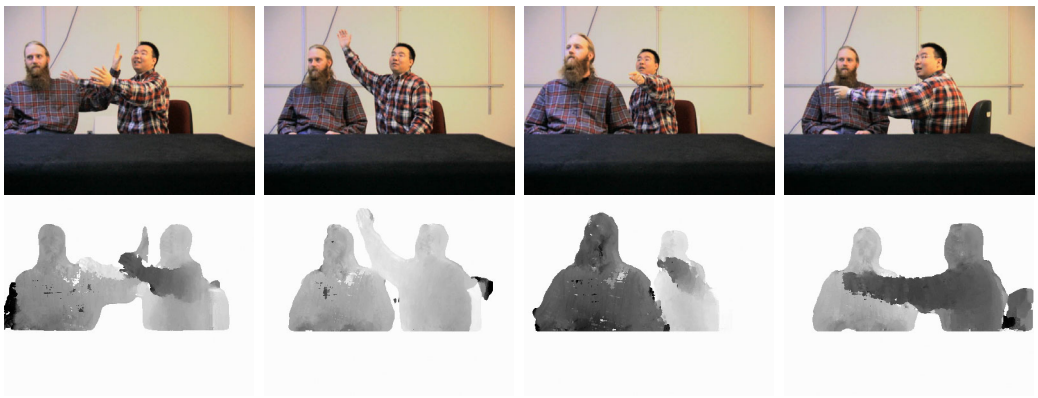


Figure 4.14: More results from our real-time online stereo system. The first row shows the input images; The second row shows the corresponding disparity map.

Chapter 5

Conclusions and Future Work

This dissertation investigates the problem of synthesizing new images from varying viewing angles by processing a set of images gathered from different viewpoints of a scene. Driven by applications such as 3D video-teleconferencing and surgical training, this dissertation research began with the following goals:

1. Use a practical number of input images (from one to two dozen);
2. Be fully automatic;
3. Be robust and accurate for a wide variety of shapes and surfaces (convex, concave, specular, diffuse, etc.); and
4. Work toward real-time, on-line view synthesis (over 10 fps).

The framework presented in this dissertation, Dependent-View Pixel Coloring (VDPC), meets all of these goals. In the next section, I will summarize the innovations of VDPC and how each of them maps to a specific goal.

5.1 Innovations

In contrast to the way the innovations were presented in Chapter 1, I explicitly distinguish between conceptual innovations and design innovations in this section. Conceptual innovations are high-level insights and observations, while design innovations are specific realizations and implementations of conceptual innovations. The VDPC framework is comprised of the design innovations, while the conceptual innovations can lead to new algorithms and techniques.

This dissertation contains the following conceptual innovations:

- **Hybrid and view-dependent estimation.** Unlike traditional approaches in which a complete model is first extracted, I recognized that, for the sake of view synthesis, I only need to recover *enough* information to synthesize the current view. Details can be found in Section 3.3.1.

- **The recognition of the potential power in commodity graphics hardware.** There is a great synergy between image processing and graphics rendering. Both frequently use simple operations applied thousands or even millions of times. Any application having similar characteristics can exploit the fast speed in modern graphics boards to accelerate their computation.

Bearing the above conceptual innovations in mind, I have developed a novel and practical view synthesis framework, *View-Dependent Pixel Coloring*. The main practical innovations of VDPC from a design standpoint are:

- **Hybrid and view-dependent formulation.** Results from controlled experiments shows the advantages of my formulation over traditional view-independent approaches, such as stereo or space carving. Details can be found in Section 3.3.1.

- **Progressive refinement with a view-dependent smoothness constraint.** Experiments results in Section 3.4 have shown that this scheme can recover highly complex shapes (such as a human hand), and, at the same time, be robust in textureless regions. Details can be found in Section 3.3.1.

- **A novel physically-based photo-consistency measure.** Strong results have been obtained using this photo-consistency measure. In addition, this measure requires no light calibration or surface normal estimation, thus it can be used in any existing stereo or space carving algorithms to extend their applicability to a specular environment. Details can be found in Section 3.3.1.

- **Real-time VDPC on commodity graphics hardware.** Details can be found in Chapter 4. At the time of this writing (May 2003), my implementation, when operating to output a depth map, is as fast as the fastest commercial software package available [Inc]. In addition, the graphics processing unit (GPU) continues to be evolving at a faster pace than CPU.

5.2 Historical Notes

The chronological order in which this dissertation research was conducted is, in fact, the reverse of the way it is laid out in this dissertation. I have briefly introduced the history in Chapter 1, here is a more detailed account.

I started this dissertation with research on real-time VDPC, which was motivated by the 3D Tele-Immersion project (http://www.advanced.org/tele-immersion/) in which a real-time solution for view synthesis was essential. Together with Prof. Greg Welch, my thesis advisor, we made the observation that for the sake of view synthesis, we did not need to estimate a full 3D model, we only needed to estimate a color for each pixel in the desired image. Based on this concept, I developed a view-dependent formulation for view synthesis that can be effectively accelerated on commodity graphics hardware to achieve real-time performance.

While real-time VDPC's speed is surprisingly fast, its reconstruction quality is mediocre. The lack of visibility handling substantially limits the usable viewing range. Around that time, the space carving framework was introduced [KS00]. Its elegant visibility handling attracted me. The formulation looked quite simple, yet from the results, it seemed to be very effective. But space carving also has a number of problems, like its sensitivity to the global threshold and difficulties with textureless regions and specular highlights. I decided to focus my dissertation research on view synthesis. My basic goal was to combine some ideas from space carving with real-time VDPC and extend them for more general scenes, in particular, scenes with a substantial amount of textureless regions and specular highlight—the kind of scenes commonly found in a surgical environment. After careful deliberation and discussion with Prof. Welch, it seemed at that time that a full probabilistic modeling of geometry, surface materials, and lighting had a lot of promise.

During the course of developing the full probabilistic framework, several papers about probabilistic modeling of the space carving problem came out [dBV99, BDC01, AD01, BFK02]. Some of the ideas in them were actually quite similar to what I had intended to do then. However, I found that the results from these new probabilistic space carving algorithms were not significantly better than those from the original algorithm. In fact, one author even mentioned in his dissertation [Bro01] that if he were to choose a space carving algorithm, he probably would *not* choose the full probabilistic framework he developed. The additional sophistication required in the implementation outweighs the improvement in the results. The difficulty is in the handling of visibility.

