Towards Real-Time Novel View Synthesis Using Visual Hulls

Ming Li

Max-Planck-Institut für Informatik
Saarbrücken, Germany

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing)
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

Eingereicht am 23. September 2004 in Saarbrücken.

Betreuender Hochschullehrer — Supervisor
Prof. Dr. Hans-Peter Seidel, MPI für Informatik, Saarbrücken, Germany
Dr. Marcus Magnor, MPI für Informatik, Saarbrücken, Germany

Gutachter — Reviewers
Prof. Dr. Hans-Peter Seidel, MPI für Informatik, Saarbrücken, Germany
Dr. Marcus Magnor, MPI für Informatik, Saarbrücken, Germany

Dekan — Dean
Prof. Dr. Jörg Eschmeier, Universität des Saarlandes, Saarbrücken, Germany

Datum des Kolloquiums — Date of Defense
Feb 09, 2005

Ming Li
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany
`ming@mpi-sb.mpg.de`

Abstract

This thesis discusses fast novel view synthesis from multiple images taken from different viewpoints. We propose several new algorithms that take advantage of modern graphics hardware to create novel views. Although different approaches are explored, one geometry representation, the *visual hull*, is employed throughout our work.

First the visual hull plays an auxiliary role and assists in reconstruction of depth maps that are utilized for novel view synthesis. Then we treat the visual hull as the principal geometry representation of scene objects. A hardware-accelerated approach is presented to reconstruct and render visual hulls directly from a set of silhouette images. The reconstruction is embedded in the rendering process and accomplished with an alpha map trimming technique. We go on by combining this technique with hardware-accelerated CSG reconstruction to improve the rendering quality of visual hulls. Finally, photometric information is exploited to overcome an inherent limitation of the visual hull. All algorithms are implemented on a distributed system. Novel views are generated at interactive or real-time frame rates.

Kurzzusammenfassung

In dieser Dissertation werden mehrere Verfahren vorgestellt, mit deren Hilfe neue Ansichten einer Szene aus mehreren Bildströmen errechnet werden können. Die Bildströme werden hierzu aus unterschiedlichen Blickwinkeln auf die Szene aufgezeichnet. Wir schlagen mehrere Algorithmen vor, welche die Funktionen moderner Grafikhardware ausnutzen, um die neuen Ansichten zu errechnen. Obwohl die Verfahren sich methodisch unterscheiden, basieren sie auf der gleichen Geometriedarstellung, der *Visual Hull*.

In der ersten Methode spielt die *Visual Hull* eine unterstützende Rolle bei der Rekonstruktion von Tiefenbildern, die zur Erzeugung neuer Ansichten verwendet werden. In den nachfolgend vorgestellten Verfahren dient die *Visual Hull* primär der Repräsentation von Objekten in einer Szene. Eine hardware-beschleunigte Methode, um *Visual Hulls* direkt aus mehreren Silhouettenbildern zu rekonstruieren und zu rendern, wird vorgestellt. Das Rekonstruktionsverfahren ist hierbei Bestandteil der Renderingmethode und basiert auf einer *Alpha Map Trimming* Technik. Ein weiterer Algorithmus verbessert die Qualität der gerenderten *Visual Hulls*, indem das *Alpha-Map*-basierte Verfahren mit einer hardware-beschleunigten CSG Rekonstruktiontechnik kombiniert wird. Eine vierte Methode nützt zusätzlich photometrische Information aus, um eine grundlegende Beschränkung des *Visual-Hull*-Ansatzes zu umgehen. Alle Verfahren ermöglichen die interaktive oder Echtzeit- Erzeugung neuer Ansichten.

Summary

Over the last decade, novel view synthesis from multiple images has become an interdisciplinary research field between computer graphics and computer vision. Since real photographs or videos can be used as input, this technique lends itself to preserving fine details and complex lighting effects present in the original images. However, a number of appealing applications, such as 3D-TV, computer games and immersive tele-presence, require not only high-quality virtual views but also real-time performance.

This work presents solutions to these demanding applications and explores different approaches to the problem of novel view synthesis. We propose a set of algorithms characterized by two common features. First, a geometry proxy, the *visual hull*, is employed by all the algorithms, either playing an auxiliary role or serving as the principal shape representation. Secondly, all algorithms take advantage of increasingly powerful graphics hardware to achieve high performance.

The first algorithm adopts the stereo vision approach, in which multiple depth maps of scene objects are recovered and warped into novel views along with color information. For this approach, the visual quality of the novel views is highly dependent on the accuracy of the depth maps. We reconstruct the visual hulls of the scene objects and rasterize them using graphics hardware to obtain a depth range for each depth map. The depth ranges are then utilized to reconstruct better depth maps using a stereo matching procedure.

Then, we use the visual hull as the core representation of scene objects. An innovative visual hull reconstruction technique is presented. We encode foreground object masks in the alpha channel of textures and project them onto silhouette cones extruded from 2D silhouette contours. By modulating the projected alpha values and enabling the alpha test, visual hulls are implicitly reconstructed in the novel view in form of depth maps. We name this technique projective alpha map trimming. Based on this technique, a single-pass visual hull rendering algorithm is proposed. The color values of reconstructed visual hulls are computed by compositing the color information of multiple reference views. In order to allow more views to be taken as input, we also provide a multi-pass algorithm to synthesize novel views of visual hulls.

Furthermore, we enhance a hardware-accelerated CSG reconstruction algorithm and combine it with alpha map trimming. The resulting hybrid algorithm benefits from both the high quality of the CSG reconstruction and the fast speed of the alpha map trimming. An advanced per-fragment blending scheme is presented to achieve smoother transitions between different projective textures.

Finally, we go beyond the visual hull and explore another representation, the *photo hull*, defined by photometric consistency across multiple reference views. We propose an algorithm to synthesize novel views of photo hulls interactively. This algorithm rasterizes a stack of slicing planes. Graphics hardware is exploited to check photo-consistency as well as to maintain visibility information with respect to reference views. Although we are focusing on photo hulls in this algorithm, visual hulls help to improve the rendering performance by constraining the number and the size of the slicing planes.

We implement all algorithms in a distributed system. Experiments are conducted on a number of videos of both real and synthetic objects. Visual results and measured performance demonstrate that we are able to generate photo-realistic novel views at interactive or real-time frame rates.

Zusammenfassung

Während der letzten zehn Jahre hat sich die Synthese neuer Ansichten einer Szene aus mehreren Ursprungsbildern als interdisziplinäres Forschungsgebiet zwischen Computergrafik und Computervision entwickelt. Da echte Fotografien oder Videos als Eingabe verwendet werden können, hat diese Technik den Vorteil, dass feine Details und komplexe Beleuchtungseffekte aus den Originalbildern erhalten bleiben. Allerdings benötigen einige ansprechende Anwendungen, wie z.B. 3D-TV, Computerspiele und realistische Telepräsenz, nicht nur qualitativ hochwertige virtuelle Ansichten, sondern verlangen auch Ausführung in Echtzeitperformanz.

Diese Arbeit bietet Lösungen für diese anspruchsvollen Anwendungen und untersucht verschiedene Ansätze, um neue Ansichten einer Szene zu generieren. Wir stellen mehrere Algorithmen vor, die durch zwei gemeinsame Eigenschaften charakterisiert sind. Erstens verwenden alle Algorithmen einen Geometrieproxy, die *Visual Hull*, entweder in einer untergeordneten Funktion oder als primäre Methode der Formdarstellung. Zweitens machen sich alle Algorithmen die Funktionen zunehmend leistungsstärkerer Grafikhardware zunutze, um eine gute Performanz zu erzielen.

Der erste Algorithmus verwendet eine Variante des *Stereo Vision* Ansatzes, bei dem mehrere Tiefenbildern von Objekten in einer Szene errechnet werden, die dann zusammn mit der Farbinformation in eine neue Ansicht gewarpt werden. Bei diesem Ansatz ist die visuelle Qualität der neuen Ansichten hochgradig von der Genauigkeit der Tiefenbildern abhängig. Daher rekonstruieren wir die *Visual Hulls* der Objekte in der Szene und rasterisieren sie mit Hilfe der Grafikhardware, um einen Tiefenbereich für jedes Tiefenbild zu erhalten. Die Tiefenbereiche werden anschließend verwendet, um durch eine *Stereo Matching* Prozedure genauere Tiefenbildern zu rekonstruieren.

In allen nachfolgenden Algorithmen verwenden wir die *Visual Hull* als grundlegende Darstellungsform der Objekte in der Szene. Zunächst wird eine innovative *Visual Hull* Berechnungsmethode vorgestellt. Wir kodieren Objekte im Szenenvordergrund in Form von Masken im Alpha-Kanal von Texturbildern und projizieren sie auf Silhouettenkegel, die aus den Kontouren der 2D Silhouetten extrudiert wurden. Durch Modulation der projizierten Alpha Werte und Aktivierung des Alpha Blendings auf der Grafikkarte werden *Visual Hulls* in neuen Ansichten implizit als Tiefenbildern rekonstruiert. Wir nennen diese Technik *Projective Alpha Map Trimming*. Aufbauend auf dieser Technik stellen wir einen Single-Pass *Visual Hull* rendering Algorithmus vor. Die Oberflaechentextur der berechneten *Visual Hulls* wird aus den Farbwerten mehrerer Referenzansichten erzeugt. Damit eine größere Anzahl an Bildströmen simultan als Eingabedaten fungieren können, haben wir auch einen Multi-Pass Algorithmus zur Synthese neuer *Visual Hull* Ansichten entwickelt.

In einem weiteren Algorithmus passen wir eine hardware-beschleunigte CSG-Rekonstruktionstechnik an das Problem an und kombinieren sie mit Alpha Map Trimming. Das Ergebnis ist ein hybrider Algorithmus, der sowohl von der hohen Qualität der CSG-Rekonstruktion als auch der Schnelligkeit des Alpha Map Trimmings profitiert. Ein hochentwickeltes Blendingschema für jedes einzelne Fragment wird vorgestellt, um glattere Übergänge zwischen den verschiedenen projektiven Texturen zu erreichen.

Im vierten Algorithmus schließlich erforschen wir eine andere Darstellung, die *Photo Hull*, die über die *Visual Hull* hinausgeht und durch photometrische Übereinstimmung über mehrere Ansichten definiert ist. Wir schlagen einen Algorithmus vor, der neue Ansichten von *Photo Hulls* interaktiv erzeugt. Dieser Algorithmus rasterisiert einen Stack von Schnittebenen. Wir nutzen Grafikhardware aus, sowohl um photometrische Übereinstimmung zu überprüfen als auch um Sichtbarkeitsinformation in Bezug auf Referenzansichten zu verwalten. Obwohl der Schwerpunkt dieses Algorithmus auf *Photo Hulls* liegt, helfen *Visual Hulls* die Renderingleistung zu verbessern, indem sie die Anzahl und die Größe der Schnittebenen begrenzen.

Wir implementieren alle oben genannten Algorithmen in einem verteilten System. Experimente mit mehreren Videos von realen und virtuellen Objekten werden durchgeführt. Die visuellen Ergebnisse und die gemessene Performanz zeigen, dass wir in der Lage sind, photorealistische neue Ansichten mit interaktiven oder Echtzeitbildraten zu erzeugen.

Acknowledgements

First of all, I wish to express my gratitude towards my supervisor Prof. Dr. Hans-Peter Seidel for his constant support and encouragement, his valuable comments, as well as for providing me the opportunity of working in an excellent research environment.

I am very grateful to my second supervisor Dr. Marcus Magnor for his instructive guidance and inspiring suggestions. He helps me to forge my soft skills of scientific research. I also would like to thank him for acting as my second reviewer and providing valuable feedbacks.

Special thanks go to Hartmut Schirmacher. It was in the course of the cooperation with him that my research began to take root. We spent days and nights to catch submission deadlines. I would further like to thank my co-worker Christian Theobalt for many constructive discussions. We put much effort in building up the research facilities for this work.

Furthermore, I would like to express my thankfulness to all my colleagues in the graphics group. They are happy to share their ideas, ready to help wherever there is need. I really enjoy working with them. I cannot name them all here, but I would like especially to thank the following people (alphabetical order): Irene Albrecht, Stefan Brabec, Katja Daubert, Michael Gösele, Jörg Haber, Kolja Kähler, Jan Kautz, Jochen Lang, Hendrik Lensch, Karol Myszkowski, Christian Rössel, Annette Scheel, Philipp Slussalek, Marc Stamminger, Jens Vorsatz, Hitoshi Yamauchi. Sincere thanks go to our secretary Sabine Budde and Conny Liegl, who always make paper submissions and conference travels go smoothly.

I also owe thanks to my Chinese friends around me. Many hilarious parties and pleasant excursions helped me to enjoy life outside the lab.

Finally, I wish to thank my beloved family and my dear girl friend Jing, for their everlasting love and support.

Contents

Chapter 1

Introduction

Light is omnipresent in the real world. It starts from light-emitting objects, propagates through space and interacts with other objects, either scatters in their volumes or bounces on their surfaces. Finally, the light is perceived by our eyes and we form mental images in our brains. In a similar way, photographs are taken by cameras when the light rays are captured through the lenses.

The above imaging process, depicted in the upper part of Figure 1.1, replicates itself in the virtual world of computer graphics, where objects are described with their photometric and geometric properties. In this world, light interaction is simulated according to physics laws. The mathematical models of eyes or lenses, often simplified and defined by a few parameters, establish the geometric relationship between 3D objects and their 2D projections.

Aiming at creating photo-realistic digital images, traditional computer graphics research strives to model the real world as precisely as possible. However, real objects exhibit extreme complexity both in terms of geometric and photometric properties. It turns out to be a very laborious task to model these properties with software tools or measuring instruments. In addition, accurate simulation of the light transportation between objects remains intricate and demands enormous computational power.

All these difficulties faced by computer graphics researchers have provoked another line of thought — generating novel views from photographs. For brevity,

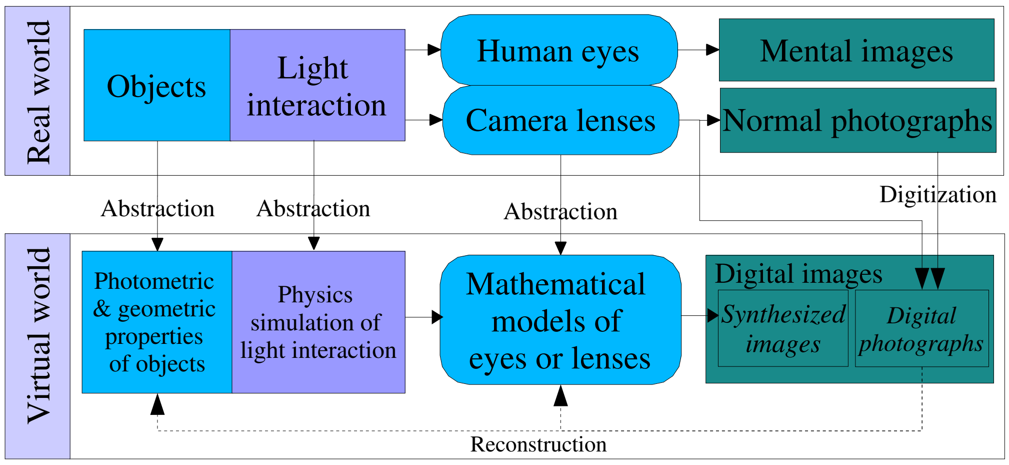


Figure 1.1: Imaging pipelines in real world and virtual world

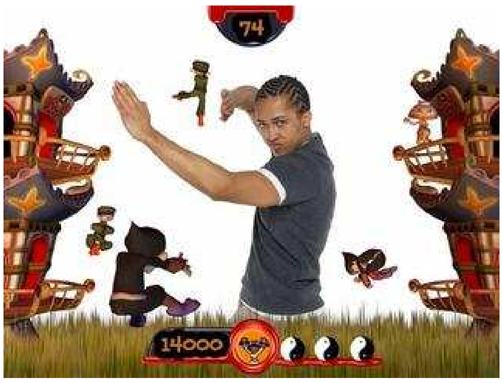
we call this new approach *novel view synthesis*. In the literature, it is also well known as *image-based modeling and rendering*. Usually, this approach exploits computer vision techniques to reconstruct object properties and camera models from photographs, as indicated by dotted lines in Figure 1.1. Then, given a virtual viewpoint, the corresponding novel view is created using the original images and the recovered information. Compared with traditional approaches, this new approach has two main advantages. First, photographs are by nature photo-realistic. Novel views generated from them are potentially able to retain the same degree of photo-realism. Second, photographs are readily available since cameras are off-the-shelf commodities and fairly easy to use. For a recent comprehensive survey on image-based rendering, interested readers are referred to [ZC04].

Novel view synthesis has many appealing applications in a wide range of fields, such as education, medical training, entertainment, communication, *etc*. Some of them are time-critical and demand real-time performance. Figure 1.2 shows several application scenarios of this kind. The upper three images illustrate the concept of live 3D-TV. In this example, a live sport event, Super-Bowl, is recorded with multiple cameras located at different viewpoints [CR01]. When the event is broadcast, viewers are able to choose whatever viewpoints

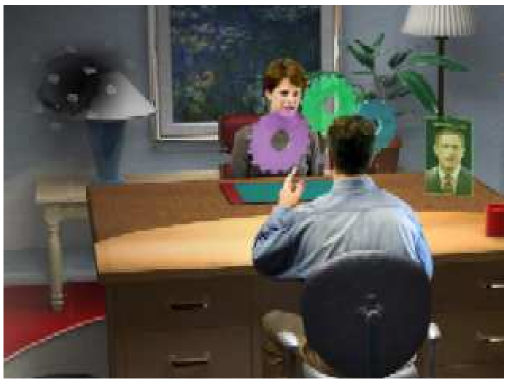
(a)

(b)

(c)



(d)



(e)

Figure 1.2: Potential application scenarios for novel view synthesis. **(a-c)** Live broadcasting of 3D-TV. A live sport event is recorded using multiple cameras. When the event is broadcast, viewers are able to choose whatever viewpoints they like to watch**.** **(d)** Screenshot of Eyetoys. The video stream of a person is superimposed with virtual scenes. The person can interact with virtual objects. **(e)** Tele-immersion. Two persons at geographically distributed sites are discussing and interacting with virtual objects.

they like to watch. Figure 1.2(d) is a screen shot from a console game — Eyetoys [Son03]. Currently, this game only acquires one video stream from camera and superimposes the video with virtual scenes. If several views are available and novel views can be synthesized, we can integrate multiple players into a common virtual environment. Figure 1.2(e) demonstrates a typical session of immersive tele-presence [ABUU00], in which two persons at geographically distributed sites are discussing and interacting with virtual objects as if they were in a shared physical space.

To meet the demands of the above applications, novel view synthesis must be performed on-line. However, for a long period of time, expensive digital imaging devices, tedious video acquisition, inadequate processing power of computers have been the limiting factors for fast on-line novel view synthesis. In recent years, these limiting factors have started to disappear as high-speed digital video cameras become available at affordable prices and the processing power of personal computers keeps increasing at rapid pace. More dramatically, the performance of graphics hardware advances even faster than that of CPUs. The architecture of graphics chips is also going through a revolutionary transition. Thanks to the fast progress of technologies, a number of on-line systems for novel view synthesis have been developed [MBR$^+$00, SLS01, MBM01, YEBM02, YWB02]. In order to push the performance further and to improve the rendering quality, we present several new algorithms that take advantage of visual hull representation [Lau94] and graphics hardware to generate realistic novel views of dynamic scenes.

1.1 Problem Statement

The problem we are going to solve in this thesis is stated as follows:

For moving objects in a scene, recorded with a set of calibrated digital cameras from different viewpoints, how does one synthesize novel views of the objects from arbitrary novel perspective in real time?

The recorded digital images are referred to as *reference views*. Reference views can be captured in two configurations. The first configuration uses a moving camera which takes images from multiple viewpoints at different times. The second one has multiple cameras mounted at different places. These cameras take images of the scene simultaneously. For on-line novel view synthesis of dynamic objects, we need images taken from multiple viewpoints at each moment of time. Therefore, we choose the second configuration to acquire reference views. Details about the acquisition setup will be explained in Chapter 3. Since in this configuration the reference viewpoints are fixed, the camera models, including intrinsic and extrinsic parameters, can be pre-determined and used for novel view synthesis later. An image, of which the corresponding camera model is known, is referred to as a *calibrated image*.

If novel views can be synthesized at more than 20 frames per second, we claim that real-time performance is achieved. Interactive frame rates are considered to be between 2 and 20 frames per second. For a number of reference views (three to eight), the algorithms that we will present run at least interactively, and some of them achieve real-time frame rates.

1.2 Main Contributions

Parts of the work have already been published in a number of scientific articles [SLS01, LSMS02, TLMS03, LMS03a, LMS03b, LMS03c, LMS04a, LMS04b]. This thesis presents these publications in a common framework, reveals their relationships and gives more insights. The main contributions of this thesis are summarized as follows:

- A novel algorithm that combines stereo and visual hull [Lau94] information to synthesize novel views. The visual hull information helps stereo pairs to reconstruct better depth maps which lead to improved rendering quality.

- An innovative technique that makes efficient use of graphics hardware to reconstruct visual hulls implicitly. This technique significantly accelerates visual hull reconstruction.

- A hybrid hardware-accelerated algorithm to render high-quality visual hulls from multiple reference views.

- A hardware-accelerated algorithm to create novel views of photo hulls [KS99]. This algorithm overcomes the inherent drawback of the visual hull representation. However, it utilizes the novel visual hull reconstruction technique for acceleration.

All algorithms are implemented in a distributed system that is capable of synthesizing novel views from a set of video streams on line. These algorithms share two common features. First, a geometry proxy, the *visual hull*, is employed by all the algorithms, either playing an auxiliary role or serving as the principal shape representation. Second, all algorithms exploit powerful graphics hardware to achieve high performance.

1.3 Chapter Organization

The rest of this thesis is organized as follows. Chapter 2 reviews previous work related to novel view synthesis and gives some background knowledge about graphics hardware. Chapter 3 describes the acquisition facilities that provide input video data for all of our algorithms. In Chapter 4 we show how to use visual hull information to improve the quality of stereo reconstruction for better view synthesis. Chapter 5 presents a hardware-accelerated approach to synthesize novel views of visual hulls. A basic single-pass rendering algorithm and its extensions are explained in this chapter. In Chapter 6 we propose a hybrid hardware-accelerated algorithm to render high-quality novel views of visual hulls. Then we address an inherent drawback of the visual hull representation in Chapter 7. Color consistency checks across multiple views are performed by graphics hardware to synthesize novel views of photo hulls. Finally, Chapter 8 concludes this thesis and discusses future research directions.

Chapter 2

Related Work

In this chapter, we start with an overview of existing methods for novel view synthesis. Then we concentrate on three kinds of geometry representations of scene objects, and review previous work related to reconstruction and rendering algorithms based on these representations. At last, a short tour of state-of-the-art graphics hardware gives useful background knowledge that helps to understand the algorithms presented in this thesis.

2.1 Novel View Synthesis

As explained in the introduction, novel view synthesis technique generates novel views from a set of reference images. In a simple case, the reference images are taken from a fixed viewpoint towards different viewing directions. Since there is only a purely rotational relationship between two sets of camera parameters, it can be shown that the coordinates of corresponding pixels in two reference images are related by a 3×3 matrix, known as *homography* [SS97]. This matrix is computed either from pre-calibrated camera parameters or by image registration. Then, a panoramic image is created by using the matrices between reference image pairs [Che95, SS97]. Finally, virtual images from novel viewing directions can be synthesized from the panoramic image. This method is called *image mosaicing* and illustrated by an example in Figure 2.1. Although it

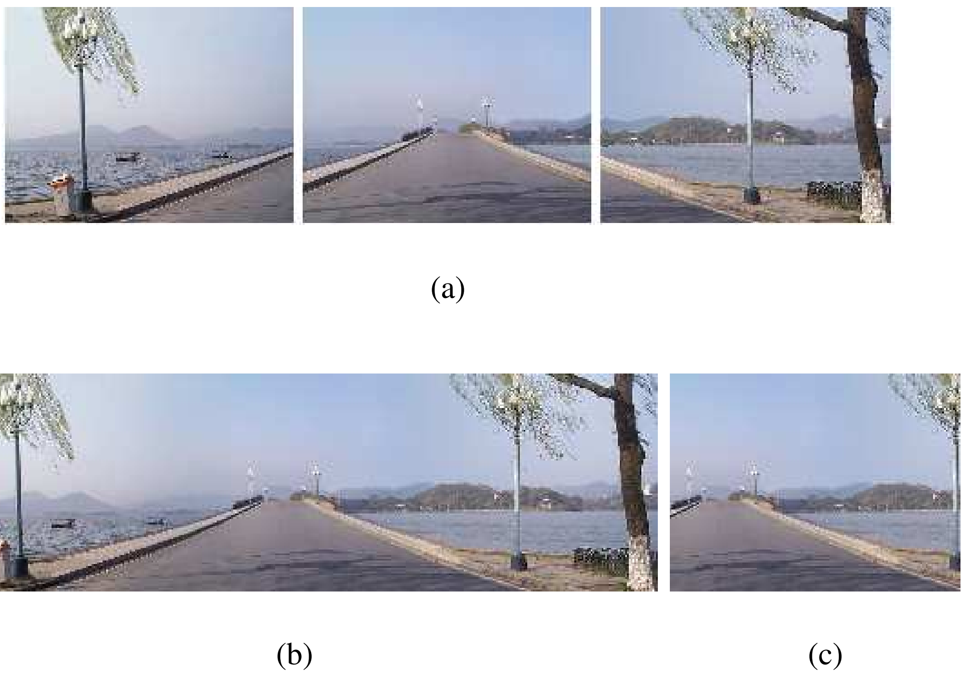
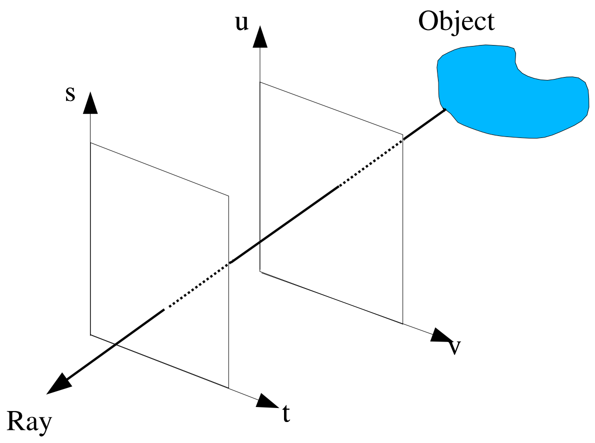


Figure 2.1: Image mosaicing. **(a)** three reference images. **(b)** panorama created from reference images. **(c)** a novel view synthesized from the panorama.

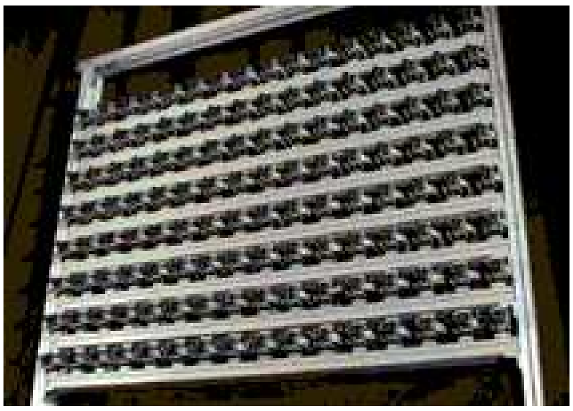
is possible to build a real-time image mosaicing system using omni-directional cameras [Poia], one limitation of the method is that the novel viewpoint has to be the same as the fixed viewpoint of the reference images. This drawback severely restricts the freedom of users' navigation.

In order to allow users to navigate in a scene more freely, reference images from different viewpoints should be taken. Levoy *et al.* [LH96] employs a computer-controlled gantry to position a camera at regular grids on a plane. All captured images are considered to record radiance values of ray samples in a scene. These rays are parameterized as a 4D *light field* function with the help of two parallel planes (see Figure 2.2a). The rays between the planes collectively form a *light slab*. Novel views are synthesized by quadrilinear interpolation of the ray samples. If more light slabs from different directions are available, novel viewpoints can be placed anywhere outside the convex hull of the scene objects. Shum *et al.* propose to reduce the 4D function to 3D by using a representation termed as *concentric mosaic* [SH99]. They move a camera along a set of coplanar concentric circles and capture rays pointing at tangent directions. Novel views can be generated for the viewpoints on the plane where the concentric circles lie. For other ways of parameterization of the light field, readers can refer to Schirmacher's PhD thesis [Sch03]. The light field methods circumvent the difficult 3D reconstruction problem and synthesize novel views directly from images. The rendering is efficient since only simple interpolation is involved. However, the interpolation could lead to blurry artifacts in the novel view because neighboring ray samples may correspond to scene points at different depths. To alleviate the artifacts, the scene has to be densely sampled with a large amount of reference images. Thus, a huge ray database created from the reference images places heavy burdens on image storage, processing and transmission. In addition, for dynamics scenes, many cameras are required to capture the reference images simultaneously. For example, Yang *et al.* design a real-time light field rendering system with 64 cameras [YEBM02]. The Stanford multi-camera array [WL03] has up to 128 cameras shown in Figure 2.2b. Even using a large number of cameras, only a limited field of view is covered by their system. Overall, it is quite challenging to build a low-cost real-time rendering system based on the light field methods.

To reduce the number of reference images while staying away from the blurry artifacts, we need some knowledge about the 3D scene geometry. According to the formal analysis conducted by Chai *et al.* [CCST00], more accurate geometry information is available, fewer reference images are needed to maintain the same level of rendering quality. In the work of *surface light fields* [WAA$^+$00], Wood *et al.* use 3D scanners to obtain accurate geometry. Although high rendering quality is achieved, it is difficult to accomplish 3D acquisition at interactive frame rates in a fully automatic way. Moreover, 3D scanners are usually quite expensive. As an alternative way, 3D geometry information can be reconstructed from 2D reference images. The reconstruction technique is also referred to as *image-based modeling* in literature. Researchers have explored different geometry representations, for instance, disparity or depth maps [CW93, SLS01], visual hulls [Lau94], photo hulls [KS99], parameterized prim-



(a)



(b)

Figure 2.2: Light Field Rendering. **(a)** Two plane parameterization of a light ray. 4D light field function L=L(s,t,u,v). **(b)** Stanford Light Field Multi-Camera Array.

itives [Low91, DTM96]. The first three representations are most relevant to our work. We will review three categories of reconstruction techniques based on these representations, i.e. depth from stereo, shape from silhouette and shape from photo-consistency. Corresponding rendering algorithms will be discussed as well.