In theory, a probability formulation should consider *all* possible visibility configurations for a voxel. For a set of n input images, there are 2^n configurations for each voxel. To avoid a combinatorial search, the visibility tests in these published works are either approximated based on heuristics or solved in a stochastic manner through hundreds of iterations. I believed that an accurate treatment of visibility is crucial for a multi-view reconstruction. The results in these papers seemed to support my belief. So I decided to treat visibility exactly in my dissertation work. Given the demonstrated effectiveness of the deterministic visibility treatment in the original space carving algorithm, I decided to adopt it without further change.

While reading published papers related to space carving, I found that many of them mentioned the problem of specular highlights and the lack of regularization terms (shape smoothness). I was working on these problems and was encouraged to see that many of my fellow researchers agreed that they were the right and important ones. Even if I did not make a contribution to visibility treatment, solving the other problems would still have a significant impact. Research on these problems finally led to this dissertation.

I would also like to acknowledge that the original idea for the photo-consistency measure I developed is attributable to Prof. Marc Pollefeys. My first idea was to use singular value decomposition to find the diffuse component. As I discussed the problem with Prof. Pollefeys, he pointed out that there had already been some work in computer vision on detecting specular highlights based on color sample distributions in the RGB color space. Based on his advice, I developed a novel photo-consistency measure that has been very effective in practice.

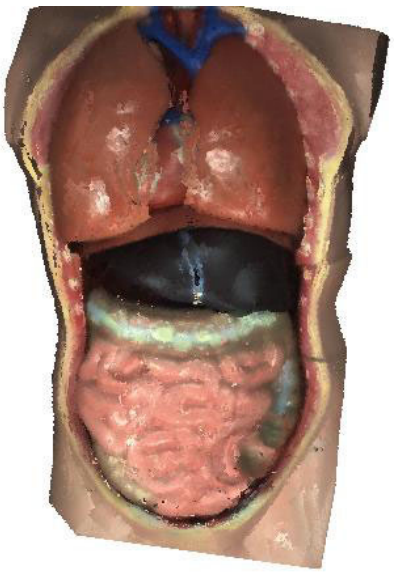
In order to evaluate the improvements of my approach over existing approaches, in particular, these probabilistic space carving approaches [dBV99, BDC01, AD01, BFK02], I have attempted to contact several authors, asking them if they would be willing to apply their methods to my data set or share their implementations with me. It seems that this is the only practical way to compare my results with others since there is no established data set or standard to evaluate multi-view reconstruction algorithms. (I will discuss possible ways for quantitative evaluations in the next section) Only Dr. Adrian Broadhurst, an author of [BDC01], kindly agreed. He generated only a top view of the teaching model data set (the images shown in Figure 1.1). Figure 5.1(a) shows the result I received. For comparison, I also included images synthesized by VDPC (a top view and a side view) in Figure 5.1(b). It is difficult to judge the quality of the recovered shape from a single view. Comparing the two top views, surface boundaries with substantial specular highlights, such the intestine region and the area between the

lungs and the heart, are less clear in Figure 5.1(a). Part of the legs (textureless regions) are also missing in Figure 5.1(a). However, the blue color of the vein near the neck is more obvious in Figure 5.1(a).

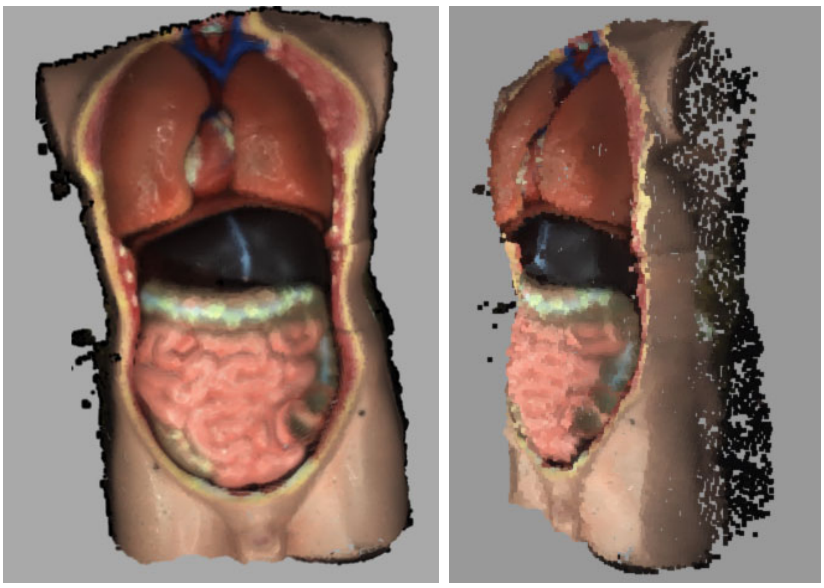
5.3 Future work

I believe the major difficulty in creating photo-realistic imagery, which used to be in rendering, is now shifting to modeling—how to create a model in minute detail so that it will match the visual acuity of human vision system. The essence of this dissertation is in creating models optimized for view synthesis, including the representation of the model, the formulation of novel constraints, and the realization of several practical algorithms. Thinking along these directions will lead to new algorithms and techniques for view synthesis. For instance, how can we explore the temporal coherence of a dynamic scene in VDPC? Would it be possible to reliably estimate parametric surface properties? Is there an optimal sampling pattern for a given scene, i.e., how many cameras are needed and where are the optimal locations for them? Can we deal with un-calibrated cases in which cameras are moving? Or is there an accurate and efficient probabilistic treatment of visibility to make a full probabilistic estimation possible? Questions like these are very interesting to explore.

One primary motivation of this dissertation is to find an effective solution for real-time online view synthesis to enable tele-immersion. Traditionally, tele-immersion incorporates a centralized architecture. That is, input images are gathered and sent to a central server in which a complete view-independent model is exacted, then the same model is distributed to every participating site to be rendered from a desired viewpoint. In contrast to this centralized "push" model, VDPC can be used to facilitate a distributed "pull" model. That is, each site can directly *pull* the necessary raw input pixels from a broadcasting (or multi-casting) network, and then render the desired images from the pixel data directly. This architecture eliminates the need for a powerful central server. In addition, it is more effective in today's heterogenous computing environment. For example, a mobile user is calling in through her cell phone with a color screen, she probably cannot make full use of the model that is optimized for high resolution displays. It would be very interesting to design both software and hardware for this kind of "just-in-time" and "just-enough" distributed visualization systems. From an even higher level, I believe computer graphics is migrating from a centralized paradigm to a distributed paradigm in which a cluster of inexpensive PCs



(a) A top view synthesized using
the method in [BDC01].



(b) A top view and a side view synthesized using VDPC

Figure 5.1: Comparing VDPC with a probabilistic space carving algorithm.

are used to accomplish complex rendering tasks. This shift follows the general trend in computing. When computers first emerged a few decades ago, the concept of centralized computing was popular (and only feasible at that time too), but it turned out that distributed solutions are more viable and popular, for example, the world wide web and file sharing programs.