2.2 Depth from Stereo

2.2.1 Depth map reconstruction

"Depth from stereo" [TV98] is a classical 3D reconstruction technique which has been extensively investigated for decades in the computer vision community. This technique is based on the simple fact illustrated in Figure 2.3: when a single point in 3D space is observed by two cameras placed at different locations, it projects to a pair of pixels with different coordinates on the two image planes. "Depth from stereo" is trying to solve this problem inversely, i.e. reconstructing the 3D position of the scene point by locating the corresponding pixels in the two images. The 3D information is typically represented as a depth map. Each value in the depth map encodes the distance between the associated scene point and the viewpoint along the principal axis of a view.

The process of the correspondence search is also known as *stereo matching*, which is the key issue for all stereo algorithms. Depending on the primitives to be matched, stereo algorithms can be divided into two classes: *feature-based methods* and *area-based methods* [1]. Feature-based methods match features, such as edges or contours, extracted from images. They are fast since only a small number of features are matched. However, feature-based methods only produce very sparse depth maps, which are not suitable for realistic novel view synthesis. Therefore, we will discuss more details for the second class of methods.

Area-based methods compare a small area surrounding the pixel in question in one image with small areas in the other image. The corresponding pixel should lie in the most similar area in the other image. The area-based methods

[1]also known as *correlation-based methods*

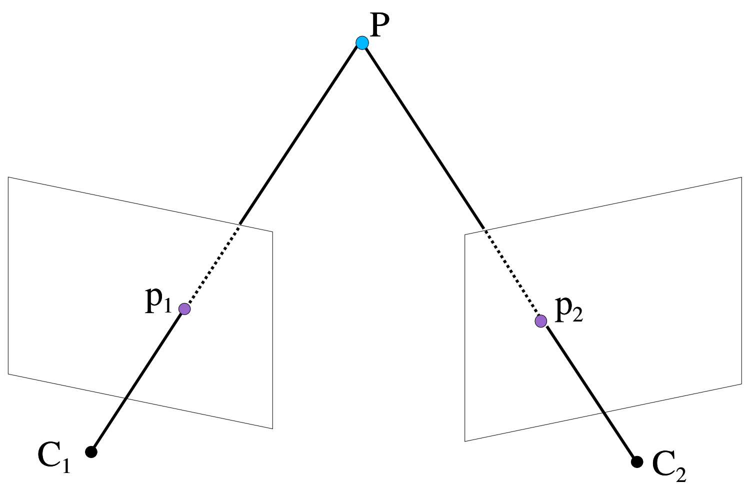


Figure 2.3: Depth from stereo. The viewing parameters of the two views C_1 and C_2 are known. The 3D point P projects to different image locations p_1 and p_2 on the image planes associated with the two viewpoints C_1 and C_2, respectively. In reverse, if the correspondence between p_1 and p_2 is identified, the 3D position of the point P can be reconstructed.

are able to generate dense depth maps because the matching process is carried out for each pixel in an image. The matching criterion is based on image intensities over the local support area. Various metrics can be employed. Among the most common ones are Normalized Cross-Correlation (NCC), Sum of Squared Differences (SSD), Normalized SSD and Sum of Absolute Differences (SAD). To make the matching criterion more reliable, rank transforms can be applied to image intensities [ZW94]. The shape of the matching area is usually chosen as a square window with the pixel of interest centered in it. Perspective distortion of local regions can be compensated by estimating local affine transformations [MC04]. If the pixels in a local region have different depth values, the similarity metric might be biased. In this case, multiple shifted windows are used to cover a relatively constant depth region [FRT97]. For dynamic scenes, the matching window is extended to time domain in space-time stereo methods [ZCS03]. Frame coherence is exploited to reduce matching ambiguity.

Area-based methods find the correspondence for each pixel in a reference view. Despite that more computation is needed compared with feature-based methods, area-based methods can be implemented very efficiently. Earlier systems use specialized hardware to carry out stereo matching [FHM+93, Kan94]. Recently, some commercial products [Poib, Vid] as well as academic research prototypes [DMM+00, SLS01] have demonstrated that area-based methods are able to run on general purpose computers in real time. Modern graphics hardware is utilized to execute fast depth map reconstruction as well, and achieves real-time performance [YP03].

Although area-based methods are able to reach real-time performance, a naive implementation could produce depth maps in poor quality. This is mainly because the area matching is error-prone in ambiguous regions where occlusions or uniform textures exist. To improve the depth map quality, more constraints should be exploited, such as the epipolar constraint [Fau93], the symmetry constraint [Fua93] and global constraints, *etc*. The epipolar constraint is stated as follows: for a pixel p in one image, the corresponding pixel in the other image must lie on the epipolar line, which is defined by projecting the viewing ray associated with the pixel p into the other image. This constraint reduces the dimensionality of the search space from 2D to 1D. Symmetry constraint ensures the consistency between the results of the left-to-right and the right-to-left matches. This is useful for detecting occlusion areas. Global constraints imposed on scan-lines or on entire images formulate the correspondence problem in a global optimization framework. The algorithm robustness can be greatly improved, but the computational cost is high. Typical stereo algorithms incorporated with global constraints find optimal solutions by using dynamic programming [OK85] or graph cuts [KZ01].

Depth maps reconstructed from only one pair of reference images cannot represent the full 3D geometry of a scene object. In order to extract more geometry information, multiple reference views are needed to recover depth maps from different perspectives. In addition, extra reference views are also helpful for eliminating false stereo matches. Narayanan *et al.* build a "virtualized reality" system consisting of 51 video cameras installed on a hemi-spherical dome [NRK98]. A multi-baseline stereo algorithm [OK93] is employed to produce a depth map for each reference view. The on-line processing system developed by Schirmacher *et al.* reconstructs live depth maps of three stereo pairs [SLS01].

Dhond and Aggarwal have surveyed stereo algorithms developed until late 1980s [DA89]. Recent advances are reviewed in [BBH03]. A comparative evaluation of a large number of stereo algorithms is conducted by Scharstein *et al.* [SS02].

2.2.2 Rendering with depth maps

Depth maps recovered from stereo algorithms represent scene geometry as observed from reference views. Together with the color information in reference views, they can be used to render realistic novel views in various ways.

1. **Point-based rendering**. For a calibrated reference view, knowing the depth value at one pixel, the associated 3D point can be generated through back-projection. This way, all pixels in a reference view produce a point cloud. To synthesize a novel view, we can simply render these points using standard graphics hardware. The color of each point is assigned to the color of the corresponding pixel. Since the points are discrete primitives, holes might appear in the novel view due to under-sampling. The surface splatting technique [ZPvG01] amortizes this under-sampling artifacts. A related technique, billboard rendering [YSK$^+$02], could also be applied. Another way of rendering a point cloud is based on a stack of planes sweeping through the volume space containing the points [KS01]. Each plane is textured with color images and depth maps. Those parts on the plane that do not match the depth values will be discarded.

2. **Mesh-based rendering**. As stated before, each depth sample in a reference view corresponds to a 3D point. When we connect those points whose projections are neighboring samples with similar depth values, a dense triangle mesh can be obtained. If multiple triangle meshes are constructed from multiple reference views, these partial triangle patches can be registered together to form a more complete mesh by using an *iterative closest point* (ICP) algorithm [TL94]. Alternatively, unorganized

point clouds generated from multi-view depth images are first integrated into a volumetric model [CL96]. Then the isosurface could be extracted and triangulated into a polyhedral mesh. Once the mesh representing the scene geometry is available, for each polygon on the mesh, several texture patches can be collected from reference views. These texture patches are blended into a single one and mapped to the polygon during rendering [RCMS99]. In the case of dynamic scenes, the textures change for every frame. *Projective texture mapping* [SKvW+92, DBY98] should be used to blend the textures at run time.

3. **3D warping**. McMillan proposes a forward 3D warping method that eliminates the need of rendering 3D geometry [McM97]. This method operates directly on images and transforms each pixel in the reference view to a new location in the desired view. The computation can be performed incrementally by taking advantage of the regular structure of images. Popescu *et al.* present the WarpEngine which is a hardware architecture designed for efficient 3D warping [PEL+00]. Oliveira *et al.* factorize the 3D warping into a simple 1D pre-warp followed by a planar projective mapping [OBM00]. The pre-warp is accomplished with only a few arithmetic instructions and the projective mapping is accelerated by texture mapping hardware. This rendering method is called *relief texture mapping*. When multiple depth maps are available, *Layered Depth Images* [SGHS98] can be generated through pre-processing and then rendered by 3D warping methods. Similar as in the point-based rendering, splatting techniques [SGHS98] should be applied to fill holes. Inverse 3D warping discussed in [McM97] could also be helpful, but the warping process is less efficient.

2.3 Shape from Silhouette

"Shape from silhouette" is another widely-used 3D reconstruction technique. This technique assumes the foreground object in an image can be separated from the background. Under this assumption, the original image can be thresholded

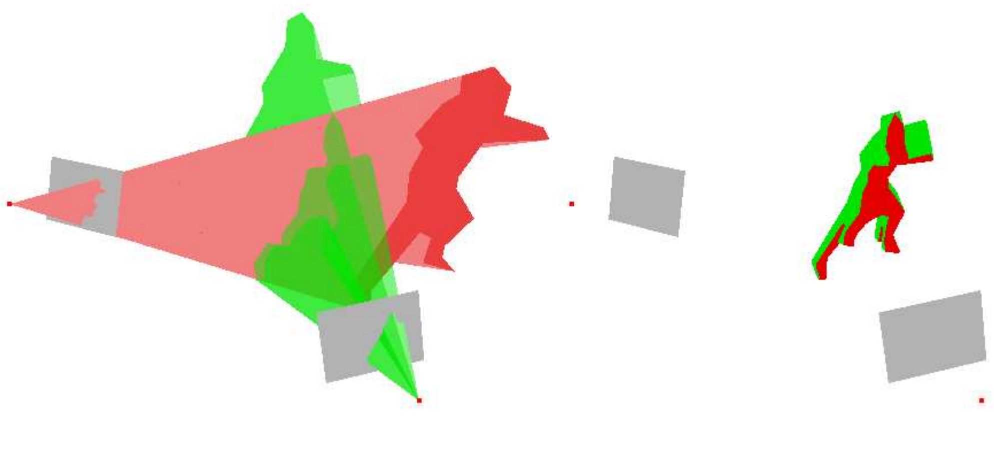


Figure 2.4: Visual hull reconstruction. **(a)** Two silhouette cones are extruded from two silhouette images. **(b)** The visual hull is created by intersecting the silhouette cones.

into a foreground/background binary image, which we call a *silhouette image*. The foreground mask, known as a *silhouette,* is the 2D projection of the corresponding foreground object in 3D. Along with the calibration information, the silhouette defines a back-projected generalized cone that contains the actual object. This cone is called a *silhouette cone* [2], composed by a set of triangular *silhouette faces*. Figure 2.4a shows two such cones produced from two silhouette images taken from different viewpoints. The intersection of the two cones is called a *visual hull* [Lau94], which is a bounding geometry of the actual 3D object (see Figure 2.4b).

Strictly speaking, a visual hull is defined as the maximal shape that gives the same silhouette of the 3D object for any possible view outside the convex hull of the object. But, in practice, only a limited number of views can be taken into account. For simplicity, we use the same term *visual hull* to describe the shape representation reconstructed from a limited number of views.

The visual hull representation has a main drawback: certain types of con-

[2]known as visual cone or viewing cone in some literature.

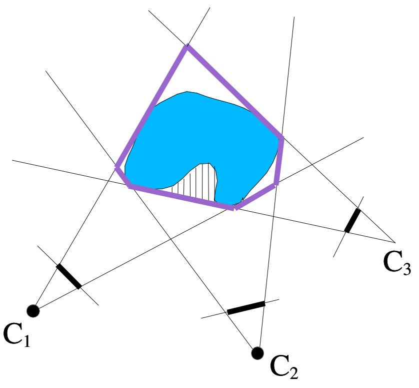


Figure 2.5: Visual hull limitation. A visual hull created from three reference views, shown in 2D. The thick lines in image planes indicate silhouettes. The hatched concave area can never be reconstructed.

cave regions on the object cannot be reconstructed since these regions are not apparent in the silhouette from any view, as illustrated in Figure 2.5. This limitation is inherent and cannot be overcome even with an infinite number of views. Despite the limitation, we still employ the visual hull representation in all algorithms presented in this thesis because it has several advantages. First, the visual hull is a conservative geometric entity that guarantees to enclose the actual 3D object. Second, the visual hull is able to provide a close approximation of the 3D object from a few number of widely-separated reference views. In addition, the accuracy of the reconstruction increases monotonically as more reference views are used. Third, compared with depth-from-stereo, visual hull reconstruction is more robust with regard to glossy or textureless surfaces.

2.3.1 Visual hull reconstruction

Research on visual hull reconstruction dates back to Baumgart's PhD thesis [Bau74] in the 1970s. Since then, a large number of reconstruction methods have been proposed. Some researchers consider that reference images are taken by a moving camera. The viewpoints are dense samples along a continuous trajectory. In this case, a visual hull with curved surfaces can be reconstructed as the envelope of all tangent planes of the target objects by using epipolar geometry and differential techniques [GW87, CB92, BB97]. However, these reconstruction methods assume that the target objects are smooth. In addition, the moving camera configuration is not suitable for handling dynamic objects. Therefore, in the following we will focus on the methods that reconstruct visual hulls from a discrete set of reference views. According to the underlying representation, such visual hull reconstruction methods are grouped in two categories: voxel-based and boundary-based methods.

2.3.1.1 Voxel-based methods

Voxel-based methods reconstruct volumetric visual hulls composed by voxels, typically small cubes. These methods operate in a tessellated 3D space called working volume. This 3D space should be large enough to contain all the objects to be reconstructed. Earlier voxel-based methods [MA83, Pot87] first construct volumetric silhouette cones by reprojecting silhouette masks into the scene, and then intersect the volumetric cones. In fact, the visual hull reconstruction can be carried out more efficiently by performing two kinds of tests on each voxel in the working volume.

1. Silhouette-cone test. The test is to verify whether a voxel belongs to a silhouette cone. This can be accomplished by projecting the voxel to a reference view and examining the relationship between its 2D projection and the silhouette. If the projection is inside the silhouette mask or on the silhouette boundary, the voxel belongs to the corresponding silhouette cone. The silhouette-cone test should be iterated for all reference views.

2. Silhouette-consistency test. A voxel is said to be silhouette-consistent if the voxel passes the silhouette-cone test for all reference views. A straightforward corollary is that a silhouette-consistent voxel must belong to the intersection of all silhouette cones, i.e. the visual hull. Therefore, the complete volumetric visual hull can be obtained by identifying all the silhouette-consistent voxels. Figure 2.6 shows a 2D example.

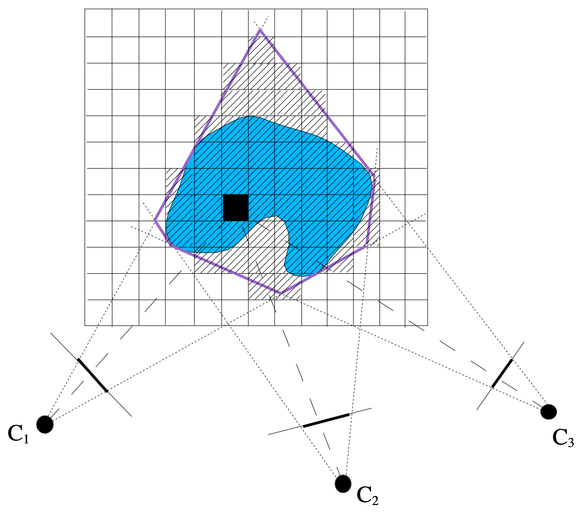


Figure 2.6: The volumetric visual hull reconstructed from three views (2D illustration). The hatched area is the volumetric visual hull. The black voxel passes the silhouette-consistency test. Therefore, it belongs to the visual hull.

Szeliski [Sze93] employs octree data structure to enhance the standard method described above. The octree structure not only increases the speed and the space efficiency of the algorithm, but also provides a way to refine the reconstruction result hierarchically. Moezzi *et al.* [MKKJ96] develop an off-line system to synthesize novel views based on volumetric visual hulls. Video sequences acquired from multiple reference views are used as input. To deal with noise existing in silhouette images, they loosen the criterion of the silhouette-consistency test. A voxel is identified as silhouette-consistent as long as the voxel passes certain a number of silhouette-cone tests and this number is greater than a pre-defined threshold. Saito *et al.* [SK99] reconstruct volumetric visual hulls in projective space if the reference cameras are weakly-calibrated.

Recently, increased computational power has led to the emergence of several real-time systems. For example, volumetric visual hulls are reconstructed in real-time for ellipsoid fitting of human figures [CKBH00] or estimation of human motion parameters [TMSS02]. Another real-time system is built by Matsuyama *et al.* [MT02]. They group 3D voxels as a stack of planes. On each plane, a projected silhouette for each reference view is computed. The projected silhouettes from all views are then intersected to produce a cross-section of the visual hull. Parallel processing is conducted by a 16-node PC cluster for accelerating the intersection computation. Lok [Lok01] adopts a similar 3D space discretization. Unlike the work [MT02], graphics hardware features are employed to accelerate the projection of silhouette images and the intersection of projected silhouettes on stack planes. This algorithm distinguishes itself from all the above algorithms because the visual hull reconstruction is embedded in the rendering process, and the reconstruction result is a view-dependent visual hull represented as a depth map in a novel view.

Voxel-based methods are able to reconstruct very complex objects, like trees or hairy animals. The computation is simple and stable. The big disadvantage is that objects represented as voxels always have quantization artifacts (Notice the boundary voxels in Figure 2.6). To reduce these artifacts, one needs a very fine space discretization which will increase the memory requirement and slow down the speed. For more detailed reviews on volumetric visual hull reconstruction approaches, we refer interested readers to [SCMS01] and [Dye01].

2.3.1.2 Boundary-based methods

Unlike voxel-based methods, boundary-based methods do not use space discretization. Silhouette cones are represented as boundary elements, such as surfaces or lines. The intersection between them is normally computed analytically. The intersection results, visual hulls, can be composed by surface patches, line segments or even points.

Baumgart [Bau74] approximates silhouette contours in multiple views using 2D polygons and extrudes them to obtain polyhedral silhouette cones. By applying general 3D intersection between the cones, polyhedral visual hulls are created. Matusik *et al.* [MBM01] exploit the projective structure of polyhedral silhouette cones to reduce the intersection computation from 3D to 2D. This algorithm is implemented in their real-time system and demonstrates great improvement of the intersection speed. Franco and Boyer [FB03] take advantage of the projective structure of silhouette cones for fast computation of edge segments associated with vertices of polygonal silhouette contours. These edge segments belong to the final polyhedral visual hull, but they are only a subset of all the edges. Those missing edges are found by applying local orientation and connectivity rules.

Polyhedral visual hulls usually contain sharp corners and rugged surfaces because silhouette contours are approximated as piecewise-linear polygons. However, some applications favor models with smooth surfaces. To get such a smooth model, Sullivan *et al.* [SP98] first computes a polyhedral visual hull as an initial estimation and fits G^1-continuous spline surfaces to the polyhedal model. Then these surfaces are deformed to minimize the difference between the projections of the smooth model and the silhouette contours. Lazebnik *et al.* [LBP01] adopt a different strategy and use 2D B-spline silhouette contours as inputs. In this case, the boundary representation of silhouette cones is a curved surface. The reconstruction result, a *generalized polyhedral visual hull,* consists of curved surface patches.

Instead of using a polyhedral representation, Cheung *et al.* [CBK03b] reconstruct visual hulls as a dense set of line segments, which they call *bounding edges*. Würmlin *et al.* [WLSG02] represent visual hulls as surface points observed from reference views. If we treat the surface points as irregular sampled voxels, the reconstruction result can be regarded as a volumetric visual hull. Matusik *et al.* [MBR$^+$00] present an algorithm to reconstruct visual hulls as surface points seen by a target view. The result is named as *image-based visual hull* (IBVH). The above three algorithms perform similar intersection computations. First, a viewing line intersects with a polyhedral silhouette cone to produce a list of line segments. Then, by applying such intersection for different silhouette cones, multiple line segment lists can be obtained. The intersection of these segment lists produces boundary elements of visual hulls. Note that the 3D intersection between a viewing line and a polyhedral cone can be reduced to 2D as explained in the work [MBM01]. The difference of the three algorithms lies in the way how to generate viewing lines. The first algorithm generates viewing lines from point samples of silhouette contours in reference views. The second one generates viewing lines from silhouette pixels in all reference views. The third one generates viewing lines from all pixels in a target view.

Interactive CSG rendering [Wie96] exploits graphics hardware to generate view-dependent visual hulls by rendering polyhedral silhouette cones. The intersection parts are identified by counting stencil values. In this thesis, we propose two other hardware-accelerated algorithms to reconstruct view-dependent visual hulls. More details will be presented in Chapter 5 and Chapter 6.

Generally, boundary-based methods consume little memory and are executed faster than voxel-based methods. In addition, they do not suffer from quantization artifacts while rendering. However, analytic intersection computation required by most boundary-based methods is susceptible to numerical instability and often results in corrupted surfaces. By reducing the computation from 3D to 2D [MBM01], this problem can be alleviated. Hardware-accelerated methods, such as interactive CSG rendering, totally avoid the stability problem, but they do not produce explicit 3D models, which are useful in many applications.

2.3.2 Visual hull rendering

Visual hull rendering bears some similarities with the point-based rendering and the mesh-based rendering methods discussed in Section 2.2.2. View-independent volumetric visual hulls can be rendered directly as points [WLSG02] or billboards [GM03]. Since polygons are usually more suitable rendering primitives for graphics hardware, it is more common to convert a volumetric model into a polyhedral one, and then to render it [MKKJ96]. View-independent polyhedral visual hulls can be rendered with projective texture mapping and texture blending.

To render view-dependent visual hulls, Lok [Lok01] builds a coarse mesh from the depth map in the target view and computes a color value for each vertex of the mesh. Then the mesh is rendered through color interpolation by graphics hardware. Matusik *et al.* [MBR$^+$00] compute a color value for each pixel in the novel view of visual hulls. The implementation is purely software-based. In our work, we render view-dependent visual hulls using projective texture mapping. Color values from multiple reference views are blended. Both operations are fully accelerated by graphics hardware.

2.4 Shape from Photo-consistency

Shape-from-photo-consistency methods take advantage of color information in input images to reconstruct the *photo hull* representation [KS99], which can represent the concave regions that the visual hull fails to do. The photo hull is defined as the maximal shape that consistently reproduces all reference color images, or equivalently, it is defined as the union of all *photo-consistent* objects with respect to a given set of color images. Suppose that the object to be reconstructed has approximately Lambertian surfaces, the object is said to be *photo-consistent*, if for each point on the object surfaces, its corresponding projections in all visible reference views have similar color intensities within a pre-defined threshold. The photo-consistency definition can be further generalized to non-Lambetian cases by incorporating more sophisticated reflectance models [YPW03].

Both the depth-from-stereo and the shape-from-photo-consistency techniques exploit color information in reference views. However, depth-from-stereo methods solve the reconstruction problem by searching correspondences in 2D images, whereas shape-from-photo-consistency methods start from voxels or surfaces in 3D space, and reconstruct objects by verifying photo-consistencies of voxels or deforming 3D surfaces. Working directly in 3D space has the advantage of being able to model visibilities with respect to reference views explicitly. As a result, shape-from-photo-consistency methods can handle widely-separated reference views, which are difficult for depth-from-stereo methods to process.

Shape-from-photo-consistency can be viewed as generalization of the shape-from-silhouette technique. For 3D reconstruction, shape-from-silhouette methods only need binary reference images and create visual hulls by checking silhouette-consistency, whereas shape-from-photo-consistency methods take color images as input to reconstruct photo hulls by verifying color consistency.

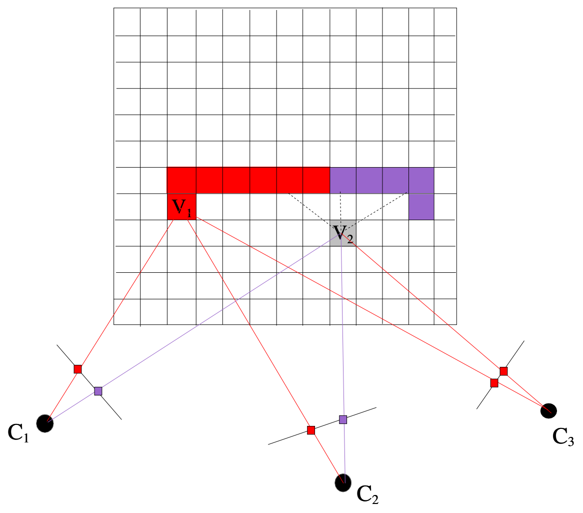


Figure 2.7: Photo-consistency test during volumetric photo hull reconstruction (2D illustration). Let the true scene object be composed of the red and the purple parts. The voxel V_1 is photo-consistent since its projections in three reference views are all red. The voxel V_2 fails the photo-consistency test since its projections are red, red and purple, as observed from the reference viewpoints C_1, C_2, C_3, respectively. Therefore, V_2 is carved away.

2.4.1 Photo hull reconstruction and rendering

Similar to voxel-based visual hull reconstruction, photo hulls can be reconstructed in a discretized working volume. The photo-consistency of each voxel in the volume is verified. Those voxels which fail to pass the photo-consistency test are carved away. The remaining voxels belong to the photo hull. This process is illustrated in 2D using Figure 2.7.

The first *shape-from-photo-consistency* work that we are aware of is presented by Fromherz and Bichsel [FB95]. Seitz and Dyer [SD97] propose a plane-sweeping approach called *voxel coloring*. During photo-consistency check, this approach traverses the scene in an order that accounts for each voxel's visibility with respect to each reference view. One constraint is that no scene point is allowed to be within the convex hull formed by the optical centers of cameras. Kutulakos and Seitz [KS99] establish a formal theory of the photo hull representation. A *space carving* algorithm is presented for photo hull reconstruction. This algorithm allows arbitrary camera placement by performing scene traversal several times from different directions. Since then, many variations or extensions of the space carving algorithm have been proposed. Eisert *et al.* [ESG99] take a multi-hypothesis approach to verify photo-consistency of voxels. Probabilistic frameworks of space carving are proposed in [BFK02, BDC01]. Another work done by Kutulakos [Kut00] improves the stability of the space carving algorithm by taking into account image noise and calibration errors explicitly. Prock *et al.* [PD98] and Sainz *et al.* [SBS02] attempt to accelerate part of the photo hull computation using graphics hardware. Transparent objects are treated in the work [dBV99]. Specular surfaces are investigated by Chhabra [Chh01] and Bonfort *et al.* [BS03]. Vedula *et al.* [VBSK00] extend space carving to time domain and recover a motion vector for each voxel of the photo hull.

Instead of verifying the photo-consistency of each voxel, some other methods deform a surface embedded in the discretized working volume. For example, level-set methods [SSH02] evolve an implicit surface according to photo-consistency measurement. Graph-cut methods [KZ02] define an energy function that represents global photo-inconsistency over all the voxels on an embedded surface. By minimizing the energy function, the surface converges to the shape of the photo hull.

Apart from voxel-based methods, some researchers build a polyhedral visual hull mesh as an initial estimate, and then move the vertices on the mesh so that they become photo-consistent [ESG00, IS02, KHV04]. Since the photo-consistency is only evaluated for each vertex, the computational cost is normally lower than those of the voxel-based methods.

All the above algorithms reconstruct a view-independent model. However, for novel view synthesis, a view-dependent model would be enough. As a natural extension of the image-based visual hull, image-based photo hull [SSH03] is reconstructed by checking photo-consistencies of the sample points along each viewing ray of a target view. Yang *et al.* [YWB02] exploit graphics hardware to render a stack of planes and perform the photo-consistency check for each rasterized fragment. This can be regarded as an extension to Lok's visual hull reconstruction method [Lok01].

The two surveys [SCMS01, Dye01], already mentioned in Section 2.3.1.1, also provide thorough reviews on reconstruction of photo hulls.

Visual hulls and photo hulls share the same underlying geometry primitives such as voxels, points or polygons. Therefore, if the same geometry primitives are used, the visual hull rendering algorithms in Section 2.3.2 apply to photo hulls as well. However, compared with the visual hull representation, the photo hull is a more accurate geometric representation, which generally leads to more photo-realistic rendering of novel views.

2.5 Graphics Hardware

Graphics hardware is generally understood as the hardware resources available on the graphics board [3] of a computer. The hardware resources typically include a *graphics processing unit* (GPU), dedicated graphics memory, and some other auxiliary chips and circuits. Modern GPUs implement a *rendering pipeline* that is very efficient at processing vector data streams and synthesizing images. Recently, the processing power of graphics hardware has been exploited to accelerate a diverse range of tasks, such as complex shading, solving equation systems, physically-based animation, image processing, novel view synthesis as presented in this work, and so on.

To utilize GPUs, there exist two major graphics programming interfaces: OpenGL [SA04] and Direct3D [Mic04a]. Although OpenGL is selected for the implementation of our algorithms, this work is not restricted by the programming interface. Direct3D offers almost the same functionality and can be used as well.

In the rest of this section, we will briefly describe the rendering pipeline of OpenGL and introduce some terminology that will be used later in the thesis.

2.5.1 Overview of the rendering pipeline

The rendering pipeline in OpenGL consists of three stages: geometry processing, rasterization and per-fragment operations, depicted in Figure 2.8. At the

[3]Graphics board is sometimes called graphics card, video card, or graphics accelerator.

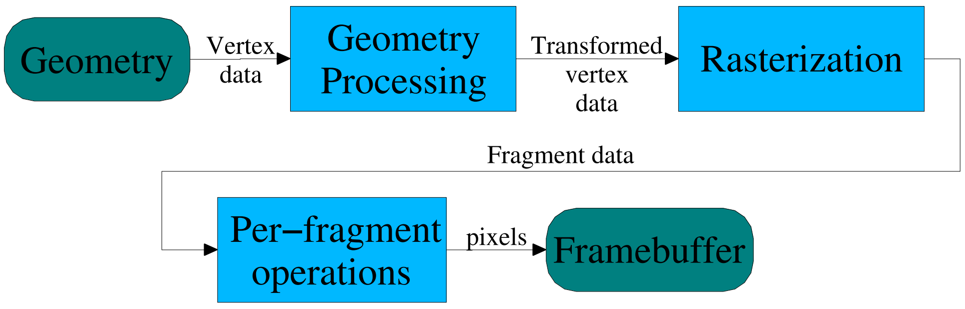


Figure 2.8: The OpenGL rendering pipeline.

beginning, a geometry primitive, represented by a group of vertices, is sent to the graphics pipeline. In the geometry processing stage, the vertex attributes, such as positions, colors and texture coordinates, are transformed or evaluated. During rasterization, the per-vertex attributes are rasterized into *fragments*. The color value of a fragment could be further modulated with texture and fog. Finally, in the stage of per-fragment operations, the fragments undergo a series of tests. Those fragments passing all the tests are called *pixels*, and will be used to update the appropriate locations in the *framebuffer*. Finally, the color content of the framebuffer will be displayed on the screen.

All functionalities of the traditional graphics pipeline are fixed. Users only have the choice to configure the parameters of some functional units, or simply to switch them on or off. Such pipeline is referred to as *fixed-function pipeline*. Recently, it becomes clear that the fixed pipeline is too limited to perform ever more sophisticated shading tasks. Consequently, parts of the traditional pipeline are replaced by fully programmable units. In addition, floating-point storage and computation become available throughout the pipeline which provides high precision. This evolution is transforming the graphics hardware into a flexible streaming processor.

In the following, we will look into more details of the fixed-function pipeline as well as the programmable units that are lately developed.

2.5.2 Fixed-function pipeline

2.5.2.1 Geometry processing

Geometry primitives are specified with multiple vertices that represent points, lines or polygons. The geometry processing mainly consists of three tasks:

1. Geometric transformation of vertex coordinates and normals:
   Vertex coordinates are represented with homogeneous coordinates in object space. They are first transformed into eye space by a *modelview* matrix. Then, the coordinates in eye space go through a set of processing steps: projection, view volume clipping, perspective division and viewport transformation. Finally, they are converted to window coordinates on the output screen. Sometimes, vertex normals also need to be transformed to eye space for lighting computation.

2. Computation of vertex colors:
   Vertex colors can be directly specified by the users. When the lighting is turned on, material and lighting properties are utilized to compute vertex colors according to a built-in equation, which is based on the Blinn-Phong model [Bli77].

3. Generation of fog coordinate and texture coordinates
   For each vertex, a fog coordinate and a set of texture coordinates can be either manually assigned or automatically generated from other vertex data, typically vertex positions and normals. For each texture coordinate, it is then multiplied by a texture matrix. If the texture matrix defines a perspective projection, the texturing operation is often called *projective texture mapping* [SKvW+92], which is widely used in our algorithms.

2.5.2.2 Rasterization

Rasterization converts primitives to a two-dimensional array of grids which contain the color and depth information. The rasterizer first figures out which grids are occupied by a primitive. For each of these grids, the z window coordinate (depth value), color, fog coordinate and texture coordinates are interpolated across the primitive from its corresponding vertex data. A grid, together with its associated data, is called a *fragment*. Then texture values are obtained by looking up texture maps using the interpolated texture coordinates. The output color of a fragment is evaluated from the interpolated color values, the texture values, and optionally modified by a fog value computed from the fog coordinate.

2.5.2.3 Per-fragment operations

Per-fragment operations control how the *framebuffer* is updated by the fragments coming from the rasterization stage. The framebuffer is composed by a two-dimensional array of pixels. Each pixel contains color, depth, stencil and accumulation values. Accordingly, the framebuffer is divided into several logical buffers, which are named as *color buffer, depth buffer, stencil buffer* and *accumulation buffer*, respectively. The content of the color buffer is used for the final display. The depth buffer and the stencil buffer assist in the per-fragment operations. The accumulation buffer can accumulate a series of color buffers and copy the accumulated result back to the color buffer for viewing.

The per-fragment operations start from a series of tests performed on the incoming fragments. The *scissor test* selects the fragment that lies in a rectangular region in the viewport. The *alpha test* compares the alpha component of the fragment's color with a reference value. The *stencil test* conditionally discards a fragment based on the result of a comparison between the value in the stencil buffer at the fragment's location and a reference value. The *depth test* compares the fragment's depth value with the depth value stored in the depth buffer. Depending on the results of the stencil test and the depth test, the stencil buffer is updated by some pre-defined operations.

If a fragment passes all the tests, its depth value is written to the depth buffer directly. The color value of the fragment might be modified with optional *blending*, *dithering* and *logic* operations, before being written to the color buffer.

For some multi-pass rendering algorithms, off-screen rendering is needed to carry out some computation. OpenGL provides a *P-buffer* extension (short for pixel buffer) [Sil, Oped] to address this issue. All the operations for the framebuffer apply to the P-buffer as well.

2.5.3 Programmable pipeline

2.5.3.1 Vertex program unit

A vertex program [Opec] is an assembler program that is written by a user and executed on the graphics board. For each vertex, the vertex program takes a set of vertex attributes as input data, performs computations, and outputs a new set of vertex data for rasterization. When a vertex program is enabled, certain tasks in the geometric processing stage of the fixed pipeline are bypassed. Therefore, depending on applications, a vertex program should take over some of the basic tasks, such as modelview and projection transformation, normal transformation, color computation, texture and fog coordinate generation, *etc*.

The latest version of the vertex program [NVIb] features a very flexible instruction set, which supports basic arithmetic operators, dot product, comparison, trigonometric and exponential functions, vector component swizzling and masking, branching and subroutine call, texture lookup, etc. Since the input vertex attributes are mostly 4-component floating-point vectors, the instruction set is especially designed to process such vectors efficiently. The vertex program also has access to OpenGL state variables and a number of environment variables specified by applications.

2.5.3.2 Fragment program unit

Initially, NVIDIA and ATI add limited programmability to replace parts of the rasterization unit of the fixed pipeline. More flexible mixture of fragment color and multiple texture values is supported by OpenGL extensions like register combiners [NVIc] and fragment shaders [ATI]. Later, the programmability evolves into a general model that allows a user to run a customized fragment program [Opea] on graphics hardware. A fragment program takes interpolated fragment data and computes a color and a depth value for each fragment. The instruction set is very similar to that of a vertex program.

2.5.3.3 High level shading languages

Vertex and fragment programs must be fed to programmable graphics hardware in assembler languages. However, as various applications demand ever more complex programs to accomplish computation-intensive tasks, it becomes impractical to write and to debug these programs in assembly languages. Therefore, a hardware independent high-level shading language needs to take this challenge since it is easy to use and provides better portability. The graphics industries have already released several languages of such kind, for example, *Cg* [MGAK03] coming from NVIDIA, *glslang* [KBR04] designed for OpenGL 2.0 and *Microsoft HLSL* [Mic04b] shipped together with Direct3D. The *Cg* language, which stands for "C for graphics", is cross-platform and works with both OpenGL and Direct3D. Therefore in our work, the hardware-accelerated algorithms are implemented using *Cg*.

Chapter 3

Synchronized Acquisition of Pre-calibrated Multi-view Videos

The input data of the novel view synthesis algorithms to be presented are a set of color video sequences taken from multiple viewpoints. Since visual hulls are used in all our algorithms, these video sequences need to be segmented into foreground and background parts so that silhouettes can be identified. In addition, the video sequences must be synchronized for every frame to ensure that the multi-view image data contain consistent information. Although synthetic data can serve as inputs, the ultimate goal of our algorithms is to process data taken from the real world. This chapter explains how we acquire the synchronized video data in an indoor environment. Specifically, we will describe our camera system, the computer infrastructure and the image processing algorithms.

3.1 Camera system

We choose Sony DFW-V500 digital video camera [Son] for image acquisition because it has the following features:

- High-speed acquisition and data transfer.
  The Sony video camera can capture color images of up to 640×480-pixel

resolution at a frame rate of 30 fps. The image data are encoded in a compact format (YUV4:2:2 or YUV4:1:1) and transferred digitally to a host computer through IEEE1394 interface (also known as Firewire). Therefore, no Analog-to-Digital conversion is needed. The transfer rate is 400Mbps. The bandwidth is sufficient to transfer videos at maximum resolution and highest frame rate.

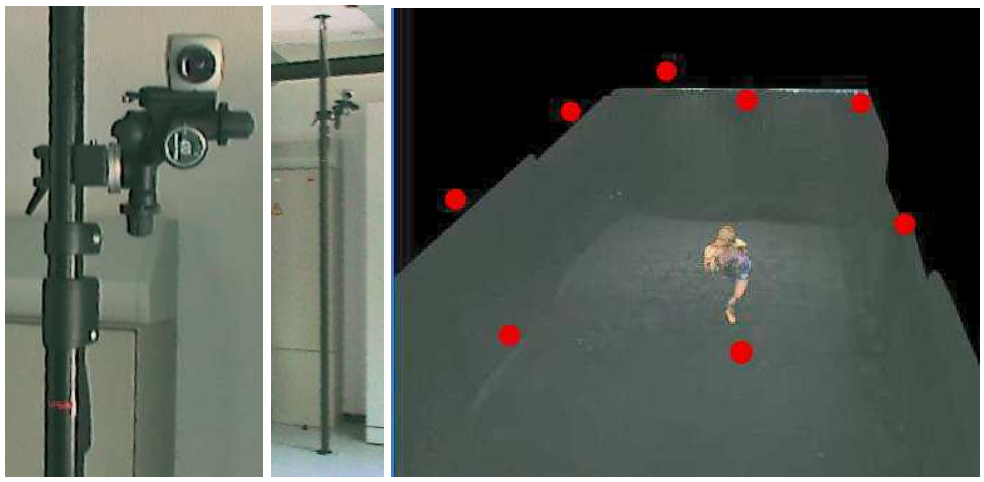
- Synchronization.
  The camera has an external trigger containing four pins. When is trigger function is set to ON, the camera does not start image acquisition until a pulse is detected in one of the four pins. This feature is of vital importance for synchronizing multiple view streams.

- Programmable control.
  A set of registers on the camera store the settings related to image color, filtering, acquisition and transfer. It is possible to save all settings in a channel for later use. A control program running on the host computer can adjust all the settings, load or unload them and start or stop the image acquisition. This programmable control greatly facilitates the management of the camera system.

We have eight cameras[1] installed in a room that is approximately $11m\times5m$ large and $3m$ high. The controlling computers only need a small area of the room on one side. The remaining space is called working space and reserved for the dynamic objects to be captured. In order to simplify the later image segmentation task, dramatic lighting changes and complex inter-reflections should be prevented. Therefore, the working space is surrounded by black curtains and the floor is covered with a black carpet.

Each camera is mounted on a geared head which can be adjusted in three directions. The head is attached to a pole via a clamp. We can fix the pole between the ceiling and the floor (See Figure 3.1b). This way, the camera can be positioned in any free space in the room. Figure 3.1(c) shows eight surrounding

[1] At the early stage of developing this system, we had six cameras. Thus, the algorithm presented in Chapter 4 is based on a camera system only containing six cameras.



(a) (b) (c)

Figure 3.1: Camera mounting, positioning and arrangement. **(a)** The camera is flexibly mounted on geared head attached to a pole. **(b)** The poles can be fixated at arbitrary location by jamming them between floor and ceiling. **(c)** A possible camera arrangement. Red circles indicate the positions of the cameras.

cameras pointing to the central area of the working space. This is a common way to arrange cameras. Most video data used in our algorithms are captured using this configuration.

3.1.1 Camera calibration

Camera calibration is a rudimentary task for 3D reconstruction algorithms. It calculates a set of camera parameters which determine the geometric relationship between a 3D object and its 2D projection on the image. Most of calibration algorithms are based on a pinhole camera model (see Figure 3.2). This model is simple and well accepted both in computer vision and computer graphics communities. The parameters of a pine hole camera can be classified into intrinsic and extrinsic ones. The intrinsic parameters are related to the imaging system of the camera itself. In a widely-used camera model, they include radial distortion coefficient κ, focal length f, principal point c and aspect ratio s. The extrin-

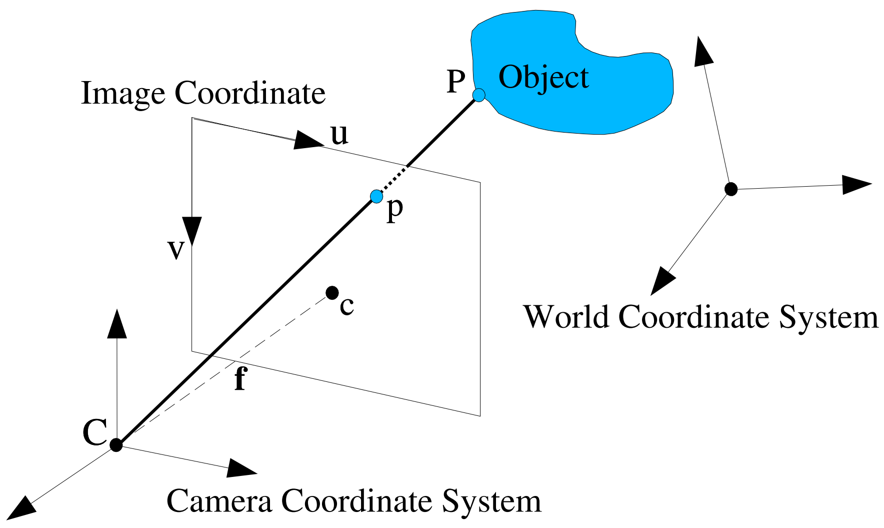


Figure 3.2: Pinhole camera model. C is the optical center of the camera, also called the viewpoint. The dotted line passing the optical center is the principal axis. c is the principal point, which is the intersection point of the principal axis and the image plane. f is the focal length of the camera. It is the distance between the optical center and the principal point.

sic parameters describe the camera's 3D position and orientation relative to the world coordinate system where the coordinates of scene objects are normally specified.

To calibrate a pinhole camera, classic methods optimize the mappings between known 3D features of a calibration target and their corresponding 2D positions in the image. Among these methods, a widely-used algorithm is proposed by Tsai [Tsa86]. This algorithm employs Levenberg-Marquardt optimization method [PTVF92] to compute camera parameters and works very well in practice. The input data are a number of 3D points and its 2D projection in the image. The 3D points could be coplanar or non-coplanar. We use planar ones since our calibration pattern is a $2m \times 2m$ checkerboard constructed on the floor (See Figure 3.3). In the Tsai algorithm, the calibration of the full camera model needs at least 11 points. However, in order to increase calibration accuracy, we adjust cameras so that at least 18 corner points of the checkerboard can be seen

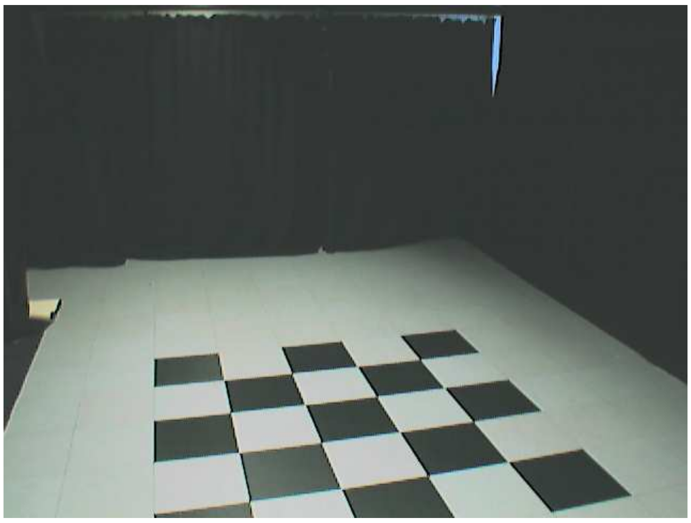


Figure 3.3: Camera calibration pattern on the stage floor, viewed from one camera.

by each camera. We have developed an interactive calibration program based on Tsai's algorithm. The user can easily specify the 2D-3D point correspondences. The whole calibration process takes less than half an hour for eight cameras. The average reprojection error is less than one pixel for images at a resolution of 640×480 pixels. This is precise enough for our purpose. It should be mentioned that during camera calibration, the black carpet on the floor needs to be removed to reveal the calibration pattern.

To further simplify the calibration task, more advanced calibration techniques could be employed. For example, automatic 2D-3D correspondence assignment can eliminate the need of manual work. Another calibration technique, *self calibration* [PKG99], does not need a specific calibration target. This technique exploits robust estimators like RANSAC [HZ00] to match image features between different views. Full calibration information could be computed from the feature correspondences. A scene image usually contains enough feature points that can be automatically located. In case of lacking feature points, one can move an LED or a small colored ball around the scene and capture these images for calibration.

Since the positions and orientations of all cameras keep unchanged during acquisition, the calibration procedure only needs to be carried out once for each camera. The acquired videos are called *pre-calibrated videos*.

3.1.2 Color calibration

Different cameras have their unique imaging systems. Therefore, even if their color controls, apertures and shutter times are set to the same, they still produce images with inconsistent colors. To deal with this problem, we first manually tune the camera settings to make the captured images have as consistent colors as possible. Then a color calibration matrix is estimated between each pair of cameras using least square methods. At runtime, these matrices are used to perform color transformation on the original images.

3.1.3 Camera synchronization

Camera synchronization guarantees that multi-view images contain information consistent for the same time instant. This is crucial to 3D reconstruction and rendering quality. In the *Virtualized Reality* system [KNR95], VITC (Vertical Interval Time Code) is embedded into every frame while capturing. During the off-line analysis of the recorded video sequences, the time stamp in VITC is examined to perform synchronization.

The above method is only suitable for off-line systems. For live video streams, external triggers available on the cameras can be used to implement synchronization. We connect the trigger sockets of all cameras to a pin of the parallel port on a computer. When an image frame needs to be captured from each camera, a pulse signal is generated by the computer to notify all cameras to start capturing. This synchronization scheme is very precise. Depending on the length of trigger pulse and the internal reference signals of the cameras, the time shift between two images captured from different cameras is within the range of several milliseconds. When we take into account the client-server communication [2], 15 fps is the highest frame rate we can reach under the triggered mode.

[2]The client-server architecture of our computer system is explained in the next section.

If the objects in a scene do not move very fast, synchronization using software [HLS04] or internal computer clocks [YEBM02] can be achieved at the price of losing precision.

3.2 Computer Infrastructure

Our system is based on a client-server architecture. Connections between computers, cameras, and the synchronization device are illustrated in Figure 3.4. A pair of cameras is connected to one client computer. Four client computers are connected to a rendering server via 100Mbps Ethernet network. The synchronization device links the parallel port of the server computer to the external triggers of the cameras.

Each client computer has an Athlon 1.1GHz CPU and 768Mbyte main memory. The client computers are responsible for synchronized video acquisition as well as real-time image processing tasks. The graphics cards installed on the client computers feature NVIDIA GeForce2 GPUs. Novel views are synthesized on the server machine, which we have equipped with different CPUs and graphics cards during the course of the development of our system. In the following chapters, when we present the performance, we will give the specific types of the CPUs and graphics cards.

Most of the software packages are developed using open source projects. All the computers are installed with Debian Linux operating system. The 1394-based DC control library [DUD+04] provides a high-level API to control the firewire cameras. We create C++ wrapper classes around this API and build some command-line utility tools. The client-server communication and the multi-thread management are implemented based on a C++ framework, ACE (`http://www.cs.wustl.edu/~schmidt/ACE.html`). A highly optimized computer vision library, OpenCV [Int], is exploited to perform image processing tasks described in the next section.

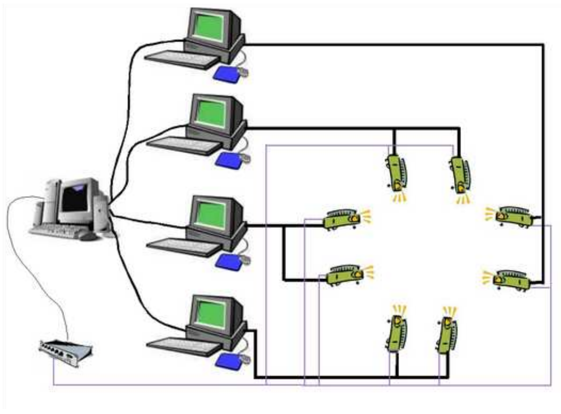


Figure 3.4: Client-server architecture of our system. Connections between computers, cameras, and the synchronization device (the box at the lower left corner) are shown.

3.3 Image Processing Algorithms

Real images acquired by our cameras need to be processed before they are passed to the novel view synthesis algorithms. This section describes two common image processing algorithms applied to the images: radial undistortion and image segmentation. Other specific processing algorithms will be discussed in the later chapters when they are required.

3.3.1 Radial distortion correction

Optic lens cause radial distortion in real images. Before we can employ an ideal camera model, the image distortion must be corrected. The distortion coefficient κ, an intrinsic camera parameter in Tsai's calibration algorithm, relates a real image coordinate (X_d, Y_d) and its undistorted counterpart (X_u, Y_u) by the following set of equations:

$$\left\{\begin{array}{lcl} X_u & = & X_d \cdot (1 + \kappa \cdot r_d^2) \\ Y_u & = & Y_d \cdot (1 + \kappa \cdot r_d^2) \\ r_d^2 & = & X_d^2 + Y_d^2 \end{array}\right. \tag{3.1}$$

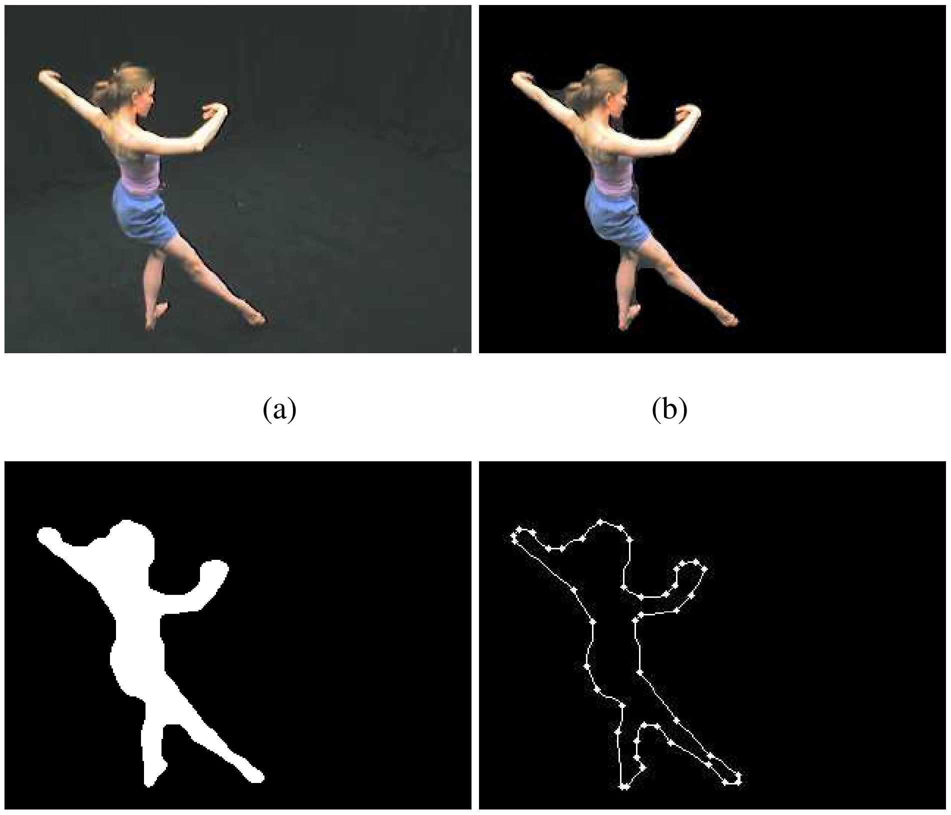
From the above equation, we can derive a cubic polynomial equation, which solves the distorted coordinate (X_d, Y_d) from a given undistorted coordinate (X_u, Y_u). The correspondences for all pixels of the undistorted image are pre-computed and stored in a lookup table. At runtime, we use inverse mapping to perform radial distortion correction.

3.3.2 Image segmentation

Since all novel view synthesis algorithms in this thesis employ the visual hull representation, silhouettes of foreground objects are needed to reconstruct visual hulls. Thus a common task is segmenting the whole image into "foreground" and "background" parts in order to identify the silhouette. A commonly-used technique to accomplish this is *bluescreen matting* [SB96]. But this technique requires a background with uniform color. To remove this restriction, a more general technique — *image differencing* [Bic94] is used in our system.

First, we accumulate 30-50 image frames when there is no object moving in the scene. From these images, we build a simple statistical background model. This model is composed by two floating-point images which contain mean values and standard deviation values, respectively. When a foreground object moves into the scene, we subtract the statistical background information from every new frame and apply thresholding. The resulting binary image contains the object mask (the silhouette), isolated noise pixels in the background and some small holes within the mask due to misclassification. We remove the noise with a median filter and apply morphological operators to fill the small holes. Finally, the contour of the silhouette is retrieved as a 2D polygon [DP73]. A parameter controls how well the silhouette is approximated by the polygon. Figure 3.5 shows an original image, the extracted foreground object, the corresponding mask and the polygonal silhouette contour.

The image differencing technique encounters difficulties in shadow regions in the scene. When a moving foreground object casts shadow onto a static surface, the shadow region can be wrongly classified as a fake foreground object due to color changes. However, the color is mainly altered in intensity. The hue keeps roughly unchanged. We exploit this fact to make the classification


(a) (b)

(c) (d)

Figure 3.5: Image segmentation and silhouette extraction. **(a)** Original image. Radial distortion correction is already performed. **(b)** Foreground object. **(c)** Foreground mask (silhouette). **(d)** 2D polygon approximation of the silhouette contour.

more stable. Silhouette extraction is critical to the later reconstruction and rendering quality. More sophisticated methods are suggested in the future work in Chapter 8.

3.4 Summary

Our acquisition facility is able to record multi-view video sequences on hard disks, or to acquire and to process live video streams on the fly. The video frames are pre-calibrated and synchronized at high precision. The camera system is easy to control and flexible enough for various applications. The distributed architecture makes the video acquisition and processing very efficient. The off-the-shelf hardware and open source software keep the overall system cost at a moderate level.

Chapter 4

Novel View Synthesis Based on a Visual Hull-Assisted Stereo Algorithm

In this chapter, we present a novel view synthesis algorithm based on improved reconstruction of dense depth maps. As mentioned in Section 2.2 in the related work, area-based depth-from-stereo methods generate dense depth maps that provide rich surface details. However, such methods are sensitive to image noise which often leads to poor depth map reconstruction. One way to improve the stability is to limit search ranges when matching corresponding pixels. This can be achieved by constructing initial estimates of scene objects and deriving search ranges from the estimated shapes. In our approach, we employ visual hulls as the initial estimates and exploit them to improve a depth-from-stereo algorithm. The better reconstructed depth maps are utilized to synthesize novel views of dynamic objects through 3D warping [McM97].

The rest of the chapter is organized as follows. We first explain the basic depth-from-stereo algorithm, and then present the visual hull-assisted stereo reconstruction as well as the rendering method. After describing our system implementation and giving the system performance, we compare our algorithm with other similar work and conclude this chapter with a summary.

4.1 Basic Stereo Algorithm

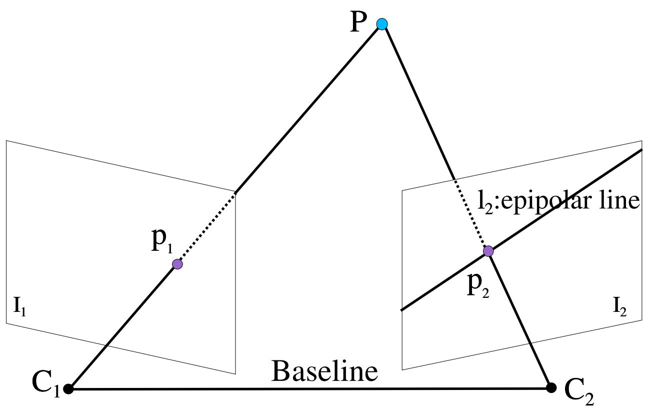
Matching corresponding pixels is the key step in stereo algorithms. Given two stereo images I_1 and I_2, the matching procedure finds the corresponding pixel in the image I_2 for each pixel in the image I_1. A naive search procedure has to examine all pixels in the image I_2. One can employ epipolar constraints to limit the search range along an epipolar line. Figure 4.1(a) shows the epipolar line l_2 corresponding to the pixel p_1. Notice that under a general stereo configuration, the epipolar line is not aligned with image scanlines, which makes the pixel traversal less efficient.

To address this issue, a perspective image warping can be applied to both stereo images so that the epipolar line coincides with an image scanline (see Figure 4.1b). The perspective warping step is called *rectification* [Fau93]. Since fixed stereo camera configuration in our system results in constant transformation between the original image and the rectified image, we accelerate the rectification operation using a pre-computed lookup table. After rectification, the rectified images $I_1^{'}$ and $I_2^{'}$ form a canonical stereo configuration. Under such a configuration, the pixel traversal becomes more efficient, and the pixel correspondence is expressed with a disparity value, which is the horizontal coordinate difference between the matching pixels $p_1^{'}$ and $p_2^{'}$.

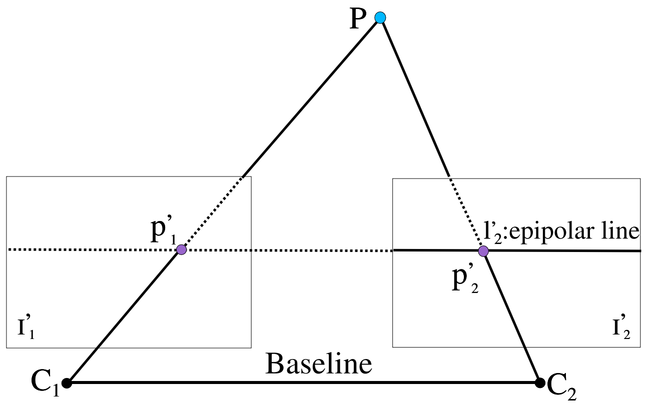
The disparity value is determined by examining rectangular regions around the pixel $p_1^{'}$ and a pixel $p^{'}$ along the epipolar line $l_2^{'}$. We choose *sum of absolute difference* (SAD) as the similarity metric because it is fast to compute and suitable for real-time systems. Suppose that the coordinates of the pixel $p_1^{'}$ are (x, y), the horizontal coordinate difference between $p_1^{'}$ and $p^{'}$ is d, the intensity functions of the stereo images for a pixel (u, v) are $I_1(u, v)$ and $I_2(u, v)$, respectively, the size of the rectangular regions is $M \times N$. Then the SAD can be written as the following Equation:

$$SAD(x,y,d) = \sum_{i=1}^{M} \sum_{j=1}^{N} |I_1(x+i, y+j) - I_2(x+i+d, y+j)|$$

By exploiting the coherence of neighboring pixels, the SAD computation is carried out in a way that the cost is independent of the window size [FHM$^+$93].