Another important thing to look at is the potential of commodity graphics hardware for other tasks. The development of commodity graphics hardware during the last few years has been phenomenal, both in term of speed and capability. Today (as of May 2003), the latest cards on the market achieve over 50 gigaflops, and the program languages for these cards, Cg for example [NVI02], are comparable to C code. One can almost claim there is already a very powerful Digital Signal Processor (DSP) on every desktop PC that can be used for computing problems both within the computer graphics domain or outside it.

Looking into the immediate future, there are a number of issues and possible extensions related to this dissertation work.

Extension to more general materials The photo-consistency measure introduced is valid only for dielectric materials. Metal, for example, is not a dielectric material. However, it is possible to extend the basic idea to include more general materials. In the RGB color space, the reflected light from a piece of metal is bound to have a different "signature" than that from a piece of dielectric material. It is likely to be a high order curve or surface, instead of a line or a point. While detecting any signature algorithmically is possible in theory (a pattern recognition problem), it has yet to be seen whether or not a high-order curve or surface can be detected reliably from a small collection of pixel samples. And if a more sophisticated signature can be detected, does it provide enough disambiguating power to be useful in practice? Any greater degree of freedom in the signature is likely to reduce the robustness. Or does it help to formulate a photo-consistency measure in other color spaces? Questions like these remain to be answered.

Put back specular highlights Careful readers may have already noticed that all the synthesized images presented lacked specular highlights. This is because I chose to assign an optimized object color to every pixel. On the one hand, my method is quite effective in removing specular highlights to reveal the true color of the object. On the other hand, scenes without specular highlights may sometimes seem two-dimensional.

A quick fix is to use view-dependent texture mapping, i.e., using the pixel color from the closest input image. A more elaborated approach is to include surface normal estimation. Surface normals can be estimated as a post-processing step. Alternatively, VDPC can be extended to estimate surface normals directly, probably as a part of the photo-consistency evaluation. With surface normal information, one can re-light the scene with synthetic lighting.

Quantitative evaluations I believe that the results presented in this dissertation (Section 3.4) demonstrate visually substantial improvements over the state of the art, especially in textureless and/or specular regions. However, the results from real imagery lack *quantitative* evaluations. In general, quantitative evaluation of any view-synthesis/reconstruction method is difficult. Different methods make different assumptions. So one method might perform well with one data set, but badly with another set, and another method might do the contrary.

Recently, Scharstein and Szeliski proposed an evaluation method of stereo algorithms [SS02]. Their evaluation is designed for dense (pixel-by-pixel) two-frame stereo algorithms only, under the Lambertian surface assumption. Two metrics are used. One is geometry-based, comparing the estimated depth map with the ground truth. The other is image-based: first render the textured depth map from a different viewpoint, then compare the synthesized image with a ground-truth image.

While both metrics can be used to evaluate VDPC, there are a number of practical issues. First, for multi-view methods like VDPC, there are many more compounding factors that could affect the final results, such as the number of input images, their spatial locations, and the reconstruction resolution. Secondly, it is my belief that any reconstruction/view-synthesis algorithm should be evaluated on *real* scenes. In a synthetic scene, everything is so ideal that the quantitative results from it rarely indicate an algorithm's performance in practice. But how can we obtain the ground-truth depth data for a real scene? In [SS02], depth maps from real scenes were manually labelled. The alternative image-based metric seems more practical, but measuring image fidelity is a very tricky business. The fact that VDPC deals with view-dependent specular highlights makes the evaluation more complicated. Using a perceptually-based error metric, such as the Sarnoff Just-Noticeable Difference (JND) [LF97], may provide a more meaningful measure of image fidelity.

In conclusion, the design of quantitative evaluation methodology and procedures for multi-view techniques is a research venue that deserves attention in both the computer

vision and the computer graphics communities.

Real-time VDPC Improvement Real-time VDPC was implemented two years ago when programmable graphics hardware just began to emerge. Today's graphics hardware has much more programmability, thus some simplifications that had to be imposed previously to fit VDPC on graphics hardware two years ago are no longer necessary. For instance, the implementation of the LMF measure (introduced in Section 3.1.4) on graphics hardware is entirely possible. In addition, the occlusion problem may be solved using the new texture-look-up functionality to see if a color sample should be included for the photo-consistency evaluation. If real-time VDPC incorporates these changes, I expect to see substantial improvement in image quality with little speed degradation.

As graphics hardware is getting more and more powerful, I could even imagine a "smart" display device with built-in cameras, a graphics card, and a network port. Such a device can be deployed anywhere on the world (as long as it can receive raw video streams) to allow interactive 3D distributed meetings as envisioned in the Office of the Future [RWC+98].

Bibliography

[1.301] OpenGL Specification 1.3, August 2001. http://www.opengl.org/developers/documentation/ version13/glspec13.pdf.

[AB91] E. H. Adelson and J. Bergen. The Plenoptic Function and the Elements of Early Vision. In *Computational Models of Visual Processing*, page 320, Cambridge, MA, August 1991. MIT Press.

[AD01] M. Agrawal and L. Davis. A Probabilistic Framework for Surface Reconstruction from Multiple Images. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[AH88] N. Ayache and C. Hansen. Rectification of Images for Binocular and Trinocular Stereovision. In *Proceedings of International Conference on Pattern Recognition*, pages 11–16, 1988.

[Ana89] P. Anandan. A Computational Framework and an Algorithm for the Measurement of Visual Motion. *International Journal of Computer Vision (IJCV)*, 2(3):283–310, 1989.

[And82] D. Anderson. Hidden Line Elimination in Projected Grid Surfaces. *ACM Transactions on Graphics*, October 1982.

[Arn83] R. D. Arnold. Automated Stereo Perception. Technical Report AIM-351, Artificial Intelligence Laboratory, Stanford University, 1983.

[Ash93] I. Ashdown. Near-field photometry: A new approach. *Journal of the Illuminating Engineering Society*, 22(1):163–180, Winter 1993.

[AV89] N. Ahuja and J. Veenstra. Generating Octrees from Object Silhouettes in Orthographic Views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(2):137–149, 1989.

[BA93] M. J. Black and P. Anandan. A Framework for the Robust Estimation of Optical Flow. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 231–236, 1993.

[Bar89] S. T. Barnard. Stochastic Stereo Matching over Scale. *International Journal of Computer Vision (IJCV)*, 3(1):17–32, 1989.

[Bas92] Ronen Basri. On the uniqueness of correspondence under orthographic and perspective projections. In *Proceeding of Image Understanding Workshop*, page 875C884, 1992.

[Bas95] M. Bass. *Handbood of Optics*. McGraw-Hill, New York, 1995.

[BBH93] R. C. Bolles, H. H. Baker, and M. J. Hannah. The JISCT Stereo Evaluation. In *DARPA Image Understanding Workshop*, pages 263–274, 1993.

[BBM+01] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured Lumigraph Rendering. In *Proceedings of SIGGRAPH 2001*, Los Angels, August 2001.

[BC00] Adrian Broadhurst and Roberto Cipolla. A Statistical Consistency Check for the Space Carving Algorithm. In *Proceedings of 11th British Machine Vision Conference*, pages 282–291, 2000.

[BDC01] A. Broadhurst, T. Drummond, and R. Cipolla. A Probabilistic Framework for Space Carving. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 388–393, 2001.

[Bel96] P. N. Belhumeur. A Bayesian Approach to Binocular Stereopsis. *International Journal of Computer Vision (IJCV)*, 19(3):237–260, 1996.

[BF82] S.T. Barnard and M.A. Fischler. Computational Stereo. *Computer Surveys*, 14(4):553–572, 1982.

[BFK02] R. Bhotika, D. J. Fleet, and K. N. Kutulakos. A Probabilistic Theory of Occupancy and Emptiness. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 112–132, 2002.

[BGCM02] C. Buehler, S. J. Gortler, M. Cohen, and L. McMillan. Minimal Surfaces for Stereo Vision. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 885–899, 2002.

[BI99] A. F. Bobick and S. S. Intille. Large occlusion stereo. *International Journal of Computer Vision (IJCV)*, 33(3):181–200, 1999.

[BJ80] P. Burt and B. Julesz. A Gradient Limit for Binocular Fusion. *Science*, 208:615–617, 1980.

[BM92] P. N. Belhumeur and D. Mumford. A Bayesian Treatment of the Stereo Correspondence Problem Using Half-occluded Regions. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 506–512, 1992.

[BN76] J.F. Blinn and M.E Newell. Texture and Reflection in Computer Generated Images. *CACM*, 19(10):542–547, October 1976.

[BN95] D. Bhat and S. Nayar. Stereo in the presence of specular reflection. In *Proceedings of International Conference on Computer Vision (ICCV)*, page 1086C1092, 1995.

[BR96] M. J. Black and A. Rangarajan. On the Unification of Line Processes, Outlier Rejection, and Robust Statistics with Applications in Early Vision. *International Journal of Computer Vision (IJCV)*, 19(1):57–91, 1996.

[Bro01] Adrian Broadhurst. *A Probabilistic Framework for Space Carving*. PhD thesis, Trinity College, University of Cambridge, 2001.

[BSA98] S. Baker, R. Szeliski, and P. Anandan. A layered approach to stereo reconstruction. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, Santa Barbara, CA, June 1998.

[BT98] S. Birchfield and C. Tomasi. A Pixel Dissimilarity Measure That Is Insensitive to Image Sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(4):401–406, 1998.

[BVZ98] Y. Boykov, O. Veksler, and R. Zabih. A Variable Window Approach to Early Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12), December 1998.

[BVZ01] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 23(11):1222–1239, 2001.

[BZ87] A. Blake and A. Zisserman. *Visual Reconstruction*. MIT Press, 1987.

[CA86] C.H. Chien and J.K. Aggarwal. Volume surface octrees for the presentation of 3d objects. *Computer Vision, Graphics and Image Processing*, 36:100–113, 1986.

[CB90] P. B. Chou and C. M. Brown. The Theory and Practice of Bayesian Image Labeling. *International Journal of Computer Vision (IJCV)*, 4(3):185–210, 1990.

[CBL99] Chun-Fa Chang, Gary Bishop, and Anselmo Lastra. LDI tree: A hierarchical representation for image-based rendering. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 291–298, Los Angeles, 1999. Addison Wesley Longman.

[CF99] E. Camahort and D. Fussell. A geometric study of light field representations. Technical Report TR99-35, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas, 1999.

[Che95] S. E. Chen. Quicktime VR: An Image-Based Approach to Virtual Environment Navigation. In *Proceedings of SIGGRAPH 1995*, pages 29–38, 1995.

[Chh01] Vikram Chhabra. Reconstructing specular objects with image based rendering using color caching. Master's thesis, Worcester Polytechnic Institute, 2001.

[CHRM96] I. J. Cox, S. L. Hingorani, S. B. Rao, and B. M. Maggs. A Maximum Likelihood Stereo Algorithm. *Computer Vision and Image Understanding (CVIU)*, 63(3):542–567, 1996.

[CK01] Rodrigo L. Carceroni and Kiriakos N. Kutulakos. Multi-View Scene Capture by Surfel Sampling: From Video Streams to Non-Rigid 3D MotionShape and Reflectance. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2001.

[CKBH00] G. K. M. Cheung, T. Kanade, J-Y. Bouguet, and M. Holler. A Real time System for Robust 3D Voxel Reconstruction of Human Motions. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 714–720, 2000.

[CLF98] Emilio Camahort, Apostolos Lerios, and Donald Fussell. Uniformly Sampled Light Fields. In *Eurographics Rendering Workshop 1998*, pages 117–130, 1998.

[CMS99] B. Culbertson, T. Malzbender, and G. Slabaugh. *Generalized Voxel Coloring*, volume 1883 of *Lecture Notes in Computer Science*, pages 100–115. Springer-Verlag, 1999.

[Col96] R. Collins. A Space-Sweep Approach to True Multi-Image Matching. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 358–363, June 1996.

[CTCS00] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic Sampling. In *Proceedings of SIGGRAPH 2000*, page 307318, New Orleans, August 2000.

[DA89] U. Dhond and J. Aggrawal. Structure from stereo: a review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):14891510, 1989.

[Dav97] E.R. Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Academic Press, 1997.

[dBV99] J.S. de Bonet and P. Viola. Poxels: Probabilistic Voxelized Volume Reconstruction. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 418–425, 1999.

[DBY98] Paul E. Debevec, George Borshukov, and Yizhou Yu. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. In *9th Eurographics Rendering Workshop*, Vienna, Austria, June 1998.

[DSV97] Lucia Darsa, Bruno Costa Silva, and Amitabh Varshney. Navigating Static Environments Using Image-Space Simplification and Morphing. In *Proceedings of Symposium on I3D Graphics*, page 2534, 1997.

[DTM96] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *SIGGRAPH*, pages 11–20, August 1996.

[Dye01] C. R. Dyer. Volumetric scene reconstruction from multiple views. In L. S. Davis, editor, *Foundations of Image Understanding*, pages 469–489. Kluwer, 2001.

[Fau93] O. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.