(a)



(b)

Figure 4.1: Stereo rectification. **(a)** General stereo configuration. Images are not in a common plane. Epipolar lines are not aligned with image scanlines. **(b)** Canonical stereo configuration. Images are in a common plane and parallel to the baseline. Epipolar lines are aligned with their corresponding scanlines.

Once the SAD is computed for each integer value d within a range $[d_{min}, d_{max}]$, the disparity value representing the correspondence of the pixel $p_1^{'}$ can be expressed as:

$$Disp(x,y) = \underset{d}{argmin}\, SAD(x,y,d), \quad d_{min} \leq d \leq d_{max} \tag{4.1}$$

We reverse the roles of $I_1^{'}$ and $I_2^{'}$, and compute the disparity value $Disp_{reverse}$ for $p_2^{'}$. According to the symmetry constraint [Fua93], the disparity values $Disp$ and $Disp_{reverse}$ should differ only in signs. In other words, the sum of the two disparity values should be zero. This left-right consistency check is very effective to remove false matches caused by occlusions. If the disparity value is identified to be a valid one, sub-pixel accuracy can be obtained by interpolating the neighboring SAD scores [FRT97]. Finally, the depth value can be computed directly from the disparity value:

$$Depth = \frac{Baseline * FocalLength}{Disp} \tag{4.2}$$

where $Baseline$ is the distance between the optical centers of a stereo pair, $FocalLength$ is the effective focal length, which is one of the intrinsic camera parameters.

4.2 Visual Hull-Assisted Stereo Algorithm

Equation 4.1 suggests that, if the value d varies in a large range, more pixels need to be examined. This will not only hinder the overall performance but also increase the possibility to match wrong pixels. Therefore, one key factor influencing the correspondence search is the disparity range. Based on this observation, we exploit the conservative shape approximation, the visual hull, to narrow disparity search range during stereo matching. This is the basic idea of our approach.

Depth-from-stereo algorithms require that two viewpoints are close to each other. On the other hand, shape-from-silhouette algorithms can reconstruct better shapes if viewpoints are distributed more evenly. Hence, we use a mixed

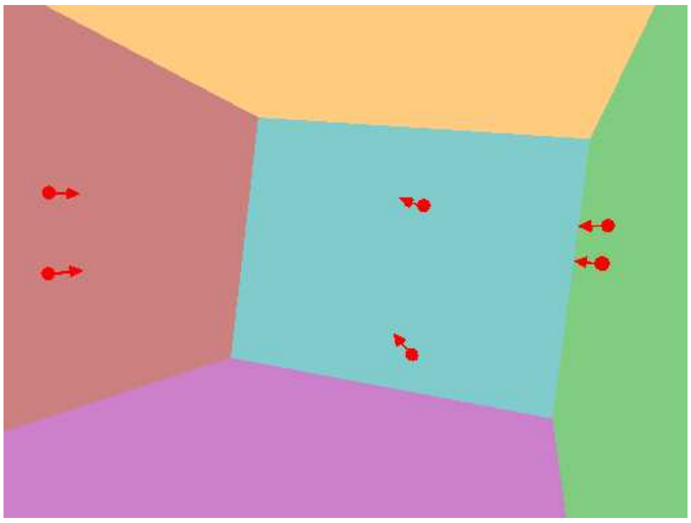


Figure 4.2: Mixed configuration for visual hull-assisted stereo algorithm. Three vertical stereo pairs are arranged in a convergent way.

configuration as shown in Figure 4.2, where three vertical stereo pairs are arranged in a convergent way. Since a pair of stereo cameras are close to each other, using both of them for visual hull reconstruction does not improve the shape accuracy very much. Therefore, the visual hulls of scene objects are only reconstructed from three camera views, each coming from one stereo pair.

We choose polyhedral representations of visual hulls because they are more suitable for generating disparity ranges. To reconstruct a polyhedral visual hull, we extract 2D silhouette polygons as described in Section 3.3.2, and then back-project these polygons into 3D space to create silhouette cones. Finally, an open source library, Breplibrary [Bek], is employed to carry out 3D intersection of silhouette cones. Figure 4.3 shows the wireframe model of a polyhedral visual hull.

Now that the visual hull information is available, we proceed to explain how to exploit this information. In the following sections, we describe two ways to limit disparity ranges. The first one imposes a global range constraint on stereo computation for all pixels. The second one generates per-pixel disparity ranges using off-screen rendering of visual hulls.

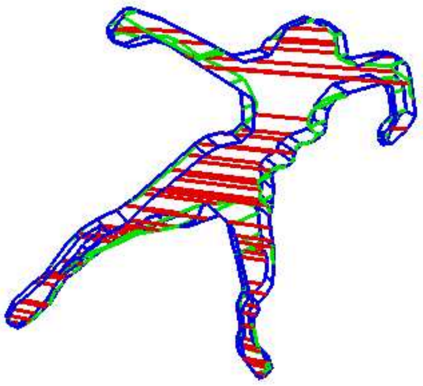


Figure 4.3: The wireframe model of a polyhedral visual hull

4.2.1 Global disparity range constraint

We have already known that a visual hull is a conservative approximation to the actual object, which means its bounding box must also be conservative. Therefore, we can compute the bounding box of a visual hull and derive the disparity range as follows.

For each reference view, we transform eight vertices of the bounding box from the world coordinate system to the camera coordinate system of the reference view. Then the minimum and maximum depth values are calculated from these transformed vertices. The depth range can be converted to a disparity range using the following formula:

$$Disp = \frac{Baseline * FocalLength}{Depth}. \tag{4.3}$$

The main advantage of the above method is that the bounding box computation can be determined from the polyhedral model of a visual hull very quickly. The downside is that the disparity range is a global range. For most foreground object pixels in a reference view, it is over-conservative.

4.2.2 Per-pixel disparity range constraint

To tackle the drawback of the global disparity range constraint, we refine the disparity range to a per-pixel level. This way, the stereo algorithm can further benefit from more precise constraints. In this section, we will present two schemes for generating per-pixel range constraint. One is not conservative but faster, the other is truly conservative at the expense of more computation. Since both of them need a disparity map generated from the visual hull, we will first discuss this problem.

4.2.2.1 Disparity map computation

Given the polyhedral visual hull and the camera calibration information, we are able to generate a depth map for each reference view. This is performed by using OpenGL off-screen rendering which is supported by graphics hardware. The generated depth maps can be easily converted to disparity maps by applying Equation 4.3. Since the depth value in this equation is different from the depth values stored in the OpenGL depth buffer, we need to do this conversion first.

According to the geometric transformation defined in OpenGL [SA04], we relate the OpenGL depth value Z_{gl} and the real depth value with the following formula:

$$Z_{gl} = (\frac{Depth \cdot (Far + Near) + 2 \cdot Far \cdot Near}{Depth \cdot (Far - Near)} + 1)/2, \tag{4.4}$$

where *Far* and *Near* are the far and near clipping planes used in the viewing volume setup, respectively.

We rewrite Equation 4.4 and represent the real depth value as a function of Z_{gl}:

$$Depth = \frac{Far \cdot Near}{Far - Z_{gl} \cdot (Far - Near)} \tag{4.5}$$

Then substituting Equation 4.5 to Equation 4.3 leads to:

$$Disp = (\frac{1}{Near} - \frac{Z_{gl} \cdot (Far - Near)}{Far \cdot Near}) \cdot Baseline \cdot FocalLength \tag{4.6}$$

Since Far, $Near$, $Baseline$, $FocalLength$ are all constants, the above equation suggests that the disparity can be computed as a multiplication followed by an addition from the OpenGL depth map. Such kind of arithmetic image operations are supported by the OpenCV library [Int], which is highly optimized for modern CPUs.

4.2.2.2 Approximate per-pixel range

Normally, the depth buffer contains the depth value for the front surfaces. Thus, the disparity map obtained from the depth buffer corresponds to the upper bounds of disparity values. In order to generate the lower bounds, we can simply subtract the upper bounds by a constant threshold. The selection of the threshold depends on the maximum concavity of actual scene objects. However, since the concavity of the object is unknown in advance, this scheme does not guarantee the correct disparity range coverage for the scene object. Consider the situation of recovering the depth value of point F in Figure 4.4. If the threshold is not big enough, the correct disparity value will fall outside the approximate range and cannot be recovered.

For a moving object that does not exhibit noticeable concavities, the approximate per-pixel range works quite well. Figure 4.5 shows the improved reconstruction results. Note that in Figure 4.5(a) the disparity map is very smooth, which means the surface detail of the object is poorly represented by the visual hull. In Figure 4.5(b), there is a region of false depth values on the right lower part of the body due to large disparity range. The false depth recovery is corrected in Figure 4.5(c) when we apply the approximate per-pixel range constraint. Meanwhile, more details are recovered by the stereo algorithm compared with the visual hull.

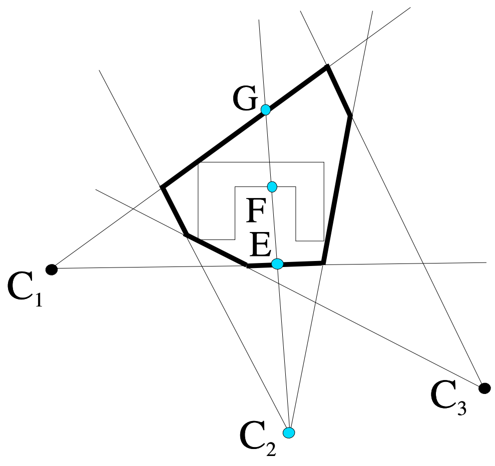


Figure 4.4: Per-Pixel range constraint. The visual hull represented in 2D is drawn as thick lines.

4.2.2.3 Conservative per-pixel range

In the previous scheme, we only make use of front surfaces of visual hulls to generate approximate disparity ranges. If we also know depth values of back surfaces, we can obtain both upper and lower bounds of disparity values that form conservative ranges. For example, in Figure 4.4, to recover the depth of the point F, we can create a disparity range from the point E and the point G. Within this range, it is possible to find the disparity value that corresponds to the correct depth of the point F.

To generate the depth map of the back surfaces of a visual hull, we use the following OpenGL settings:

```
glClearDepth(0)
glDepthFunc(GL_GREATER)
```

Then the generated depth map is converted to the lower-bound disparity map using the method described in Section 4.2.2.1.

One disadvantage of this scheme is that it needs two rendering passes to generate depth maps, one pass for front surfaces and the other for back surfaces.

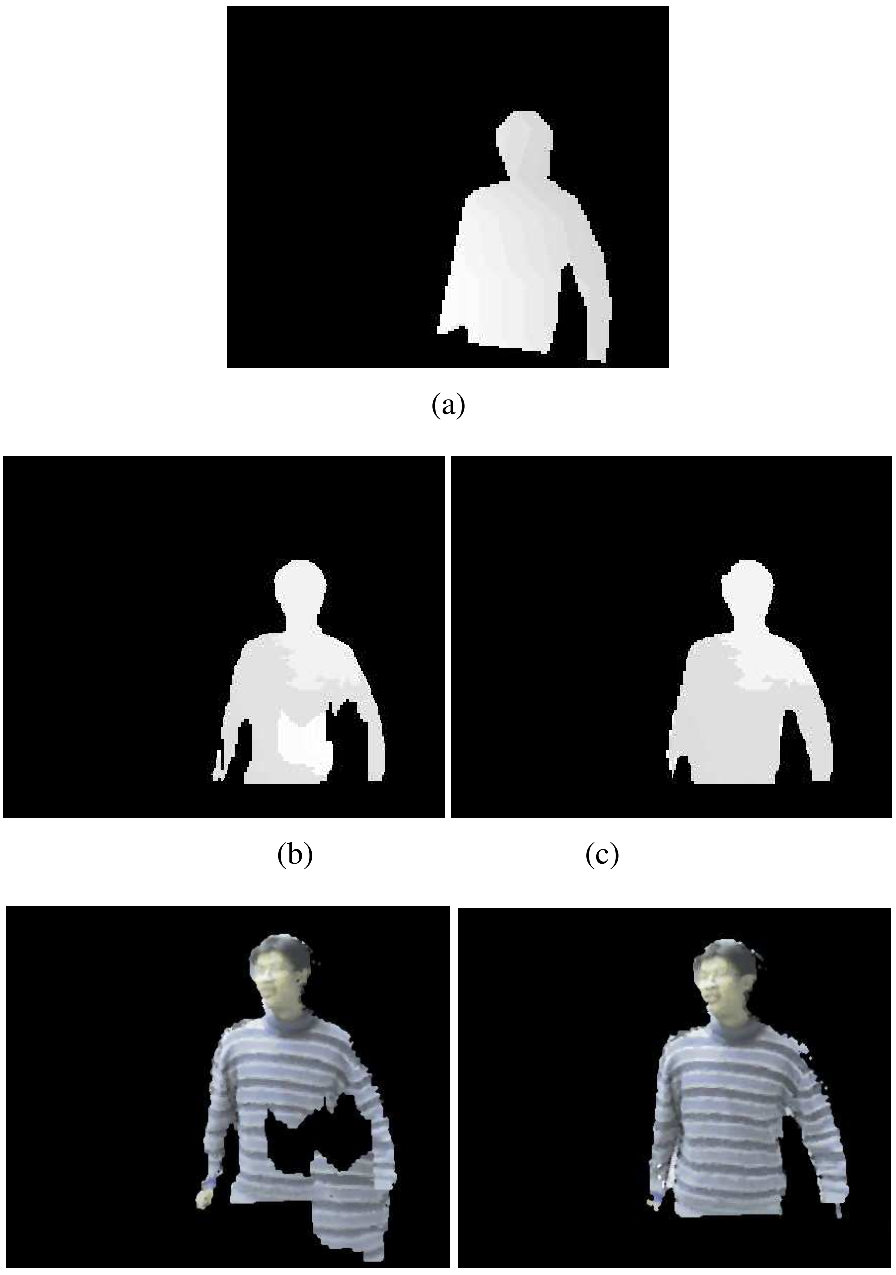


Figure 4.5: Improved reconstruction using the approximate per-pixel range constraint. **(a)** Disparity map generated from the visual hull. **(b)** Depth map without using visual hull information. **(c)** Improved depth map using approximate range limit by a threshold. **(d)** Rendered view using the depth map of Figure *4.5*(b). The rendering method will be presented in Section 4.3. **(e)** Rendered view using the depth map from Figure 4.5(c).

Fortunately, the depth map generation is performed with graphics hardware. The real-time performance does not drop too much according to our experiments.

4.3 Rendering

One can merge the depth maps recovered from multiple stereo pairs into a single polyhedral model and render the model with texture mapping. However, the polyhedral model extraction is time-consuming and not suitable for real-time applications. Therefore, we use warping-based methods for rendering. The 3D warping equation is expressed as follows:

$$\begin{cases} x_2 & = & ((w_{11} \cdot x_1 + w_{12} \cdot y_1 + w_{13}) \cdot z_1 + w_{14})/z_2 \\ y_2 & = & ((w_{21} \cdot x_1 + w_{22} \cdot y_1 + w_{23}) \cdot z_1 + w_{24})/z_2 \\ z_2 & = & (w_{31} \cdot x_1 + w_{32} \cdot y_1 + w_{33}) \cdot z_1 + w_{34} \end{cases} \tag{4.7}$$

(x_1, y_1) and (x_2, y_2) are the pixel coordinates in a reference view and a novel view, respectively. z_1 and z_2 are the depth values corresponding to the pixel (x_1, y_1) and the warped pixel (x_2, y_2), respectively. The constant values w_{ij} $(1 \leq i \leq 3, 1 \leq j \leq 4)$ are computed from the pinhole camera models of the reference view and the novel view. By exploiting the spatial coherence between neighboring pixels in the reference view, the warped pixel coordinates are computed incrementally for efficiency [PEL+00]. To avoid aliasing artifacts, we have implemented the splatting technique described in the work [SGHS98], in which each splat is associated with a Gaussian kernel.

We use a fuzzy Z-buffer algorithm to resolve visibilities in the novel view. Warped pixels having similar depth values within a pre-defined threshold may correspond to the same point in 3D. They are combined into a final color C using the following formula:

$$C = \left[\sum_{k=1}^{N} W_{ks} \cdot C_k \right] / \sum_{k=1}^{N} W_{ks}, \tag{4.8}$$

where N is the number of warped pixels that have similar depth values, W_{ks} is the splatting weight, C_k is the color of a warped pixel.

Figure 4.6 shows the rendering results of a moving object composed in a virtual room. So far, our current rendering algorithm does not take advantage of hardware-acceleration. If graphics hardware features like point sprite [Opeb] and fragment program [Opea] are available, hardware-accelerated 3D warping [HH02] or point rendering [CH02] can be implemented to increase the rendering speed.

4.4 System Implementation and Performance

We distribute the reconstruction and rendering computation in the client-server system described in Chapter 3. Six cameras are used as indicated in Figure 4.2. They are connected to three client computers. The server machine has the same configuration with the client computers, i.e. 768Mbyte main memory, an AMD Athlon 1.1GHz CPU and an NVIDIA GeForce2 graphics card.

The system initialization consists of creating a background model for each camera and sending camera parameters of reference views to the server. After the initialization, the system enters a *processing cycle*, which is defined as the time of processing one synchronized image set collected by all cameras. According to the direction of the network transfer, each cycle is divided into three stages as illustrated in Figure 4.7.

Stage 1 (Figure 4.7a) is mainly responsible for rectification and image segmentation. First, each stereo pair is rectified to align epipolar lines along with image scanlines. This is necessary for the fast stereo computation as already explained earlier in Section 4.1. Then the moving foreground object in each reference view is segmented out from the background. Finally, for one camera of each stereo pair, the 2D polygonal silhouette contour is retrieved and sent to the server.

In stage 2 (Figure 4.7b), when all silhouette information is available, the server computes a polyhedral visual hull using general 3D CSG intersection and then broadcasts visual hull information back to all clients. If global range constraint is employed, only eight vertices of the visual hull's bounding box need to be transferred, which costs trivially. If per-pixel range constraint is

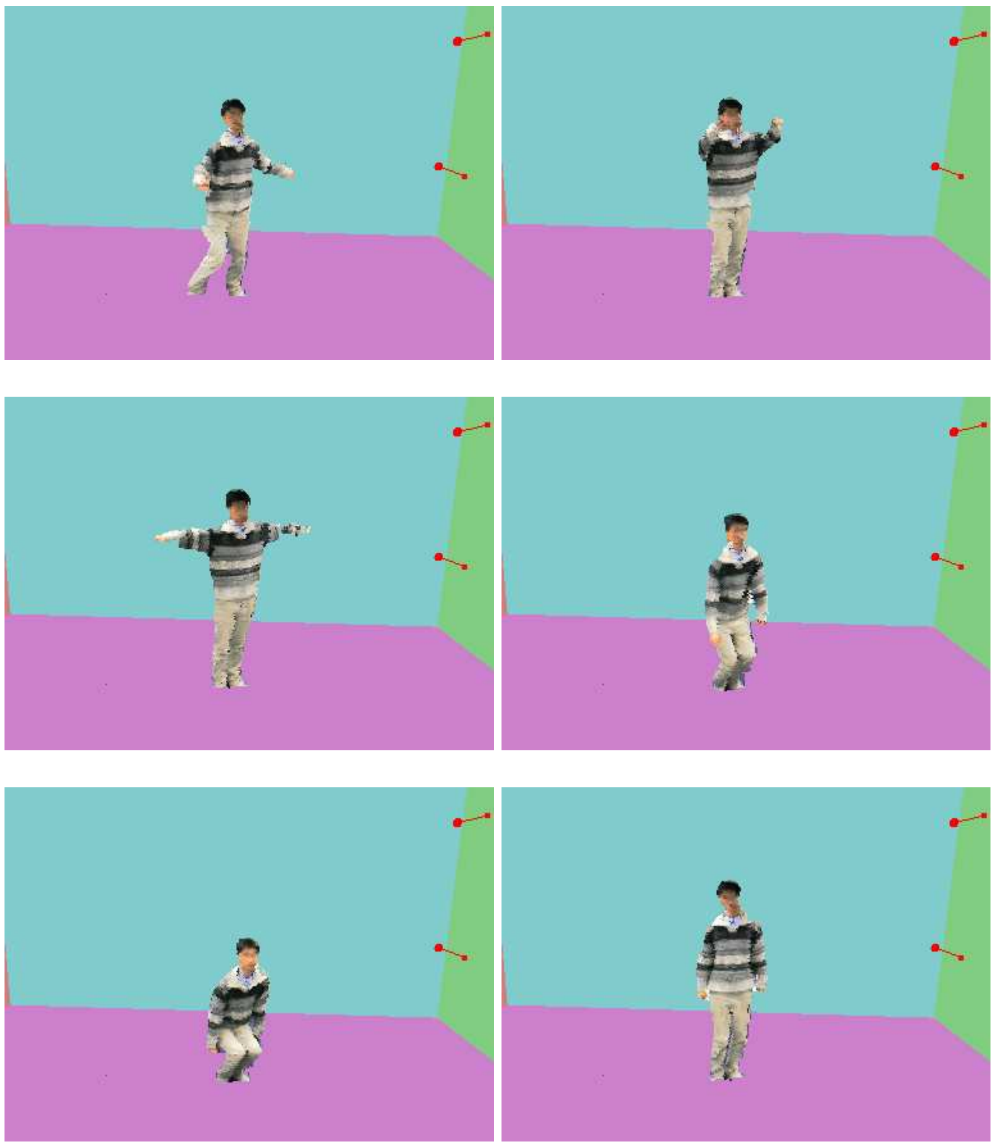


Figure 4.6: Rendering results of 3D warping. A sequence of dynamic events rendered from a novel viewpoint at about 10 fps. The two arrows show the positions of a camera pair (about 30° away from the current viewpoint).

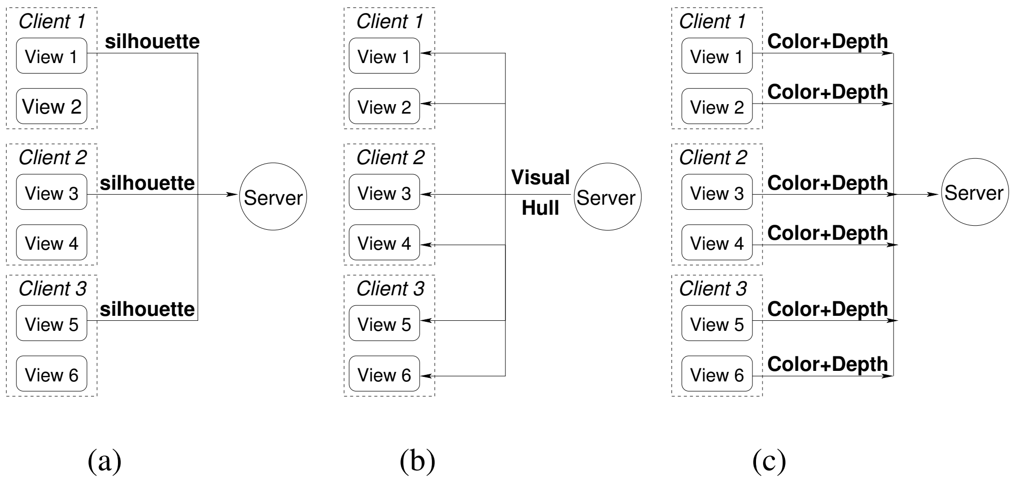


Figure 4.7: Processing stages of our system. Each client computer processes two views that form a stereo pair.

required, the polyhedral visual hull itself will be sent back to client computers.

In the last stage (Figure 4.7c), all client computers use visual hull information to guide stereo computation and generate depth maps with improved quality. Since we already have silhouette information, stereo computation only needs to be performed on foreground object masks instead of whole images. Finally, the depth maps, together with the color images, are sent to the server for rendering. For the network transfer, the silhouettes in reference views are again exploited to create bounding rectangular regions of interest (ROI). Only the ROIs of the color and depth data are sent to the server. This way, the network bandwidth is greatly reduced.

From the processing stages we can see that the computation is well distributed among the computers. The time needed for image capturing, rectification, silhouette extraction and stereo reconstruction is independent of the number of stereo pairs. This provides good scalability and is one of the important reasons why we can build an on-line novel view synthesis system.

In order to further increase parallelism within each computer, we use multi-thread implementation for our novel view synthesis system. On each client computer, five threads are running. Two of them acquire video streams from

Processing	**Timing (ms)**
Image acquisition & Rectification	29/29
Silhouette extraction	29/43
Visual hull reconstruction	15
Approximate disparity computation from visual hulls	87/74
Disparity computation from stereo	30
Depth computation	25/20
Network transfer of color+depth maps	57/48
Miscellaneous	27/13
Total time	299/272

Table 4.1: Timings measured for one client computer in our system. The approximate range constraint (see Section 4.2.2.2) is employed. The two numbers for the timing correspond to different images acquired by one of the stereo pairs.

a stereo pair. The other two perform rectification, silhouette extraction, disparity range generation from visual hull and communication with the server. The main thread is responsible for the stereo computation. On the server side, the main thread visualizes the reconstruction result. The rendering thread is decoupled from the visual hull reconstruction and runs at a faster speed in order to provide interactivity of novel view navigation. The other thread is used to reconstruct the visual hull and generates synchronization signals to trigger all the cameras. The remaining threads are employed to receive silhouette information and color-plus-depth images from the client computers.

To evaluate our system performance, we fix the video acquisition resolution at 320×240 pixels. The silhouette mask in each reference view occupies around 1/6 of the whole image. The off-screen rendering of depth maps and stereo computation are performed at half of the acquisition resolution in order to get higher frame rate. The size of stereo matching window is chosen as 11×11 pixels. Under such settings, we achieve 2-4 fps for the depth map reconstruction, while the rendering algorithm runs at about 10 fps. We measure the timing of each step in the processing cycle on one of the client computers and present the results in Table 4.1.

4.5 Discussion

One idea to use approximate geometry for stereo reconstruction has been proposed by Debevec *et al.* [DTM96]. With the help of user interaction, they reconstruct a hierarchy of parameterized primitives instead of visual hulls to approximate scene objects. Using the geometry information of the primitives, one image of the stereo pair is warped onto the image plane of the other image. Thus, the foreshortening problem for the far-apart stereo pair is eliminated and the stereo reconstruction can be done more robustly. The disadvantage of this algorithm is that it is not fully automatic. In addition, the types of scene objects are also limited by the supported primitive types.

Vedula *et al.* [VRSK98] present a similar way to enhance the stereo algorithm using silhouette information. They execute stereo algorithm and merge the recovered depth maps to obtain a volumetric model, from which a polygonal mesh is extracted. The polygonal mesh is reprojected to generate per-pixel search bounds for stereo matching. Meanwhile, silhouette masks are extracted from depth discontinuities. Then depth map reconstruction is refined by using the search bounds. The improved depth maps are merged again into a volumetric model. The above steps are carried out iteratively. Finally, the refined volumetric model is carved using the silhouette information, converted to a polygonal model and rendered with texture mapping. They call the reconstruction method *Model-Enhanced Stereo* (MES) algorithm. Our algorithm shares the same spirit with theirs in the aspect of employing the restricted search bounds for stereo computation. However, there are some noteworthy distinctions. First, we extract silhouette information from original images in the earlier processing stage, while they generate the silhouettes from the estimated 3D objects in the later stage. Second, we use a polyhedral visual hull representation for direct and faster reconstruction of an initial geometry estimate, whereas they use volumetric representation and need to extract polyhedral models, which is very time-consuming. Third, also as a result of the above two points, our system can do on-line processing while theirs cannot.

4.6 Summary

In this chapter we have presented a novel view synthesis algorithm based on visual hull-assisted stereo computation. The algorithm is demonstrated to run interactively on an on-line processing system. We adopt the polyhedral visual hull representation as an initial estimate of scene objects. It can be quickly reconstructed, and hence meets the requirements of a real-time system. On the one hand, the estimated visual hull imposes a bounding volume constraint or per-pixel constraints on the depth-from-stereo algorithm to improve the result of depth map recovery. On the other hand, stereo algorithm is able to recover concave regions on the object, which compensates for the inherent limitation of the visual hull representation. We produce better 3D reconstruction results and still keep interactive performance. This is made possible by fast reconstruction of visual hulls, distributed computation across several machines, multi-thread implementation, and hardware-accelerated depth map generation from visual hulls.

Chapter 5

Hardware-Accelerated Novel View Synthesis of Visual Hulls

The novel view synthesis algorithm presented in the previous chapter is based on the depth map representation. Although visual hull information is exploited to improve the quality of depth map reconstruction, there might still exists false depth recovery, which can affect the rendering quality significantly. In practice, given multiple reference views (usually more than three), the visual hull representation offers a fairly good approximation to the true geometry of a 3D object. Therefore, when a scene object does not exhibit noticeable concavities, we can simply reconstruct a visual hull and use it for novel view synthesis.

In this chapter we present an innovative technique to reconstruct visual hull implicitly by exploiting graphics hardware capabilities. The key idea is trimming polyhedral silhouette cones with projective alpha maps during rendering. We refer to the reconstruction technique as projective alpha map trimming. A novel view synthesis algorithm based on this technique has several advantages:

1. The visual hull reconstruction is robust and has no voxelization artifacts in final rendering results.

2. The novel view synthesis algorithm is more efficient because explicit visual hull reconstruction is avoided.

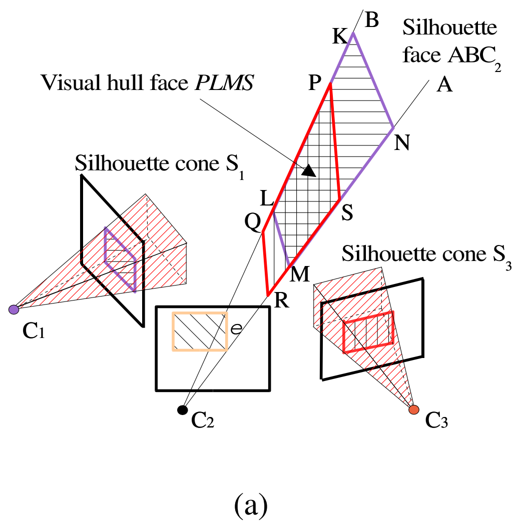
3. The novel view synthesis algorithm is fully hardware-accelerated and shows significant performance improvements over previous ones.

The remaining part of this chapter is organized as follows. Although the visual hull reconstruction is coupled with rendering, we will explain the reconstruction technique separately in order to single out how the computation is mapped onto graphics hardware (Section 5.1). In Section 5.2, a basic single-pass rendering algorithm is presented to render textured visual hulls. A simple extension is discussed as well. To support more reference views, a multi-pass visual hull rendering algorithm is proposed in Section 5.3. After giving the system performance, we compare our reconstruction techniques with other similar ones in Section 5.5 and summarize this chapter in the last section.