[FHM+93] O. Faugeras, B. Hotz, H. Mathieu, T. Viville, Z. Zhang, P. Fua, E. Thron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy. Real time sorrelation-based stereo: Algorithm, amplementations and application. Technical Report 2013, INRIA, August 1993.

[FP03] D. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*, chapter 6, page 119. Prentice Hall, 2003.

[FS00] T. Feng and H.-Y. Shum. An optical analysis of light field rendering. Technical report MSR-TR-2000-38, Microsoft Research, May 2000.

[GG84a] D. Geiger and F. Girosi. Parallel and Deterministic Algorithms for MRF's: Surface Reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 6(6):721–741, 1984.

[GG84b] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distribution, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 6(6):721–741, 1984.

[GGSC96] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *Proceedings of SIGGRAPH 1996*, pages 43–54, New Orleans, August 1996.

[GLY92] D. Geiger, B. Ladendorf, and A. Yuille. Occlusions and Binocular Stereo. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 425–433, 1992.

[Gre86] N Greenem. Environment Mapping and Other Applications of World Projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

[Gri85] W. E. L. Grimson. Computational experiments with a feature based stereo algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 7(1):17–34, 1985.

[HA00] Nicolas Holzschuch and Laurent Alonso. Using graphics hardware to speed-up visibility queries. *Journal of Graphics Tools*, 5(2):33–47, 2000.

[Hal94] M. Halle. Holographic stereograms as discrete imaging systems. In *Proceedings of SPIE*, volume 2176 of *Practical Holography VIII*, pages 73–84, May 1994.

[Han74] M. J. Hannah. *Computer Matching of Areas in Stereo Images*. PhD thesis, Stanford University, 1974.

[HiAA97] Youichi Horry, Ken ichi Anjyo, and Kiyoshi Arai. Tour Into the Picture: Using a Spidery Mesh Interface to Make Animation from a Single Image. In *SIGGRAPH*, June 1997.

[HIG03] H. Hirschmuller, P. Innocent, and J. Garibaldi. Real-Time Correlation-Based Stereo Vision with Reduced Border Errors. *International Journal of Computer Vision*, 47(1-3), April-June 2003.

[Hir01] Heiko Hirschmuler. Improvements in Real-Time Correlation-Based Stereo Vision. In *Proceedings of IEEE Workshop on Stereo and Multi-Baseline Vision*, pages 141–148, Kauai, Hawaii, December 2001.

[Hor86] B.K.P. Horn. *Robot Vision*. MIT Press, 1986.

[IAH95] M. Irani, P. Anandan, and S. Hsu. Mosaic based representations of video sequences and their applications. In *Proceedings of ICCV*, page 605611, Cambridge, Massachusetts, June 1995.

[IG98] H. Ishikawa and D. Geiger. Occlusions, Discontinuities, and Epipolar Lines in Stereo. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 232–248, 1998.

[IKL+99] Kenneth E. Hoff III, John Keyser, Ming C. Lin, Dinesh Manocha, and Tim Culver. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. In *Proceeding of SIGGRAPH 99*, pages 277–286, August 1999.

[IMG00] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically Reparameterized Light Fields. In *Proceedings of SIGGRAPH 2000*, pages 297–306, August 2000.

[Inc] Point Grey Research Inc. http://www.ptgrey.com.

[IPL98] Insung Ihm, Sanghoon Park, and Rae Kyoung Lee. Rendering of Spherical Light Fields. In *Pacific Graphics 97*, Seoul, Korea, 1998.

[IZLM01] Kenneth E. Hoff III, Andrew Zaferakis, Ming C. Lin, and Dinesh Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. In *2001 ACM Symposium on Interactive 3D Graphics*, pages 145–148, March 2001. ISBN 1-58113-292-1.

[Joh00] Richard Arnold Johnson. *Miller and Freund's Probability and Statistics for Engineers*. Prentice Hall, 2000.

[JYS01] H. Jin, A. Yezzi, and S. Soatto. Variational multiframe stereo in the presence of specular reflections. Technical Report TR01-0017, UCLA, 2001.

[Kan94] T. Kanade. Development of a Video-rate Stereo Machine. In *DARPA Image Understanding Workshop*, page 549C557, Monterey, CA, 1994. Morgan Kaufmann Publishers.

[Kil00] Mark J. Kilgard. A Practical and Robust Bump-mapping Technique for Today's GPUs. In *Game Developers Conference 2000*, San Jose, California, March 2000.

[KO94a] T. Kanade and M. Okutomi. A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 16(9):920–932, 1994.

[KO94b] T. Kanade and M. Okutomi. A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):920 – 932, September 1994.

[KS99] K.N. Kutulakos and S. M. Seitz. A Theory of Shape by Space Carving. In *Proceedings of International Conference on Computer Vision (ICCV)*, page 307C314, 1999.

[KS00] K. Kutulakos and S. M. Seitz. A Theory of Shape by Space Carving. *International Journal of Computer Vision (IJCV),*, 38(3):199–218, 2000.

[KSC01] S. B. Kang, R. Szeliski, and J. Chai. Handling Occlusions in Dense Multi-view Stereo. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[KSK98] Reinhard Klette, Karsten Schlns, and Andreas Koschan. *Computer Vision: Three-Dimensional Data from Images*. Springer, 1998.

[Kut00] K. N. Kutulakos. Approximate N-view Stereo. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 67–83, 2000.

[KYO+96] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka. A Stereo Engine for Video-rate Dense Depth Mapping and Its New Applications. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 196–202, June 1996.

[KZ01] V. Kolmogorov and R. Zabih. Computing Visual Correspondence with Occlusions Using Graph Cuts. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 508–515, 2001.

[KZ02a] Vladimir Kolmogorov and Ramin Zabih. Multi-camera Scene Reconstruction via Graph Cuts. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 82–96, 2002.

[KZ02b] Vladimir Kolmogorov and Ramin Zabih. What Energy Functions Can Be Minimized via Graph Cuts? In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 65–81, 2002.

[Lau94] A. Laurentini. The Visual Hull Concept for Silhouette Based Image Understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, February 1994.

[Lau95] A. Laurentini. How Far 3D shapes Can be Understood from 2D Silhouettes? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):188–195, 1995.

[Lau97] A. Laurentini. How Many 2D Silhouettes Does It Take to Reconstruct a 3D Object? *Computer Vision and Image Understanding*, 67(1):81–87, 1997.

[LB92] S. W. Lee and R. Bajcsy. Detection of Specularity using Color and Multiple Views. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 643–653, 1992.

[Lev71] R. Levin. Photometric characteristics of light controlling apparatus. *Illuminating Engineering*, 66(4):205–215, 1971.

[LF97] J. Lubin and D. Fibush. Sarnoff JND vision model, 1997. T1A1.5 Working Group Document 97-612, ANSI T1 Standards Committee, 1997.

[LH96] M. Levoy and P. Hanrahan. Light Field Rendering. In *Proceedings of SIGGRAPH 1996*, pages 31–42, New Orleans, August 1996.

[LK81] B. D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application in Stereo Vision. In *Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 674–679, Vancouver, Canada, 1981.

[LKHZ98] Wei Li, Qi Ke, Xiaohu Huang, and Nanning Zheng. Light field rendering of dynamic scenes. *Machine Graphics and Vision*, 7(3):551–563, 1998.

[LLF98] Y. G. Leclerc, Q.-T. Luong, and P. Fua. Self-consistency: A novel approach to characterizing the accuracy and reliability of point correspondence algorithms. In *DARPA Image Under-standing Workshop*, pages 793–807, November 1998.

[LLK+02] Stephen Lin, Yuanzhen Li, Sing Bing Kang, Xin Tong, and Heung-Yeung Shum. Diffuse-Specular Separation and Depth Recovery from Image Sequences. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 210–224, 2002.

[LLL+02] Y. Li, S. Lin, H. Lu, S.B. Kang, and H-Y Shum. Multibaseline Stereo in the Presence of Specular Reflections. In *International Conference on Pattern Recognition*, pages 573–576, 2002.

[LM01] E. Scott Larsen and David K. McAllister. Fast Matrix Multiplies using Graphics Hardware. In *Proceeding of Super Computer 2001*, November 2001.

[Lok01] B. Lok. Online Model Reconstruction for Interactive Virtual Environments. In *Proceedings 2001 Symposium on Interactive 3D Graphics*, pages 69–72, Chapel Hill, North Carolina, March 2001.

[LS97] Jed Lengyel and John Snyder. Rendering with Coherent Layers. In *SIGGRAPH*, June 1997.

[LS00] Z.-C. Lin and H.-Y. Shum. On the numbers of samples needed in light field rendering with constant-depth assumption. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2000.

[LTH86] D. Laidlaw, W. Trumbore, and John F. Hughes. Constructive solid geometry for polyhedral objects. *ACM Computer Graphics (SIGGRAPH)*, 20(4):161–170, 1986.

[LZ99a] David Liebowitz and Andrew Zisserman. Combining Scene and Auto-Calibration Constraints. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 293–300, 1999.

[LZ99b] C. Loop and Z. Zhang. Computing Rectifying Homographies for Stereo Vision. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 125–131, 1999.

[MA83] W. N. Martin and J. K. Aggarwal. Volumetric Description of Objects from Multiple Views. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 5(2):150–158, 1983.

[Mar82] D. Marr. *Vision*. W. H. Freeman and Company, 1982.

[MB95] L. McMillan and Gary Bishop. Plenoptic Modeling: An Image-Based Rendering System. In *Proceedings of SIGGRAPH 1995*, pages 39–46, 1995.

[MBR+00] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image-Based Visual Hulls. In *Proceedings of SIGGRAPH 2000*, pages 369–374, New Orleans, August 2000.

[McM97] L. McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997.

[Mee90] J. Meehan. *Panoramic Photography*. Watson-Guptill, 1990.

[MID02] J. Mulligan, V. Isler, and K. Daniilidis. Trinocular Stereo: A New Algorithm and its Evaluation. *International Journal of Computer Vision (IJCV), Special Issue on Stereo and Multi-baseline Vision*, 47:51–61, 2002.

[MKJ96] S. Moezzi, D.Y. Kuramura, and R. Jain. Reality Modeling and Visualization from Multiple Video Sequences. *IEEE Computer Graphics and Applications*, 16(6):58–63, 1996.

[MMB97] W. Mark, L. McMillan, and G. Bishop. Post-rendering 3d warping. In *Proceedings of Symposium on I3D Graphics*, pages 7–16, 1997.

[MMP87] J. Marroquin, S. Mitter, and T. Poggio. Probabilistic Solution of Ill-posed Problems in Computational Vision. *Journal of the American Statistical Association*, 82(397):76–89, 1987.

[MP76] D. Marr and T. Poggio. Cooperative Computation of Stereo Disparity. *Science*, 194:283–287, 1976.

[MP94] S. Mann and R. W. Picard. Virtual bellows: Constructing high-quality images from video. In *Proceedings of ICIP*, volume I, page 363367, Austin, Texas, November 1994.

[MSK89] L. Matthies, R. Szeliski, and T. Kanade. Kalman Filter-based Algorithms for Estimating Depth from Image Sequences. *International Journal of Computer Vision (IJCV)*, 3:209–236, 1989.

[Nay97] Shree K. Nayar. Catadioptric Omnidirectional Camera. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, Peurto Rico, June 1997.

[NFA88] H. Noborio, S. Fukada, and S. Arimoto. Construction of the Octree Approximating Three-Dimensional Objects by Using Multiple Views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):769–782, 1988.

[Nie97] W. Niem. Error analysis for silhouette-based 3D shape estimation from multiple views. In *Proc. Int. Workshop on Synthetic-Natural Hybrid Coding and Three-Dimensional Imaging*, 1997.

[Nis84] H. Nishihara. PRISM, a Pratical Real-Time Imaging Stereo Matcher. Technical Report A.I. Memo 780, MIT, 1984.

[Nvi] Nvidia. http://www.nvidia.com.

[NVI02] NVIDIA. Cg: C for Graphics, 2002. http://www.cgshaders.org/.

[OK92] M. Okutomi and T. Kanade. A Locally Adaptive Window for Signal Matching. *International Journal of Computer Vision (IJCV)*, 7(2):143–162, 1992.

[OK93] M. Okutomi and T. Kanade. A Multiple-baseline Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 15(4):353–363, 1993.

[Oko76] Takanori Okoshi. *Three-Dimensional Imaging Techniques*. Academic Press, Inc., New York, 1976.

[O.V01] O.Veksler. Stereo Matching by Compact Windows via Minimum Ratio Cycle. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 540–547, 2001.

[PD96] D. Papadimitriou and T. Dennis. Epipolar line estimation and rectification for stereo image pairs. *IEEE Transactions on Image Processing*, 5(4):672–676, 1996.

[PG97] M. Pollefeys and L. Van Gool. A Stratified Approach to Self-calibration. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 407–412. IEEE Computer Society Press, 1997.

[Pho75] Bui-Tuong Phong. Illumination for computer generated pictures. *CACM*, 18(6):3111–317, June 1975.

[PKG98] M. Pollefeys, R. Koch, and L. Van Gool. Self-Calibration and Metric Reconstruction in spite of Varying and Unknown Internal Camera Parameters. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 90–95. Narosa Publishing House, 1998.