5.1 Hardware-Accelerated Visual Hull Reconstruction

As we have already known, visual hulls can be reconstructed by intersecting polyhedral silhouette cones generated from multiple reference views. This section first identifies different types of intersections and then explains how to utilize graphics hardware to carry out these intersections.

We classify intersection of silhouette cones into two types: face-cone intersection and polygon-polygon intersection. The first one is the intersection of a silhouette face with a silhouette cone from another viewpoint. It produces one polygon on the silhouette face. When repeating this intersection for multiple silhouette cones, we obtain a set of polygons on the silhouette face. These polygons serve as inputs of the second kind of intersection — polygon-polygon intersection. The result is one visual hull face. An example is shown in Figure 5.1(a). Consider the silhouette face ABC_2. The face-cone intersection $ABC_2 \bigcap S_1$ and $ABC_2 \bigcap S_3$ produce the polygon $KLMN$ and $PQRS$, respectively (for clarity, the actual face-cone intersections are not shown in the figure). By applying



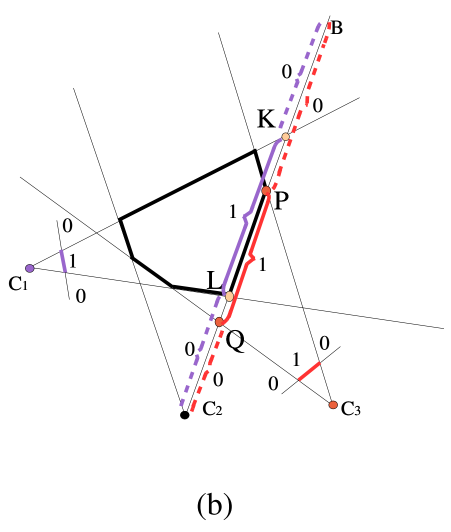


Figure 5.1: The principle of visual hull reconstruction with alpha map trimming. C_1,C_2,C_3 are reference viewpoints. We assume the visual hull is reconstructed from three views. **(a)** Face-cone intersection and polygon-polygon intersection. The hatched area on each image plane is the silhouette of the object. For the silhouette face ABC_2, the polygons *KLMN* and *PQRS* are the face-cone intersection results with respect to silhouette cone S_1 and S_3. The polygon-polygon intersection between *KLMN* and *PQRS* defines the visual hull face *PLMS*. **(b)** Top view of Figure 5.1(a). The alpha values on *KL* and *PQ* are set to one by projecting foreground masks of view 1 and view 3, respectively. If the alpha values are multiplied per-fragment, only the common part *PL* gets the final value 1 in the alpha channel.

polygon-polygon intersection between *KLMN* and *PQRS*, the visual hull face *PLMS* is obtained.

In [MBM01], the face-cone intersection is computed by projecting the silhouette face into each image plane and intersecting the projected face with the silhouette edges. These intersection results are lifted from the image planes back to the silhouette face again. Then the polygon-polygon intersection on the silhouette face is computed. The image-based visual hull technique [MBR+00] shares the same spirit in reducing the intersection computation from 3D to 2D. The difference is that the two kinds of intersections are discretized into line-polygon and segment-segment intersections. Nevertheless, both approaches carry out the intersections analytically and suffer from numerical instability problems. Our hardware-accelerated reconstruction algorithm performs the intersection in image space of a novel view. The computation is robust for arbitrarily complex silhouettes cones.

The face-cone intersection is computed by rendering a silhouette face from a novel viewpoint using projective texture mapping. We load a silhouette image from another reference view as a texture. The corresponding projective texture matrix is calculated from the calibration data associated with that view. The alpha value of the texture is set to 1 for foreground objects and 0 for the background. When the silhouette face is rasterized in the novel view, an alpha value 1 is assigned to each fragment in the intersection part. By iterating such processing for all reference views except for the one that produces the silhouette face currently being rendered (the index of this special view is denoted as $\hat{k}$), we obtain a set of rasterized polygons marked with the alpha value 1. The reason for excluding the view $\hat{k}$ is simple. We need to intersect the silhouette face in question with other silhouette cones, not the silhouette cones containing this face.

The second type of intersection, polygon-polygon intersection, is accomplished by multiplying the alpha values of the rasterized face-cone-intersection polygons. The modulation is expressed with the following formula:

$$A = \prod_{\substack{k=1 \\ k \neq \widehat{k}}}^{N} A_k \, , \tag{5.1}$$

where N is the total number of input images, A_k denotes the alpha channel of the $\boldsymbol{k}$-th projective texture, and A is the resulting alpha value. Obviously, only the common part of all face-cone-intersection polygons has a resulting alpha value 1. The common part corresponds to exactly one visual hull face and can be retrieved by enabling the alpha test. Figure 5.1(b) illustrates this idea in 2D.

So far we have described the intersections for one silhouette face of a silhouette cone. When iterating this computation over all silhouette faces of all silhouette cones, we obtain a visual hull, the intersection result of multiple silhouette cones. For example, in Figure 5.1(b), the area enclosed by thick lines is a cross-section of the resulting visual hull. Notice that since the reconstruction is accomplished by rasterization, during which hidden surfaces of visual hulls are removed. As a result, the reconstruction technique finally produces depth maps of visual hulls instead of complete geometry information.

5.2 Single-Pass Visual Hull Rendering

The depth masks generated by the above reconstruction technique only indicate where visual hulls should be rendered. To achieve realistic rendering results, 3D objects must be textured with color images. In the context of novel view synthesis, original color images taken from different viewpoints are often used to map onto the recovered 3D geometry. Thus, the user can freely choose a novel viewpoint to examine the object resembling its counterpart in the real world. A technical problem to be tackled is that one single image generally cannot cover all surfaces of the object. Therefore, we should stitch multiple textures together. For the area covered by more than one texture, we need to blend multiple textures to achieve smooth appearance.

5.2.1 Multiple texture blending

Suppose we have $\boldsymbol{N}$ input images. Then, for a visual hull face f, the rasterized fragment color C_f can be computed using the following formula:

$$C_f = \left[\sum_{k=1}^{N} W_k \cdot T_k \right] / \sum_{k=1}^{N} W_k = \left[\sum_{k=1}^{N} V_{k,f} \cdot W_{kd} \cdot T_k \right] / \sum_{k=1}^{N} V_{k,f} \cdot W_{kd}\,, \tag{5.2}$$

where T_k is the texture color from the $\boldsymbol{k}$-th input image, W_{kd} is a weighting factor (see Equation 5.4) and $V_{k,f}$ is the visibility function for the face f with regard to view k:

$$V_{k,f} = \begin{cases} 1 & , \text{ f is visible from view k} \\ 0 & , \text{ otherwise} \end{cases} \tag{5.3}$$

There are two points that need to be clarified for this visibility function. First, for the reference view $\hat{k}$ from which the face f is generated, we state that f is invisible from this view (namely, $V_{\hat{k},f} = 0$) since the view $\hat{k}$ does not contribute to the rendering of the face f. Second, the surface normal can be used to compute the visibility if the silhouette cone is convex. For concave geometry, partially visible or self-occluded faces may occur. This visibility issue will be addressed in a more complex rendering algorithm presented in the next chapter by exploiting advanced fragment programmability on modern graphics card.

We achieve view-dependent texturing [DBY98] by including a weighting factor W_{kd}. The weight depends on the angle deviation between the principal axis of a reference view and that of a novel view. Normally, a smaller angle means that the corresponding reference view has similar orientation with the novel view. Therefore, the texels in this reference view should get higher weights. A weighting function can be chosen as:

$$W_{kd} = 1/acos(\bar{d}_k \bullet \bar{d}), \tag{5.4}$$

where $\bar{d}_k$ and $\bar{d}$ represent the principal axis of the reference view k and the novel view, respectively. Strictly speaking, the angle deviation should be computed from the reference and the novel viewing directions for each visible surface point. But if the principal axis of a view does not depart too much from the center of an object, the principal axis can approximate the viewing directions for all surface points on the object. This approximation leads to faster renderig algorithm as well as less demanding graphics hardware.

In Equation 5.2, there is an expensive per-pixel division operation. Since both $V_{k,f}$ and W_{kd} are fixed values for each face, we avoid this per-pixel division by normalizing their product $V_{k,f} \cdot W_{kd}$ in advance:

$$\widehat{W_{k,f}} = V_{k,f} \cdot W_{kd} / \sum_{k=1}^{N} V_{k,f} \cdot W_{kd} \tag{5.5}$$

By substituting this normalized weight into Equation 5.2, we obtain the multiple texture blending function that is to be evaluated on graphics hardware:

$$C_f = \sum_{k=1}^{N} \widehat{W_{k,f}} \cdot T_k \tag{5.6}$$

5.2.2 Basic rendering algorithm

In order to implement multiple texture blending, we need a more complex fragment coloring mechanism than the simple OpenGL texture environment. The OpenGL extension *Register Combiners* [NVIc] serves well for this purpose.

The register combiners take interpolated colors, filtered texel values as input. After some computations are performed in a number of general combiner stages, a final combiner stage output an RGBA value for each fragment. On a Geforce4 graphics card, four texture units are available. This means we are able to handle four silhouette images in one rendering pass. To evaluate the fragment color expressed in Equation 6.1, we make use of four general combiner stages and one final combiner stage.

To render textured visual hulls, we need to load silhouette color images as an RGBA texture. The RGB channels store the color information, whereas the alpha channel stores the silhouette mask information. When rendering a silhouette face f with projective textures, for each vertex of the face, we encode the normalized weights $\widehat{W_{k,f}}$ in the color/alpha channel of the primary vertex color and the red/green channel of the secondary vertex color[1].

The register combiners are configured to perform the following tasks. At the first general combiner stage, we use the dot product to separate the weights stored in the secondary color into two registers. For the general combiner stage $k\,(k = 2,3,4)$, we modulate the texel value T_k with $\widehat{W_{k,f}}$ and accumulate the

[1]The function glSecondaryColor only accepts 3-component color. Therefore, we cannot encode the weights in the same way as the primary color.

```
Set and enable alpha test
loop $i$ over reference views
   Load the silhouette color image $i$ as an RGBA texture $T_i$
   Set up the projective texture matrix for $T_i$
Configure register combiners and enable this extension
Loop $S_i$ over silhouette cones
 Loop $\triangle_j$ over silhouette faces of $S_i$
    Compute visibility and weight vector for all views
    Compute normalized weight vector
    Encode normalized weight vector in the primary
      and secondary color of each vertex of $\triangle_j$
    Draw $\triangle_j$
```

Figure 5.2: Basic single-pass rendering algorithm. The rendering is very efficient because the rendered primitives are triangles.

result. The final stage adds the contribution from $\widehat{W_{1,f}} \cdot T_1$. This way, multiple textures are blended together with appropriate weights to produce the color values of the silhouette face f.

The above register combiner configuration is used only for the color evaluation of the silhouette face. In addition, we must compute the alpha value and enable the alpha test to remove the extra part of the silhouette face. The alpha portion of the register combiners is configured to simulate the texture environment GL_MODULATE in order to modulate the alpha values sampled from different textures.

Once we finish the configuration, we can simply render the silhouette faces. The single-pass rendering algorithm performs visual hull reconstruction and color computation simultaneously. The pseudo code of the rendering algorithm is presented in Figure 5.2. We show snapshots of the rendering results generated for two novel views in Figure 5.3.

Notice that the algorithm is not limited to NVIDIA graphics cards. Similar multi-texture blending computations can be performed on ATI's graphics cards as well by using the OpenGL extension ATI_fragment_shader [ATI]. Recently-released more advanced graphics cards, like GeForce FX 5800/5900 series and Radeon 9700/9800/X800 series, have more powerful and flexible programma-

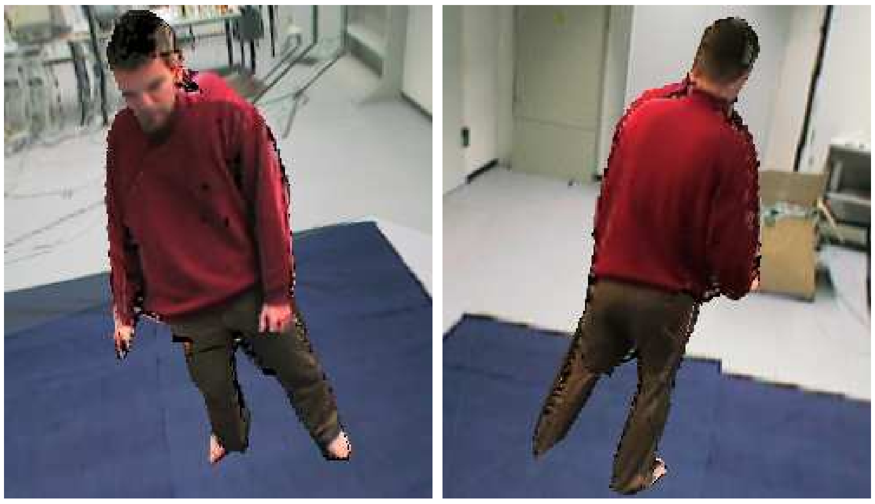


Figure 5.3: Two textured visual hulls. Four reference views are used. **(a)** Novel front view. **(b)** Novel back view. The visual hull is rendered by blending the textures from four viewpoints. The background scene is rendered as a textured box.

bility. With such cards, the color channel and the alpha channel computations described above can be implemented more straightforward using the standardized OpenGL extension ARB_fragment_program [Opea]. The algorithm is also not limited to the OpenGL API. The Direct3D API [Mic04a] provides an alternative for implementation.

5.2.3 Extended single-pass rendering

The basic rendering algorithm implementation assigns each silhouette image and projective texture matrix to a fixed texture unit. However, when a silhouette cone is rendered, only silhouette images from the other views should be projected onto the cone. Therefore, the silhouette texture corresponding to this cone must be ignored. For the color computation, it is done by setting the weighting factor associated with this texture to 0. For the alpha modulation, the alpha value 1 is used instead of the value sampled from the silhouette mask. Thus, it

	Texture object & Projective texture matrix
Cone 1	2 3 4 5
Cone 2	1 3 4 5
Cone 3	1 2 4 5
Cone 4	1 2 3 5
Cone 5	1 2 3 4

Table 5.1: Rendering visual hulls from five reference images. Four texture units are available. Each texture object or each projective texture matrix is assigned with an ID number from 1 to 5.

turns out that this texture unit contributes nothing to the final rendering result. Or in other words, the hardware resource is wasted.

In order to make use of the wasted texture unit, we switch textures and projective texture matrices when rendering different silhouette cones. The *texture object*, a feature introduced as early as OpenGL 1.1, allows us to load several textures at one time and switch them very quickly during rendering. This way, supporting one more input silhouette image becomes straightforward. For example, if four texture units are ready to use on a graphics card, we can render silhouette cones generated from five reference images as shown in Table 5.1.

5.3 Multi-Pass Visual Hull Rendering

Although the number of supported input images is only increased by one in the extended single-pass visual hull rendering, the idea of switching texture objects can be employed to generalize this method for accepting arbitrary number of reference images. This is accomplished by taking a multi-pass rendering approach. Assuming we have four texture units, and nine input silhouette images, the texture object and matrix switching should look like this:

However, only switching texture objects and texture matrices is not enough to implement multi-pass visual hull rendering. Within each rendering pass, the alpha values from different texture units can be multiplied by utilizing the fragment programmability of graphics hardware, exposed in the form of OpenGL

		Texture object & Projective texture matrix
Cone 1	pass1	2 3 4 5
	pass2	6 7 8 9
Cone 2	pass1	1 3 4 5
	pass2	6 7 8 9
...	...	...

Table 5.2: Switching texture objects and matrices when rendering visual hulls from nine reference images using four texture units. Each texture object or each projective texture matrix is assigned with an ID number from 1 to 9. The switching of texture objects and projective texture matrices is only shown for rendering the first two silhouette cones.

extensions such as NV_register_combiners [NVIc]. However, we still need to find a way to modulate the alpha multiplication results from different passes.

Our solution is to use the stencil test, a standard feature available on most commodity graphics cards. The stencil test operates on the stencil buffer, which associates a stencil value to each rasterized fragment. Each stencil value is compared with a reference value. The comparison result decides whether the fragment should be discarded or not. Meanwhile, the stencil value is modified according to the stencil test result as well as the depth test result.

In the OpenGL graphics pipeline, the alpha test is ahead of the depth test. Therefore, the alpha test can control the update of the stencil value indirectly through its influence on the depth test. We exploit this fact to use the stencil buffer to record the alpha multiplication results of each rendering pass. Initially, the stencil values in the stencil buffer are all cleared to 0. The alpha test is set to "GL_GREATER than 0". The depth test is set to GL_ALWAYS. During rendering, the stencil comparison is configured to test whether the stencil value equals the index number of the current rendering pass. The stencil update operation is specified in such a way that the stencil value is increased by 1 only if both the depth test and the stencil test are passed.

An example shown in Figure 5.4 justifies such settings. In the first pass, the fragment with the alpha value 1 survives the alpha test and hence passes the

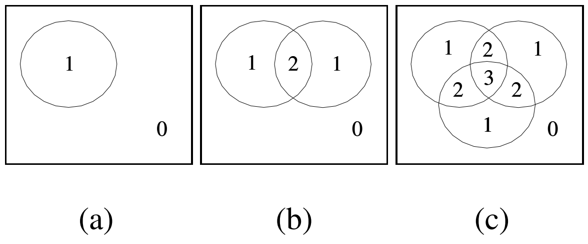


Figure 5.4: Stencil buffer update for modulating the alpha results from multiple rendering passes. The three figures show the stencil buffer after the first, second and third alpha rendering pass, respectively. The numbers are stencil values in the stencil buffer.

later depth test. According to our stencil operation setting, the stencil value is updated to 1 where the alpha value is 1 (see Figure 5.4a). In the second rendering pass, the stencil value increases to 2 only for those fragments having the stencil value 1 and the alpha value 1 (see Figure 5.4b). If this process continues, after the third rendering pass the stencil buffer will look like Figure 5.4(c). The region where the stencil values are equal to three contains the modulation result of alpha values from all three passes.

Compared with the alpha channel, the computation in the color channel is less complex. Through OpenGL framebuffer blending, we accumulate the weighted sum results from all color rendering passes. The frame buffer blending function is set to GL_ADD. Both the source and destination blending factors are set to 1. The color computation cannot be executed together with the alpha computation. The reason is that coloring each visual hull face needs the stencil mask, i.e. the alpha modulation results from all alpha rendering passes.

Since we have to use additional rendering passes for the color computation, we can take this chance to reset the stencil values modified during the alpha channel computation. This avoids clearing the whole stencil buffer after rendering each visual hull surface, which is too expensive. Notice that, for the first $(nPassNum - 1)$ passes, the regions tagged as the visual hull surface by alpha rendering passes should be kept intact in order to continue to serve as a mask for the subsequent color rendering pass. Figure 5.5 illustrates how to clear the stencil buffer generated in Figure 5.4(c). We give the pseudo code of the multi-pass

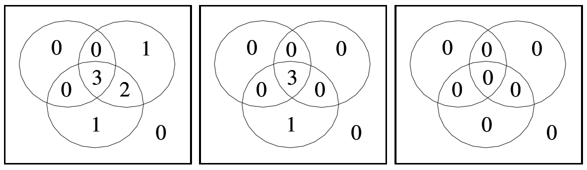


Figure 5.5: Stencil buffer clearing during the color rendering passes. The three figures show the stencil buffer after the first, second and third color rendering pass, respectively. Each color rendering pass corresponds to the alpha rendering pass used for stencil buffer update in Figure 5.4.

rendering algorithm in Figure 5.6. Figure 5.7 shows some rendering results.

The rendering passes required by either the alpha or the color computation can be determined from the number of reference views and the number of available texture units using the following formula:

$$nPassNum = \left\lceil \frac{nRefViews - 1}{nTextureUnit} \right\rceil$$

The total number of rendering passes required to render visual hulls is twice this value, namely $2 \cdot nPassNum$. However, the rendering cost is not doubled. This is because we only need to update the stencil buffer during alpha rendering passes.

5.4 System Performance

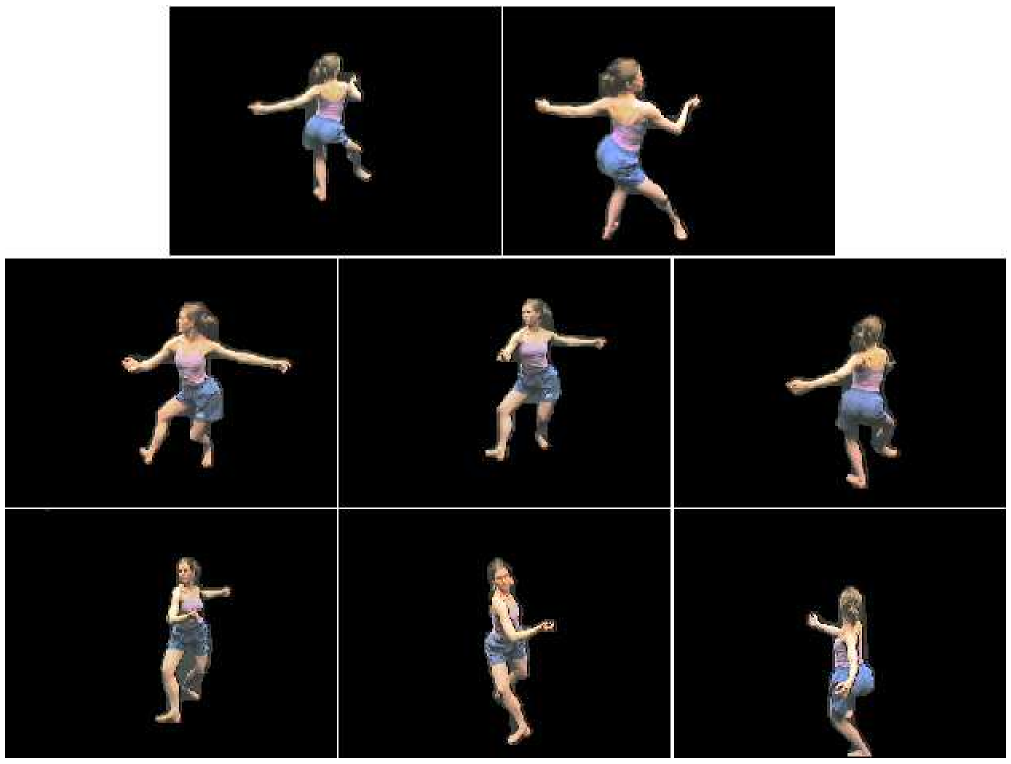
The single-pass and multi-pass visual hull rendering algorithms are implemented in the distributed system described in Chapter 3. For visual hull reconstruction, the cameras need to be arranged in a convergent way roughly along a circle (See Figure 3.1c). The video images are acquired at 320×240-pixel resolution. The server is a P4 1.7GHz dual-processor machine with a Quadro4 700 XGL (GeForce4-class) graphics card [2].

The resolution of the rendered novel view is set to 640×480 pixels. Each 2D silhouette polygon is approximated with 100 to 120 edges. For the basic

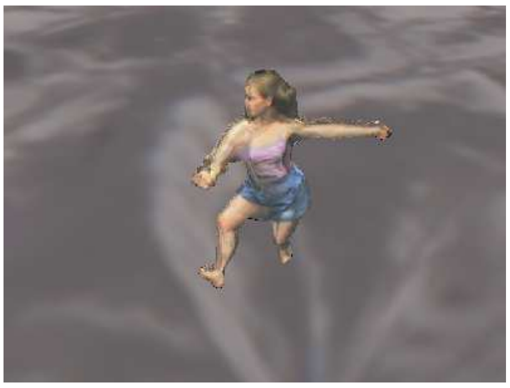
[2]Notice that in the original paper [LMS03a] we use a GeForce3 graphics card. Now we test this algorithm on the GeForce4 card and have a slightly higher performance as shown later.

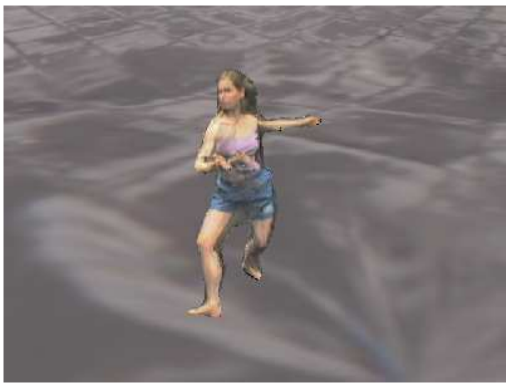
```
//Initialization and Texture setup
Clear stencil buffer
Enable stencil, depth and alpha test
Set alpha test to “GL_GREATER than 0”
Set source and destination blending factors both to 1
Set blending function to GL_ADD
Configure and enable register combiners
Compute $nPassNum$
loop $i$ over reference views
  Load silhouette image $i$ as an RGBA texture
  Name it as texture object $T_i$

Loop $i$ over reference views //start rendering
  Loop $S_i$ over silhouette cones
    Loop $\triangle_j$ over silhouette faces of $S_i$
      Compute view-dependent weights for all views except $i$

      //alpha modulation using the stencil test
      Set stencil update operation to GL_KEEP, GL_KEEP,GL_INCR
      Set depth test to GL_ALWAYS
      Disable color and depth buffer writing
      Loop $p$ over alpha computation passes
        Set stencil test to “GL_EQUAL to $p$”
        Setup texture object and projective texture matrix for each texture unit
        Draw $\triangle_j$

      //color accumulation using blending
      Set stencil test to “GL_EQUAL to $nPassNum$”
      Set depth test to GL_LEQUAL
      Enable color and depth buffer writing
      Loop $p$ over color computation passes
        Set stencil operation to GL_ZERO, GL_KEEP, GL_KEEP for
             the first $nPassNum - 1$ passes,
       Set stencil operation to GL_ZERO, GL_ZERO, GL_ZERO for the last pass
        Encode view-dependent weights in the color of each vertex of $\triangle_j$
        Setup texture object and projective texture matrix for each texture unit
        Draw $\triangle_j$
```

Figure 5.6: Multi-pass hardware-accelerated algorithm for novel view synthesis of visual hulls.



(a)





(b) (c)

Figure 5.7: Results of multi-pass visual hull rendering. **(a)** Segmented reference views. **(b)** and **(c)** Rendering results from novel viewpoints.

single-pass visual hull rendering algorithm, the experiment is performed using synchronized video sequences recorded from four views. We have achieved 86 fps for rendering textured visual hulls. A performance comparison between our system and similar systems is given in Table 5.3. From this table, we can see that our algorithm shows considerable performance improvement.

We record another set of synchronized video sequences from eight reference views for testing the performance of the extended single-pass rendering algorithm as well as the multi-pass one. In Table 5.4, we give the performance of the rendering algorithms. Notice that when we use the multi-pass algorithm to render visual hulls from eight reference views, the frame rate drops significantly compared with the basic algorithm. There are two reasons for the drop of the frame rate. First, the number of silhouette cones to be rendered is doubled. Second, four rendering passes are spent to obtain the final result. Despite more geometry primitives and more rendering passes, the frame rate is still kept at near real-time frame rates, 18 fps.

5.5 Discussion

Our visual hull reconstruction technique can be viewed as an hardware-accelerated version of the work [MBM01]. Both techniques take silhouette cones as the input data. The main difference is that we do not reconstruct a view-independent 3D model of visual hulls while they do. Since the explicit reconstruction is skipped, our algorithm is more efficient. However, the implicit geometry reconstructed by our technique is the depth map of a visual hull. It is not suitable for further geometry processing like editing and deformation.

Similar to our technique, the *image-based visual hull* technique [MBR+00] reconstructs visual hull implicitly as well. The reconstruction is performed in software in a ray-tracing manner, whereas we carry out the reconstruction using triangle rasterization, which is well accelerated by commodity graphics hardware.

Our reconstruction technique is also related to Lok's work [Lok01]. Both reconstruct visual hulls implicitly using graphics hardware rasterization. However,

	number of cameras	input image size (pixels)	processing power	frame rate (fps)
IBVH	4	256×256	quad 500MHz PC (4x600MHz PC)	8
GVE	9	640×480	unknown (9x600MHz PC)	offline (8.77)
OMR	5	720×486	SGI Reality Monster	12-15
PVH	4	320×240	dual 933MHz PC (4x600MHz PC)	30 (15)
HAVH	4	320×240	dual 1.7GHz PC (4x1.1GHz PC)	86

Table 5.3: Performance comparison. In the column *processing power*, if the system is in client-server mode, we distinguish the rendering server from the clients. The machines listed in parentheses are client PCs. For the frame rate, the number in parenthesis is the reconstruction frame rate when rendering is decoupled from reconstruction. **IBVH**: Image-Based Visual Hulls [MBR$^+$00]. **GVE**: Generation, Visualization and Editing of 3D Video [MT02]. **OMR**: Online Model Reconstruction for Interactive Virtual Environments [Lok01]. **PVH**: Polyhedral Visual Hulls for Real-Time Rendering [MBM01]. **HAVH**: Our system. Hardware-Accelerated Visual Hull Novel View Synthesis.

rendering algorithm	Basic	Extended single-pass	Extended multi-pass
number of referednce views	4	5	8
number of rendering passes	1	1	4
frame rate (fps)	86	69	18

Table 5.4: Performance of the basic and the extended algorithms for novel view synthesis from visual hulls. For the multi-pass algorithm, the rendering passes consist of two alpha passes and two color ones.

in Lok's work the 3D space is discretized with a stack of planes. Quantization artifacts arise from space tessellation. We do not have such artifacts because the input data for the visual hull reconstruction are polyhedral silhouette cones.

5.6 Summary

In this chapter we have presented a new hardware-accelerated algorithms to reconstruct and render visual hulls from multiple-view video streams in real time. The visual hull reconstruction is accomplished during rendering process and accelerated by graphics hardware. Thanks to the graphics hardware acceleration, the speed of novel view synthesis is greatly increased compared with the performance of previously reported similar systems. Furthermore, since the reconstruction is performed in the image space of a novel view, it is robust and does not suffer from numerical instabilities.

We have implemented a basic and an extended single-pass rendering algorithm as well as a multi-pass extension. All of them run in real-time for up to eight reference views. For the single-pass algorithm, we employ projective alpha map trimming and alpha texture modulation to reconstruct visual hulls. The color computation can be carried out parallel to the visual hull reconstruction. The single-pass algorithm is very fast, but has one limitation. The number of input reference images is limited by the number of texture units. If a graphics card has few texture units, the visual hull reconstruction is coarse and the rendering quality is low. Therefore, we present a multi-pass extension to address this issue. Our multi-pass rendering algorithm exploits additional stencil buffer operations to reconstruct visual hulls. It overcomes the limitation of graphics hardware resources and is able to take arbitrary number of reference views as input.

Chapter 6

Hybrid Hardware-Accelerated Novel View Synthesis of Visual Hulls

In the previous chapter, the hardware-accelerated algorithm exploits alpha map trimming to perform novel view synthesis of visual hulls. High rendering speed has been achieved. However, projective texture mapping causes aliasing artifacts between visual hull faces generated from different silhouette cones. These artifacts become especially noticeable when visual hulls are examined in a close-up view. Figure 6.1 shows such artifacts with a flat-shaded visual hull reconstructed by the alpha map trimming technique. This drawback arises from projecting discrete silhouette contours in reference views onto silhouette cones. Using high-resolution reference images only alleviates the problem, but cannot completely eliminate it.

In order to overcome the drawback, this chapter presents a hybrid approach to synthesize novel views of visual hulls. This approach combines the alpha map trimming with a hardware-accelerated CSG reconstruction technique. The alpha map trimming offers fast speed while the CSG reconstruction generates visual hulls without the aforementioned aliasing artifacts. In addition, we propose an advanced view-dependent texturing scheme which further improves the

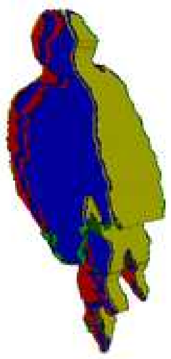


Figure 6.1: A flat-shaded visual hull shows the aliasing artifacts of the projective alpha map trimming technique. Different colors represent different silhouette cones. The visual hull is rendered from four reference images.

rendering quality by applying more accurate *per-fragment* blending weight computation.

The remainder of the chapter is organized as follows. Section 6.1 briefly describes the principle of a hardware-accelerated CSG reconstruction technique. Section 6.2 explains in detail our hybrid algorithm for synthesizing novel views of visual hulls. Section 6.3 presents the rendering performance. Finally, we summarize the chapter in Section 6.4.

6.1 Hardware-Accelerated CSG Reconstruction

Hardware-accelerated CSG reconstruction methods [Wie96, SLJ98, GKMV03] exploit the stencil buffer to perform 3D Boolean operations. Since no projective texture mapping is involved, this class of methods do not suffer from the aliasing artifacts.

Generally, CSG operations include union, intersection and difference. For visual hull reconstruction, the relevant operation is intersection and the intersection primitives are silhouette cones. Therefore, we are only interested in CSG intersection operation. The basic idea of hardware-accelerated CSG intersection is counting front- and back-facing fragments with the help of the stencil

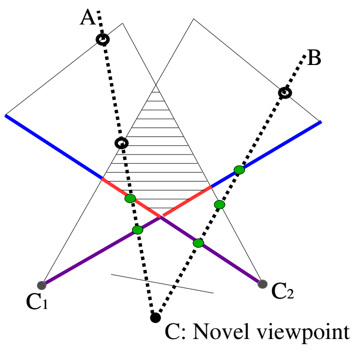


Figure 6.2: Principle of hardware-accelerated CSG reconstruction shown in 2D. Two silhouette cones extruded from C_1 and C_2 are intersected. The intersection region is filled with horizontal lines. The first depth layer is drawn in purple and the second layer is colored in red and blue. The red parts are the intersection result rendered from the novel viewpoint. The hollow circles are fragments behind the second layer. The green circles are fragments in front of or on the second layer.

buffer. A method proposed by Guha *et al.* [GKMV03] is adopted by us. We will describe this method in detail.

Given a novel viewpoint, all depth layers of front faces are traversed from front to back relative to this viewpoint. For each layer, the depth test is set to “less or equal”. Then the front faces of all silhouette cones are rasterized. If a fragment passes the depth test, the corresponding stencil value is *increased* by 1. When back faces are rendered, the stencil operation is reversed. Namely, the stencil value is *decreased* by 1 if a fragment survives the depth test. After all depth layers are traversed, only those fragments in the intersection parts are marked with the stencil values equal to the total number of the objects. In the end, a complete depth map of the visual hull for the current novel viewpoint is generated. As an example, the intersection of two silhouette cones is illustrated in 2D using Figure 6.2. For the second depth layer, two front-facing fragments pass the depth test along the viewing ray CA, and hence yield a stencil value of 2 (equal to the number of the silhouette cones). For the viewing ray CB, two front-facing fragments and one back-facing fragment pass the depth test and produce a stencil value of 1.

Most earlier CSG reconstruction methods [Wie96, SLJ98] need to copy depth buffers to main memory. The operation is typically slow and becomes the main performance bottleneck. The method proposed by Guha *et al.* [GKMV03] utilizes *depth peeling* [Eve02] for depth layer traversal. The *depth peeling* technique copies depth buffers to texture memory, which is an on-board operation and can be performed much faster. In spite of the performance improvement, its speed is still far below that of the alpha map trimming. The reason is that the minimum number of rendering passes required by CSG reconstruction is equal to the number of silhouette cones (denoted as n) for visual hull reconstruction. This means all silhouette cones must be rendered at least n times. In contrast, the single-pass alpha map trimming technique renders all silhouette cones only once, and therefore is n times faster than the CSG reconstruction. Even with the multi-pass alpha map trimming, the rendering speed is still k times faster, where k is the number of texture units.

6.2 Hybrid Visual Hull Rendering

Aiming at attaining good quality of the CSG reconstruction while keeping fast speed of the alpha map trimming, we design a hybrid approach combining the strengths of both to synthesize novel views of visual hulls. The whole process consists of four steps (Figure 6.3) which are described in the rest of this section.

6.2.1 Valid region determination

When we apply the hardware-accelerated CSG reconstruction technique, polygons of silhouette cones have to be rasterized multiple times. These polygons are usually very large and consume a huge amount of fill-rate capability of graphics hardware. On the other hand, the intersected results, i.e. visual hulls, only occupy partial region of the output window. We call the region a *valid region*. If we know this region *a priori*, the fill-rate can be greatly reduced by applying the scissor test. In addition, another operation could benefit from predicting the valid region. During CSG reconstruction, the *depth peeling* technique

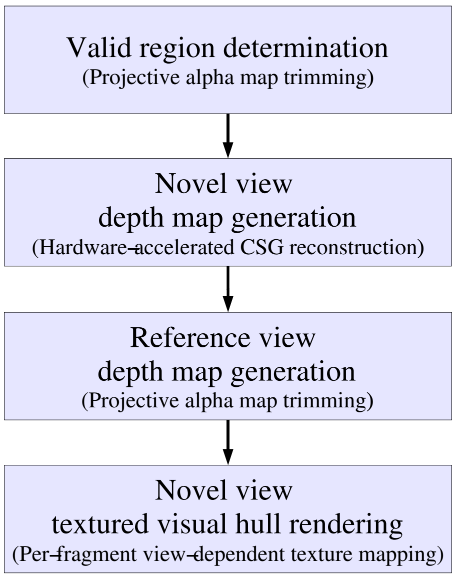

Valid region determination
(Projective alpha map trimming)
Novel view
depth map generation
(Hardware-accelerated CSG reconstruction)
Reference view
depth map generation
(Projective alpha map trimming)
Novel view
textured visual hull rendering
(Per-fragment view-dependent texture mapping)

Figure 6.3: Workflow of our hybrid approach. The texts in parentheses indicate the technique utilized in each rendering step.

requires frame-buffer-to-texture copying operations when stepping through all depth layers. With the knowledge of the valid region, the copying operations can be carried out on this region only instead of the whole frame buffer.

For the determination of the valid region, we employ the projective alpha map trimming technique to render a visual hull in a very small off-screen window (e.g. 80×60 pixels). Writing to the color buffer and to the depth buffer are both disabled. The stencil buffer is enabled and only the pixels covered by the visual hull are written. The content of the stencil buffer is read back to the main memory, and a rectangular region enclosing the visual hull is calculated. This region is expanded by one pixel in order to tolerate some rounding errors introduced by rendering a smaller version of the visual hull. Finally, the region is scaled by the size ratio between the actual novel viewport and the small off-screen window.

Notice that the aliasing artifacts of the alpha map trimming do not have any effect on the valid region determination. One concern is that some thin features on the visual hull might be lost when we render down-scaled version of visual hulls. In this case, we should choose the size of the off-screen window more carefully. Although buffer readback operations are generally expensive, reading the stencil buffer of a small window does not cost much.

6.2.2 Novel view depth map generation

Once the valid region has been identified, silhouette cones can be rasterized in the constrained area to perform the CSG reconstruction. The result is the depth map of a visual hull in the novel view. This depth map guarantees that the final visual hull rendering is free of aliasing artifacts. Although this rendering step only produces depth maps, we present a final rendering quality comparison in Figure 6.4 to demonstrate that the CSG reconstruction method is able to generate significantly better rendering results compared with the alpha map trimming method.

The original CSG method used by Guha *et al.* performs well for visual hull reconstruction as long as novel viewpoints are outside of any silhouette cone. Once a novel viewpoint falls inside a silhouette cone, parts of the silhouette cone's faces will be clipped by the near plane of the view volume. This leads to wrong face counting and incorrect rendering results. Such limitation is undesirable in the context of novel view synthesis of visual hulls.

To solve this problem, we come up with an idea inspired by Carmack's *zfail* shadow volume algorithm [Car00]. Instead of counting front- and back-facing fragments between the novel viewpoint and the first visible fragment, we examine those fragments between *infinity* and the first fragment. For example, in Figure 6.2, such fragments are depicted as hollow circles. Accordingly, the updating strategy for the stencil buffer is also changed. A stencil value is decreased for front faces or increased for back faces when the depth test *fails*. Since we can place the far plane of the view volume very distant to the viewpoint, all *zfail* fragments will be counted. This way, correct counting results can be achieved without worrying about the near clipping plane. Apart from this improvement, we provide two other performance enhancements to the original CSG reconstruction method, terminating depth traversal using hardware-accelerated occlusion tests [HP] and reducing the number of rendering passes with two-sided stencil operations [NVIa].

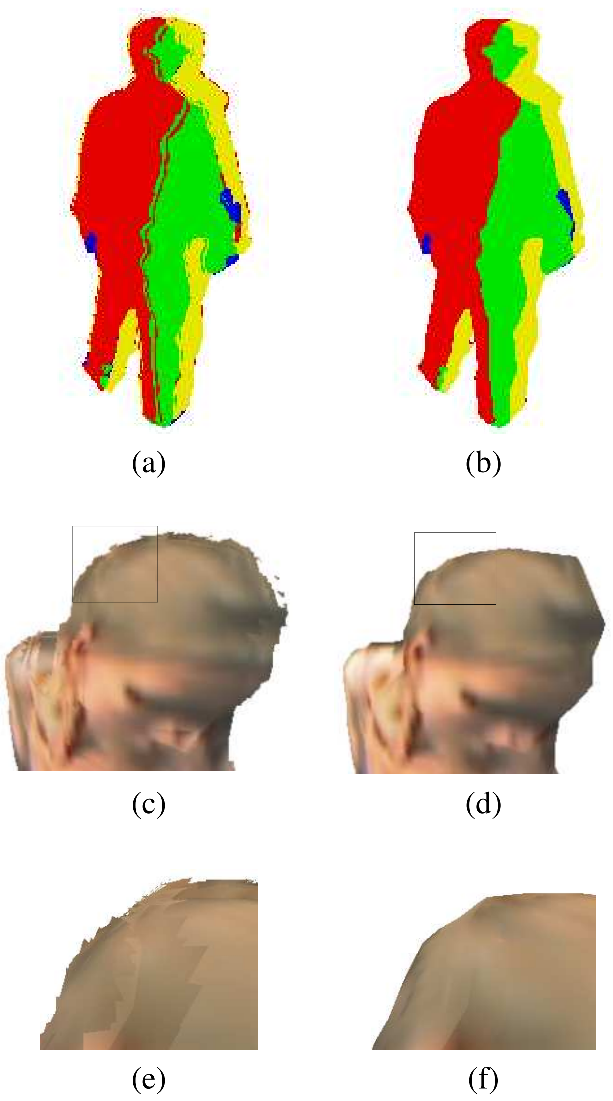


Figure 6.4: Rendering quality comparison between visual hulls reconstructed by the projective alpha map trimming and the hardware-accelerated CSG reconstruction techniques. For (a), (c) and (e), the alpha map trimming technique is employed, whereas for (b), (d) and (f), the CSG reconstruction technique is applied. (a) and (b) are rendered in flat-shaded style. One can clearly observe the inter-penetrations along intersection boundaries. (c)(d)(e)(f) show textured visual hulls. (e) and (f) are close views of the boxed regions in (c) and (d), respectively.

6.2.3 Reference view depth map generation

Visibilities with respect to reference views are critical to multi-view texture mapping. For those parts that are invisible in a reference view, the corresponding color information should be neglected when blending multiple textures. This issue can be addressed in object space. In previous chapter, we use the normal vectors to compute the reference view visibilities of silhouette surfaces. When we render a surface invisible in a reference view, the visibility information is encoded in the weighting factor corresponding to this view. This approach works fine if the reconstructed visual hull is convex, but it fails for concave surface regions. To overcome this limitation, Debevec *et al.* split triangles of objects so that each triangle is either fully visible or fully invisible to any reference view [DBY98]. However, this process takes a long time even for a moderately complex object.

As an alternative, the visibility issue can be addressed in image space using the *shadow mapping* technique [Wil78]. We adopt this technique since it has been implemented on common graphics hardware [SKvW+92, Hei99] and is very suitable for real-time applications. In order to solve the visibility problem for all reference views, we generate a depth map of the visual hull for each view. The rendering can be performed by either the projective alpha map trimming or the CSG reconstruction technique. We choose the former one because of its considerably higher speed. Although depth maps produced by the alpha map trimming have aliasing artifacts and the quality is not as good as that produced by the CSG reconstruction technique, in practice, due to smooth blending of multiple textures, the artifacts introduced by the quality difference of depth maps are hard to notice in the final rendering result.

6.2.4 Textured visual hull rendering

Since the depth map of a visual hull has already been created for the novel view as described in the section 6.2.2, we render the textured visual hulls by enabling the “equal” depth test and rasterizing all silhouette cones using the multi-view texture mapping. Note that when a silhouette cone is textured, the reference view corresponding to this cone should not be used because it would project texels along silhouette contours throughout the whole surfaces of the cone.

6.2.4.1 View-dependent multi-view texture mapping

The essential idea of the multi-view texture mapping is the same as that expressed in Equation 5.2 in the previous chapter. We represent a fragment's color C as a convex combination of the texture values in the reference views:

$$C = \left[\sum_{k=1}^{N} W_k \cdot T_k \right] / \sum_{k=1}^{N} W_k \, , \tag{6.1}$$

where N is the number of reference views, T_k and W_k are the texture color and its associated weighting factor for the k-th reference image, respectively.

The algorithms presented in the previous chapter target graphics hardware with preliminary fragment programmability (e.g. NVIDIA GeForce2/3/4, ATI Radeon 8500). As a result, in Equation 5.2, W_k only consists of two components: the face visibility factor and the view-dependent weight factor. Tricks are needed to fit the weight blending computation into a set of register combiner stages. The rendering quality is limited. Advanced graphics cards, such as NVIDIA GeForce FX and ATI Radeon 9700/9800/X800, provide more flexible fragment programmability, which allow us to perform more complex blending computation. In our new per-fragment view-dependent texture mapping scheme, we decompose weight W_k into four components:

$$W_k = V_k \cdot W_{kf} \cdot W_{ks} \cdot W_{kd} \, . \tag{6.2}$$

The first component V_k is the visibility function with respect to the reference view k. Since the depth map corresponding to the view k has been created (see Section 6.2.3), V_k can be easily determined by the standard shadow mapping algorithm [SKvW$^+$92].

The weight W_{kf} is fetched from a featuring map which helps to eliminate seams arising from sampling different reference views at silhouette boundaries [DTM96, SS97, PCD$^+$97]. The feathering map is generated by applying the

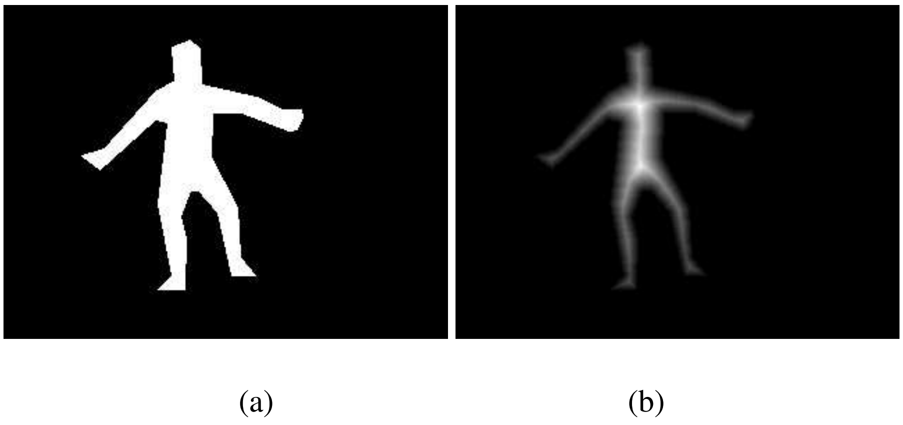


Figure 6.5: Distance transformation. **(a)** Original foreground object mask. **(b)** Foreground object mask after applying distance transformation. It looks thinner because the pixels close to the contour are too dark to be seen.

distance transformation [Bor86] on a binary silhouette mask. It is then stored in the corresponding alpha channel. For each pixel in the silhouette mask, the distance transformation calculates the distance from the pixel to the nearest zero-value pixel. Mathematically, it can be expressed in the following formula:

$$W(p) = min\{dist\,(p,q)\,,\, q \in R\}\,,$$

where p and q are pixel positions, R stands for the region where the pixel values are zeros, $dist$ denotes the distance between p and q. Figure 6.5 gives an example of the effect of the distance transformation. Notice that different silhouette masks have different maximum distance values. To ensure the distance values in different silhouette masks have the same range (in our case 0 – 255), an additional scaling operation is required.

W_{ks} is referred to as the surface obliqueness weight. It penalizes surfaces that are oblique when observed from a reference viewpoint. We use the following definition:

$$W_{ks} = \left[\max(\tilde{d}_k \bullet n\,,\, 0)\right]^{\alpha}\,. \tag{6.3}$$

For a point p that is associated with the fragment under consideration, the vector $\tilde{d}_k$ is the viewing direction from p toward the k-th reference viewpoint. The vector n denotes the normal vector of the point p. When the angle between $\tilde{d}_k$ and n is greater than 90°, their dot-product becomes negative. This means that we are processing a back-facing fragment with respect to the view k and should set the weight W_{ks} to zero. The *max* function in Equation 6.3 serves for this purpose. The constant α in the equation is a tunable parameter, which emphasizes a larger weight.

The last weighting component W_{kd} is dependent on the novel viewpoint. It gives a higher weight to the fragment whose reference viewing direction $\tilde{d}_k$ is closer to its novel viewing direction $\tilde{d}$. The weight function is written as:

$$W_{kd} = (\tilde{d}_k \bullet \tilde{d} + 1)^{\beta}. \tag{6.4}$$

The constant β plays a similar role as the constant α in Equation 6.3. In order to obtain a convex combination for the final pixel color, we add one to the dot-product of $\tilde{d}_k$ and $\tilde{d}$ so that W_{kd} is guaranteed to be non-negative. This view-dependent weight can be represented as a function of the angle between $\tilde{d}_k$ and $\tilde{d}$. However, computing the angle requires an inverse cosine operation, which is costly on current graphics hardware.

6.2.4.2 Per-fragment blending weight computation

In order to compute the weighting factors W_{ks} and W_{kd} accurately, the 3D coordinate associated with each fragment must be used to determine the viewing vectors $\tilde{d}_k$ and $\tilde{d}$. In previous chapter, due to graphics hardware constraints, the principal axes of reference views and novel views are used to approximate these two vectors. This approximation is only valid when the object in the scene is relatively small. Other previous view-dependent texture mapping methods [PCD+97, DBY98, PHL+98, BBM+01] compute the blending weights for each vertex of the object and then perform linear interpolation across surfaces to obtain the blending weights for each fragment. However, hardware-accelerated visual hull rendering methods do not produce vertex information of visual hulls. The only available geometry entities are silhouette cones. In this case, the

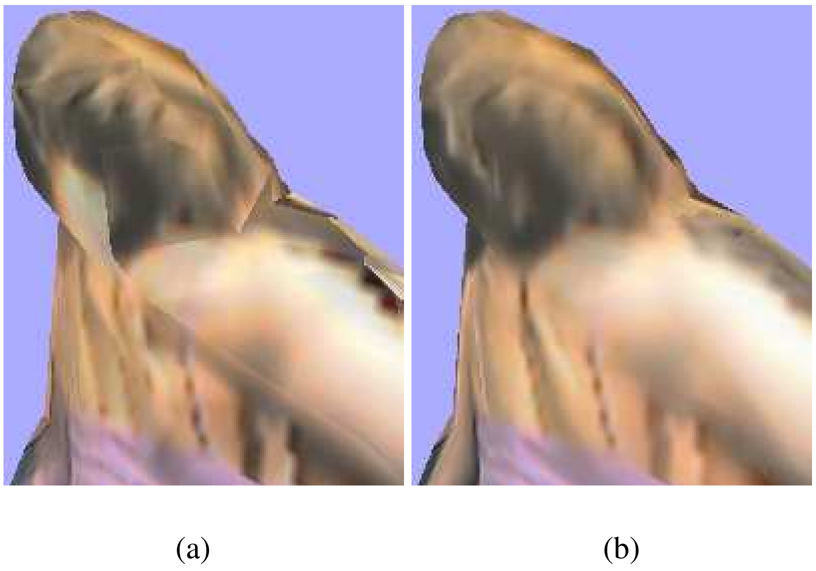


Figure 6.6: Comparison of rendering quality between per-vertex and per-fragment blending weight computations. (a) and (b) are rendered from the same viewpoint. **(a)** Per-vertex weight computation. **(b)** Per-fragment weight computation. Rendering artifacts are suppressed considerably because of more accurate weight computation.

weight computation based on a per-vertex basis has large errors and causes noticeable rendering artifacts.

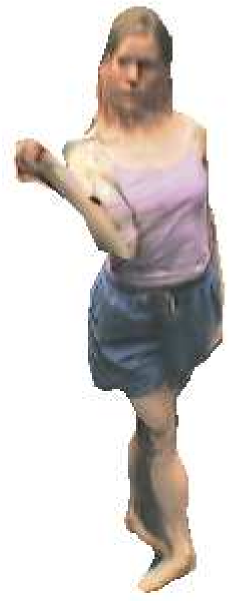
Fortunately, the vertex and fragment programmability [Opec, Opea] of recent graphics hardware can assist us in calculating these weights on a per-fragment basis. With the more accurate weights, we achieve smooth texture blending on the objects and continuous texture transitions when changing novel viewpoints. Our per-fragment view-dependent texture mapping procedure is detailed as follows. First, 3D coordinates of silhouette cone vertices and normal vectors of silhouette cone faces are passed to a vertex program [Opec], through which a 3D coordinate together with a normal vector is generated for each fragment by the graphics rasterizer. Then, a fragment program [Opea] takes these inputs to compute the per-fragment weights W_{ks} and W_{kd}. Both constants α and β are set to five in our implementation. The viewing vector normalizations in Equation 6.3 and 6.4 are replaced by *normalization cube map* [WD] lookups for acceleration purpose. In the same fragment program, V_k is computed from the depth map created for the k-th reference view, while W_{kf} and T_k are sampled from the alpha and color channels of the reference view k, respectively. Once all weighting factors are available, the final fragment color is evaluated with Equation 6.1. We compare the rendering results using per-vertex and per-fragment weight computations in Figure 6.6, and demonstrate the advantage of using per-fragment computation. More snapshots of the novel view synthesis results of visual hulls generated from both real video streams and synthetic animation sequences are presented in Figure 6.7.

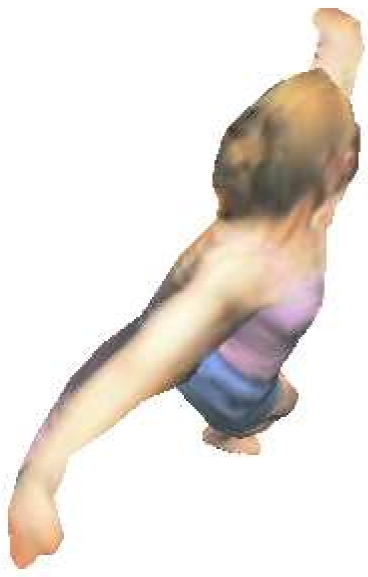
6.3 System Performance

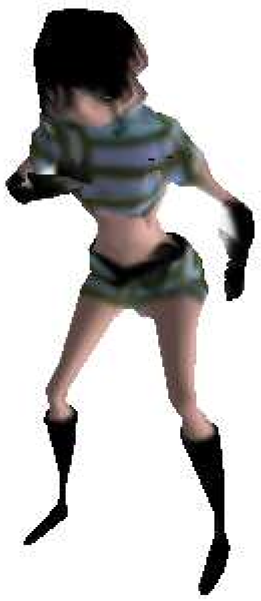
The hybrid algorithm for synthesizing novel views of visual hulls has been integrated into the distributed system described in Chapter 3. The client computers are responsible for synchronized video acquisition as well as real-time image processing tasks, such as foreground/background segmentation, silhouette contour extraction and distance transformation. The server is a P4 1.7GHz processor machine and carries out the hybrid rendering algorithm on a GeForce FX 5800 Ultra graphics card.

The high-level shading language Cg [MGAK03] is employed to implement the per-fragment view-dependent blending scheme presented in Section 6.2.4. Our Cg code can be compiled into low-level shaders suitable for different graphics hardware and different graphics APIs (e.g. OpenGL, Direct3D).

We set the resolution of the rendered novel view to 640×480 pixels and execute the hybrid rendering algorithm using various numbers of reference views from one common dataset. The average number of polygons of each silhouette cone is about 50. The valid region occupies about one third of the entire viewport. Under this setting, we measure the rendering performances and show the results in Table 6.1. For eight reference views, we also render visual hulls with pure alpha mapping trimming and CSG reconstruction techniques. The frame rates are 7.5 fps and 1.7 fps, respectively. If valid region determination is switched on, we achieve 2.9 fps when the CSG reconstruction is used for depth


(a)


(b)


(c)

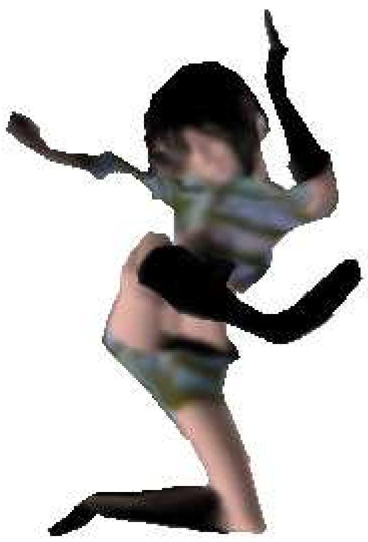

(d)

Figure 6.7: Novel views of visual hulls. All images are generated from eight reference views. Each view is rendered at least 20° away from the closest reference view. Real video streams are used to render (a) and (b). Synthetic animation sequences are employed to create (c) and (d) .

# Reference Views	Rendering Performance(fps)
2	48.4
3	27.0
4	13.2
5	8.8
6	6.4
7	4.7
8	4.0

Table 6.1: Performance of our hybrid hardware-accelerated algorithm for novel view synthesis of visual hulls.

map generation both for reference views and the novel view.

Furthermore, for the hybrid approach, we compare the performances measured with and without the valid region determination. Table 6.2 presents the timings of individual rendering steps for both situations. Notice that the rendering speed is approximately doubled when employing the valid region determination because both hardware-accelerated CSG reconstruction and final view-dependent texture mapping profit from the scissor test. For depth map rendering in reference views, using the alpha map trimming technique, it takes 51 ms to generate eight depth maps at 320×240-pixel resolution. If the CSG method is employed, it takes about 130 ms according to our measurement. Obviously, the alpha map trimming technique performs better. Another characteristic of our rendering algorithm is that more graphics power is consumed at the fragment processing stage. This suggests that the geometric complexity of silhouette cones does not affect overall performance very much. In our tests, if the number of silhouette faces is increased by a factor of two, the rendering speed drops by about 10%.

6.4 Summary

In this chapter we have presented a hybrid hardware-accelerated algorithm to render high-quality visual hulls from multiple reference views. The performance ranges from interactive to real-time frame rates depending on the number of ref-

	Timings (ms)	
	w/ valid region determination	w/o valid region determination
Valid region determination	5	N/A
Novel view depth map generation	40	115
Reference view depth map generation	51	51
View-dependent texture mapping	96	300
Texture loading and miscellaneous	58	58
Total	250	524
Overall performance	4.0 fps	1.9 fps

Table 6.2: Comparison of the rendering performances that are obtained with and without the valid region determination. Eight reference views are used. Timings of individual rendering steps are presented.

erence views. Compared with analytical intersection methods, our algorithm works robustly for highly complex objects since all intersections are performed in image space. The two techniques that form the basis of our algorithm, projective alpha map trimming and hardware-accelerated CSG reconstruction, benefit from each other. The inherent discretization problem limiting the alpha map trimming method is elegantly solved, and the slow performance of the CSG reconstruction method is accelerated. Additionally, rendering quality of textured visual hulls has been improved with per-fragment view-dependent texture mapping by taking advantage of programmability of more advanced graphics hardware.

Chapter 7

Hardware-Accelerated Novel View Synthesis of Photo Hulls

In Chapter 5 and Chapter 6, we use visual hulls to represent dynamic scene objects for novel view synthesis. Although the algorithms run at interactive or real-time frame rates, the 3D reconstruction has an inherent limitation, i.e. certain concave parts of objects cannot be reconstructed. This limitation leads to rendering artifacts in synthesized novel views.

To overcome the limitation, the *shape-from-photo-consistency* approach exploits color information in reference views to perform more accurate 3D reconstruction. The reconstruction results are known as *photo hulls* [KS99]. Although photo hulls generally provide better rendering results, the performances of photo hull reconstruction algorithms are rather slow. This chapter addresses the performance issue and presents an efficient hardware-accelerated algorithm to synthesize novel views of photo hulls. The algorithm runs on off-the-shelf graphics hardware at interactive frame rates. Compared with a purely software-based implementation, the performance is approximately seven times faster.