[PKG99] M. Pollefeys, R. Koch, and L. Van Gool. A Simple and Efficient Rectification Method for General Motion. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 496–501, Corfu, Greece, 1999.

[PMF85] S. B. Pollard, J. E. W. Mayhew, and J. P. Frisby. PMF: A Stereo Correspondence Algorithm Using a Disparity Gradient Limit. *Perception*, 14:449–470, 1985.

[Pol98] M. Pollefeys. Visual 3D Modeling from Images. online tutorial, 1998. http://www.cs.unc.edu/ marc/tutorial/index.html.

[Pot87] M. Potmesil. Generating Octree Models of 3D Objects from their Silhouettes in a Sequence of Images. *Computer Vision, Graphics and Image Processing*, 40:1–20, 1987.

[Pow78] M.J.D Powell. A Fast Algorithm for Nonlinearly Constrained Optimization Calculations. *Numerical Analysis*, 630, 1978. Lecture Notes in Mathematics, Springer Verlag.

[PPMF86] S. Pollard, J. Porrill, J. Mayhew, and J. Frisby. Disparity Gradient, Lipschitz Continuity, and Computing Binocular Correspondance. In O.D. Faugeras and G. Giralt, editors, *Robotics Research: The Third International Symposium*, volume 30, pages 19–26. MIT Press, 1986.

[Pra85] K. Prazdny. Detection of Binocular Disparities. *Biological Cybernetics*, 52(2):93–99, 1985.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, 1985.

[PTK85] Tomaso Poggio, Vincent Torre, and Christof Koch. Computational Vision and Regularization Theory. *Nature*, 317(314-319), 1985.

[RC98] S. Roy and I. J. Cox. A Maximum-flow Formulation of the N-camera Stereo Correspondence Problem. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 492–499, 1998.

[RGH80] T. W. Ryan, R. T. Gray, and B. R. Hunt. Prediction of Correlation Errors in Stereo-pair Images. *Optical Engineering*, 19(3):312–322, 1980.

[RWC+98] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The Office of the Future: A Unified Approach to Image-Based Modeling and Spatially Immersive Displays. *Computer Graphics*, 32(Annual Conference Series):179–188, 1998.

[SA90] S. K. Srivastava and N. Ahuja. Octree Generation from Object Silhouettes in Perspective Views. *Computer Vision, Graphics and Image Processing*, 49(1):68–74, 1990.

[SCG97] P.-P. Sloan, M. F. Cohen, and S. J. Gortler. Time Critical Lumigraph Rendering. In *Symp. on Interactive 3D Graphics*, April 1997.

[Sch94] D. Scharstein. Matching Images by Comparing Their Gradient Fields. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, volume 1, pages 572–575, 1994.

[Sch99] D. Scharstein. View Synthesis Using Stereo Vision. *Lecture Notes in Computer Science (LNCS)*, 1583, 1999.

[SCMS00] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer. Improved Voxel Coloring via Volumetric Optimization. Technical Report 3, Center for Signal and Image Processing, Georgia Institute of Technology, 2000.

[SCMS01] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer. A Survey of Methods for Volumetric Scene Reconstruction from Photographs. 1, Center for Signal and Image Processing, Georgia Institute of Technology, 2001.

[SD96] S.M. Seitz and C.R. Dyer. View Morphing. In *SIGGRAPH 96 Conference Proceedings*, volume 30 of *Annual Conference Series*, pages 21–30, New Orleans, Louisiana, 1996. ACM SIGGRAPH, Addison Wesley.

[SD99] S. M. Seitz and C. R. Dyer. Photorealistic Scene Reconstruction by Voxel Coloring. *International Journal of Computer Vision (IJCV),*, 35(2):151–173, 1999.

[Sei89] P. Seitz. Using Local Orientation Information as Image Primitive for Robust Object Recognition. *SPIE Visual Communications and Image Processing IV*, 1199:1630C1639, 1989.

[SGHS98] Jonathan Shade, Steven J. Gortler, Li Wei He, and Richard Szeliski. Layered Depth Images. In *Proceedings of SIGGRAPH 98*, pages 231–242, August 1998.

[SH85] R. Szeliski and G. Hinton. Solving Random-dot Stereograms Using the Heat Equation. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 284–288, 1985.

[SH97] H. Y. Shum and L. W. He. Rendering with Concentric Mosaics. In *Proceedings of SIGGRAPH 1997*, pages 299–306, 1997.

[SH99] P.-P. Sloan and C. Hansen. Parallel Lumigraph Reconstruction. In *Proc. Symposium on Parallel Visualization and Graphics*, pages 7–15, 1999.

[Sha93] J. Shah. A Nonlinear Diffusion Model for Discontinuous Disparity and Half-occlusion in Stereo. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 34–40, 1993.

[SHS98] H.Y. Shum, M. Han, and R. Szeliski. Interactive construction of 3D models from panoramic mosaics. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR*, pages 427–433, Santa Barbara, CA, June 1998.

[SHSd00] H. Schirmacher, W. Heidrich, and H.-P. Sei-del. High-quality Interactive Lumigraph Rendering through Warping. In *Proc. Graphics Interface 2000*, Montreal, Canada, 2000.

[SLS+96] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical Image Caching for Accelerated Walk-throughs of Complex Environments. In *SIGGRAPH*, August 1996.

[SMS01] Hartmut Schirmacher, Li Ming, and Hans-Peter Seidel. On-the-Fly Processing of Generalized Lumigraphs. *EUROGRAPHICS 2001*, 20(3), 2001.

[SS96] Gernot Schaufler and Wolfgang Stürzlinger. A Three-Dimensional Image Cache for Virtual Reality. In *Proceedings of Eurographics '96*, August 1996.

[SS97] Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. *Computer Graphics*, 31(Annual Conference Series):251–258, 1997.

[SS98] D. Scharstein and R. Szeliski. Stereo Matching with Nonlinear Diffusion. *International Journal of Computer Vision (IJCV)*, 28(2):155–174, 1998.

[SS02] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1):7–42, May 2002.

[SVSG01] Hartmut Schirmacher, Christian Vogelgsang, Hans-Peter Seidel, and Gnther Greiner. Efficient Free Form Light Field Rendering. In *Proc. Vision, Modeling, and Visualization 2001 (VMV01)*, pages 249–256, 2001.

[SVZ00] D. Snow, P. Viola, and R. Zabih. Exact Voxel Occupancy with Graph Cuts. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 345–352, 2000.

[Sze93] R. Szeliski. Rapid Octree Construction from Image Sequences. *Computer Vision, Graphics and Image Processing*, 58(1):23–32, 1993.