We briefly describe our algorithm in Section 7.1. In the subsequent three sections the algorithm is explained in detail. After giving the performance measurements, we conclude this chapter.

7.1 Algorithm Overview

Traditionally, the view synthesis problem is treated as a two-step process. First, explicit 3D models of objects are reconstructed from reference views. Then a novel view is synthesized by projecting the 3D models onto the image plane of the view. Unlike the traditional way, we do not explicitly reconstruct complete photo hulls of 3D objects. Instead, reconstruction is performed implicitly and embedded in rendering process. This is achieved by adopting a view-dependent plane-sweeping strategy to render a stack of planes directly into the novel view. During rendering, photo-consistency is checked on the fly using graphics hardware, and photo hulls are reconstructed in the form of depth maps.

The algorithm runs as follows. First, to reduce rasterization cost, we compute an effective bounding volume of the object under consideration. This volume is discretized into a set of slicing planes parallel to the image plane of the novel view. Then the planes are processed in a front-to-back order. While each plane is rendered, we determine a set of active reference views and project them onto the plane. The photo-consistency of each rasterized fragment is checked by exploiting the programmability of graphics hardware. Visibility masks associated with the active reference views are maintained on the graphics board and play a critical role during photo-consistency checks. For display, the color of a photo-consistent fragment is evaluated as the weighted average of visible samples in the active reference views, and photo-inconsistent fragments are discarded. With the help of the photo-consistency information of each fragment, we are able to update the visibility masks that are used to process the next slicing plane. The outline of our novel view synthesis algorithm is given in pseudo-code in Figure 7.1.

Our algorithm is most relevant to the following two photo hull rendering algorithms. Slabaugh *et al.* [SSH03] step along each viewing ray of the novel view and find for each ray a sample that is photo-consistent with reference views. All the photo-consistency and color computations are performed on CPUs and are relatively slow. We share the same strategy with the work presented by Yang *et al.* [YWB02], namely, rendering a stack of planes and exploiting graph-

```
Generate slicing planes depending on the novel view
foreach slicing plane $\mathcal{H}$
  Render $\mathcal{H}$ with multi-view texture mapping and photo-consistency check
  Update visibility masks for active reference views
end foreach
```

Figure 7.1: Outline of our hardware-accelerated algorithm to synthesize novel views of photo hulls.

ics hardware to perform computations. However, the photo-consistency check in their work only uses a simple criterion due to graphics hardware limitation. With more advance graphics hardware, we are able to apply more robust photo-consistency check. Another limitation of their algorithm is that visibilities with respect to reference views are completely ignored during the consistency check. To minimize the reconstruction error caused by this limitation, they require all reference views should be very close to each other. This imposes severe restrictions on the freedom of choosing novel viewpoints. Our algorithm is not only fully hardware-accelerated, but also takes the visibility issue into consideration. Table 7.1 summarizes the most relevant work and compares them with our new algorithm.

7.2 Slicing Plane Generation

We are interested in synthesizing novel views of a dynamic scene object. The 3D location of the object is unknown. To ensure full coverage of the object, a great number of planes has to be rasterized along the novel viewing direction, each of them covering the whole output window. This kind of space discretization quickly exhausts the fill rate capability of graphics hardware. Therefore, it is highly desirable to generate slicing planes within a pre-determined bounding volume of the object.

By specifying a fixed bounding box, one can compute a set of planes by stepping through this box along the novel viewing direction. However, this solution is not optimal since a fixed bounding box is typically too conservative. We provide a better solution by computing an *effective bounding volume* us-

	hardware-accelerated	considering visibility in photo-consistency check
CBSR	√	×
IBPH	×	√
HAPH	√	√

Table 7.1: Comparison of most relevant work. **CBSR**: Real-Time Consensus-Based Scene Reconstruction [YWB02]. **IBPH**:Image-Based Photo Hulls [SSH03]. **HAPH**, our algorithm: Hardware-Accelerated Novel View Synthesis of Photo Hulls

ing the visual hull, which bounds the actual object more tightly than the fixed bounding box. The effective volume is dynamically adjusted each time a novel view is synthesized. This volume remains conservative since the visual hull is a superset of the photo hull. This property guarantees that no geometry will be missed when rendering the photo hull.

To determine the effective bounding volume, a depth map of the visual hull is created for the novel view. From this depth map, we compute both the visual hull's bounding rectangle on the image plane and its depth range in the principal axis of the novel view. Then the effective bounding volume can be constructed by using the bounding rectangle and the depth range. Note that the depth range does not span the entire visual hull. It covers only the visible parts of the visual hull. Thus, we avoid discretizing space of occluded parts that contribute nothing to the final rendering result.

To create the depth map of the visual hull, we extract silhouette contours of the foreground object and employ the projective alpha map trimming technique (see Section 5) to render the visual hull in a small off-screen window (e.g. 80×60 pixels). The depth buffer is read back to main memory in floating point format, and a rectangular region enclosing the visual hull is calculated. We expand this bounding rectangle by one pixel in order to tolerate rounding errors introduced by rendering a down-sampled visual hull. The expanded rectangle is then scaled by the size ratio between the actual novel view and the small off-screen window. Meanwhile, the minimum and maximum depth values are determined from the depth map. Since we adopt the OpenGL graphics API, the

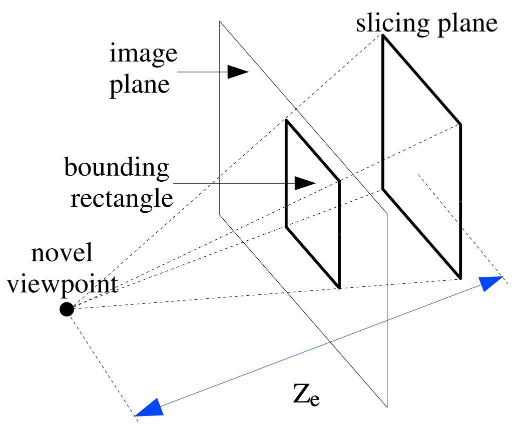


Figure 7.2: Slicing plane generation. Given the bounding rectangle enclosing the visual hull, and the distance Z_e along the viewing direction, the slicing plane can be directly generated .

depth values must be converted from window space to eye space using Equation 4.5.

Once we know the depth range in eye space, we discretize the continuous depth within this range. For each discretized depth value, its corresponding slicing plane can be directly derived given the bounding rectangle on the image plane and the novel viewing parameters. This is illustrated in Figure 7.2. To further tighten the bounding volume, we divide the depth range into several subranges and compute a smaller bounding rectangle for each of them.

Both visual hull rendering and depth buffer reading do not take much time because the off-screen window is very small. In our tests, this step amounts to only 2% of total rendering time. This cost greatly pays off later in our rendering algorithm.

The idea for reducing rasterization costs is very similar to the valid region determination used in previous chapter. However, there are two differences. Fist, effective bounding volume construction requires not only a bounding rectangle but also a depth range. Second, the ways to utilize the bounding information are different. Valid regions are used to set the scissor test, whereas effective bounding volume are used to generate slicing planes.

7.3 Slicing Plane Rendering

Since we reconstruct photo hulls implicitly, there is no explicit voxel representation existing in traditional photo hull reconstruction algorithms. The 3D space is discretized into a set of slicing planes, which are sent through the graphics pipeline, transformed by the target viewing parameters, and rasterized into fragments. These fragments actually correspond to discrete 3D points in space. Therefore, we can check the photo-consistency of each fragment and decide whether to draw the fragment in the output frame buffer. This way, photo hull reconstruction and rendering are combined into a single process.

To perform the photo-consistency check for a fragment, we need to locate those corresponding visible samples in the reference views. Notice that it is not necessary to process all reference views. Those views behind the frontmost slicing plane can be simply excluded. The remaining views are called *active reference views*, in which a 3D point on a slicing plane is more likely visible. We make use of projective texture mapping to sample the pixels in active views. Texture matrices are set up to match the viewing parameters associated with the reference images. To account for visibilities, a visibility map associated with each active view is kept on the graphics board. The visibility maps indicate which color pixels should participate in the photo-consistency check as well as the output color composition. Each visibility map coincides with its associated active view and shares the same set of texture coordinates. Figure 7.3 gives an example of the photo-consistency check as well as the visibility issue involved in the process.

We choose the variance of corresponding visible pixel samples in active reference views as the photo-consistency metric. Yang *et al.* [YWB02] approximate the variance using the *sum-of-squared-difference* (SSD). Unlike their method, we compute the true variance value, giving more reliable results. In the fragment program the variance σ^2 is computed as follows:

$$\sigma^2 = \left[\sum_{k=1}^{M} (R_k - \bar{R})^2 + \sum_{k=1}^{M} (G_k - \bar{G})^2 + \sum_{k=1}^{M} (B_k - \bar{B})^2 \right] / M \, ,$$

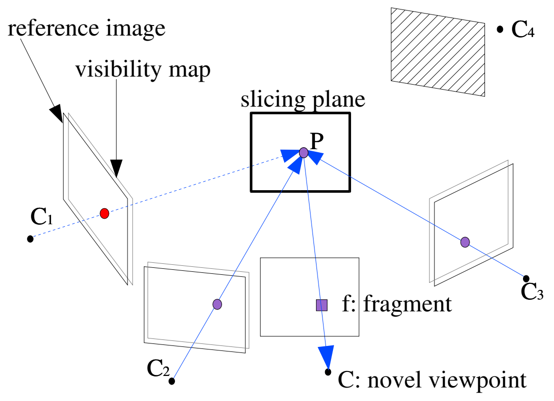


Figure 7.3: Photo-consistency check of a fragment. C_1, C_2, C_3 and C_4 are reference viewpoints. The views corresponding to C_1, C_2, C_3 are active views. The dotted frame attached to each of them represents a visibility map. Each reference view is associated with a visibility map. From C_1, the point P is occluded according to its visibility map, and the red pixel in C_1 is not used for the photo-consistency check. The fragment f is photo-consistent because the corresponding pixels in C_2 and C_3 have the same color.

where M is the number of those active views in which the 3D point associated with the fragment is visible, (R_k, G_k, B_k) is the sampled pixel color from the k-th view, and $(\bar{R}, \bar{G}, \bar{B})$ is the mean color of the corresponding pixels in all M views. Once we know the variance, the photo-consistency can be expressed as a threshold function:

$$photo\text{-}consistency = \begin{cases} 1, & \sigma^2 < T \\ 0, & \text{otherwise} \end{cases},$$

where T is a user-defined threshold. Currently, the variance computation is based on a single sample from each active reference view. Therefore, calibration errors and image noise can introduce instabilities to the photo-consistency check process. Incorporating local neighborhood information will provide more robust reconstruction results. The mipmapping technique utilized in [YP03] could be adopted in this context.

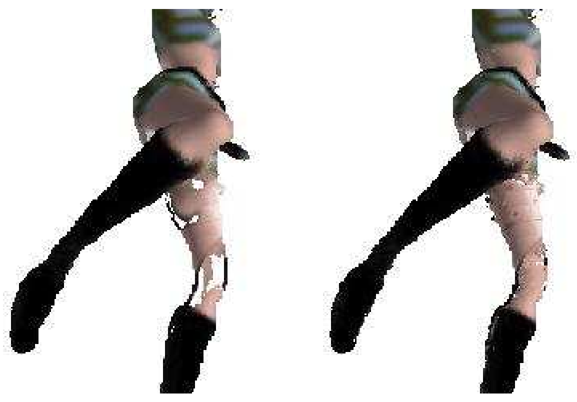
Uncertainties of photo-consistency checks could happen to the pixels close to the silhouette contours in reference images. Such pixels should not contribute to the decision on photo-consistency because they often represent mixtures of the foreground and the background pixels. In order to leave them out, we perform distance transformation [Bor86] and compute a feathering map for each reference view. This map is stored in the alpha channel. The values in the map are one in the central part of the foreground object mask and drop gradually to zero at its boundary. By setting a small threshold, we prevent the pixels near the boundary from being sampled.

Feathering maps are also employed to perform the silhouette-consistency check, which is useful for rejecting most inconsistent fragments efficiently. In a feathering map, the value zero represents scene background, whereas non-zero values indicate the foreground object. A fragment is silhouette-consistent only if the alpha values sampled from all reference views are greater than zero. This check is performed before the photo-consistency check in the same fragment program. It amounts to the visual hull computation as described in [Lok01].

Finally, if a fragment passes both the silhouette-consistency and the photo-consistency check, RGB color values are assigned to the fragment using the following formula:

$$(R,G,B) = \frac{\sum_{k=1}^{M} W_{kf} * (R_k, G_k, B_k)}{\sum_{k=1}^{N} W_{kf}},$$

where W_{kf} is the sampled alpha value of the k-th feathering map. This weight eliminates the seam artifacts between different textures projected on the slicing plane. If view-dependent effect is desired, the view-dependent weighting factor like the one used in the previous chapter could be incorporated in the color computation. Parallel to processing the color channels, we set the alpha channel of the output frame buffer to 1 for the fragments passing the photo-consistency test. This information will be used for updating the visibility maps in the next step.



(a) (b)

Figure 7.4: Influence of visibility maps in photo hull rendering. Eight reference views are used. **(a)** Without considering visibilities with respect to reference views, black colors from the raised leg participate in the photo-consistency check. Therefore, some regions on the thigh of the other leg fail to be reconstructed. **(b)** This artifacts are removed when visibility maps are employed.

7.4 Visibility Map Updating

Initially, all visibility maps contain the value zero, which suggests that the first slicing plane is visible for all pixels in the active reference views. While we proceed to render each slicing plane from front to back, implicit geometric information of the object is produced in the form of alpha maps in the novel view. This information must be reflected in the visibility maps in order to be able to check the photo-consistency and to composite final colors correctly. To show the importance of the visibility maps, we compare the rendering results obtained with and without visibility maps in Figure 7.4.

The algorithm for updating visibility maps is as follows:

1. Copy the alpha channel of the novel view to a texture T_a.
   Since a bounding rectangle is determined in the subsection 7.2, the copying operation only needs to be carried out for the area of the bounding rectangle which is usually much smaller than the whole output window.

2. Render the current slicing plane for each active reference view using the

texture T_a.
The rendering is performed in an off-screen buffer on the graphics card and does not interfere with the frame buffer of the novel view. The current slicing plane, together with the texture T_a, represents one layer of the photo hull in object space. Therefore, by projecting this layer to a reference view, we are able to generate the occlusion information for the planes behind the current one. We divide off-screen buffer into rectangular tiles and assign them to different reference views. For each active reference view, the viewport is set to its corresponding tile. Then the slicing plane is rendered with the texture T_a. The visibility maps are produced in the alpha channel of the off-screen buffer and copied to separate textures in preparation for rendering the next plane. If render-to-texture is supported, the overhead cost of copying textures could be saved.

3. Clear the alpha channel of the novel view.
   The alpha channel of the novel view corresponds to the current slicing plane. It should be erased before processing the next plane. We achieve this by clearing the output frame buffer with the color channels disabled for writing.

All these operations are executed on the graphics board. No copying operation from the frame buffer to main memory is required. Rendering slicing planes textured by alpha maps is a trivial task for modern graphics hardware.

7.5 System Performance and Results

Like other algorithms presented in this thesis, the hardware-accelerated algorithm for novel view synthesis of photo hulls is integrated into our distributed system (see Chapter 3). The client computers are responsible for synchronized video acquisition as well as real-time image processing tasks, such as foreground/background segmentation, silhouette contour extraction and distance transformation. The server is a P4 1.7GHz processor machine equipped with a GeForce FX 5800 Ultra graphics card. The resolution of the rendered novel view is set to 320×240 pixels. The bounding rectangle enclosing the photo hull occupies about one third of the entire viewport. As the object moves, the space is discretized into 30-60 slicing planes orthogonal to the novel viewing direction. The fragment computation code for slicing plane rendering is written in Cg [MGAK03] and compiled into a fragment program [Opea] that can be directly executed on graphics hardware.

We have carried out our experiments using three datasets. One consists of the real image data of a toy puppy. Eight reference cameras are placed in front of the puppy and span about 120° along an arc. In Figure 7.5, the comparison between the visual hull and the photo hull shows that the photo hull-based algorithm is capable of recovering general concave regions and generating more realistic novel views.

The second dataset is a synthetic animation sequence of a girl performing Kungfu. Eight cameras are evenly distributed and along the circumference around the girl. Figure 7.6(a) illustrates the configuration and Figure 7.6(b) shows a novel view of the photo hull. We generate slicing planes using the fixed bounding volume and the effective bounding volume, respectively. The rendering speed of the latter is approximately 2.5 times higher.

As the third dataset, real video sequences are recorded from eight surrounding Firewire cameras to make a parody of the bullet time scene in the movie “The Matrix”. A snapshot rendered from these videos is presented in Figure 7.6(c).

For the *Kungfu girl* and *Matrix parody* datasets, all reference views are used during rendering. Three of them are the active views for most novel views. The rendering speed is about 2 fps. For the puppy dataset, the reference views are closer to each other and the number of active views ranges from 5-8. In this case, we need more time to update the visibility maps. Therefore, the rendering speed decreases to 1.7 fps.

The *image-based photo hull* technique (IBPH) [SSH03] can be regarded as a software implementation of our algorithm. In order to compare the performance with it, we apply our algorithm to a subset of the puppy dataset and keep the rendering configuration as close as possible to theirs. Five reference views are

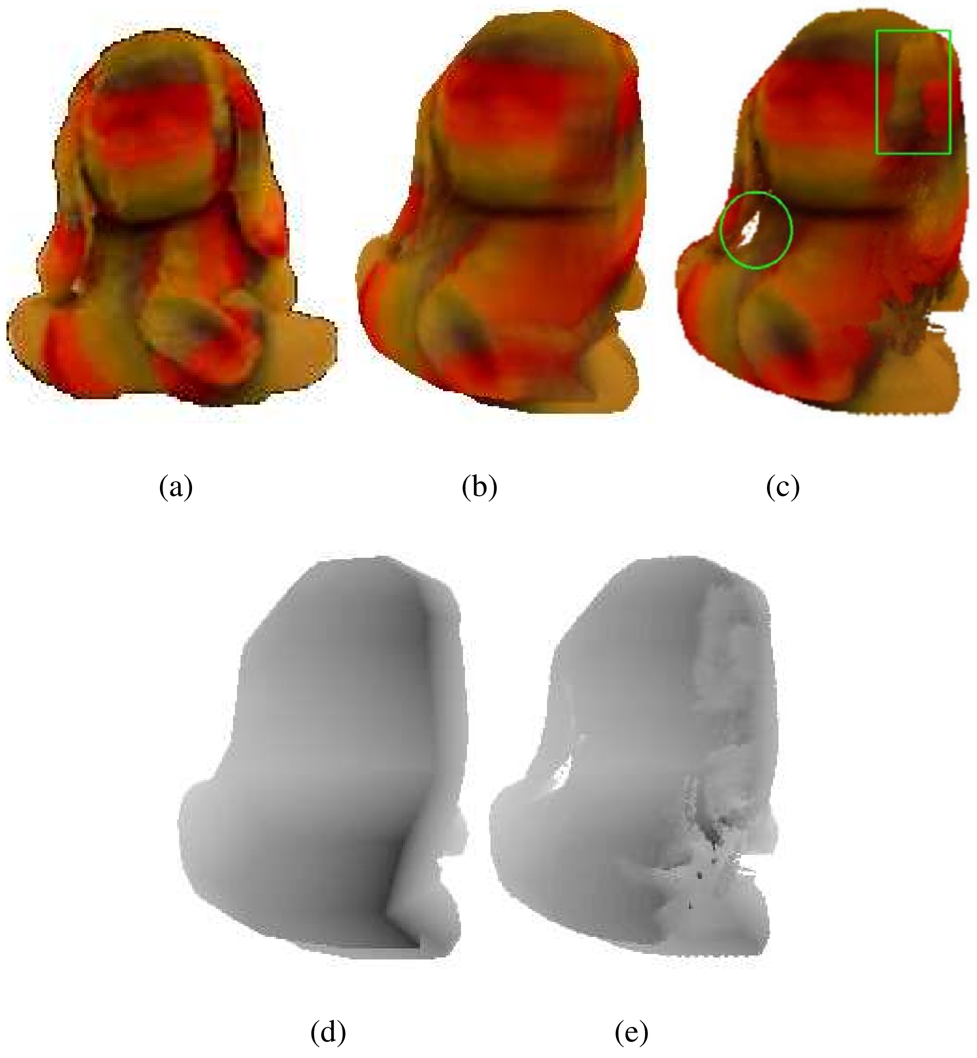
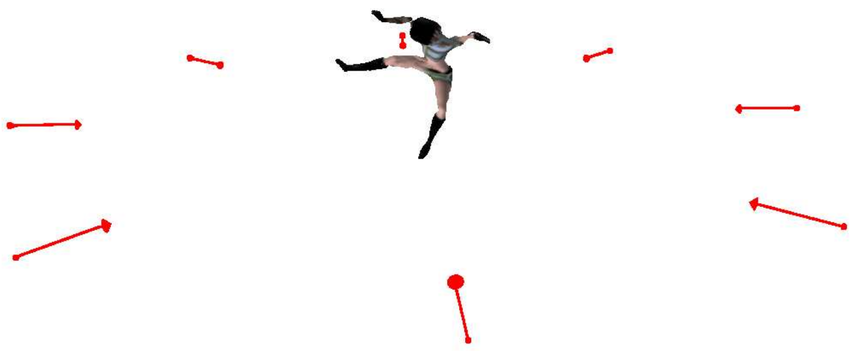
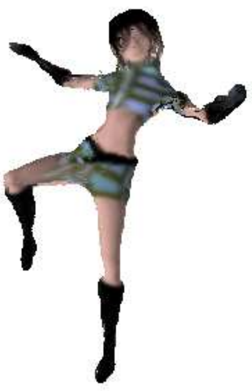
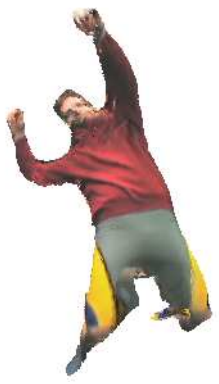


Figure 7.5: Comparison of the rendering results of the visual hull and the photo hull. **(a)** One of the eight input images. **(b)** A novel view of the visual hull of the puppy. **(c)** A novel view of the photo hull generated with our hardware-accelerated algorithm. In the circle, the empty space between the left ear and the body are correctly carved away. Also note that the boxed region is less blurry than the corresponding region in (b). **(d)** and **(e)** are the depth maps for (b) and (c), respectively. Note the fine details revealed in (e).



(a)





(b) (c)

Figure 7.6: Photo hull rendering results generated from eight surrounding reference views. **(a)** The arrangement of the cameras **(b)** A novel view from the *Kungfu girl* synthetic sequence. **(c)** A novel view from the *matrix parody* real video sequence.

used, and the novel viewpoint is specified such that all views are active. While the IBPH runs at 0.4 fps on a machine with dual 2GHz processors, we achieve frame rates of 2.7 fps, approximately 7 times faster. Notice that in the IBPH work some higher frame rates are obtained by computing down-sampled photo hulls in the novel view (For instance, 6.8 fps for a grid size of 4×4 pixels). For fair comparison, we do not use them here.

The number of reference images supported by our algorithm is restricted by the maximum number of texture units available on the graphics hardware. However, we expect it will not take long to see the debut of powerful graphics hardware with more texture units. Another factor influencing the performance of our algorithm is that the graphics card that we use does not have full support of branch instructions. Therefore, in our fragment program, final color evaluation has to be carried out not only on the photo-consistent fragments but also on rejected fragments. This inefficiency will disappear with the latest graphics cards featuring real branch instruction support (for example, NVIDIA's GeForce FX 6800 series), and the performance of the proposed rendering algorithm will be notably improved.

7.6 Summary

In this chapter we have presented a hardware-accelerated algorithm to synthesize novel views of photo hulls from a set of images or videos. By exploiting color information, our algorithm is able to recover concave parts of an object. We generate novel views without explicit photo hull reconstruction by rendering a set of slicing planes depending on the novel viewpoint. Graphics hardware is extensively employed to constrain discretization space, to check the photo-consistency of fragments and to maintain the visibility maps. Thanks to hardware acceleration, the rendering algorithm runs at interactive frame rates, and is much faster than a software implementation.

Chapter 8

Conclusions and Future Work

In this thesis we have presented a set of new algorithms for interactive and real-time novel view synthesis.

First, we have shown that visual hulls can assist in the depth-from-stereo technique by limiting the disparity range effectively. The improved depth reconstruction leads to better visual quality of the rendered novel views.

Second, scene objects are approximated as visual hulls. An innovative technique has been proposed to use graphics hardware features to reconstruct visual hulls. We have explained a basic algorithm to synthesize novel views of visual hulls, and then extended it so that more reference views can be used as input in case of limited graphics hardware resources.

Third, we have combined our new visual hull reconstruction technique with hardware-accelerated CSG reconstruction to obtain an optimal balance between speed and quality. For the hybrid novel view synthesis algorithm, the color values of visual hulls are computed by a per-fragment view-dependent texture mapping technique, which improves rendering quality further.

Finally, we adopt photo hulls to represent scene objects in order to achieve more accurate reconstruction and even better rendering quality. We create novel views of photo hulls by rendering a stack of planes through the working volume. The photo-consistency of each rasterized fragment is checked during rendering. Visibilities with respect to reference views are fully taken into account. To reduce rendering cost, visual hulls are employed to limit the number of planes as well as the area of each plane.

Conclusions

Based on the experiences obtained during the development of the above algorithms, the following conclusions can be drawn.

1. Today's graphics hardware is a powerful tool to improve performance of novel view synthesis algorithms. Nowadays, GPUs are becoming more and more flexible and programmable. The processing power of the GPU is growing at a faster pace than that of the CPU. The GPU is very good at processing streaming vector data like geometry primitives and images. Fortunately, the input data for novel view synthesis algorithms are also of that kind. By migrating more computation onto the GPU, CPU resources can be saved for other tasks. Graphics hardware is extensively exploited in the algorithms that we have presented. Depending on the number of reference views and the expected rendering quality, the performance of the algorithms ranges from interactive to real-time frame rates.

2. The visual hull representation is quite useful for fast novel view synthesis. As presented in the rendering results, a visual hull reconstructed from eight surrounding reference views has already provided an acceptable approximation to the true shape of the 3D object. The reconstructed visual hull can be directly rendered to generate novel views. Alternatively, other reconstruction and rendering algorithms can employ the visual hull to improve speed and/or quality.

3. View-dependent 3D reconstruction is suitable for on-line novel view synthesis of dynamic objects. On-line systems process input data streams on the fly. As such, explicit reconstruction of a complete 3D model is unnecessary for novel view synthesis. In addition, view-dependent reconstruction is performed during the rendering process, which is in most cases

accelerated by graphics hardware. The algorithms presented in Chapter 5–7 all perform view-dependent reconstruction and prove to be very efficient.

Future Work

Despite recent progress in fast novel view synthesis, there remain plenty of issues to be addressed and many interesting areas to be explored. This section discusses some of them and suggests possible research directions.

- **Accurate foreground/background segmentation**
  When the visual hull is used for novel view synthesis either as the main representation or as the auxiliary one, the final rendering quality heavily depends on whether the foreground objects can be accurately segmented from the background. However, in practice, it is hard to achieve this in a complex environment. Even under controlled environment conditions like our video studio, the segmentation may still exhibit noise. To tackle this problem, Grauman *et al.* assume that some prior knowledge about the object is known. They devise a learning-based method to formulate the silhouette extraction problem in a Bayesian framework [GSD03]. When lighting conditions change significantly, gradient information could be exploited to stabilize the segmentation [JSS02].

- **Beyond the visual hull**
  Although visual hulls are good approximations to true 3D objects and they can be rendered very efficiently by graphics hardware, ever-increasing demand on high-quality rendering calls for more accurate models. As we have presented in Chapter 4 and Chapter 7, the depth-from-stereo and the shape-from-photo-consistency techniques provide two ways to incorporate color information in order to reconstruct surface details. Other variations of these two techniques could also be combined with shape-from-silhouette techniques. Time-domain extensions [MG04], Differential methods [SSH02], global optimization methods [KZ02] and prob-

abilistic methods [BDC01, BFK02] deserve further investigation. For complex objects like plants and hairs or natural phenomena like fire, tailored algorithms can be designed when considering their specific features [IM04, RMD04]. Some active 3D reconstruction methods such as *shape-from-structured-light* could also help to refine visual hulls [KTS02].

- **Beyond re-rendering from novel viewpoints**
  In this thesis, we restrict ourselves to dealing with the problem of re-rendering from novel viewpoints. Actually, relighting or reanimating real objects in a scene are challenging problems as well. Recovering reflectance properties of object surfaces would make it possible to relight the object under different lighting conditions. In an augmented reality system where real and virtual objects are mixed, it is very important to maintain consistent illumination. Most existing algorithms to recover reflectance properties rely on an accurate object model known beforehand [RH01]. In case of an unknown object, the problem is ill-posed. However, we can subdivide the problem into two stages: geometry reconstruction and reflectance reconstruction, and then find the optimal solution iteratively. Notice that the geometry reconstruction should take into account non-Lambertian surfaces [JSY03, YPW03, ZPQ04]. Re-animating real objects requires recovering motion represented as scene flow [VBK02], rigid transformations or articulated rigid transformations [CTMS03, CBK03a]. Then motion could be interpolated, re-defined or re-targeted to other objects. If the class of object is known, a generic model could simplify the task dramatically. An attempt has also been made to recover geometry, reflectance and motion simultaneously by using a *surfel* representation [CK01]. All these methods require a large amount of computational power, and are not ready for real-time applications yet.

- **Graphics hardware**
  For the visual hull rendering algorithm presented in Chapter 6, a depth map for each reference view is needed when projecting multiple color textures on visual hulls. For the photo hull rendering algorithm presented

in Chapter 7, a visibility map for each active reference view is created when evaluating photo-consistency and rendering on each slice plane. In both cases, the same geometry primitives are rendered several times from multiple viewpoints. Current graphics hardware has to use multi-pass rendering. If future graphics hardware is able to perform this in parallel, speed and scalability of the above two rendering algorithms will be greatly improved. The SLI (Scalable Link Interface) multi-GPU technology developed by NVIDIA has already taken the first step in this direction. Graphics hardware is not limited to accelerating the algorithms presented in this thesis. One should also think of new ways to employ latest graphics hardware to design new algorithms or to speed up existing ones.

- **Evaluation and validation**
  In the recent years, many novel view synthesis algorithms have been presented. Thorough analysis and evaluation is desperately needed to recommend application-specific solutions. One can evaluate rendering results objectively by comparison to ground truth image data. On the other hand, perception and psychology come into play when human experiments are carried out to assess realism of synthesized novel views. Algorithm and system design will benefit from the thorough consideration of the subjective aspects.