[Sze94] R. Szeliski. Image mosaicing for tele-reality applications. In *IEEE Workshop on Applications of Computer Vision*, page 4453, Sarasota, Florida, December 1994.

[Sze96] R. Szeliski. Video mosaics for virtual environments. In *IEEE Computer Graphics and Applications*, page 2230, March 1996.

[Sze99] R. Szeliski. Prediction Error as a Quality Metric for Motion and Stereo. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 781–788, Sept 1999.

[Tec] BeHere Technology. http://www.behere.com.

[TH86] Q. Tian and M. N. Huhns. Algorithms for Subpixel Registration. *Computer Vision, Graphics and Image Processing*, 35:220–233, 1986.

[TK92] C. Tomasi and T. Kanade. Shape and Motion from Image Streams under Orthography: A Factorization Approach. *International Journal of Computer Vision*, 9(2):137–154, 1992.

[TV98] Emanuele Trucco and Allessandro Verri. *Introductory Techniques for 3D Computer Vision*. Prentice Hall, 1998.

[Vek99] O. Veksler. *Efficient Graph-based Energy Minimization Methods in Computer Vision*. PhD thesis, Cornell University, 1999.

[Wes90] L. A. Westover. Footprint Evaluation for Volume Rendering. In *Proceedings of SIGGRAPH 1990*, August 1990.

[WH97] John Woodfill and Brian Von Herzen. Real-Time Stereo Vision on the PARTS Reconfigurable Computer. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 201–210, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[Wil83] Lance Williams. Pyramidal Parametrics. In *Computer Graphics (SIGGRAPH 1983 Proceedings)*, volume 17, pages 1–11, July 1983.

[WND96] Mason Woo, Jackie Neider, and Tom Davic. *"OpenGL Programming Guide"*. Addison-Wesley, second edition, 1996.

[Wol89] L. B. Wolff. Using Polarization to Separate Reflection Components. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, page 363C369, 1989.

[Wol90] G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1990.

[YPW03] R. Yang, M. Pollefeys, and G. Welch. Dealing with Textureless Regions and Specular Highlights—A Progressive Space Carving Scheme Using a Novel Photo-consistency Measure. In *submitted for publication, under review*, 2003.

[YZ02] Ruigang Yang and Zhengyou Zhang. Eye Gaze Correction with Stereovision for Video-Teleconferencing. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 479–494, 2002.

[ZS01] Z. Zhang and Y. Shan. A Progressive Scheme for Stereo Matching. In M. Pollefeys et al, editor, *Springer LNCS 2018: 3D Structure from Images - SMILE 2000*, pages 65–85. Springer-Verlag, 2001.

[ZW94] R. Zabih and J. Woodfill. Non-parametric Local Transforms for Computing Visual Correspondence. In *Proceedings of European Conference on Computer Vision (ECCV)*, page 151C158, 1994.

Appendix A

Sample Code for Real-time VDPC on Graphics Hardware

I present here complete pseudo code and detailed settings in Pixel Shader for real-time VDPC (Chapter 4). The code for Pixel Shader is written roughly following the syntax of `nvparser`, a generalized compiler for NVIDIA extensions. Documentation about `nvparser` can be found on NVIDIA's web site at http://www.nvidia.com.

A.1 Pseudo code for an OpenGL implementation

Algorithm 3 outlines the implementation of real-time VDPC on graphics hardware, discussed in Section 4.3.

A.2 Code to compute the squared difference

The piece of code presented here (Algorithm 4) corresponds to function `setupPixelShaderForSDD` in Algorithm 3. It assumes that there are m input images. The alpha channel of the input images contains a gray scale copy of the image, and the base reference image is stored in tex0. The squared difference is computed on the gray scale images. The scales in the code are necessary because the unsigned char values are converted to floating point values between [0, 1] within Pixel Shader. If no scale is applied, the output squared value (in unsigned char) will be $\text{floor}((a-b)^2/256)$, where a and b are the input values (in unsigned char). In my implementation, I use a combined scale factor of 32, effectively computing $\text{floor}((a-b)^2/32)$.

A.3 Code to select the best color

The piece of code presented here (Algorithm 5) corresponds to function `setupPixelShaderForMinMax` in Algorithm 3. It assumes that the mean colors are

stored in the RGB channel while the SSD scores are stored in the alpha channel. It will select the pixel color (or color-coded depth) with the smaller alpha value.

Algorithm 3 Pseudo code for an OpenGL implementation

```
createTex(workingTexture);
createTex(frameBufferTexture);
for (i = 0; i< steps; i++) {
   // the scoring stage;
   setupPerspectiveProjection();
   glEnable(GL_BLEND);
   glBlendFunc(GL_ONE, GL_ONE);
   setupPixelShaderForSDD();
   for (j = 0; j< inputImageNumber; j++)
     projectImage(j, baseReferenceImage);

   // the OPTIONAL aggregation stage
   // MML is the Maxinum Mipmap Level
   if (MML > 0)
       sumAllMipLevels(MML);

  //  the selection stage;
  if (i == 0) {
   copyFrameToTexture(frameBufferTexture);
   continue;
  } else
   copyFrameToTexture(workingTexture);

  setupPixelShaderForMinMax();
  setupOrthogonalProjection();
  renderTex(workingTexture,
            frameBufferTexture);
  copyFrameToTexture(frameBufferTexture);
}
```

Algorithm 4 Pixel Shader code to compute the SSD score.

```
const1 = {1/m, 1/m, 1/m, 1};
// the base reference image  will be added
// m-1 times more than  the other images;
const0 = {1/((m)(m-1)), 1/((m)(m-1)),  1/((m)(m-1)), 1};

// **** combiner stage 0;
{
    rgb {
      spare0 = tex1*const1 + tex0*const0;
    }
    alpha {
      spare0 = tex1 - tex0;
      scale_by_four ();
    }
}
// **** combiner stage 1
{
    alpha{
        spare0 = spare0*spare0;
        scale_by_four ();
    }
}
// **** final output
{
    out.rgb = spare0.rgb;
    out.alpha = spare0;
}
```

Algorithm 5 Pixel Shader code to do the minimum alpha test.

```
// **** combiner stage 0;
{
    alpha {
        // spare0 = tex1 - tex0 + 0.5
      spare0 = tex1 - half_bias(tex0);
    }
}
// **** combiner stage 1
{
    rgb{
    // select the color with  the smaller alpha;
    // spare0 =  (spare0.alpha < 0.5) ? (tex1) : (tex0);
        spare0 = mux();
    }
    alpha{
    // select the smaller  alpha value
        spare0 = mux();
    }
}
// **** final output
{ out = spare0; }
```