- **Practical system issues**
  If more reference views and higher resolution is desired, network transfer and data compression turn into the limiting factors, which should be considered in the system implementation. For example, currently we transfer the color and silhouette mask information of all reference views to the central server. Actually, given a destination viewpoint, color information from those views which only contribute to back-facing surfaces is not needed for the final rendering. We can take advantage of this fact to devise a view-dependent image transfer scheme for reducing network bandwidth. To mix novel views of real and virtual objects and allow real-time interaction between them, new paradigms of human-computer interaction are

required. A recent system shows some promising results [HLS04]. Meanwhile, tracking devices and wearable displays could be integrated into the system. Live interaction with virtual objects or virtual representations of persons at distant places would enable new ways of learning, playing and communicating. Furthermore, it provides a great feeling of immersion as well as a lot of fun!

Bibliography

[ABUU00] Advanced Network and Services, Brown University, University of North Carolina at Chapel Hill, and University of Pennsylvania (Current participants). National tele-immersion initiative, 1998–2000. http://www.advanced.org/teleimmersion2.html. 3

[ATI] ATI Technologies Inc. ATI_fragment_shader OpenGL extension. http://oss.sgi.com/projects/ogl-sample/registry/ATI/fragment_shader.txt. 30, 70

[Bau74] Baumgart, B. G. *Geometric modeling for computer vision*. Ph.D. thesis, Stanford University, October 1974. 17, 20

[Bek] Bekaert, P. boundary representation library. http://breplibrary.sourceforge.net/. 49

[BFK02] Bhotika, R., Fleet, D., and Kutulakos, K. A probabilistic theory of occupancy and emptiness. In *6th European Conference on Computer Vision (ECCV 2002)*, volume III, pages 112–132. May-June 2002. 25, 114

[Bic94] Bichsel, M. Segmenting simply connected moving objects in a static scene. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 16(11):1138–1142, November 1994. 41

[Bli77] Blinn, J. F. Models of light reflection for computer synthesized pictures. In *SIGGRAPH 1977*, pages 192–198. July 1977. 28

[BS03] Bonfort, T. and Sturm, P. Voxel carving for specular surfaces. In *9th International Conference on Computer Vision (ICCV 2003)*, pages 591–596. October 2003. 25

[Bor86] Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371, 1986. 90, 104

[BB97] Boyer, E. and Berger, M.-O. 3D surface reconstruction using occluding contours. *International Journal of Computer Vision*, 22(3):219–233, 1997. 18

[BDC01] Broadhurst, A., Drummond, T., and Cipolla, R. A probabilistic framework for space carving. In *8th International Conference on Computer Vision (ICCV 2001)*, volume I, pages 388–393. July 2001. 25, 114

[BBH03] Brown, M., Burschka, D., and Hager, G. Advances in computational stereo. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(8):993–1008, August 2003. 14

[BBM$^+$01] Buehler, C., Bosse, M., McMillan, L., Gortler, S. J., and Cohen, M. F. Unstructured lumigraph rendering. In *SIGGRAPH 2001*, pages 425–432. August 2001. 91

[CK01] Carceroni, R. and Kutulakos, K. Multi-view scene capture by surfel sampling: from video streams to non-rigid 3D motion, shape and reflectance. In *8th International Conference on Computer Vision (ICCV 2001)*, volume II, pages 60–67. July 2001. 114

[Car00] Carmack, J. *zfail* stenciled shadow volume rendering. Unpublished correspondence. http://developer.nvidia.com/attach/5628, Early 2000. 86

[CTMS03] Carranza, J., Theobalt, C., Magnor, M. A., and Seidel, H.-P. Free-viewpoint video of human actors. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):569–577, July 2003. 114

[CR01] CBS Broadcasting Inc. and Robotics Institute of Carnegie Mellon University. Eye vision, January 2001. http://www.ri.cmu.edu/events/sb35/tksuperbowl.html. 2

[CCST00] Chai, J.-X., Chan, S.-C., Shum, H.-Y., and Tong, X. Plenoptic sampling. In *SIGGRAPH 2000*, pages 307–318. July 2000. 9

[Che95] Chen, S. E. QuickTime VR – an image-based approach to virtual environment navigation. In *SIGGRAPH 1995*, pages 29–38. August 1995. 7

[CW93] Chen, S. E. and Williams, L. View interpolation for image synthesis. In *SIGGRAPH 1993*, pages 279–288. August 1993. 9

[CBK03a] Cheung, K.-M., Baker, S., and Kanade, T. Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture. In *2003 Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, volume I, pages 77–83. June 2003. 114

[CBK03b] Cheung, K.-M., Baker, S., and Kanade, T. Visual hull alignment and refinement across time: A 3D reconstruction algorithm combining shape-from-silhouette with stereo. In *2003 Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, volume II, pages 375–382. June 2003. 21

[CKBH00] Cheung, K.-M., Kanade, T., Bouguet, J.-Y., and Holler, M. A real time system for robust 3D voxel reconstruction of human motions. In *2000 IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 714–720. June 2000. 19

[Chh01] Chhabra, V. *Reconstructing specular objects with image based rendering using color caching*. Master's thesis, Worcester Polytechnic Institute, 2001. 25

[CB92] Cipolla, R. and Blake, A. Surface shape from the deformation of apparent contours. *International Journal of Computer Vision*, 9(2):83–112, 1992. 18

[CH02] Coconu, L. and Hege, H.-C. Hardware-accelerated point-based rendering of complex scenes. In *13th Eurographics workshop on Rendering*, pages 43–52. June 2002. 56

[CL96] Curless, B. and Levoy, M. A volumetric method for building complex models from range images. In *SIGGRAPH'96 Proceedings*, pages 303–312. August 1996. 15

[DMM$^+$00] Daniilidis, K., Mulligan, J., McKendall, R., Kamberova, G., Schmid, D., and Bajcsy, R. *Real-Time 3D Tele-immersion*, pages 253–266. Kluwer Academic Publishers, 2000. 13

[dBV99] de Bonet, J. and Viola, P. Roxels: Responsibility weighted 3d volume reconstruction. In *7th International Conference on Computer Vision (ICCV 1999)*, volume I, pages 418–425. September 1999. 25

[DBY98] Debevec, P. E., Borshukov, G., and Yu, Y. Efficient view-dependent image-based rendering with projective texture-mapping. In *9th Eurographics Rendering Workshop*, pages 105–116. June 1998. 15, 68, 88, 91

[DTM96] Debevec, P. E., Taylor, C. J., and Malik, J. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH 1996*, pages 11–20. August 1996. 11, 60, 89

[DUD$^+$04] Dennedy, D., Urmson, C., Douxchamps, D., Peters, G., Ronneberger, O., and Evers, T. 1394-based DC control library, version 0.9.5, 2004. http://sourceforge.net/projects/libdc1394. 39

[DA89] Dhond, U. R. and Aggarwal, J. K. Structure from stereo – A review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1489–1510, 1989. 14

[DP73] Douglas, D. H. and Peucker, T. K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, December 1973. 41

[Dye01] Dyer, C. R. Volumetric scene reconstruction from multiple views. In *Foundations of Image Understanding*, pages 469–489. Kluwer Academic Publishers, 2001. 20, 26

[ESG99] Eisert, P., Steinbach, E., and Girod, B. Multi-hypothesis, volumetric reconstruction of 3-D objects from multiple calibrated camera views. In *International Conference on Acoustics Speech and Signal Processing, ICASSP 1999*, pages 3509–3512. March 1999. 25

[ESG00] Eisert, P., Steinbach, E., and Girod, B. Automatic reconstruction of stationary 3-D objects from multiple uncalibrated camera views. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(2):261–277, March 2000. 25

[Eve02] Everitt, C. Interacive order-independent transparency. http://developer.nvidia.com/object/Interactive_Order_Transparency.html, 2002. 84

[Fau93] Faugeras, O. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993. 13, 46

[FHM$^+$93] Faugeras, O., Hotz, B., Mathieu, H., Vieville, T., Z., Z., Fua, P., Theron, E., Laurent, M., Berry, G., Vuillemin, J., Bertin, P., and Proy, C. Real time correlation based stereo: algorithm implementations and applications. Technical Report 2013, INRIA, 1993. 13, 46

[FB03] Franco, J.-S. and Boyer, E. Exact polyhedral visual hulls. In *14th British Machine Vision Conference (BMVC 2003)*, pages 329–338. September 2003. 20

[FB95] Fromherz, T. and Bichsel, M. Multiple depth and normal maps for shape from multiple views and visual cues. In *ISPRS (International Society for Photogrammetry and Remote Sensing): Intercommission Workshop 'From Pixels to Sequences'*, pages 186–194. 1995. 24

[Fua93] Fua, P. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6:35–49, 1993. 13, 48

[FRT97] Fusiello, A., Roberto, V., and Trucco, E. Efficient stereo with multiple windowing. In *1997 Conference on Computer Vision and Pattern Recognition (CVPR 1997)*, pages 858–863. June 1997. 12, 48

[GW87] Giblin, P. and Weiss, R. Reconstruction of surfaces from profiles. In *1st International Conference on Computer Vision (ICCV 1987)*, pages 136–144. June 1987. 18

[GM03] Goldlücke, B. and Magnor, M. Real-time microfacet billboarding for free-viewpoint video rendering. In *2003 International Conference on Image Processing (ICIP 2003)*, volume 3, pages 713–716. September 2003. 22

[GSD03] Grauman, K., Shakhnarovich, G., and Darrell, T. A bayesian approach to image-based visual hull reconstruction. In *2003 Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, volume I, pages 187–194. June 2003. 113

[GKMV03] Guha, S., Krishnan, S., Munagala, K., and Venkat, S. Application of the two-sided depth test to CSG rendering. In *2003 Symposium* *on Interactive 3D Rendering*, pages 177–180. April 2003. 82, 83, 84

[HZ00] Hartley, R. and Zisserman, A. *Multiple view geometry in computer vision*. Cambridge University Press, 2000. 37

[HLS04] Hasenfratz, J.-M., Lapierre, M., and Sillion, F. A real-time system for full body interaction. In *10th Eurographics Symposium on Virtual Environments*, pages 147–156. June 2004. 39, 116

[Hei99] Heidrich, W. *High-quality Shading and Lighting for Hardware-accelerated Rendering*. Ph.D. thesis, University of Erlangen, Computer Graphics Group, 1999. 88

[HH02] Hidalgo, E. and Hubbold, R. J. Hybrid geometric-image-based-rendering. *Computer Graphics Forum (Eurographics 2002)*, 21(3):471–482, September 2002. 56

[HP] HP Corporation. HP_occclusion_test OpenGL extension. http://oss.sgi.com/projects/ogl-sample/registry/HP/occlusion_test.txt. 86

[IM04] Ihrke, I. and Magnor, M. Volumetric reconstruction and interactive rendering of trees from photographs. In *Eurographics Symposium on Computer Animation 2004*, pages 367–375. September 2004. 114

[Int] Intel Corporation. OpenCV library beta 3.1 for Linux. http://www.intel.com/research/mrl/research/opencv/. 39, 52

[IS02] Isidoro, J. and Sclaroff, S. Stochastic mesh-based multiview reconstruction. In *1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT 2002)*, pages 568–577. June 2002. 25

[JSS02] Javed, O., Shafique, K., and Shah, M. A hierarchical approach to robust background subtraction using color and gradient information. In *IEEE Workshop on Motion and Video Computing 2002*, pages 22–27. December 2002. 113

[JSY03] Jin, H.-L., Soatto, S., and Yezzi, A. Multi-view stereo beyond lambert. In *2003 Conference on Computer Vision and Pattern Recognition. (CVPR 2003)*, pages 171–178. June 2003. 114

[KTS02] Kampel, M., Tosovic, S., and Sablatnig, R. Octree-based fusion of shape from silhouette and shape from structured light. In *1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT 2002)*, pages 754–757. June 2002. 114

[Kan94] Kanade, T. Development of a video-rate stereo machine. In *1994 DARPA Image Understanding Workshop (IUW 1994)*, pages 549–558. November 1994. 13

[KNR95] Kanade, T., Narayanan, P. J., and Rander, P. W. Virtualized reality: concepts and early results. In *IEEE Workshop on Representation of Visual Scenes*, pages 69–76. June 1995. 38

[KS01] Kautz, J. and Seidel, H.-P. Hardware accelerated displacement mapping for image based rendering. In *Graphics Interface 2001*, pages 61–70. June 2001. 14

[KHV04] Kück, H., Heidrich, W., and Vogelgsang, C. Shape from contours and multiple stereo – a hierarchical, mesh-based approach. In *1st Canadian Conference on Computer and Robot Vision (CRV 2004)*, pages 76–83. May 2004. 25

[KBR04] Kessenich, J., Baldwin, D., and Rost, R. *The OpenGL Shading Language v1.10.59*. 3DLabs Inc., April 2004. http:// oss.sgi.com/ projects/ ogl-sample/ registry/ ARB/ GLSLang Spec.Full.1.10.59.pdf. 31

[KZ02] Kolmogorov, V. and Zabih, R. Multi-camera scene reconstruction via graph cuts. In *7th European Conference on Computer Vision*, volume III, pages 82–96. May-June 2002. 25, 113

[KZ01] Komolgorov, V. and Zabih, R. Computing visual correspondence with occlusions using graph cuts. In *8th International Conference on Computer Vision (ICCV 2001)*, volume II, pages 508–515. July 2001. 13

[KS99] Kutulakos, K. and Seitz, S. A theory of shape by space carving. In *7th International Conference on Computer Vision (ICCV 1999)*, pages 307–314. July 1999. 5, 9, 23, 24, 97

[Kut00] Kutulakos, K. N. Approximate N-view stereo. In *6th European Conference on Computer Vision (ECCV 2000)*, pages 67–83. July 2000. 25

[Lau94] Laurentini, A. The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 16(2):150–162, February 1994. 4, 5, 9, 16

[LBP01] Lazebnik, S., Boyer, E., and Ponce, J. On computing exact visual hulls of solids bounded by smooth surfaces. In *2001 IEEE Conference on Computer Vision and Pattern Recognition*, volume I, pages 156–151. December 2001. 21

[LH96] Levoy, M. and Hanrahan, P. Light field rendering. In *SIGGRAPH 1996*, pages 31–42. August 1996. 8

[LMS03a] Li, M., Magnor, M., and Seidel, H.-P. Hardware-accelerated visual hull reconstruction and rendering. In *Graphics Interface 2003*, pages 65–71. June 2003. 5, 75

[LMS03b] Li, M., Magnor, M., and Seidel, H.-P. Improved hardware-accelerated visual hull rendering. In *8th Fall Workshop on Vision, Modeling, and Visualization (VMV 2003)*, pages 151–158. November 2003. 5

[LMS03c] Li, M., Magnor, M., and Seidel, H.-P. Online accelerated rendering of visual hulls in real scenes. *Journal of WSCG*, 11(2):290–297, February 2003. 5

[LMS04a] Li, M., Magnor, M., and Seidel, H.-P. Handware-accelerated rendering of photo hulls. *Computer Graphics Forum (Eurographics 2004)*, 23(3):635–642, September 2004. 5

[LMS04b] Li, M., Magnor, M., and Seidel, H.-P. A hybrid hardware-accelerated algorithm for high quality rendering of visual hulls. In *Graphics Interface 2004*, pages 41–48. May 2004. 5

[LSMS02] Li, M., Schirmacher, H., Magnor, M., and Seidel, H.-P. Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes. In *5th Conference on Multimedia Signal Processing (MMSP 2002)*, pages 9–12. December 2002. 5

[Lok01] Lok, B. Online model reconstruction for interactive virtual environments. In *2001 Symposium on Interactive 3D Graphics*, pages 69–72. March 2001. 20, 22, 26, 78, 79, 104

[Low91] Lowe, D. Fitting parameterized three-dimensional models to images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(5):441–450, May 1991. 11

[MG04] Magnor, M. and Goldlücke, B. Spacetime-coherent geometry reconstruction from multiple video streams. In *2nd International Symposium on 3D Data Processing Visualization and Transmission (3DPVT 2004)*, pages 1–6. September 2004. 113

[MGAK03] Mark, W. R., Glanville, R. S., Akeley, K., and Kilgard, M. J. Cg: a system for programming graphics hardware in a C-like language. In *SIGGRAPH 2003*, pages 896–907. July 2003. 31, 93, 107

[MA83] Martin, W. and Aggarwal, J. Volumetric descriptions of objects from multiple views. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 5(2):150–158, March 1983. 18

[MT02] Matsuyama, T. and Takai, T. Generation, visualization, and editing of 3D video. In *1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT 2002)*, pages 234–245. June 2002. 19, 20, 79

[MBR$^+$00] Matusik, W., Buehler, C., Raskar, R., Gortler, S. J., and McMillan, L. Image-based visual hulls. In *SIGGRAPH 2000*, pages 369–374. July 2000. 4, 21, 22, 65, 78, 79

[MBM01] Matusik, W., Bueler, C., and McMillan, L. Polyhedral visual hulls for real-time rendering. In *12th Eurographics Rendering Workshop*, pages 115–125. June 2001. 4, 20, 21, 22, 65, 78, 79

[McM97] McMillan, L. *An Image-Based Approach to Three-Dimensional Computer Graphics*. Ph.D. thesis, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 1997. 15, 45

[MC04] Megyesi, Z. and Chetverikov, D. Affine propagation for surface reconstruction in wide baseline stereo. In *17th Internation Conference on Pattern Recognition (ICPR 2004)*, volume 4, pages 76–79. August 2004. 12

[Mic04a] Microsoft Corporation. Direct3D Reference, 2004. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/reference/d3d/d3dreference.asp. 26, 71

[Mic04b] Microsoft Corporation. HLSL Shader Reference, 2004. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/reference/hlslreference/hlslreference.asp. 31

[MKKJ96] Moezzi, S., Katkere, A., Kuramura, D. Y., and Jain, R. Reality modeling and visualization from multiple video sequences. *IEEE Computer Graphics and Applications*, 16(6):58–63, November 1996. 19, 22

[NRK98] Narayanan, P., Rander, P., and Kanade, T. Constructing virtual worlds using dense stereo. In *6th International Conference on Computer Vision (ICCV 1998)*, pages 3–10. January 1998. 13

[NVIa] NVIDIA Corporation. EXT_stencil_two_side OpenGL extension. http://oss.sgi.com/projects/ogl-sample/registry/EXT/stencil _two_side.txt. 86

[NVIb] NVIDIA Corporation. NV_vertex_program3 OpenGL extension. http://oss.sgi.com/projects/ogl-sample/registry/NV/vertex_prog-ram3.txt. 30

[NVIc] NVIDIA Corporation. NV_register_combiners OpenGL extension. http://oss.sgi.com/projects/ogl-sample/registry/NV/register _combiners.txt. 30, 69, 73

[OK85] Ohta, Y. and Kanade, T. Stereo by intra- and inter-scanline search using dynamic programming. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 7(2):139–154, March 1985. 13

[OK93] Okutomi, M. and Kanade, T. A multiple-baseline stereo. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(4):353–363, April 1993. 13

[OBM00] Oliveira, M. M., Bishop, G., and McAllister, D. Relief texture mapping. In *SIGGRAPH 2000*, pages 359–368. July 2000. 15

[Opea] OpenGL Architectural Review Board. ARB_fragment_program OpenGL extension. http://oss.sgi.com/projects/ogl-sample/regis-try/ARB/fragment_program.txt. 30, 56, 71, 92, 107

[Opeb] OpenGL Architectural Review Board. ARB_point_sprite OpenGL extension. http://oss.sgi.com/projects/ogl-sample/registry/ARB/point_sprite.txt. 56

[Opec] OpenGL Architectural Review Board. ARB_vertex_program OpenGL extension. http://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_program.txt. 30, 92

[Oped] OpenGL Architectural Review Board. WGL_ARB_pbuffer OpenGL extension. http://oss.sgi.com/projects/ogl-sample/registry/ARB/wgl_pbuffer.txt. 29

[PHL$^+$98] Pighin, F., Hecker, J., Lischinski, D., Szeliski, R., and Salesin, D. H. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH 1998*, pages 75–84. July 1998. 91

[Poia] Point Grey Research. Ladybug. http://www.ptgrey.com/products/ladybug/index.html. 8

[Poib] Point Grey Research. Digiclops stereo vision systems. http://www.ptgrey.com/products/digiclops/index.html. 13

[PKG99] Pollefeys, M., Koch, R., and Gool, L. V. Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters. *International Journal of Computer Vision*, 32(1):7–25, 1999. 37

[PEL$^+$00] Popescu, V., Eyles, J., Lastra, A., Steinhurst, J., England, N., and Nyland, L. The WarpEngine: an architecture for the post-polygonal age. In *SIGGRAPH 2000*, pages 433–442. July 2000. 15, 55

[Pot87] Potmesil, M. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1):1–29, October 1987. 18

[PTVF92] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992. 36

[PD98] Prock, A. C. and Dyer, C. R. Towards real-time voxel coloring. In *1998 DARPA Image Understanding Workshop (IUW 1998)*, pages 315–321. November 1998. 25

[PCD$^+$97] Pulli, K., Cohen, M., Duchamp, T., Hoppe, H., Shapiro, L., and Stuetzle, W. View-based rendering: Visualizing real objects from scanned range and color data. In *8th Eurographics Workshop on Rendering*, pages 23–34. June 1997. 89, 91

[RH01] Ramamoorthi, R. and Hanrahan, P. A signal-processing framework for inverse rendering. In *SIGGRAPH 2001*, pages 117–128. August 2001. 114

[RMD04] Reche, A., Martin, I., and Drettakis, G. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3):720–727, August 2004. 114

[RCMS99] Rocchini, C., Cignomi, P., Montani, C., and Scopigno, R. Multiple textures stitching and blending on 3D objects. In *10th Eurographics Rendering Workshop*, pages 119–130. June 1999. 15

[SBS02] Sainz, M., Bagherzadeh, N., and Susin, A. Hardware accelerated voxel carving. In *1st Ibero-American Symposium in Computer Graphics*, pages 289–297. July 2002. 25

[SK99] Saito, H. and Kanade, T. Shape reconstruction in projective grid space from large number of images. In *1999 Conference on Computer Vision and Pattern Recognition (CVPR 1999)*, pages 49–54. June 1999. 19

[SS02] Scharstein, D. and Szeliski, R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 48(1-3):7–42, April-June 2002. 14

[Sch03] Schirmacher, H. *Efficient Acquisition, Representation, and Rendering of Light Fields*. Ph.D. thesis, University of Saarland, 2003. 9

[SLS01] Schirmacher, H., Li, M., and Seidel, H.-P. On-the-fly processing of generalized lumigraphs. *Computer Graphics Forum (Eurographics 2001)*, 20(3):C165–C173, September 2001. 4, 5, 9, 13, 14

[SA04] Segal, M. and Akeley, K. *The OpenGL Graphics System: A Specification (Version 2.0)*. Silicon Graphics, Inc., http://www.opengl.org/documentation/specs/version2_0/glspec20.pdf, September 2004. 26, 51

[SKvW$^+$92] Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., and Haeberli, P. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH 1992*, pages 249–252. July 1992. 15, 28, 88, 89

[SD97] Seitz, S. M. and Dyer, C. R. Photorealistic scene reconstruction by voxel coloring. In *1997 Conference on Computer Vision and Pattern Recognition (CVPR 1997)*, pages 1067–1073. June 1997. 24

[SGHS98] Shade, J., Gortler, S., He, L.-W., and Szeliski, R. Layered depth images. In *SIGGRAPH 1998*, pages 231–242. July 1998. 15, 55

[SH99] Shum, H.-Y. and He, L.-W. Rendering with concentric mosaics. In *SIGGRAPH 1999*, pages 299–306. August 1999. 9

[Sil] Silicon Graphics Inc. GLX_SGIX_pbuffer OpenGL extension. http://oss.sgi.com/projects/ogl-sample/registry/SGIX/pbuffer.txt. 29

[SSH02] Slabaugh, G. G., Schafer, R. W., and Hans, M. C. Multi-resolution space carving using level sets methods. In *2002 International Conference on Image Processing (ICIP 2002)*. September 2002. 25, 113

[SSH03] Slabaugh, G. G., Schafer, R. W., and Hans, M. C. Image-based photo hulls for fast and photo-realistic new view synthesis. *Real-Time Imaging*, 9(5):347–360, October 2003. 25, 98, 100, 107

[SCMS01] Slabaugh, G. G., Culbertson, B., Malzbender, T., and Schafer, R. A survey of methods for volumetric scene reconstruction from photographs. In *Volume Graphics 2001*, pages 81–100. June 2001. 20, 26

[SB96] Smith, A. R. and Blinn, J. F. Blue screen matting. In *SIGGRAPH 1996*, pages 259–268. August 1996. 41

[Son03] Sony Computer Entertainment Europe. Eyetoys: Play, 2003. http://www.scee.presscentre.com/Software/detail.asp?SoftwareID =24. 3

[Son] Sony Corporation. Brochure of Sony DFW-500 digital video camera. http:// www.sony.net/ Products/ISP/pdf/guide/GDFWV 500E.pdf. 33

[SLJ98] Stewart, N., Leach, G., and John, S. An improved Z-buffer CSG rendering algorithm. In *1998 Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 25–30. August 1998. 82, 84

[SP98] Sullivan, S. and Ponce, J. Automatic model construction and pose estimation from photographs using triangular splines. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20(10):1091–1097, 1998. 21

[Sze93] Szeliski, R. Rapid octree construction from image sequences. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 58(1):23–32, July 1993. 19

[SS97] Szeliski, R. and Shum, H.-Y. Creating full view panoramic image mosaics and environment maps. In *SIGGRAPH 1997*, pages 251–258. August 1997. 7, 89

[TLMS03] Theobalt, C., Li, M., Magnor, M., and Seidel, H.-P. A flexible and versatile studio for multi-view video recording. In *Vision, Video and Graphics 2003*, pages 9–16. July 2003. 5

[TMSS02] Theobalt, C., Magnor, M., Schüler, P., and Seidel, H.-P. Combining 2D feature tracking and volume reconstruction for online video-based human motion capture. In *Pacific Graphics 2002*, pages 96–103. October 2002. 19

[TV98] Trucco, E. and Verri, A. *Introductory techniques for 3-D computer vision*, chapter 7. Stereopsis. Prentice Hall, 1998. 11

[Tsa86] Tsai, R. Y. An efficient and accurate camera calibration technique for 3-D machine vision. In *1986 Conference on Computer Vision and Pattern Recognition (CVPR 1986)*, pages 364–374. June 1986. 36

[TL94] Turk, G. and Levoy, M. Zippered polygon meshes from range images. In *SIGGRAPH 1994*, pages 311–318. September 1994. 14

[VBK02] Vedula, S., Baker, S., and Kanade, T. Spatio-temporal view interpolation. In *13th Eurographics Workshop on Rendering*, pages 65–76. June 2002. 114

[VBSK00] Vedula, S., Baker, S., Seitz, S., and Kanade, T. Shape and motion carving in 6D. In *2000 Conference on Computer Vision and Pattern Recognition (CVPR 2000)*, volume II, pages 592–598. June 2000. 25

[VRSK98] Vedula, S., Rander, P., Saito, H., and Kanade, T. Modeling, combining, and rendering dynamic real-world events from image sequences. In *4th Conference on Virtual Systems and Multimedia (VSMM 1998)*, pages 326–332. November 1998. 60

[Vid] Videre Design. Small vision system. http://www.ai.sri.com/~konolige/svs/svs.htm. 13

[Wie96] Wiegand, T. F. Interactive rendering of CSG models. *Computer Graphics Forum (Eurographics 1996)*, 15(4):249–261, August 1996. 22, 82, 84

[WL03] Wilburn, B. and Levoy, M. Standford multi-camera array, 2003. http://graphics.stanford.edu/projects/array. 9

[Wil78] Williams, L. Casting curved shadows on curved surfaces. In *SIGGRAPH 1978*, pages 270–274. August 1978. 88

[WAA$^+$00] Wood, D. N., Azuma, D. I., Aldinger, K., Curless, B., Duchamp, T., Salesin, D. H., and Stuetzle, W. Surface light fields for 3D photography. In *SIGGRAPH 2000*, pages 287–296. July 2000. 9

[WLSG02] Würmlin, S., Lamboray, E., Staadt, O., and Gross, M. 3D video recorder. In *Pacific Graphics 2002*, pages 96–103. October 2002. 21, 22

[WD] Wynn, C. and Dietrich, S. Cube maps. http://developer.nvidia.com/object/cube_maps.html. 92

[YSK$^+$02] Yamazaki, S., Sagawa, R., Kawasaki, H., Ikeuchi, K., and Sakauchi, M. Microfacet billboarding. In *13th Eurographics workshop on Rendering*, pages 169–180. June 2002. 14

[YEBM02] Yang, J. C., Everett, M., Buehler, C., and McMillan, L. A real-time distributed light field camera. In *13th Eurographics workshop on Rendering*, pages 77–86. June 2002. 4, 9, 39

[YP03] Yang, R. and Pollefeys, M. Multi-resolution real-time stereo on commodity graphics hardware. In *2003 Conference on Computer Vision and Pattern Recognition*, pages 211–220. June 2003. 13, 103

[YPW03] Yang, R., Pollefeys, M., and Welch, G. Dealing with textureless regions and specular highlights: A progressive space carving scheme using a novel photo-consistency measure. In *9th International Conference on Computer Vision (ICCV 2003)*, pages 576–584. October 2003. 23, 114

[YWB02] Yang, R., Welch, G., and Bishop, G. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Pacific Graphics 2002*, pages 225–235. October 2002. 4, 25, 98, 100, 102

[ZW94] Zabih, R. and Woodfill, J. Non-parametric local transforms for computing visual correspondence. In *3rd European Conference on Computer Vision*, pages 150–158. May 1994. 12

[ZPQ04] Zeng, G., Paris, S., and Quan, L. Robust carving for non-lambertian objects. In *Proceedings of International Conference on Pattern Recognition (ICPR 2004)*, volume 3, pages 119–122. August 2004. 114

[ZC04] Zhang, C. and Chen, T. A survey on image-based rendering: Representation, sampling and compression. *Signal Processing: Image Communication*, 19(1):1–28, January 2004. 2

[ZCS03] Zhang, L., Curless, B., and Seitz, S. M. Spacetime stereo: Shape recovery for dynamic scenes. In *2003 Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, volume II, pages 367–374. June 2003. 12

[ZPvG01] Zwicker, M., Pfister, H., van Baar, J., and Gross, M. Surface splatting. In *SIGGRAPH 2001*, pages 371–378. August 2001. 14